MC-202 — Aula 2
Estruturas Ligadas, Noções de Eficiência

Lehilton Pedrosa

Instituto de Computação – Unicamp

Segundo Semestre de 2015

Roteiro

Um jogo para exercitar a memória

Jogo para memória

Cada um joga uma vez:

Um jogo para exercitar a memória

Jogo para memória

Cada um joga uma vez:

1. O primeiro jogador fala:
   - dia do seu aniversário

Um jogo para exercitar a memória

Jogo para memória

Cada um joga uma vez:

1. O primeiro jogador fala:
   - dia do seu aniversário
2. O segundo jogador:
   - o nome do jogador anterior (do primeiro)
   - o dia de seu aniversário
   - o dia do primeiro jogador

Um jogo para exercitar a memória

Jogo para memória

Cada um joga uma vez:

1. O primeiro jogador fala:
   - dia do seu aniversário
2. O segundo jogador:
   - o nome do jogador anterior (do primeiro)
   - o dia de seu aniversário
   - o dia do primeiro jogador
3. O terceiro jogador:
   - o nome do jogador anterior (do segundo)
   - o dia de seu aniversário
   - o dia do segundo jogador
   - o dia do primeiro jogador

Um jogo para exercitar a memória

Jogo para memória

Cada um joga uma vez:

1. O primeiro jogador fala:
   - dia do seu aniversário
2. O segundo jogador:
   - o nome do jogador anterior (do primeiro)
   - o dia de seu aniversário
   - o dia do primeiro jogador
3. O terceiro jogador:
   - o nome do jogador anterior (do segundo)
   - o dia de seu aniversário
   - o dia do segundo jogador
   - o dia do primeiro jogador
4. e assim por diante...

Implementação com vetor

```
int main() {
    int dias[10], dia_lido, i, n;

    n = 0;
    do {
        scanf("%d", &dia_lido);
        if (dia_lido != -1) {
            dias[n] = dia_lido;
            n++;
            for (i = 0; i < n; i++) {
                printf("%d\n", dias[i]);
            }
        }
    } while (dia_lido != -1);

    return 0;
}
```

Implementação com vetor

```
int main() {
    int dias[10], dia_lido, i, n;

    n = 0;
    do {
        scanf("%d", &dia_lido);
        if (dia_lido != -1) {
            dias[n] = dia_lido;
            n++;
            for (i = 0; i < n; i++) {
                printf("%d\n", dias[i]);
            }
        }
    } while (dia_lido != -1);

    return 0;
}
```

O que acontece se mais de 10 números forem lidos?

Dificuldades com vetor

Problemas

- o vetor tem tamanho limitado

Dificuldades com vetor

Problemas

- o vetor tem tamanho limitado
- não adianta alocar vetor dinamicamente:
  - não sabemos quantos números serão lidos no início

Dificuldades com vetor

Problemas

- o vetor tem tamanho limitado
- não adianta alocar vetor dinamicamente:
  - não sabemos quantos números serão lidos no início

Uma solução possível

- só alocamos uma variável por vez (dinamicamente)

Dificuldades com vetor

Problemas

- o vetor tem tamanho limitado
- não adianta alocar vetor dinamicamente:
  - não sabemos quantos números serão lidos no início

Uma solução possível

- só alocamos uma variável por vez (dinamicamente)

Dificuldades com vetor

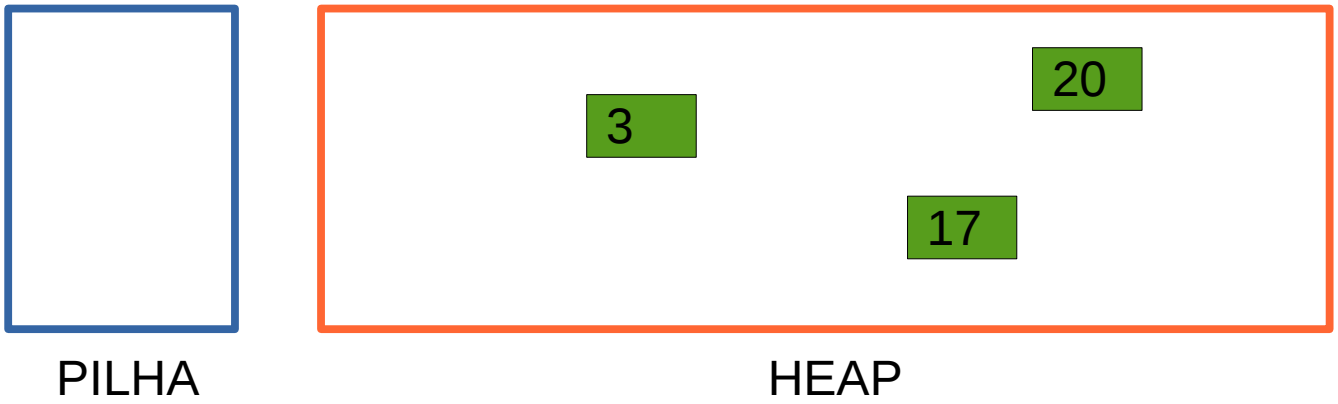
Problemas

- o vetor tem tamanho limitado
- não adianta alocar vetor dinamicamente:
  - não sabemos quantos números serão lidos no início

Uma solução possível

- só alocamos uma variável por vez (dinamicamente)

Dificuldades com vetor



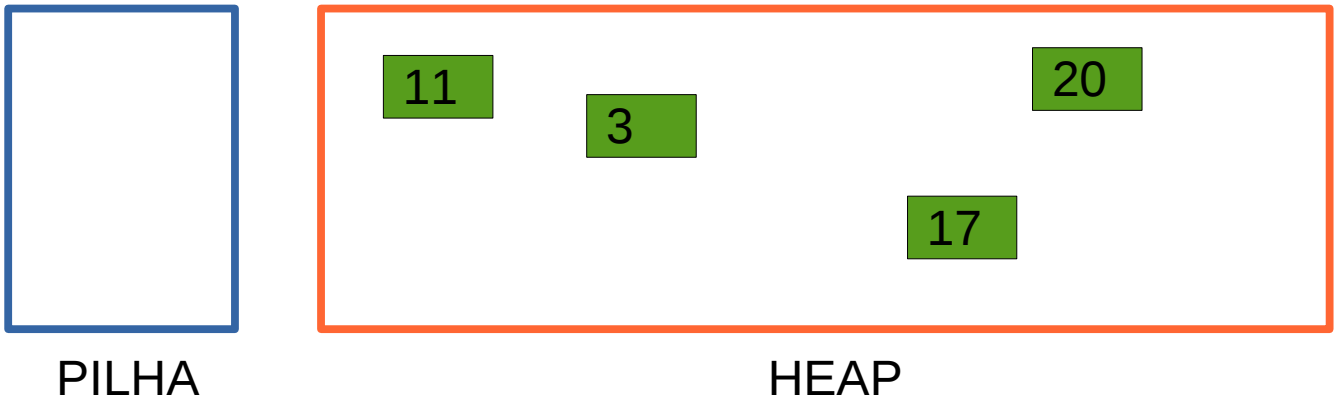
Problemas

- o vetor tem tamanho limitado
- não adianta alocar vetor dinamicamente:
  - não sabemos quantos números serão lidos no início

Uma solução possível

- só alocamos uma variável por vez (dinamicamente)

Dificuldades com vetor



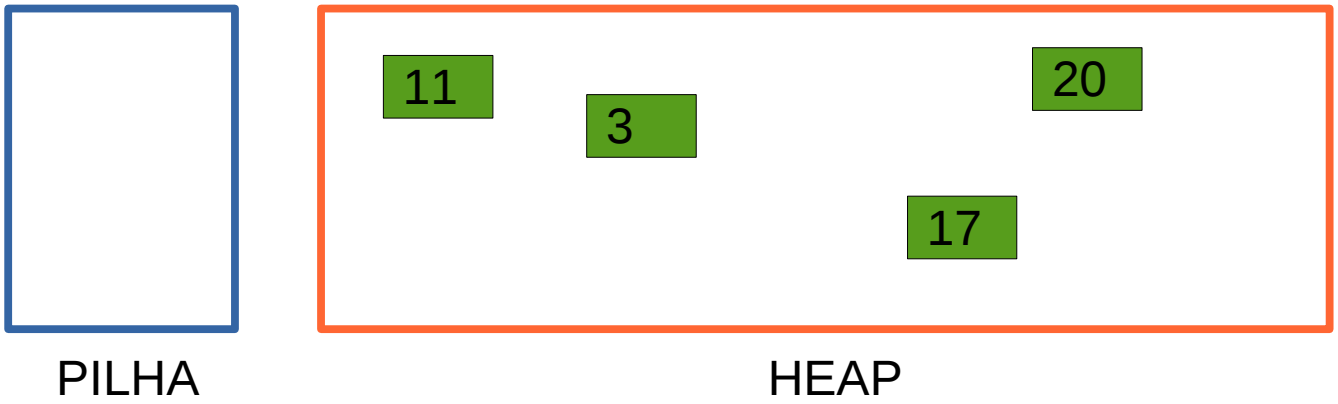
Problemas

- o vetor tem tamanho limitado
- não adianta alocar vetor dinamicamente:
  - não sabemos quantos números serão lidos no início

Uma solução possível

- só alocamos uma variável por vez (dinamicamente)

Dificuldades com vetor



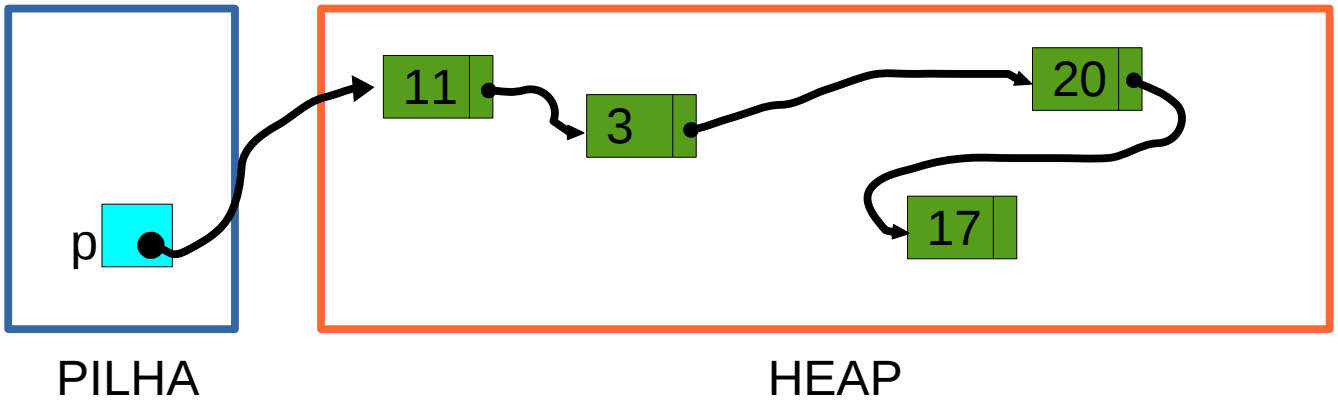
Problemas

- o vetor tem tamanho limitado
- não adianta alocar vetor dinamicamente:
  - não sabemos quantos números serão lidos no início

Uma solução possível

- só alocamos uma variável por vez (dinamicamente)
- mas, como sabemos onde está alocada?

Dificuldades com vetor



Problemas

- o vetor tem tamanho limitado
- não adianta alocar vetor dinamicamente:
  - não sabemos quantos números serão lidos no início

Uma solução possível

- só alocamos uma variável por vez (dinamicamente)
- mas, como sabemos onde está alocada?
  - salvamos ponteiros juntos dos dados!

Estruturas ligadas

Nó

Elemento na memória alocado dinamicamente que contém um dado ou conjunto de dados juntamente com um (ou mais) ponteiro para outro nó.

Estruturas ligadas

Nó

Elemento na memória alocado dinamicamente que contém um dado ou conjunto de dados juntamente com um (ou mais) ponteiro para outro nó.

Estrutura ligada

Conjunto de nós ligados entre si.

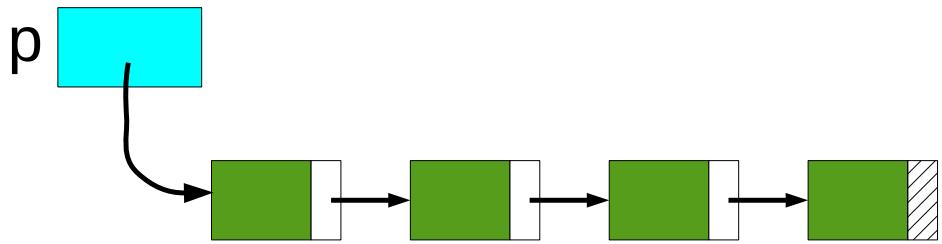
Estruturas ligadas

Nó

Elemento na memória alocado dinamicamente que contém um dado ou conjunto de dados juntamente com um (ou mais) ponteiro para outro nó.

Estrutura ligada

Conjunto de nós ligados entre si.



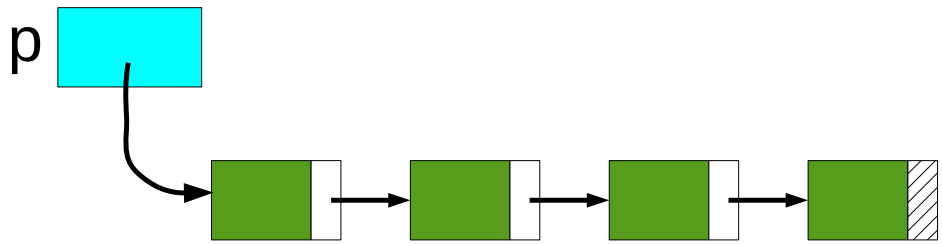
Estruturas ligadas

Nó

Elemento na memória alocado dinamicamente que contém um dado ou conjunto de dados juntamente com um (ou mais) ponteiro para outro nó.

Estrutura ligada

Conjunto de nós ligados entre si.



Observações

- uma estrutura ligada é acessada a partir de uma variável primária (da pilha)
- um ponteiro pode estar vazio (aponta para `NULL` em C)

Implementação em C

Definição do nó

```
typedef struct No {
    int dado;
    struct No *prox;
} No;
```

Implementação em C

Definição do nó

```
typedef struct No {
    int dado;
    struct No *prox;
} No;
```

Observações

- o `typedef` apenas define um apelido `No` para o tipo `struct No`
- deve-se usar `struct No` dentro do registro, porque o apelido ainda não existe ali!
- os nomes do `struct` e do `typedef` podem ser distintos

Algumas operações

Inicializa lista e adiciona elemento

```
void iniciar_lista(No **lista) {
    *lista = NULL;
}
void adicionar_elemento(No **lista, int x) {
    No *novo;
    novo = malloc(sizeof(No));
    if (novo == NULL) {
        printf("Não há memória!"); exit(1);
    }
    novo->dado = x;
    novo->prox = *lista;
    *lista = novo;
}
```

Algumas operações

Inicializa lista e adiciona elemento

```
void iniciar_lista(No **lista) {
    *lista = NULL;
}
void adicionar_elemento(No **lista, int x) {
    No *novo;
    novo = malloc(sizeof(No));
    if (novo == NULL) {
        printf("Não há memória!"); exit(1);
    }
    novo->dado = x;
    novo->prox = *lista;
    *lista = novo;
}
```

Importante!

O ponteiro da lista é passado por referência!

Reimplementando com lista ligada

```
int main() {
    No *lista, *atual;

    iniciar_lista(&lista);
    do {
        scanf("%d", &dia_lido);
        if (dia_lido != -1) {
            adicionar_elemento(&lista, dia_lido);
            atual = lista;
            while (atual != NULL) {
                printf("%d\n", atual->dado);
                atual = atual->prox;
            }
        }
    } while (dia_lido != -1);
    return 0;
}
```

Reimplementando com lista ligada

```
int main() {
    No *lista, *atual;

    iniciar_lista(&lista);
    do {
        scanf("%d", &dia_lido);
        if (dia_lido != -1) {
            adicionar_elemento(&lista, dia_lido);
            atual = lista;
            while (atual != NULL) {
                printf("%d\n", atual->dado);
                atual = atual->prox;
            }
        }
    } while (dia_lido != -1);
    return 0;
}
```

Agora não temos mais limitação no número de elementos lidos!

Reimplementando com lista ligada

```
int main() {
    No *lista, *atual;

    iniciar_lista(&lista);
    do {
        scanf("%d", &dia_lido);
        if (dia_lido != -1) {
            adicionar_elemento(&lista, dia_lido);
            atual = lista;
            while (atual != NULL) {
                printf("%d\n", atual->dado);
                atual = atual->prox;
            }
        }
    } while (dia_lido != -1);
    return 0;
}
```

Agora não temos mais limitação no número de elementos lidos!

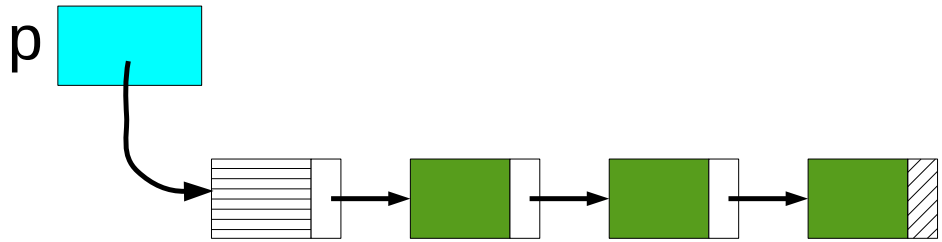
O que falta fazer???

Destruir lista

Liberando a memória de uma lista

```
void destruir_lista(No **lista) {
    No *atual, *proximo;
    proximo = *lista;
    while (prox != NULL) {
        atual = proximo;
        proximo = atual->prox;
        free(atual);
    }
    *lista = NULL;
}
```

Lista com cabeça (nó *dummy*)



Observação

- lista sempre aponta para o nó *dummy*
- evita passagem por referência
- um mesmo procedimento para adicionar elemento em qualquer posição
- percorrimento tem que ignorar cabeça!

Lista com cabeça (nó *dummy*) – operações

```
No *iniciar_lista() {
    No *dummy;
    dummy = malloc(sizeof(No));
    if (dummy == NULL) {
        printf("Não há memória!"); exit(1);
    }
    dummy->prox = NULL;
    return dummy; // retorna nó cabeça
}

void adicionar_elemento_depois(No *anterior, int x) {
    No *novo;
    novo = malloc(sizeof(No));
    if (novo == NULL) {
        printf("Não há memória!"); exit(1);
    }
    novo->dado = x;
    novo->prox = anterior->prox;
    anterior->prox = novo;
}
```

Exemplo de aplicação: polinômio

Polinômio

Um polinômio é uma expressão da seguinte forma

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

Exemplo de aplicação: polinômio

Polinômio

Um polinômio é uma expressão da seguinte forma

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

Lista de termos

```
typedef struct Termo {
    float coeficiente;
    int expoente;
    struct Termo *prox;
} Termo;
```

Exemplo de aplicação: polinômio

Polinômio

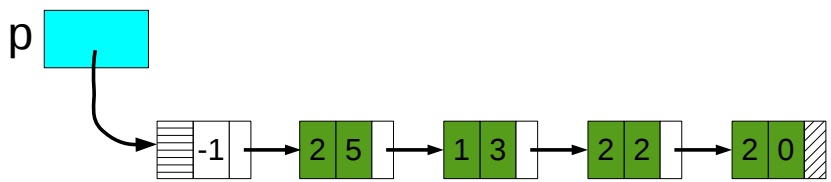
Um polinômio é uma expressão da seguinte forma

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

Lista de termos

```
typedef struct Termo {
    float coeficiente;
    int expoente;
    struct Termo *prox;
} Termo;
```

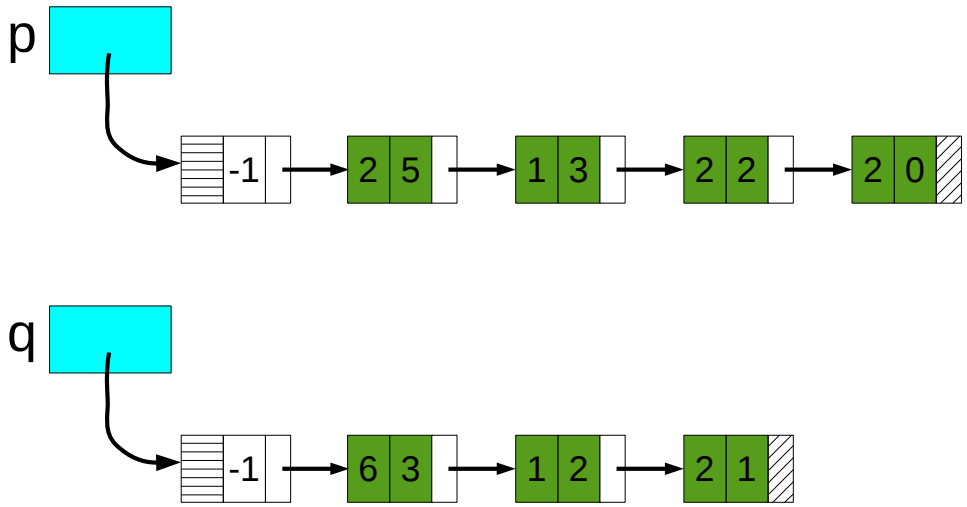
$2x^5 + x^3 + 2x^2 + 2$:



Somando polinômios

Problema

Dados dois polinômios p e q, como obter um terceiro polinômio correspondente à soma $r = p + q$?



Acessando o k-ésimo elemento

Com vetor

```
valor = vetor[k];
```

Com lista

```
int i;
No *proximo, *atual;

proximo = lista;
for (i = 0; i < k; i++) {
    atual = proximo;
    proximo = atual->prox;
}
valor = atual->dado;
```

Acessando o k-ésimo elemento

Com vetor

```
valor = vetor[k];
```

Com lista

```
int i;
No *proximo, *atual;

proximo = lista;
for (i = 0; i < k; i++) {
    atual = proximo;
    proximo = atual->prox;
}
valor = atual->dado;
```

Número de operações:

constante

Acessando o k-ésimo elemento

Com vetor

```
valor = vetor[k];
```

Número de operações:

constante

Com lista

```
int i;
No *proximo, *atual;

proximo = lista;
for (i = 0; i < k; i++) {
    atual = proximo;
    proximo = atual->prox;
}
valor = atual->dado;
```

Número de operações:

quase proporcional a k

Contando instruções[1]

(a)

```
...
x = a + b;
...
```

(b)

```
...
for (i = 0; i < n; i++)
    x = a + b;
...
```

(c)

```
...
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        x = a + b;
...
```

[1] exemplo retirado da apostila dos profs. Lucchesi e Kowaltowski

Contando instruções[1]

(a)

```
...
x = a + b;
...
```

(b)

```
...
for (i = 0; i < n; i++)
    x = a + b;
...
```

(c)

```
...
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        x = a + b;
...
```

	a	b	c
análise simplificada (1):	1	n	n^2
análise detalhada (2):	2	$5n + 2$	$5n^2 + 5n + 2$

(1) atribuições (2) atribuições, operações aritméticas e comparações

[1] exemplo retirado da apostila dos profs. Lucchesi e Kowaltowski

Contando instruções[1]

(a)

```
...
x = a + b;
...
```

(b)

```
...
for (i = 0; i < n; i++)
    x = a + b;
...
```

(c)

```
...
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        x = a + b;
...
```

	a	b	c
análise simplificada (1):	1	n	n^2
análise detalhada (2):	2	$5n + 2$	$5n^2 + 5n + 2$

(1) atribuições (2) atribuições, operações aritméticas e comparações

As duas análises produzem contagem da mesma "ordem" de grandeza!

[1] exemplo retirado da apostila dos profs. Lucchesi e Kowaltowski

Notação O

Definição

Uma função $f(n)$ é da ordem de $g(n)$, ou $f(n) = O(g(n))$, se existirem números c e n_0 tais que $f(n) \leq c\, g(n)$ para todo $n \geq n_0$.

Exemplos

$$
\begin{aligned}
1 &= O(1) \\
1000000 &= O(1) \\
5n + 2 &= O(n) \\
5n^2 + 5n + 2 &= O(n^2) \\
1000000n &= O(n^2) \\
\log_2 n &= O(\log_{10} n) \\
\log_{10} n &= O(\log_2 n) \\
n^{1000} &= O(2^n)
\end{aligned}
$$

Busca binária e busca sequencial em vetor ordenado

```
int busca_sequencial(int v[], int n, int x) {
    int i;
    for (i = 0; i < n; i++)
        if (v[i] == x) return i;
    return -1;
}
int busca_binaria(int v[], int n, int x) {
    int ini = 0, fim = n - 1;
    while (ini <= fim) {
        int meio = (ini + fim) / 2;
        if   (v[meio] == x)  return meio;
        else (v[meio] >  x)  fim = meio - 1;
        else                 ini = meio + 1;
    }
    return -1;
}
```

Para tentar em casa: Suponha que o algoritmo de busca binária é executado em um computador 20 vezes mais lento que o da sequencial. Ainda assim, ele pode ser mais rápido que o algoritmo sequencial? Para que tamanho n de vetor?

Comparando vetores e listas ligadas

Comparando

- **acesso:** acessar um elemento em certa posição em vetor é mais rápido; em uma lista é preciso primeiro obter o nó
- **inserção** para inserir um elemento no meio de um vetor, é preciso depois mover todos os elementos posteriores; em uma lista, basta atualizar os ponteiros
- **remoção:** acontece o mesmo com a inserção (por quê?)
- **criação:** para criar um vetor é imprescindível conhecer um limitante para o tamanho máximo e, se esse valor for ruim, pode haver desperdício de espaço; uma lista é mais flexível, mas gasta espaço para guardar os ponteiros

Comparando vetores e listas ligadas

Comparando

- **acesso:** acessar um elemento em certa posição em vetor é mais rápido; em uma lista é preciso primeiro obter o nó
- **inserção** para inserir um elemento no meio de um vetor, é preciso depois mover todos os elementos posteriores; em uma lista, basta atualizar os ponteiros
- **remoção:** acontece o mesmo com a inserção (por quê?)
- **criação:** para criar um vetor é imprescindível conhecer um limitante para o tamanho máximo e, se esse valor for ruim, pode haver desperdício de espaço; uma lista é mais flexível, mas gasta espaço para guardar os ponteiros

- **Pergunta:** qual é melhor?

Comparando vetores e listas ligadas

Comparando

- **acesso:** acessar um elemento em certa posição em vetor é mais rápido; em uma lista é preciso primeiro obter o nó
- **inserção** para inserir um elemento no meio de um vetor, é preciso depois mover todos os elementos posteriores; em uma lista, basta atualizar os ponteiros
- **remoção:** acontece o mesmo com a inserção (por quê?)
- **criação:** para criar um vetor é imprescindível conhecer um limitante para o tamanho máximo e, se esse valor for ruim, pode haver desperdício de espaço; uma lista é mais flexível, mas gasta espaço para guardar os ponteiros

- **Pergunta:** qual é melhor?
- **Resposta:** depende do problema e do algoritmo e de ...

Exercício 1

Além da notação O existem outras notações importantes utilizadas para descrever o tempo (ou a complexidade) de um algoritmo. Pesquise e defina o que significam as notações Ω e Θ.

Exercício 2 - Matriz esparsa

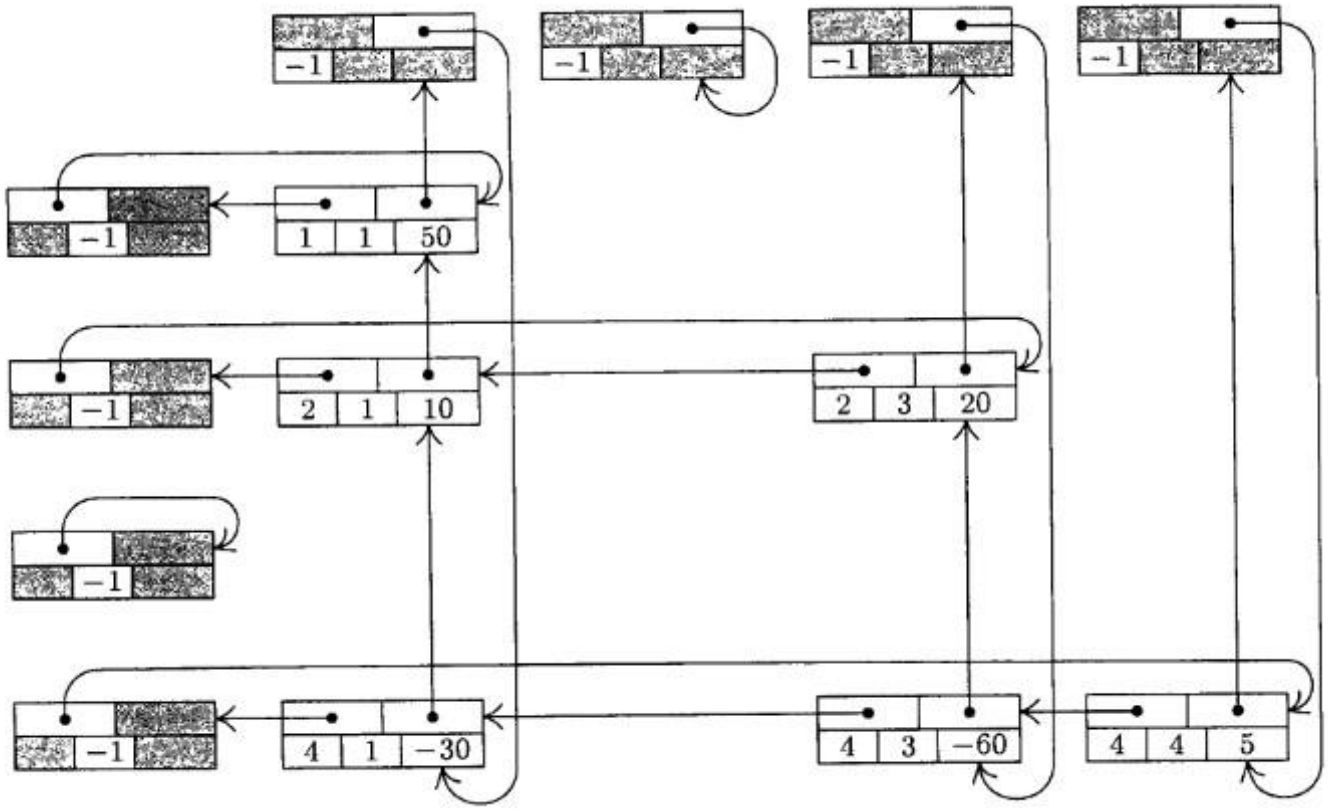
Uma matriz $n \times m$ é dita esparsa quando o número de elementos não-nulos é "pequeno" comparado ao número total de elementos nm. Nessa situação, pode ser vantajoso utilizar listas ligadas para representar uma matriz, já que os algoritmos podem supor que todos os elementos não percorridos são nulos. Por exemplo, a matriz a seguir é representada pelas listas "ortogonais" desenhadas no próximo slide:

$$\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix}$$

Exercício

Defina um novo tipo de nó (`struct`) correspondente a um elemento da matriz esparsa desenhada no próximo slide. Qual a diferença dessa estrutura para as estruturas vistas em sala? (por exemplo, para onde aponta os nós *dummies*?)

Listas ortogonais[2]



[2]Imagem do livro The Art of Computer Programming - I, Knuth.