1. Escreva uma função que dado um conjunto de n pontos no plano representados por pares de números reais e um número real R, devolve um grafo onde o conjunto de vértices é formado pelos n pontos e dois vértices são adjacentes se e somente se os pontos correspondentes estão a distância no máximo R.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Ponto {
  double x, y;
} Ponto;

typedef struct Grafo {
  Ponto *vertices;
  int **adj;
  int n;
} Grafo;

typedef Grafo * p_grafo;

double dist2(Ponto p1, Ponto p2) {
  return (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y);
}

p_grafo grafo_distancia(int n, Ponto * pontos, double R) {
  int i, j;
  p_grafo g = malloc(sizeof(Grafo));
  g->vertices = malloc(n * sizeof(Ponto));
  g->adj = malloc(n * sizeof(int *));
  for (i = 0; i < n; i++)
    g->adj = malloc(n * sizeof(int));
  g->n = n;
  for (i = 0; i < n; i++)
    g->vertices[i] = pontos[i];
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      g->adj[i][j] = (i != j && dist2(pontos[i], pontos[j]) <= R * R) ?
    1 : 0;
  return g;
}
```

2. Escreva uma função que dado um grafo G, devolve 1 se G é uma floresta e devolve 0 se G caso contrário.

```
#include <stdlib.h>

typedef struct Grafo {
  int **adj;
  int n;
} Grafo;

typedef Grafo * p_grafo;

int tem_ciclo(p_grafo g, int v, int pai, int * visitado) {
  int w;
  visitado[v] = 1;
  for (w = 0; w < g->n; w++)
    if (g->adj[v][w]) {
```

```
      if (visitado[w] && pai != w)
        return 1;
      if (!visitado[w] && tem_ciclo(g, w, v, visitado))
        return 1;
    }
  return 0;
}

int eh_floresta(p_grafo g) {
  int s, *visitado = malloc(g->n * sizeof(int));
  for (s = 0; s < g->n; s++)
    if (!visitado[s] && tem_ciclo(g, s, s, visitado)) {
      free(visitado);
      return 0;
    }
  free(visitado);
  return 1;
}
```

3. Faça uma função que dado inteiros n e m, e uma lista de m pares de nomes de pessoas (sem espaço) que se conhecem, cria um grafo de n vértices onde cada vértice é uma pessoa e há uma aresta entre dois vértices se e somente se as duas pessoas se conhecem.

```
typedef struct Grafo {
  int **adj;
  int n;
  char **nomes;
} Grafo;

typedef Grafo * p_grafo;

typedef struct Pares {
  char *nome1, *nome2;
} Pares;

int indice_vertice(p_grafo g, p_hash h, char *nome) {
  int v;
  v = busca_no_hash(h, nome);
  if (v == -1) {
    v = g->n;
    g->nomes[g->n] = malloc((strlen(nome) + 1) * sizeof(char));
    strcpy(nome, g->nomes[g->n]);
    g->n++;
    insere_no_hash(h, nome, v);
  }
  return v;
}

p_grafo cria_rede_social(int n, int m, Pares *lista) {
  int i, u, v;
  p_hash h = cria_hash();
  p_grafo g = malloc(sizeof(Grafo));
  g->adj = malloc(n * sizeof(int *));
  for (i = 0; i < n; i++)
    g->adj = calloc(n, sizeof(int));
  g->n = 0;
```

```
  for (i = 0; i < m; i++) {
    u = indice_vertice(g, h, lista[i].nome1);
    v = indice_vertice(g, h, lista[i].nome2);
    g->adj[u][v] = g->adj[v][u] = 1;
  }
  return g;
}
```

4. Seja G o grafo onde os vértices são as pessoas vivas nesse momento e onde há uma aresta entre dois vértices se duas pessoas se conhecem.

Acredita-se que tal grafo satisfaça a propriedade dos seis graus de separação, isto é, para quaisquer duas pessoas u e v, u conhece v em no máximo seis passos. Por exemplo, se u conhece w e w conhece v, mas u não conhece v diretamente, então u conhece v em dois passos.

Faça uma função que dado um grafo G e um inteiro k, verifica se G tem a propriedade dos k graus de separação.

```
int * distancias(p_grafo g, int s) {
  int w, v;
  int *dist = malloc(g->n * sizeof(int));
  p_fila f = criar_fila();
  for (v = 0; v < g->n; v++)
    dist[v] = INT_MAX;
  enfileira(f,s);
  dist[s] = 0;
  while(!fila_vazia(f)) {
    v = desenfileira(f);
    for (w = 0; w < g->n; w++)
      if (g->adj[v][w] && dist[w] == INT_MAX) {
        dist[w] = dist[v] + 1;
        enfileira(f, w);
      }
  }
  destroi_fila(f);
  return dist;
}

int k_separacao(p_grafo g, int k) {
  int v, w, *dist;
  for (v = 0; v < g->n; v++) {
    dist = distancias(g, v);
    for (w = 0; w < g->n; w++)
      if (dist[w] > k) {
        free(dist);
        return 0;
      }
    free(dist);
  }
  return 1;
}
```