

Projeto e Análise de Algoritmos

Algoritmos gulosos

Lehilton Pedrosa

Primeiro Semestre de 2023

Algoritmos gulosos

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ algoritmo de programação dinâmica:
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ algoritmo guloso:
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Problemas e subproblemas

Vamos estudar algoritmos gulosos

- ▶ decompomos um problema em vários subproblemas
- ▶ de novo, um deles corresponde à **subestrutura ótima**
- ▶ mas podemos escolher a subestrutura eficientemente

Comparando as estratégias

- ▶ **algoritmo de programação dinâmica:**
 1. primeiro resolvemos todos os subproblemas
 2. depois decidimos o subproblema ótimo
- ▶ **algoritmo guloso:**
 1. primeiro escolhemos o subproblema ótimo
 2. depois resolvemos apenas esse subproblema

Algoritmos gulosos

Ideia:

- ▶ realizamos uma sequência de passos
- ▶ a cada passo, fazemos uma escolha

Premissas dos algoritmos gulosos:

- ▶ a escolha é aquela que parece ser a melhor no momento
- ▶ essa escolha é denominada **escolha gulosa**
- ▶ e é feita de acordo com um **critério guloso**

Nem sempre um algoritmo guloso encontra uma solução ótima, mas para vários problemas é possível mostrar que sim

Algoritmos gulosos

Ideia:

- ▶ realizamos uma sequência de passos
- ▶ a cada passo, fazemos uma escolha

Premissas dos algoritmos gulosos:

- ▶ a escolha é aquela que parece ser a melhor no momento
- ▶ essa escolha é denominada **escolha gulosa**
- ▶ e é feita de acordo com um **critério guloso**

Nem sempre um algoritmo guloso encontra uma solução ótima, mas para vários problemas é possível mostrar que sim

Algoritmos gulosos

Ideia:

- ▶ realizamos uma sequência de passos
- ▶ a cada passo, fazemos uma escolha

Premissas dos algoritmos gulosos:

- ▶ a escolha é aquela que parece ser a melhor no momento
- ▶ essa escolha é denominada **escolha gulosa**
- ▶ e é feita de acordo com um **critério guloso**

Nem sempre um algoritmo guloso encontra uma solução ótima, mas para vários problemas é possível mostrar que sim

Algoritmos gulosos

Ideia:

- ▶ realizamos uma sequência de passos
- ▶ a cada passo, fazemos uma escolha

Premissas dos algoritmos gulosos:

- ▶ a escolha é aquela que parece ser a melhor no momento
- ▶ essa escolha é denominada **escolha gulosa**
- ▶ e é feita de acordo com um **critério guloso**

Nem sempre um algoritmo guloso encontra uma solução ótima, mas para vários problemas é possível mostrar que sim

Algoritmos gulosos

Ideia:

- ▶ realizamos uma sequência de passos
- ▶ a cada passo, fazemos uma escolha

Premissas dos algoritmos gulosos:

- ▶ a escolha é aquela que parece ser a melhor no momento
- ▶ essa escolha é denominada **escolha gulosa**
- ▶ e é feita de acordo com um **critério guloso**

Nem sempre um algoritmo guloso encontra uma solução ótima, mas para vários problemas é possível mostrar que sim

Algoritmos gulosos

Ideia:

- ▶ realizamos uma sequência de passos
- ▶ a cada passo, fazemos uma escolha

Premissas dos algoritmos gulosos:

- ▶ a escolha é aquela que parece ser a melhor no momento
- ▶ essa escolha é denominada **escolha gulosa**
- ▶ e é feita de acordo com um **critério guloso**

Nem sempre um algoritmo guloso encontra uma solução ótima, mas para vários problemas é possível mostrar que sim

Algoritmos gulosos

Ideia:

- ▶ realizamos uma sequência de passos
- ▶ a cada passo, fazemos uma escolha

Premissas dos algoritmos gulosos:

- ▶ a escolha é aquela que parece ser a melhor no momento
- ▶ essa escolha é denominada **escolha gulosa**
- ▶ e é feita de acordo com um **critério guloso**

Nem sempre um algoritmo guloso encontra uma solução ótima, mas para vários problemas é possível mostrar que sim

Algoritmos gulosos

Ideia:

- ▶ realizamos uma sequência de passos
- ▶ a cada passo, fazemos uma escolha

Premissas dos algoritmos gulosos:

- ▶ a escolha é aquela que parece ser a melhor no momento
- ▶ essa escolha é denominada **escolha gulosa**
- ▶ e é feita de acordo com um **critério guloso**

Nem sempre um algoritmo guloso encontra uma solução ótima, mas para vários problemas é possível mostrar que sim

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum **subconjunto** A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum **critério guloso**
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum subconjunto A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum critério guloso
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum **subconjunto** A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum **critério guloso**
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum **subconjunto** A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum **critério guloso**
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum **subconjunto** A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum **critério guloso**
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum **subconjunto** A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum **critério guloso**
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum **subconjunto** A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum **critério guloso**
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum **subconjunto** A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum **critério guloso**
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum **subconjunto** A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum **critério guloso**
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Uma receita para algoritmos gulosos

Vários problemas têm a seguinte estrutura

- ▶ existe um conjunto de elementos E
- ▶ uma solução é algum **subconjunto** A^* de E

Um estratégia genérica:

1. faça $A \leftarrow \emptyset$
2. enquanto A não é solução viável:
 - (a) escolha um elemento e com algum **critério guloso**
 - (b) certifique-se de que existe solução A^* contendo $A \cup \{e\}$
 - (c) faça $A \leftarrow A \cup \{e\}$
3. devolva A

Algoritmos gulosos

- ▶ Seleção de atividades

Seleção de atividades

Considere n atividades executadas em certo lugar

- ▶ podem ser palestras, reuniões em um sala etc.
- ▶ denote essas atividades por $S = \{a_1, \dots, a_n\}$

Duração das atividades

- ▶ a atividade a_i começa no tempo s_i e termina no tempo f_i
- ▶ assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são **compatíveis** se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- ▶ podem ser palestras, reuniões em um sala etc.
- ▶ denote essas atividades por $S = \{a_1, \dots, a_n\}$

Duração das atividades

- ▶ a atividade a_i começa no tempo s_i e termina no tempo f_i
- ▶ assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são **compatíveis** se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- ▶ podem ser palestras, reuniões em um sala etc.
- ▶ denote essas atividades por $S = \{a_1, \dots, a_n\}$

Duração das atividades

- ▶ a atividade a_i começa no tempo s_i e termina no tempo f_i
- ▶ assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são **compatíveis** se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- ▶ podem ser palestras, reuniões em um sala etc.
- ▶ denote essas atividades por $S = \{a_1, \dots, a_n\}$

Duração das atividades

- ▶ a atividade a_i começa no tempo s_i e termina no tempo f_i
- ▶ assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Exemplo

Duas atividades a_i e a_j são **compatíveis** se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- ▶ podem ser palestras, reuniões em um sala etc.
- ▶ denote essas atividades por $S = \{a_1, \dots, a_n\}$

Duração das atividades

- ▶ a atividade a_i começa no tempo s_i e termina no tempo f_i
- ▶ assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Exemplo

Duas atividades a_i e a_j são **compatíveis** se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- ▶ podem ser palestras, reuniões em um sala etc.
- ▶ denote essas atividades por $S = \{a_1, \dots, a_n\}$

Duração das atividades

- ▶ a atividade a_i começa no tempo s_i e termina no tempo f_i
- ▶ assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são **compatíveis** se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Seleção de atividades

Considere n atividades executadas em certo lugar

- ▶ podem ser palestras, reuniões em um sala etc.
- ▶ denote essas atividades por $S = \{a_1, \dots, a_n\}$

Duração das atividades

- ▶ a atividade a_i começa no tempo s_i e termina no tempo f_i
- ▶ assim, ela deve ser realizada no intervalo $[s_i, f_i)$

Definição

Duas atividades a_i e a_j são **compatíveis** se os intervalos $[s_i, f_i)$ e $[s_j, f_j)$ são disjuntos.

Problema de seleção de atividades

Problema

Entrada:

- ▶ conjunto de atividades $S = \{a_1, \dots, a_n\}$
- ▶ tempos de início s e de termino f

Solução:

- ▶ subconjunto A de atividades compatíveis

Objetivo:

- ▶ maximizar $|A|$

Problema de seleção de atividades

Problema

Entrada:

- ▶ conjunto de atividades $S = \{a_1, \dots, a_n\}$
- ▶ tempos de início s e de termino f

Solução:

- ▶ subconjunto A de atividades compatíveis

Objetivo:

- ▶ maximizar $|A|$

Problema de seleção de atividades

Problema

Entrada:

- ▶ conjunto de atividades $S = \{a_1, \dots, a_n\}$
- ▶ tempos de início s e de termino f

Solução:

- ▶ subconjunto A de atividades compatíveis

Objetivo:

- ▶ maximizar $|A|$

Problema de seleção de atividades

Problema

Entrada:

- ▶ conjunto de atividades $S = \{a_1, \dots, a_n\}$
- ▶ tempos de início s e de termino f

Solução:

- ▶ subconjunto A de atividades compatíveis

Objetivo:

- ▶ maximizar $|A|$

Problema de seleção de atividades

Problema

Entrada:

- ▶ conjunto de atividades $S = \{a_1, \dots, a_n\}$
- ▶ tempos de início s e de termino f

Solução:

- ▶ subconjunto A de atividades compatíveis

Objetivo:

- ▶ maximizar $|A|$

Problema de seleção de atividades

Problema

Entrada:

- ▶ conjunto de atividades $S = \{a_1, \dots, a_n\}$
- ▶ tempos de início s e de termino f

Solução:

- ▶ subconjunto A de atividades compatíveis

Objetivo:

- ▶ maximizar $|A|$

Problema de seleção de atividades

Problema

Entrada:

- ▶ conjunto de atividades $S = \{a_1, \dots, a_n\}$
- ▶ tempos de início s e de termino f

Solução:

- ▶ subconjunto A de atividades compatíveis

Objetivo:

- ▶ maximizar $|A|$

Problema de seleção de atividades

Problema

Entrada:

- ▶ conjunto de atividades $S = \{a_1, \dots, a_n\}$
- ▶ tempos de início s e de termino f

Solução:

- ▶ subconjunto A de atividades compatíveis

Objetivo:

- ▶ maximizar $|A|$

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- ▶ as atividades estão em ordem de tempo de término
- ▶ iremos usar essa ordem em seguida

Exemplos de soluções

- ▶ $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- ▶ $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- ▶ $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e ótimas

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- ▶ as atividades estão em ordem de **tempo de término**
- ▶ iremos usar essa ordem em seguida

Exemplos de soluções

- ▶ $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- ▶ $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- ▶ $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e **ótimas**

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- ▶ as atividades estão em ordem de **tempo de término**
- ▶ iremos usar essa ordem em seguida

Exemplos de soluções

- ▶ $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- ▶ $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- ▶ $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e **ótimas**

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- ▶ as atividades estão em ordem de **tempo de término**
- ▶ iremos usar essa ordem em seguida

Exemplos de soluções

- ▶ $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- ▶ $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- ▶ $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e **ótimas**

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- ▶ as atividades estão em ordem de **tempo de término**
- ▶ iremos usar essa ordem em seguida

Exemplos de soluções

- ▶ $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- ▶ $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- ▶ $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e **ótimas**

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- ▶ as atividades estão em ordem de **tempo de término**
- ▶ iremos usar essa ordem em seguida

Exemplos de soluções

- ▶ $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- ▶ $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- ▶ $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e **ótimas**

Uma instância

Os tempos de início e de término são

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- ▶ as atividades estão em ordem de **tempo de término**
- ▶ iremos usar essa ordem em seguida

Exemplos de soluções

- ▶ $\{a_1, a_2\}$ e $\{a_1, a_3\}$ são pares incompatíveis
- ▶ $\{a_1, a_4\}$ e $\{a_3, a_9, a_{11}\}$ são viáveis, mas não ótimas
- ▶ $\{a_1, a_4, a_8, a_{11}\}$ e $\{a_2, a_4, a_9, a_{11}\}$ são viáveis e **ótimas**

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de término
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de **término**
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de **término**
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de **término**
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de **término**
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de **término**
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de **término**
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de **término**
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de **término**
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Definições preliminares

Supomos que $f_1 \leq f_2 \leq \dots \leq f_n$

- ▶ ou seja, as atividades estão em ordem de **término**
- ▶ se não estiverem, podemos ordenar

Para um par (i, j) , defina $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- ▶ atividades que começam depois que a_i termina
- ▶ e além disso terminam antes que a_j inicie

Considere atividades dummies

- ▶ atividade a_0 com $f_0 = 0$ e atividade a_{n+1} com $s_{n+1} = \infty$
- ▶ assim S_{ij} está definido para todo $0 \leq i, j \leq n+1$
- ▶ e o conjunto de todas atividades é $S = S_{0,n+1}$

Subestrutura ótima

Considere uma solução ótima para as atividades em S_{ij}

- ▶ suponha que a_k esteja nessa solução ótima
- ▶ as demais atividades da solução devem estar
 - ▶ antes do início de a_k
 - ▶ depois do término de a_k

Descobrimos uma **subestrutura ótima**

- ▶ queremos uma solução ótima para o conjunto S_{ik}
- ▶ além de uma solução ótima para o conjunto S_{kj}

Mas **não** sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima para as atividades em S_{ij}

- ▶ suponha que a_k esteja nessa solução ótima
- ▶ as demais atividades da solução devem estar
 - ▶ antes do início de a_k
 - ▶ depois do término de a_k

Descobrimos uma **subestrutura ótima**

- ▶ queremos uma solução ótima para o conjunto S_{ik}
- ▶ além de uma solução ótima para o conjunto S_{kj}

Mas **não** sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima para as atividades em S_{ij}

- ▶ suponha que a_k esteja nessa solução ótima
- ▶ as demais atividades da solução devem estar
 - ▶ **antes** do início de a_k
 - ▶ **depois** do término de a_k

Descobrimos uma **subestrutura ótima**

- ▶ queremos uma solução ótima para o conjunto S_{ik}
- ▶ além de uma solução ótima para o conjunto S_{kj}

Mas **não** sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima para as atividades em S_{ij}

- ▶ suponha que a_k esteja nessa solução ótima
- ▶ as demais atividades da solução devem estar
 - ▶ **antes** do início de a_k
 - ▶ **depois** do término de a_k

Descobrimos uma **subestrutura ótima**

- ▶ queremos uma solução ótima para o conjunto S_{ik}
- ▶ além de uma solução ótima para o conjunto S_{kj}

Mas **não** sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima para as atividades em S_{ij}

- ▶ suponha que a_k esteja nessa solução ótima
- ▶ as demais atividades da solução devem estar
 - ▶ **antes** do início de a_k
 - ▶ **depois** do término de a_k

Descobrimos uma **subestrutura ótima**

- ▶ queremos uma solução ótima para o conjunto S_{ik}
- ▶ além de uma solução ótima para o conjunto S_{kj}

Mas **não** sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima para as atividades em S_{ij}

- ▶ suponha que a_k esteja nessa solução ótima
- ▶ as demais atividades da solução devem estar
 - ▶ **antes** do início de a_k
 - ▶ **depois** do término de a_k

Descobrimos uma **subestrutura ótima**

- ▶ queremos uma solução ótima para o conjunto S_{ik}
- ▶ além de uma solução ótima para o conjunto S_{kj}

Mas **não** sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima para as atividades em S_{ij}

- ▶ suponha que a_k esteja nessa solução ótima
- ▶ as demais atividades da solução devem estar
 - ▶ **antes** do início de a_k
 - ▶ **depois** do término de a_k

Descobrimos uma **subestrutura ótima**

- ▶ queremos uma solução ótima para o conjunto S_{ik}
- ▶ além de uma solução ótima para o conjunto S_{kj}

Mas **não** sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima para as atividades em S_{ij}

- ▶ suponha que a_k esteja nessa solução ótima
- ▶ as demais atividades da solução devem estar
 - ▶ **antes** do início de a_k
 - ▶ **depois** do término de a_k

Descobrimos uma **subestrutura ótima**

- ▶ queremos uma solução ótima para o conjunto S_{ik}
- ▶ além de uma solução ótima para o conjunto S_{kj}

Mas **não** sabemos qual é a atividade a_k na solução ótima!

Subestrutura ótima

Considere uma solução ótima para as atividades em S_{ij}

- ▶ suponha que a_k esteja nessa solução ótima
- ▶ as demais atividades da solução devem estar
 - ▶ **antes** do início de a_k
 - ▶ **depois** do término de a_k

Descobrimos uma **subestrutura ótima**

- ▶ queremos uma solução ótima para o conjunto S_{ik}
- ▶ além de uma solução ótima para o conjunto S_{kj}

Mas **não** sabemos qual é a atividade a_k na solução ótima!

Usando programação dinâmica

Definimos o seguinte **subproblema**

- ▶ considere um par (i, j) para $0 \leq i, j \leq n + 1$
- ▶ defina $c[i, j]$ o **valor de uma solução ótima** para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i, j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- ▶ o valor ótimo corresponde à entrada $c[0, n + 1]$
- ▶ é fácil calcular usando **programação dinâmica** (exercício)

Usando programação dinâmica

Definimos o seguinte **subproblema**

- ▶ considere um par (i, j) para $0 \leq i, j \leq n + 1$
- ▶ defina $c[i, j]$ o **valor de uma solução ótima** para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i, j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- ▶ o valor ótimo corresponde à entrada $c[0, n + 1]$
- ▶ é fácil calcular usando **programação dinâmica** (exercício)

Usando programação dinâmica

Definimos o seguinte **subproblema**

- ▶ considere um par (i, j) para $0 \leq i, j \leq n + 1$
- ▶ defina $c[i, j]$ o **valor de uma solução ótima** para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i, j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- ▶ o valor ótimo corresponde à entrada $c[0, n + 1]$
- ▶ é fácil calcular usando **programação dinâmica** (exercício)

Usando programação dinâmica

Definimos o seguinte **subproblema**

- ▶ considere um par (i, j) para $0 \leq i, j \leq n + 1$
- ▶ defina $c[i, j]$ o **valor de uma solução ótima** para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i, j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- ▶ o valor ótimo corresponde à entrada $c[0, n + 1]$
- ▶ é fácil calcular usando **programação dinâmica** (exercício)

Usando programação dinâmica

Definimos o seguinte **subproblema**

- ▶ considere um par (i, j) para $0 \leq i, j \leq n + 1$
- ▶ defina $c[i, j]$ o **valor de uma solução ótima** para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i, j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- ▶ o valor ótimo corresponde à entrada $c[0, n + 1]$
- ▶ é fácil calcular usando **programação dinâmica** (exercício)

Usando programação dinâmica

Definimos o seguinte **subproblema**

- ▶ considere um par (i, j) para $0 \leq i, j \leq n + 1$
- ▶ defina $c[i, j]$ o **valor de uma solução ótima** para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i, j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- ▶ o valor ótimo corresponde à entrada $c[0, n + 1]$
- ▶ é fácil calcular usando **programação dinâmica** (exercício)

Usando programação dinâmica

Definimos o seguinte **subproblema**

- ▶ considere um par (i, j) para $0 \leq i, j \leq n + 1$
- ▶ defina $c[i, j]$ o **valor de uma solução ótima** para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i, j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- ▶ o valor ótimo corresponde à entrada $c[0, n + 1]$
- ▶ é fácil calcular usando **programação dinâmica** (exercício)

Usando programação dinâmica

Definimos o seguinte **subproblema**

- ▶ considere um par (i, j) para $0 \leq i, j \leq n + 1$
- ▶ defina $c[i, j]$ o **valor de uma solução ótima** para a instância do problema com atividades S_{ij}

Podemos computar $c[i, j]$ com a recorrência

$$c[i, j] = \begin{cases} 0 & \text{se } S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{se } S_{ij} \neq \emptyset \end{cases}$$

Voltando ao problema original

- ▶ o valor ótimo corresponde à entrada $c[0, n + 1]$
- ▶ é fácil calcular usando **programação dinâmica** (exercício)

Algumas observações

- ▶ na programação dinâmica, testamos **todas** atividades a_k
- ▶ em um algoritmo guloso, queremos escolher apenas **uma**

Como escolher de forma gulosa?

- ▶ queremos uma atividade que consome menos “recursos”
- ▶ as atividades estão ordenadas por tempo de término
- ▶ a_1 é a escolha que deixa mais tempo para outras atividades

Simplificando

Algumas observações

- ▶ na programação dinâmica, testamos **todas** atividades a_k
- ▶ em um algoritmo guloso, queremos escolher apenas **uma**

Como escolher de forma gulosa?

- ▶ queremos uma atividade que consome menos “recursos”
- ▶ as atividades estão ordenadas por tempo de término
- ▶ a_1 é a escolha que deixa mais tempo para outras atividades

Simplificando

Algumas observações

- ▶ na programação dinâmica, testamos **todas** atividades a_k
- ▶ em um algoritmo guloso, queremos escolher apenas **uma**

Como escolher de forma gulosa?

- ▶ queremos uma atividade que consome menos “recursos”
- ▶ as atividades estão ordenadas por tempo de término
- ▶ a_1 é a escolha que deixa mais tempo para outras atividades

Simplificando

Algumas observações

- ▶ na programação dinâmica, testamos **todas** atividades a_k
- ▶ em um algoritmo guloso, queremos escolher apenas **uma**

Como escolher de forma gulosa?

- ▶ queremos uma atividade que consome menos “recursos”
- ▶ as atividades estão ordenadas por tempo de término
- ▶ a_1 é a escolha que deixa mais tempo para outras atividades

Simplificando

Algumas observações

- ▶ na programação dinâmica, testamos **todas** atividades a_k
- ▶ em um algoritmo guloso, queremos escolher apenas **uma**

Como escolher de forma gulosa?

- ▶ queremos uma atividade que consome menos “recursos”
- ▶ as atividades estão ordenadas por tempo de término
- ▶ a_1 é a escolha que deixa mais tempo para outras atividades

Simplificando

Algumas observações

- ▶ na programação dinâmica, testamos **todas** atividades a_k
- ▶ em um algoritmo guloso, queremos escolher apenas **uma**

Como escolher de forma gulosa?

- ▶ queremos uma atividade que consome menos “recursos”
- ▶ as atividades estão ordenadas por tempo de término
- ▶ a_1 é a escolha que deixa mais tempo para outras atividades

Simplificando

Algumas observações

- ▶ na programação dinâmica, testamos **todas** atividades a_k
- ▶ em um algoritmo guloso, queremos escolher apenas **uma**

Como escolher de forma gulosa?

- ▶ queremos uma atividade que consome menos “recursos”
- ▶ as atividades estão ordenadas por tempo de término
- ▶ a_1 é a escolha que deixa mais tempo para outras atividades

Depois de escolher a_1 , qual subproblema resta?

- ▶ quemos atividades que começam depois que a_1 termina
- ▶ definimos um novo subproblema $S_k = \{a_i \in S : s_i \geq f_k\}$
- ▶ pela definição da dummy a_0 , o problema original é $S_0 = S$

Propriedade da escolha gulosa

Considere um subproblema S_k não vazio e seja a_m uma atividade em S_k com o menor tempo de término.

Então existe uma solução ótima para S_k que contém a_m .

Depois de escolher a_1 , qual subproblema resta?

- ▶ quemos atividades que começam depois que a_1 termina
- ▶ definimos um novo subproblema $S_k = \{a_i \in S : s_i \geq f_k\}$
- ▶ pela definição da dummy a_0 , o problema original é $S_0 = S$

Propriedade da escolha gulosa

Considere um subproblema S_k não vazio e seja a_m uma atividade em S_k com o menor tempo de término.

Então existe uma solução ótima para S_k que contém a_m .

Escolha gulosa

Depois de escolher a_1 , qual subproblema resta?

- ▶ quemos atividades que começam depois que a_1 termina
- ▶ definimos um novo subproblema $S_k = \{a_i \in S : s_i \geq f_k\}$
- ▶ pela definição da dummy a_0 , o problema original é $S_0 = S$

Considere um subproblema S_k não vazio e seja a_m uma atividade em S_k com o menor tempo de término.

Então existe uma solução ótima para S_k que contém a_m .

Escolha gulosa

Depois de escolher a_1 , qual subproblema resta?

- ▶ quemos atividades que começam depois que a_1 termina
- ▶ definimos um novo subproblema $S_k = \{a_i \in S : s_i \geq f_k\}$
- ▶ pela definição da dummy a_0 , o problema original é $S_0 = S$

Escolha gulosa

Considere um subproblema S_k não vazio e seja a_m uma atividade em S_k com o menor tempo de término.

Então existe uma solução ótima para S_k que contém a_m .

Escolha gulosa

Depois de escolher a_1 , qual subproblema resta?

- ▶ quemos atividades que começam depois que a_1 termina
- ▶ definimos um novo subproblema $S_k = \{a_i \in S : s_i \geq f_k\}$
- ▶ pela definição da dummy a_0 , o problema original é $S_0 = S$

Escolha gulosa

Considere um subproblema S_k não vazio e seja a_m uma atividade em S_k com o **menor tempo de término**.

Então existe uma solução ótima para S_k que contém a_m .

Escolha gulosa

Depois de escolher a_1 , qual subproblema resta?

- ▶ quemos atividades que começam depois que a_1 termina
- ▶ definimos um novo subproblema $S_k = \{a_i \in S : s_i \geq f_k\}$
- ▶ pela definição da dummy a_0 , o problema original é $S_0 = S$

Escolha gulosa

Considere um subproblema S_k não vazio e seja a_m uma atividade em S_k com o **menor tempo de término**.

Então existe uma solução ótima para S_k que contém a_m .

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividade de A' são mutualmente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividades de A' são mutuamente compatíveis

Demonstração da escolha gulosa

Seja A^* uma **solução ótima** para S_k

- ▶ se $a_m \in A^*$, então nada há a fazer
- ▶ então suponha que $a_m \notin A^*$

Vamos criar **outra solução ótima** A' contendo a_m

- ▶ seja $a_i \in A^*$ a atividade com menor f_k
- ▶ defina $A' = A \setminus \{a_i\} \cup \{a_m\}$

Temos que provar que A' é solução ótima

- ▶ é claro que $|A^*| = |A'|$
- ▶ então resta mostrar que A' é viável
- ▶ nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_i
- ▶ daí nenhuma atividade de $A^* \setminus \{a_k\}$ começa antes de f_m
- ▶ assim as atividade de A' são mutualmente compatíveis

A discussão anterior sugere um algoritmo recursivo:

- ▶ suponha que estamos tentando resolver S_k
- ▶ seja a_m a atividade em S_k com o menor tempo de término
- ▶ resolva o subproblema S_m e junte com a_m

Resolvendo recursivamente

A discussão anterior sugere um algoritmo recursivo:

- ▶ suponha que estamos tentando resolver S_k
- ▶ seja a_m a atividade em S_k com o menor tempo de término
- ▶ resolva o subproblema S_m e junte com a_m

Resolvendo recursivamente

A discussão anterior sugere um algoritmo recursivo:

- ▶ suponha que estamos tentando resolver S_k
- ▶ seja a_m a atividade em S_k com o menor tempo de término
- ▶ resolva o subproblema S_m e junte com a_m

Resolvendo recursivamente

A discussão anterior sugere um algoritmo recursivo:

- ▶ suponha que estamos tentando resolver S_k
- ▶ seja a_m a atividade em S_k com o menor tempo de término
- ▶ resolva o subproblema S_m e junte com a_m

Algoritmo recursivo

- ▶ o vetor f está em ordem de tempo de término
- ▶ queremos atividades que começam **depois** que k termina

SELEC-ATIV-GUL-REC(s, f, k, n)

```
    ▷ acha atividade  $a_m$  em  $S_k$  que termina primeiro
1    $m \leftarrow k + 1$ 
2   enquanto  $m \leq n$  e  $s_m < f_k$ 
3        $m \leftarrow m + 1$ 

    ▷ escolhe  $a_m$  e resolve subproblema  $S_m$ 
4   se  $m \leq n$ 
5       devolva  $\{a_m\} \cup \text{SELEC-ATIV-GUL-REC}(s, f, m, n)$ 
6   senão
7       devolva  $\emptyset$ 
```

Análise

- ▶ vemos cada elemento apenas uma vez, daí o tempo é $\Theta(n)$
- ▶ pode ser que precisemos ordenar as atividades

Algoritmo recursivo

- ▶ o vetor f está em ordem de tempo de término
- ▶ queremos atividades que comecem **depois** que k termina

SELEC-ATIV-GUL-REC(s, f, k, n)

```
    ▷ acha atividade  $a_m$  em  $S_k$  que termina primeiro
1    $m \leftarrow k + 1$ 
2   enquanto  $m \leq n$  e  $s_m < f_k$ 
3        $m \leftarrow m + 1$ 

    ▷ escolhe  $a_m$  e resolve subproblema  $S_m$ 
4   se  $m \leq n$ 
5       devolva  $\{a_m\} \cup \text{SELEC-ATIV-GUL-REC}(s, f, m, n)$ 
6   senão
7       devolva  $\emptyset$ 
```

Análise

- ▶ vemos cada elemento apenas uma vez, daí o tempo é $\Theta(n)$
- ▶ pode ser que precisemos ordenar as atividades

Algoritmo recursivo

- ▶ o vetor f está em ordem de tempo de término
- ▶ queremos atividades que começam **depois** que k termina

SELEC-ATIV-GUL-REC(s, f, k, n)

```
    ▷ acha atividade  $a_m$  em  $S_k$  que termina primeiro
1   $m \leftarrow k + 1$ 
2  enquanto  $m \leq n$  e  $s_m < f_k$ 
3       $m \leftarrow m + 1$ 

    ▷ escolhe  $a_m$  e resolve subproblema  $S_m$ 
4  se  $m \leq n$ 
5      devolva  $\{a_m\} \cup \text{SELEC-ATIV-GUL-REC}(s, f, m, n)$ 
6  senão
7      devolva  $\emptyset$ 
```

Análise

- ▶ vemos cada elemento apenas uma vez, daí o tempo é $\Theta(n)$
- ▶ pode ser que precisemos ordenar as atividades

Algoritmo recursivo

- ▶ o vetor f está em ordem de tempo de término
- ▶ queremos atividades que começam **depois** que k termina

SELEC-ATIV-GUL-REC(s, f, k, n)

```
    ▷ acha atividade  $a_m$  em  $S_k$  que termina primeiro
1   $m \leftarrow k + 1$ 
2  enquanto  $m \leq n$  e  $s_m < f_k$ 
3       $m \leftarrow m + 1$ 
    ▷ escolhe  $a_m$  e resolve subproblema  $S_m$ 
4  se  $m \leq n$ 
5      devolva  $\{a_m\} \cup \text{SELEC-ATIV-GUL-REC}(s, f, m, n)$ 
6  senão
7      devolva  $\emptyset$ 
```

Análise

- ▶ vemos cada elemento apenas uma vez, daí o tempo é $\Theta(n)$
- ▶ pode ser que precisemos ordenar as atividades

Algoritmo recursivo

- ▶ o vetor f está em ordem de tempo de término
- ▶ queremos atividades que começam **depois** que k termina

SELEC-ATIV-GUL-REC(s, f, k, n)

```
    ▷ acha atividade  $a_m$  em  $S_k$  que termina primeiro
1   $m \leftarrow k + 1$ 
2  enquanto  $m \leq n$  e  $s_m < f_k$ 
3       $m \leftarrow m + 1$ 

    ▷ escolhe  $a_m$  e resolve subproblema  $S_m$ 
4  se  $m \leq n$ 
5      devolva  $\{a_m\} \cup \text{SELEC-ATIV-GUL-REC}(s, f, m, n)$ 
6  senão
7      devolva  $\emptyset$ 
```

Análise

- ▶ vemos cada elemento apenas uma vez, daí o tempo é $\Theta(n)$
- ▶ pode ser que precisemos ordenar as atividades

Algoritmo recursivo

- ▶ o vetor f está em ordem de tempo de término
- ▶ queremos atividades que começam **depois** que k termina

SELEC-ATIV-GUL-REC(s, f, k, n)

```
    ▷ acha atividade  $a_m$  em  $S_k$  que termina primeiro
1    $m \leftarrow k + 1$ 
2   enquanto  $m \leq n$  e  $s_m < f_k$ 
3        $m \leftarrow m + 1$ 

    ▷ escolhe  $a_m$  e resolve subproblema  $S_m$ 
4   se  $m \leq n$ 
5       devolva  $\{a_m\} \cup \text{SELEC-ATIV-GUL-REC}(s, f, m, n)$ 
6   senão
7       devolva  $\emptyset$ 
```

Análise

- ▶ vemos cada elemento apenas uma vez, daí o tempo é $\Theta(n)$
- ▶ pode ser que precisemos ordenar as atividades

Algoritmo iterativo

Podemos reescrever de maneira iterativa

SELEC-ATIV-GUL-ITER(s, f, n)

```
1   $A \leftarrow \{a_1\}$ 
2   $k \leftarrow 1$ 
3  para  $m \leftarrow 2$  até  $n$ 
4      se  $s_m \geq f_k$ 
5           $A \leftarrow A \cup \{a_m\}$ 
6           $k \leftarrow m$ 
7  devolva  $A$ 
```

Algoritmo iterativo

Podemos reescrever de maneira iterativa

SELEC-ATIV-GUL-ITER(s, f, n)

```
1   $A \leftarrow \{a_1\}$ 
2   $k \leftarrow 1$ 
3  para  $m \leftarrow 2$  até  $n$ 
4      se  $s_m \geq f_k$ 
5           $A \leftarrow A \cup \{a_m\}$ 
6           $k \leftarrow m$ 
7  devolva  $A$ 
```

Algoritmos gulosos

- ▶ Codificação de Huffman

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto C
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto C
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto C
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto C
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto C
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto C
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto C
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto $\mathcal{C}$
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto $\mathcal{C}$
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto $\mathcal{C}$
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto $\mathcal{C}$
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto $\mathcal{C}$
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificações

Queremos representar um **texto**

- ▶ é uma sequência de caracteres de um alfabeto $\mathcal{C}$
- ▶ cada caractere está associado a uma sequência de bits

Tipos de codificação

- ▶ comprimento fixo
 - ▶ cada sequência tem o mesmo número de bits
 - ▶ basta que elas sejam distintas
- ▶ codificação de comprimento variável
 - ▶ as sequências podem ter tamanhos diferentes
 - ▶ são **livres de prefixo**: uma sequência não é prefixo de outra

Tamanho do texto codificado

- ▶ é o número de bits usados para representar o texto
- ▶ codificações diferentes têm tamanhos diferentes

Codificação de tamanho variável

Restrição de prefixo

- ▶ o código de um caractere não pode ser prefixo de outro
- ▶ isso evita que a leitura do texto seja ambígua
- ▶ chamamos a codificação de **livre de prefixo**

Exemplo de codificação ruim

	<i>a</i>	<i>b</i>	<i>c</i>
código	01	1001	100101

- ▶ considere uma sequência de bits 100101
- ▶ a sequência corresponde ao texto *ba* ou ao texto *c*?

Codificação de tamanho variável

Restrição de prefixo

- ▶ o código de um caractere não pode ser prefixo de outro
- ▶ isso evita que a leitura do texto seja ambígua
- ▶ chamamos a codificação de **livre de prefixo**

Exemplo de codificação ruim

	<i>a</i>	<i>b</i>	<i>c</i>
código	01	1001	100101

- ▶ considere uma sequência de bits 100101
- ▶ a sequência corresponde ao texto *ba* ou ao texto *c*?

Codificação de tamanho variável

Restrição de prefixo

- ▶ o código de um caractere não pode ser prefixo de outro
- ▶ isso evita que a leitura do texto seja ambígua
- ▶ chamamos a codificação de **livre de prefixo**

Exemplo de codificação ruim

	<i>a</i>	<i>b</i>	<i>c</i>
código	01	1001	100101

- ▶ considere uma sequência de bits 100101
- ▶ a sequência corresponde ao texto *ba* ou ao texto *c*?

Codificação de tamanho variável

Restrição de prefixo

- ▶ o código de um caractere não pode ser prefixo de outro
- ▶ isso evita que a leitura do texto seja ambígua
- ▶ chamamos a codificação de **livre de prefixo**

Exemplo de codificação ruim

	<i>a</i>	<i>b</i>	<i>c</i>
código	01	1001	100101

- ▶ considere uma sequência de bits 100101
- ▶ a sequência corresponde ao texto *ba* ou ao texto *c*?

Codificação de tamanho variável

Restrição de prefixo

- ▶ o código de um caractere não pode ser prefixo de outro
- ▶ isso evita que a leitura do texto seja ambígua
- ▶ chamamos a codificação de **livre de prefixo**

Exemplo de codificação ruim

	<i>a</i>	<i>b</i>	<i>c</i>
código	01	1001	100101

- ▶ considere uma sequência de bits 100101
- ▶ a sequência corresponde ao texto *ba* ou ao texto *c*?

Codificação de tamanho variável

Restrição de prefixo

- ▶ o código de um caractere não pode ser prefixo de outro
- ▶ isso evita que a leitura do texto seja ambígua
- ▶ chamamos a codificação de **livre de prefixo**

Exemplo de codificação ruim

	<i>a</i>	<i>b</i>	<i>c</i>
código	01	1001	100101

- ▶ considere uma sequência de bits 100101
- ▶ a sequência corresponde ao texto *ba* ou ao texto *c*?

Codificação de tamanho variável

Restrição de prefixo

- ▶ o código de um caractere não pode ser prefixo de outro
- ▶ isso evita que a leitura do texto seja ambígua
- ▶ chamamos a codificação de **livre de prefixo**

Exemplo de codificação ruim

	<i>a</i>	<i>b</i>	<i>c</i>
código	01	1001	100101

- ▶ considere uma sequência de bits **100101**
- ▶ a sequência corresponde ao texto *ba* ou ao texto *c*?

Codificação de tamanho variável

Restrição de prefixo

- ▶ o código de um caractere não pode ser prefixo de outro
- ▶ isso evita que a leitura do texto seja ambígua
- ▶ chamamos a codificação de **livre de prefixo**

Exemplo de codificação ruim

	<i>a</i>	<i>b</i>	<i>c</i>
código	01	1001	100101

- ▶ considere uma sequência de bits 100101
- ▶ a sequência corresponde ao texto *ba* ou ao texto *c*?

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- ▶ com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- ▶ com a codificação de tamanho variável, usamos

$$\underbrace{(45 \cdot 1)}_a + \underbrace{13 \cdot 3}_b + \underbrace{12 \cdot 3}_c + \underbrace{16 \cdot 3}_d + \underbrace{9 \cdot 4}_e + \underbrace{5 \cdot 4}_f \cdot 1.000 = 224.000 \text{ bits}$$

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- ▶ com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- ▶ com a codificação de tamanho variável, usamos

$$\underbrace{(45 \cdot 1)}_a + \underbrace{13 \cdot 3}_b + \underbrace{12 \cdot 3}_c + \underbrace{16 \cdot 3}_d + \underbrace{9 \cdot 4}_e + \underbrace{5 \cdot 4}_f \cdot 1.000 = 224.000 \text{ bits}$$

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- ▶ com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- ▶ com a codificação de tamanho variável, usamos

$$\underbrace{(45 \cdot 1)}_a + \underbrace{13 \cdot 3}_b + \underbrace{12 \cdot 3}_c + \underbrace{16 \cdot 3}_d + \underbrace{9 \cdot 4}_e + \underbrace{5 \cdot 4}_f \cdot 1.000 = 224.000 \text{ bits}$$

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- ▶ com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- ▶ com a codificação de tamanho variável, usamos

$$\underbrace{(45 \cdot 1)}_a + \underbrace{13 \cdot 3}_b + \underbrace{12 \cdot 3}_c + \underbrace{16 \cdot 3}_d + \underbrace{9 \cdot 4}_e + \underbrace{5 \cdot 4}_f \cdot 1.000 = 224.000 \text{ bits}$$

Exemplo

Considere um texto com 100.000 caracteres de $C = \{a, b, c, d, e, f\}$:

	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Tamanho do texto codificado

- ▶ com a codificação de tamanho fixo, usamos

$$3 \cdot 100.000 = 300.000 \text{ bits}$$

- ▶ com a codificação de tamanho variável, usamos

$$\underbrace{(45 \cdot 1)}_a + \underbrace{13 \cdot 3}_b + \underbrace{12 \cdot 3}_c + \underbrace{16 \cdot 3}_d + \underbrace{9 \cdot 4}_e + \underbrace{5 \cdot 4}_f \cdot 1.000 = 224.000 \text{ bits}$$

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Exemplo:

Texto: alfabeto

Objetivo: tabela de frequências

Solução:

Exemplo: codificação de comprimento variável

Objetivo:

Exemplo: minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Exemplo

Caracteres

Tabela de frequências

Solução

Construção da árvore binária

Objetivo

Minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Entrada:

- ▶ alfabeto C
- ▶ tabela de frequências f

Solução:

- ▶ codificação de comprimento variável

Objetivo:

- ▶ minimizar o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Entrada:

- ▶ alfabeto C
- ▶ tabela de frequências f

Solução:

- ▶ codificação de comprimento variável

Objetivo:

- ▶ **minimizar** o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Entrada:

- ▶ alfabeto C
- ▶ tabela de frequências f

Solução:

- ▶ codificação de comprimento variável

Objetivo:

- ▶ **minimizar** o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Entrada:

- ▶ alfabeto C
- ▶ tabela de frequências f

Solução:

- ▶ codificação de comprimento variável

Objetivo:

- ▶ **minimizar** o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Entrada:

- ▶ alfabeto C
- ▶ tabela de frequências f

Solução:

- ▶ codificação de comprimento variável

Objetivo:

- ▶ **minimizar** o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Entrada:

- ▶ alfabeto C
- ▶ tabela de frequências f

Solução:

- ▶ codificação de comprimento variável

Objetivo:

- ▶ **minimizar** o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Entrada:

- ▶ alfabeto C
- ▶ tabela de frequências f

Solução:

- ▶ codificação de comprimento variável

Objetivo:

- ▶ **minimizar** o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Entrada:

- ▶ alfabeto C
- ▶ tabela de frequências f

Solução:

- ▶ codificação de comprimento variável

Objetivo:

- ▶ **minimizar** o tamanho do texto codificado

Codificação de Huffman

Codificação de Huffman

- ▶ problema para a compressão de dados
- ▶ dependendo da aplicação, reduz de 20 a 90%

Problema

Entrada:

- ▶ alfabeto C
- ▶ tabela de frequências f

Solução:

- ▶ codificação de comprimento variável

Objetivo:

- ▶ **minimizar** o tamanho do texto codificado

Representação de codificação

Uma codificação é representada por uma **árvore binária**

- ▶ o filho esquerdo está associado ao bit 0
- ▶ o filho direito está associado ao bit 1
- ▶ as folhas representam os caracteres do alfabeto

A codificação é livre de prefixo

- ▶ um código corresponde a um caminho até a folha
- ▶ prefixos só levam a nós internos
- ▶ então o código de um caractere não é prefixo de outro

Representação de codificação

Uma codificação é representada por uma **árvore binária**

- ▶ o filho esquerdo está associado ao **bit 0**
- ▶ o filho direito está associado ao **bit 1**
- ▶ as folhas representam os caracteres do alfabeto

A codificação é livre de prefixo

- ▶ um código corresponde a um caminho até a folha
- ▶ prefixos só levam a nós internos
- ▶ então o código de um caractere não é prefixo de outro

Representação de codificação

Uma codificação é representada por uma **árvore binária**

- ▶ o filho esquerdo está associado ao **bit 0**
- ▶ o filho direito está associado ao **bit 1**
- ▶ as folhas representam os caracteres do alfabeto

A codificação é livre de prefixo

- ▶ um código corresponde a um caminho até a folha
- ▶ prefixos só levam a nós internos
- ▶ então o código de um caractere não é prefixo de outro

Representação de codificação

Uma codificação é representada por uma **árvore binária**

- ▶ o filho esquerdo está associado ao **bit 0**
- ▶ o filho direito está associado ao **bit 1**
- ▶ as folhas representam os caracteres do alfabeto

A codificação é livre de prefixo

- ▶ um código corresponde a um caminho até a folha
- ▶ prefixos só levam a nós internos
- ▶ então o código de um caractere não é prefixo de outro

Representação de codificação

Uma codificação é representada por uma **árvore binária**

- ▶ o filho esquerdo está associado ao **bit 0**
- ▶ o filho direito está associado ao **bit 1**
- ▶ as folhas representam os caracteres do alfabeto

A codificação é livre de prefixo

- ▶ um código corresponde a um caminho até a folha
- ▶ prefixos só levam a nós internos
- ▶ então o código de um caractere não é prefixo de outro

Representação de codificação

Uma codificação é representada por uma **árvore binária**

- ▶ o filho esquerdo está associado ao **bit 0**
- ▶ o filho direito está associado ao **bit 1**
- ▶ as folhas representam os caracteres do alfabeto

A codificação é livre de prefixo

- ▶ um código corresponde a um caminho até a folha
- ▶ prefixos só levam a nós internos
- ▶ então o código de um caractere não é prefixo de outro

Representação de codificação

Uma codificação é representada por uma **árvore binária**

- ▶ o filho esquerdo está associado ao **bit 0**
- ▶ o filho direito está associado ao **bit 1**
- ▶ as folhas representam os caracteres do alfabeto

A codificação é livre de prefixo

- ▶ um código corresponde a um caminho até a folha
- ▶ prefixos só levam a nós internos
- ▶ então o código de um caractere não é prefixo de outro

Representação de codificação

Uma codificação é representada por uma **árvore binária**

- ▶ o filho esquerdo está associado ao **bit 0**
- ▶ o filho direito está associado ao **bit 1**
- ▶ as folhas representam os caracteres do alfabeto

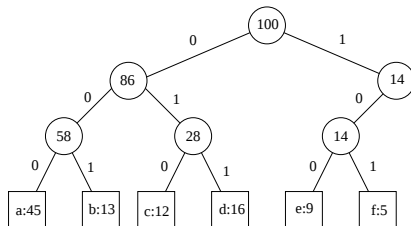
A codificação é livre de prefixo

- ▶ um código corresponde a um caminho até a folha
- ▶ prefixos só levam a nós internos
- ▶ então o código de um caractere não é prefixo de outro

Árvore binária para codificação fixa

Codificação de comprimento fixo

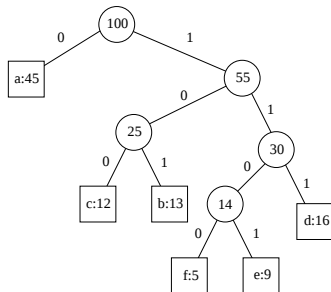
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência	45	13	12	16	9	5
Código fixo	000	001	010	011	100	101



Árvore binária para codificação variável

Codificação de comprimento variável

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência	45	13	12	16	9	5
Código variável	0	101	100	111	1101	1100



Árvores binárias **cheias**

- ▶ são árvores em que cada nó interno tem dois filhos
- ▶ assim, há $|C|$ folhas e $|C| - 1$ nós internos (exercício)

Sempre existe uma codificação ótima que é cheia

- ▶ suponha que há nó x com um único filho y
- ▶ substituimos a subárvore de x pela subárvore de y
- ▶ a nova árvore tem as mesmas folhas
- ▶ o código de cada caractere só pode diminuir

Árvores binárias **cheias**

- ▶ são árvores em que cada nó interno tem dois filhos
- ▶ assim, há $|C|$ folhas e $|C| - 1$ nós internos (exercício)

Sempre existe uma codificação ótima que é cheia

- ▶ suponha que há nó x com um único filho y
- ▶ substituimos a subárvore de x pela subárvore de y
- ▶ a nova árvore tem as mesmas folhas
- ▶ o código de cada caractere só pode diminuir

Árvores binárias **cheias**

- ▶ são árvores em que cada nó interno tem dois filhos
- ▶ assim, há $|C|$ folhas e $|C| - 1$ nós internos (exercício)

Sempre existe uma codificação ótima que é cheia

- ▶ suponha que há nó x com um único filho y
- ▶ substituimos a subárvore de x pela subárvore de y
- ▶ a nova árvore tem as mesmas folhas
- ▶ o código de cada caractere só pode diminuir

Árvores binárias **cheias**

- ▶ são árvores em que cada nó interno tem dois filhos
- ▶ assim, há $|C|$ folhas e $|C| - 1$ nós internos (exercício)

Sempre existe uma codificação ótima que é cheia

- ▶ suponha que há nó x com um único filho y
- ▶ substituimos a subárvore de x pela subárvore de y
- ▶ a nova árvore tem as mesmas folhas
- ▶ o código de cada caractere só pode diminuir

Árvores binárias cheias

- ▶ são árvores em que cada nó interno tem dois filhos
- ▶ assim, há $|C|$ folhas e $|C| - 1$ nós internos (exercício)

Sempre existe uma codificação ótima que é cheia

- ▶ suponha que há nó x com um único filho y
- ▶ substituimos a subárvore de x pela subárvore de y
- ▶ a nova árvore tem as mesmas folhas
- ▶ o código de cada caractere só pode diminuir

Árvores binárias **cheias**

- ▶ são árvores em que cada nó interno tem dois filhos
- ▶ assim, há $|C|$ folhas e $|C| - 1$ nós internos (exercício)

Sempre existe uma codificação ótima que é cheia

- ▶ suponha que há nó x com um único filho y
- ▶ substituimos a subárvore de x pela subárvore de y
- ▶ a nova árvore tem as mesmas folhas
- ▶ o código de cada caractere só pode diminuir

Árvores binárias cheias

- ▶ são árvores em que cada nó interno tem dois filhos
- ▶ assim, há $|C|$ folhas e $|C| - 1$ nós internos (exercício)

Sempre existe uma codificação ótima que é cheia

- ▶ suponha que há nó x com um único filho y
- ▶ substituimos a subárvore de x pela subárvore de y
- ▶ a nova árvore tem as mesmas folhas
- ▶ o código de cada caractere só pode diminuir

Árvores binárias cheias

- ▶ são árvores em que cada nó interno tem dois filhos
- ▶ assim, há $|C|$ folhas e $|C| - 1$ nós internos (exercício)

Sempre existe uma codificação ótima que é cheia

- ▶ suponha que há nó x com um único filho y
- ▶ substituimos a subárvore de x pela subárvore de y
- ▶ a nova árvore tem as mesmas folhas
- ▶ o código de cada caractere só pode diminuir

Estrutura de um nó z

- ▶ o filho esquerdo é denotado por $z.esq$
- ▶ o filho direito é denotado por $z.dir$
- ▶ a frequência dos caracteres na **subárvore** é $z.freq$

Estrutura de um nó z

- ▶ o filho esquerdo é denotado por $z.esq$
- ▶ o filho direito é denotado por $z.dir$
- ▶ a frequência dos caracteres na **subárvore** é $z.freq$

Estrutura de um nó z

- ▶ o filho esquerdo é denotado por $z.esq$
- ▶ o filho direito é denotado por $z.dir$
- ▶ a frequência dos caracteres na **subárvore** é $z.freq$

Estrutura de um nó z

- ▶ o filho esquerdo é denotado por $z.esq$
- ▶ o filho direito é denotado por $z.dir$
- ▶ a frequência dos caracteres na **subárvore** é $z.freq$

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- ▶ caracteres infrequentes estão em folhas mais profundas
- ▶ dois nós pouco frequentes são unidos por um nó interno
- ▶ construímos a árvore de maneira **bottom-up**

Ideia:

- ▶ começar com $|C|$ nós correspondendo aos caracteres
- ▶ juntar os dois nós menos frequentes
- ▶ repetir $|C| - 1$ vezes até sobrar apenas um nó
- ▶ o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- ▶ caracteres infrequentes estão em folhas mais profundas
- ▶ dois nós pouco frequentes são unidos por um nó interno
- ▶ construímos a árvore de maneira **bottom-up**

Ideia:

- ▶ começar com $|C|$ nós correspondendo aos caracteres
- ▶ juntar os dois nós menos frequentes
- ▶ repetir $|C| - 1$ vezes até sobrar apenas um nó
- ▶ o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- ▶ caracteres infrequentes estão em folhas mais profundas
- ▶ dois nós pouco frequentes são unidos por um nó interno
- ▶ construímos a árvore de maneira **bottom-up**

Ideia:

- ▶ começar com $|C|$ nós correspondendo aos caracteres
- ▶ juntar os dois nós menos frequentes
- ▶ repetir $|C| - 1$ vezes até sobrar apenas um nó
- ▶ o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- ▶ caracteres infrequentes estão em folhas mais profundas
- ▶ dois nós pouco frequentes são unidos por um nó interno
- ▶ construímos a árvore de maneira **bottom-up**

Ideia:

- ▶ começar com $|C|$ nós correspondendo aos caracteres
- ▶ juntar os dois nós menos frequentes
- ▶ repetir $|C| - 1$ vezes até sobrar apenas um nó
- ▶ o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- ▶ caracteres infrequentes estão em folhas mais profundas
- ▶ dois nós pouco frequentes são unidos por um nó interno
- ▶ construímos a árvore de maneira **bottom-up**

Ideia:

- ▶ começar com $|C|$ nós correspondendo aos caracteres
- ▶ juntar os dois nós menos frequentes
- ▶ repetir $|C| - 1$ vezes até sobrar apenas um nó
- ▶ o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- ▶ caracteres infrequentes estão em folhas mais profundas
- ▶ dois nós pouco frequentes são unidos por um nó interno
- ▶ construímos a árvore de maneira **bottom-up**

Ideia:

- ▶ começar com $|C|$ nós correspondendo aos caracteres
- ▶ juntar os dois nós menos frequentes
- ▶ repetir $|C| - 1$ vezes até sobrar apenas um nó
- ▶ o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- ▶ caracteres infrequentes estão em folhas mais profundas
- ▶ dois nós pouco frequentes são unidos por um nó interno
- ▶ construímos a árvore de maneira **bottom-up**

Ideia:

- ▶ começar com $|C|$ nós correspondendo aos caracteres
- ▶ juntar os dois nós menos frequentes
- ▶ repetir $|C| - 1$ vezes até sobrar apenas um nó
- ▶ o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- ▶ caracteres infrequentes estão em folhas mais profundas
- ▶ dois nós pouco frequentes são unidos por um nó interno
- ▶ construímos a árvore de maneira **bottom-up**

Ideia:

- ▶ começar com $|C|$ nós correspondendo aos caracteres
- ▶ juntar os dois nós menos frequentes
- ▶ repetir $|C| - 1$ vezes até sobrar apenas um nó
- ▶ o nó restante é a raiz da árvore devolvida

Construindo uma árvore ótima

Vamos adotar a seguinte estratégia

- ▶ caracteres infrequentes estão em folhas mais profundas
- ▶ dois nós pouco frequentes são unidos por um nó interno
- ▶ construímos a árvore de maneira **bottom-up**

Ideia:

- ▶ começar com $|C|$ nós correspondendo aos caracteres
- ▶ juntar os dois nós menos frequentes
- ▶ repetir $|C| - 1$ vezes até sobrar apenas um nó
- ▶ o nó restante é a raiz da árvore devolvida

Algoritmo de Huffman

HUFFMAN(C)

```
1   $Q \leftarrow C$ 
2  para  $i \leftarrow 1$  até  $|C| - 1$ 
3      alocar novo registro  $z$ 
4       $z.esq \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$ 
5       $z.dir \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$ 
6       $z.freq \leftarrow x.freq + y.freq$ 
7       $\text{INSERT}(Q, z)$ 
8  devolva  $\text{EXTRACT-MIN}(Q)$ 
```

- ▶ Q é uma fila de prioridades ordenada pela frequência
- ▶ EXTRACT-MIN e INSERT têm custo $\Theta(\log n)$
- ▶ O algoritmo tem complexidade de tempo $\Theta(n \log n)$

Algoritmo de Huffman

HUFFMAN(C)

```
1   $Q \leftarrow C$ 
2  para  $i \leftarrow 1$  até  $|C| - 1$ 
3      alocar novo registro  $z$ 
4       $z.esq \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$ 
5       $z.dir \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$ 
6       $z.freq \leftarrow x.freq + y.freq$ 
7       $\text{INSERT}(Q, z)$ 
8  devolva  $\text{EXTRACT-MIN}(Q)$ 
```

- ▶ Q é uma fila de prioridades ordenada pela frequência
- ▶ EXTRACT-MIN e INSERT têm custo $\Theta(\log n)$
- ▶ O algoritmo tem complexidade de tempo $\Theta(n \log n)$

Algoritmo de Huffman

HUFFMAN(C)

```
1   $Q \leftarrow C$ 
2  para  $i \leftarrow 1$  até  $|C| - 1$ 
3      alocar novo registro  $z$ 
4       $z.esq \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$ 
5       $z.dir \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$ 
6       $z.freq \leftarrow x.freq + y.freq$ 
7       $\text{INSERT}(Q, z)$ 
8  devolva  $\text{EXTRACT-MIN}(Q)$ 
```

- ▶ Q é uma fila de prioridades ordenada pela frequência
- ▶ EXTRACT-MIN e INSERT têm custo $\Theta(\log n)$
- ▶ O algoritmo tem complexidade de tempo $\Theta(n \log n)$

Algoritmo de Huffman

HUFFMAN(C)

```
1   $Q \leftarrow C$ 
2  para  $i \leftarrow 1$  até  $|C| - 1$ 
3      alocar novo registro  $z$ 
4       $z.esq \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$ 
5       $z.dir \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$ 
6       $z.freq \leftarrow x.freq + y.freq$ 
7       $\text{INSERT}(Q, z)$ 
8  devolva  $\text{EXTRACT-MIN}(Q)$ 
```

- ▶ Q é uma fila de prioridades ordenada pela frequência
- ▶ EXTRACT-MIN e INSERT têm custo $\Theta(\log n)$
- ▶ O algoritmo tem complexidade de tempo $\Theta(n \log n)$

Custo de uma codificação

Algumas notações

- ▶ T é uma árvore binária representando uma codificação
- ▶ $d_T(c)$ é a distância da raiz até o nó do caractere c
- ▶ $f(c)$ é a frequência do caractere c

Tamanho do texto codificado

- ▶ o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- ▶ dizemos que $B(T)$ é o **custo** de T

Custo de uma codificação

Algumas notações

- ▶ T é uma árvore binária representando uma codificação
- ▶ $d_T(c)$ é a distância da raiz até o nó do caractere c
- ▶ $f(c)$ é a frequência do caractere c

Tamanho do texto codificado

- ▶ o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- ▶ dizemos que $B(T)$ é o **custo** de T

Custo de uma codificação

Algumas notações

- ▶ T é uma árvore binária representando uma codificação
- ▶ $d_T(c)$ é a distância da raiz até o nó do caractere c
- ▶ $f(c)$ é a frequência do caractere c

Tamanho do texto codificado

- ▶ o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- ▶ dizemos que $B(T)$ é o **custo** de T

Custo de uma codificação

Algumas notações

- ▶ T é uma árvore binária representando uma codificação
- ▶ $d_T(c)$ é a distância da raiz até o nó do caractere c
- ▶ $f(c)$ é a frequência do caractere c

Tamanho do texto codificado

- ▶ o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- ▶ dizemos que $B(T)$ é o **custo** de T

Custo de uma codificação

Algumas notações

- ▶ T é uma árvore binária representando uma codificação
- ▶ $d_T(c)$ é a distância da raiz até o nó do caractere c
- ▶ $f(c)$ é a frequência do caractere c

Tamanho do texto codificado

- ▶ o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- ▶ dizemos que $B(T)$ é o **custo** de T

Custo de uma codificação

Algumas notações

- ▶ T é uma árvore binária representando uma codificação
- ▶ $d_T(c)$ é a distância da raiz até o nó do caractere c
- ▶ $f(c)$ é a frequência do caractere c

Tamanho do texto codificado

- ▶ o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- ▶ dizemos que $B(T)$ é o custo de T

Custo de uma codificação

Algumas notações

- ▶ T é uma árvore binária representando uma codificação
- ▶ $d_T(c)$ é a distância da raiz até o nó do caractere c
- ▶ $f(c)$ é a frequência do caractere c

Tamanho do texto codificado

- ▶ o número de bits de um texto codificado por T é

$$B(T) = \sum_{c \in C} f(c) d_T(c).$$

- ▶ dizemos que $B(T)$ é o **custo** de T

Escolha gulosa

Escolha gulosa

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa

Escolha gulosa

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa

Escolha gulosa

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa

Escolha gulosa

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa

Escolha gulosa

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa

Escolha gulosa

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa

Escolha gulosa

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa

Escolha gulosa

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa

Escolha gulosa

Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa

Escolha gulosa

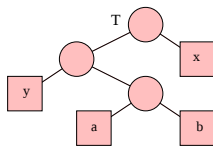
Sejam x e y os nós correspondentes aos dois caracteres com as menores frequências em C . Então:

- ▶ existe **codificação ótima** em que x e y são folhas irmãs
- ▶ elas estão tão distantes da raiz quanto qualquer folha

Demonstração:

- ▶ considere uma árvore ótima T
- ▶ sejam a e b duas folhas irmãs mais profundas
- ▶ e sejam x e y as duas folhas de menor frequência
- ▶ construa uma árvore T' trocando a e x
- ▶ e depois uma árvore T'' trocando b por y
- ▶ vamos mostrar que T'' também é uma **árvore ótima**

Escolha gulosa (cont)

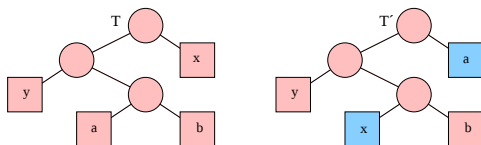


- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)

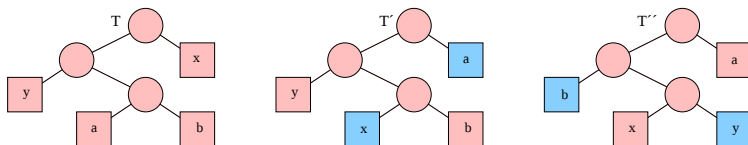


- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)

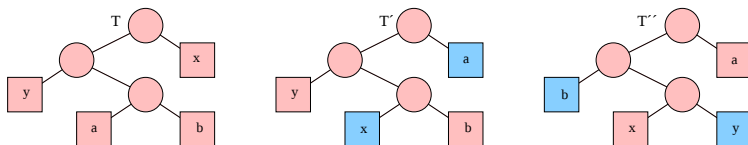


- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)

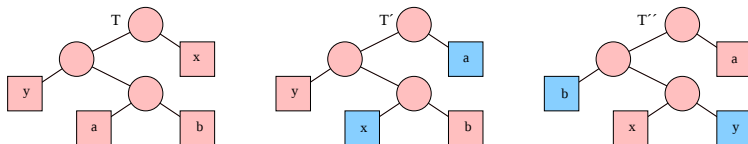


- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)

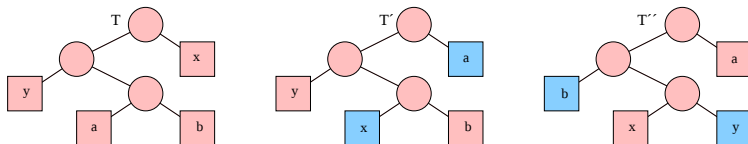


- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)

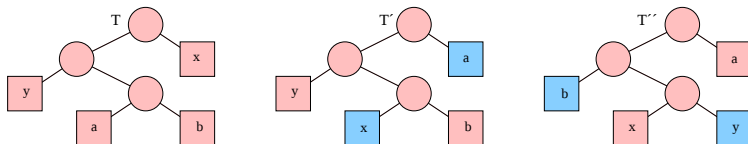


- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)

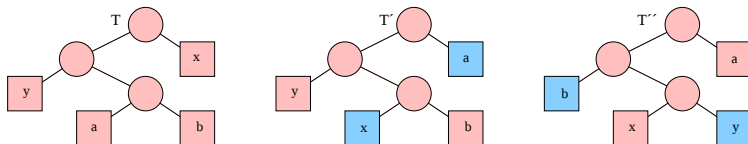


- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)

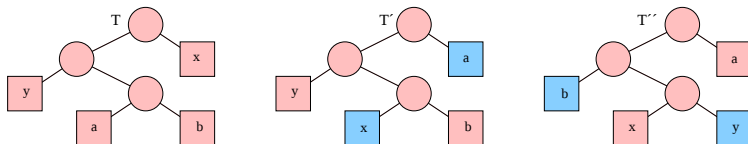


- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)

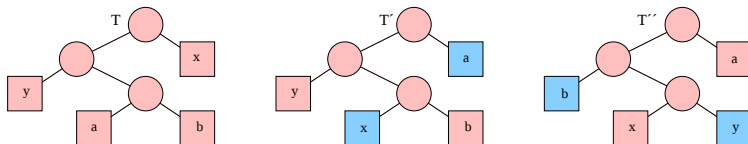


- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Escolha gulosa (cont)



- ▶ aplicando a definição de B e cancelando termos idênticos,

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_{T'}(x) - f(a) d_{T'}(a) \\ &= f(x) d_T(x) + f(a) d_T(a) - f(x) d_T(a) - f(a) d_T(x) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

- ▶ assim, $B(T) \geq B(T')$ e, analogamente, $B(T') \geq B(T'')$
- ▶ como T é ótima, T'' também é ótima e o enunciado segue

Subestrutura ótima

Subestrutura ótima

Suponha que

- ▶ x e y são os dois caracteres com menores frequências
- ▶ $C' = C \setminus \{x, y\} \cup \{z\}$ é alfabeto com $f(z) = f(x) + f(y)$
- ▶ T' é uma árvore ótima para o alfabeto C'
- ▶ T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C .

Subestrutura ótima

Subestrutura ótima

Suponha que

- ▶ x e y são os dois caracteres com menores frequências
- ▶ $C' = C \setminus \{x, y\} \cup \{z\}$ é alfabeto com $f(z) = f(x) + f(y)$
- ▶ T' é uma árvore ótima para o alfabeto C'
- ▶ T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C .

Subestrutura ótima

Subestrutura ótima

Suponha que

- ▶ x e y são os dois caracteres com menores frequências
- ▶ $C' = C \setminus \{x, y\} \cup \{z\}$ é alfabeto com $f(z) = f(x) + f(y)$
- ▶ T' é uma árvore ótima para o alfabeto C'
- ▶ T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C .

Subestrutura ótima

Subestrutura ótima

Suponha que

- ▶ x e y são os dois caracteres com menores frequências
- ▶ $C' = C \setminus \{x, y\} \cup \{z\}$ é alfabeto com $f(z) = f(x) + f(y)$
- ▶ T' é uma árvore ótima para o alfabeto C'
- ▶ T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C .

Subestrutura ótima

Subestrutura ótima

Suponha que

- ▶ x e y são os dois caracteres com menores frequências
- ▶ $C' = C \setminus \{x, y\} \cup \{z\}$ é alfabeto com $f(z) = f(x) + f(y)$
- ▶ T' é uma árvore ótima para o alfabeto C'
- ▶ T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C .

Subestrutura ótima

Subestrutura ótima

Suponha que

- ▶ x e y são os dois caracteres com menores frequências
- ▶ $C' = C \setminus \{x, y\} \cup \{z\}$ é alfabeto com $f(z) = f(x) + f(y)$
- ▶ T' é uma árvore ótima para o alfabeto C'
- ▶ T é obtida de T' substituindo-se z por duas folhas x e y

Então T é uma árvore ótima para C .

Subestrutura ótima (cont)

Comparando os custos de T e T'

- ▶ se $c \in C \setminus \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- ▶ como os códigos de x e y têm um bit a mais que z , sabemos que

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- ▶ portanto, $B(T) = B(T') + f(x) + f(y)$

Subestrutura ótima (cont)

Comparando os custos de T e T'

- ▶ se $c \in C \setminus \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- ▶ como os códigos de x e y têm um bit a mais que z , sabemos que

$$\begin{aligned}f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y))\end{aligned}$$

- ▶ portanto, $B(T) = B(T') + f(x) + f(y)$

Subestrutura ótima (cont)

Comparando os custos de T e T'

- ▶ se $c \in C \setminus \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- ▶ como os códigos de x e y têm um bit a mais que z , sabemos que

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- ▶ portanto, $B(T) = B(T') + f(x) + f(y)$

Subestrutura ótima (cont)

Comparando os custos de T e T'

- ▶ se $c \in C \setminus \{x, y\}$, então

$$f(c)d_T(c) = f(c)d_{T'}(c)$$

- ▶ como os códigos de x e y têm um bit a mais que z , sabemos que

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + (f(x) + f(y)) \end{aligned}$$

- ▶ portanto, $B(T) = B(T') + f(x) + f(y)$

Subestrutura ótima (cont)

Vamos mostrar que T é uma árvore ótima para C

- ▶ considere uma árvore ótima T^* para C
- ▶ pelo lema, supomos que x e y são folhas irmãs em T^*
- ▶ construa $\hat{T}$ a partir de T^* trocando x e y por z
- ▶ fazendo $f(z) = f(x) + f(y)$, temos $B(\hat{T}) = B(T^*) - f(x) - f(y)$
- ▶ como $\hat{T}$ é viável para C' e T' é ótima, $B(T') \leq B(\hat{T})$
- ▶ daí,

$$\begin{aligned} B(T) &= B(T') + f(x) + f(y) \\ &\leq B(\hat{T}) + f(x) + f(y) \\ &= B(T^*) - f(x) - f(y) + f(x) + f(y) \\ &= B(T^*) \end{aligned}$$

- ▶ então, de fato, T é ótima para C

Subestrutura ótima (cont)

Vamos mostrar que T é uma árvore ótima para C

- ▶ considere uma árvore ótima T^* para C
- ▶ pelo lema, supomos que x e y são folhas irmãs em T^*
- ▶ construa $\hat{T}$ a partir de T^* trocando x e y por z
- ▶ fazendo $f(z) = f(x) + f(y)$, temos $B(\hat{T}) = B(T^*) - f(x) - f(y)$
- ▶ como $\hat{T}$ é viável para C' e T' é ótima, $B(T') \leq B(\hat{T})$
- ▶ daí,

$$\begin{aligned} B(T) &= B(T') + f(x) + f(y) \\ &\leq B(\hat{T}) + f(x) + f(y) \\ &= B(T^*) - f(x) - f(y) + f(x) + f(y) \\ &= B(T^*) \end{aligned}$$

- ▶ então, de fato, T é ótima para C

Subestrutura ótima (cont)

Vamos mostrar que T é uma árvore ótima para C

- ▶ considere uma árvore ótima T^* para C
- ▶ pelo lema, supomos que x e y são folhas irmãs em T^*
- ▶ construa $\hat{T}$ a partir de T^* trocando x e y por z
- ▶ fazendo $f(z) = f(x) + f(y)$, temos $B(\hat{T}) = B(T^*) - f(x) - f(y)$
- ▶ como $\hat{T}$ é viável para C' e T' é ótima, $B(T') \leq B(\hat{T})$
- ▶ daí,

$$\begin{aligned} B(T) &= B(T') + f(x) + f(y) \\ &\leq B(\hat{T}) + f(x) + f(y) \\ &= B(T^*) - f(x) - f(y) + f(x) + f(y) \\ &= B(T^*) \end{aligned}$$

- ▶ então, de fato, T é ótima para C

Subestrutura ótima (cont)

Vamos mostrar que T é uma árvore ótima para C

- ▶ considere uma árvore ótima T^* para C
- ▶ pelo lema, supomos que x e y são folhas irmãs em T^*
- ▶ construa $\hat{T}$ a partir de T^* trocando x e y por z
- ▶ fazendo $f(z) = f(x) + f(y)$, temos $B(\hat{T}) = B(T^*) - f(x) - f(y)$
- ▶ como $\hat{T}$ é viável para C' e T' é ótima, $B(T') \leq B(\hat{T})$
- ▶ daí,

$$\begin{aligned} B(T) &= B(T') + f(x) + f(y) \\ &\leq B(\hat{T}) + f(x) + f(y) \\ &= B(T^*) - f(x) - f(y) + f(x) + f(y) \\ &= B(T^*) \end{aligned}$$

- ▶ então, de fato, T é ótima para C

Subestrutura ótima (cont)

Vamos mostrar que T é uma árvore ótima para C

- ▶ considere uma árvore ótima T^* para C
- ▶ pelo lema, supomos que x e y são folhas irmãs em T^*
- ▶ construa $\hat{T}$ a partir de T^* trocando x e y por z
- ▶ fazendo $f(z) = f(x) + f(y)$, temos $B(\hat{T}) = B(T^*) - f(x) - f(y)$
- ▶ como $\hat{T}$ é viável para C' e T' é ótima, $B(T') \leq B(\hat{T})$
- ▶ daí,

$$\begin{aligned} B(T) &= B(T') + f(x) + f(y) \\ &\leq B(\hat{T}) + f(x) + f(y) \\ &= B(T^*) - f(x) - f(y) + f(x) + f(y) \\ &= B(T^*) \end{aligned}$$

- ▶ então, de fato, T é ótima para C

Subestrutura ótima (cont)

Vamos mostrar que T é uma árvore ótima para C

- ▶ considere uma árvore ótima T^* para C
- ▶ pelo lema, supomos que x e y são folhas irmãs em T^*
- ▶ construa $\hat{T}$ a partir de T^* trocando x e y por z
- ▶ fazendo $f(z) = f(x) + f(y)$, temos $B(\hat{T}) = B(T^*) - f(x) - f(y)$
- ▶ como $\hat{T}$ é viável para C' e T' é ótima, $B(T') \leq B(\hat{T})$
- ▶ daí,

$$\begin{aligned} B(T) &= B(T') + f(x) + f(y) \\ &\leq B(\hat{T}) + f(x) + f(y) \\ &= B(T^*) - f(x) - f(y) + f(x) + f(y) \\ &= B(T^*) \end{aligned}$$

- ▶ então, de fato, T é ótima para C

Subestrutura ótima (cont)

Vamos mostrar que T é uma árvore ótima para C

- ▶ considere uma árvore ótima T^* para C
- ▶ pelo lema, supomos que x e y são folhas irmãs em T^*
- ▶ construa $\hat{T}$ a partir de T^* trocando x e y por z
- ▶ fazendo $f(z) = f(x) + f(y)$, temos $B(\hat{T}) = B(T^*) - f(x) - f(y)$
- ▶ como $\hat{T}$ é viável para C' e T' é ótima, $B(T') \leq B(\hat{T})$
- ▶ daí,

$$\begin{aligned} B(T) &= B(T') + f(x) + f(y) \\ &\leq B(\hat{T}) + f(x) + f(y) \\ &= B(T^*) - f(x) - f(y) + f(x) + f(y) \\ &= B(T^*) \end{aligned}$$

- ▶ então, de fato, T é ótima para C

Subestrutura ótima (cont)

Vamos mostrar que T é uma árvore ótima para C

- ▶ considere uma árvore ótima T^* para C
- ▶ pelo lema, supomos que x e y são folhas irmãs em T^*
- ▶ construa $\hat{T}$ a partir de T^* trocando x e y por z
- ▶ fazendo $f(z) = f(x) + f(y)$, temos $B(\hat{T}) = B(T^*) - f(x) - f(y)$
- ▶ como $\hat{T}$ é viável para C' e T' é ótima, $B(T') \leq B(\hat{T})$
- ▶ daí,

$$\begin{aligned} B(T) &= B(T') + f(x) + f(y) \\ &\leq B(\hat{T}) + f(x) + f(y) \\ &= B(T^*) - f(x) - f(y) + f(x) + f(y) \\ &= B(T^*) \end{aligned}$$

- ▶ então, de fato, T é ótima para C

HUFFMAN

HUFFMAN(C)

```
1   $Q \leftarrow C$ 
2  para  $i \leftarrow 1$  até  $|C| - 1$ 
3      alocar novo registro  $z$ 
4       $z.esq \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$ 
5       $z.dir \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$ 
6       $z.freq \leftarrow x.freq + y.freq$ 
7       $\text{INSERT}(Q, z)$ 
8  devolva  $\text{EXTRACT-MIN}(Q)$ 
```

Correção do algoritmo de Huffman

Teorema

HUFFMAN constrói uma codificação ótima.

Demonstração:

- ▶ seja T a codificação devolvida pelo algoritmo
- ▶ associamos cada nó de T a um caractere distinto
- ▶ vamos mostrar que T é ótima por indução em $|C|$

Correção do algoritmo de Huffman

Teorema

HUFFMAN constrói uma codificação ótima.

Demonstração:

- ▶ seja T a codificação devolvida pelo algoritmo
- ▶ associamos cada nó de T a um caractere distinto
- ▶ vamos mostrar que T é ótima por indução em $|C|$

Correção do algoritmo de Huffman

Teorema

HUFFMAN constrói uma codificação ótima.

Demonstração:

- ▶ seja T a codificação devolvida pelo algoritmo
- ▶ associamos cada nó de T a um caractere distinto
- ▶ vamos mostrar que T é ótima por indução em $|C|$

Correção do algoritmo de Huffman

Teorema

HUFFMAN constrói uma codificação ótima.

Demonstração:

- ▶ seja T a codificação devolvida pelo algoritmo
- ▶ associamos cada nó de T a um caractere distinto
- ▶ vamos mostrar que T é ótima por indução em $|C|$

Correção do algoritmo de Huffman

Teorema

HUFFMAN constrói uma codificação ótima.

Demonstração:

- ▶ seja T a codificação devolvida pelo algoritmo
- ▶ associamos cada nó de T a um caractere distinto
- ▶ vamos mostrar que T é ótima por indução em $|C|$

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C

Correção do algoritmo de Huffman (cont)

Considere $|C| = 1$

- ▶ nesse caso, o algoritmo devolve um único nó
- ▶ que é uma solução ótima

Agora, suponha que $|C| \geq 2$

- ▶ sejam x e y os caracteres escolhidos na primeira iteração
- ▶ seja z um caractere com $f(z) = f(x) + f(y)$
- ▶ considere um conjunto de caracteres $C' = C \setminus \{x, y\} \cup \{z\}$
- ▶ a árvore devolvida pelo algoritmo para C' é $T' = T - x - y$
- ▶ como $|C'| < |C|$, por hipótese de indução, T' é ótima para C'
- ▶ então, pela subestrutura ótima, T é ótima para C