Colectânea de Problemas
Análise e Síntese de Algoritmos

Departamento de Engenharia Informática
Instituto Superior Técnico (IST)

`Versão`: Fevereiro 2019

Resumo

Este documento agrega os exercícios dos testes e exames da unidade curricular de Análise e Síntese de Algoritmos (ASA) entre os anos lectivos 2009/2010 a 2017/2018. O principal objectivo é organizar os problemas por tópicos leccionados e pelo grau de dificuldade que apresentam. Como resultado, os alunos têm à sua disposição um conjunto alargado de problemas de apoio ao estudo das matérias leccionadas.

Parte I.
Revisão de Introdução aos Algoritmos e Estruturas de Dados

I.1. Notação assimptótica - Recorrências

I.1.1 Considere a função recursiva apresentada abaixo:

```
int xpto(int x, int y)
{
    int i, s;

    s = 0;
    if(x < y) {
        for(i = x; i <= y; i++)
            s += i;
        s += xpto(x,(x+y)/2) + xpto(x-1,(x+y)/2-1) + xpto((x+y)/2,y);
    }
    return s;
}
```

Determine o menor *majorante assimptótico* (notação $\mathcal{O}$) para o tempo de execução da função. **Sugestão:** determine a recorrência que traduz o tempo de execução da função (em termos do número de elementos, $n = \texttt{y}-\texttt{x}+1$) e de seguida, utilizando os métodos que conhece, determine o respectivo majorante assimptótico.

I.1.2 Considere a função recursiva apresentada abaixo:

```
void xpto(int a[], int l, int r)
{
    int i, j;

    for (j = 0, i = l; i < r; i += 2)
         a[j ++] = a[i] + a[i + 1];

    if (i == r)
         a[j++] = a[r];

    xpto(a, l, (r + l)/2 + 1);
}
```

Determine o menor majorante assimptótico (notação O) para o tempo de execução da função em termos do número de elementos $n = \texttt{r}-\texttt{l}+1$.

I.1.3 Indique a recorrência e a complexidade para a operação ToH, cujo pseudo-código é fornecido abaixo, em função de n. Assuma que a operação Move tem complexidade $O(1)$.

```
ToH(n,A,C,B)
1  if n > 0
2     then ToH(n-1,A,B,C)
3          Move(n,A,C)
4          ToH(n-1,B,C,A)
```

I.1.4 Encontre o menor *majorante assimptótico* (notação O) para a recursão:

$$T(n) \leq \begin{cases} c & , \ n \leq n_0 \\ 4T(n/3) + n & , \ n > n_0 \end{cases}$$

onde c e n_0 são constantes positivas.

I.1.5 Indique a recorrência e a complexidade para a operação ArrayToTree, cujo pseudo-código é fornecido abaixo, em função de $n = l - r + 1$. Assuma que a operação NewNode tem complexidade $O(1)$.

```
ArrayToTree(a,l,r)
1  if r < l
2     then return NIL
3     else m ← ⌊(l + r)/2⌋
4          left ← ArrayToTree(a, l, m − 1)
5          right ← ArrayToTree(a, m + 1, r)
6          return NewNode(a[m], left, right)
```

I.1.6 Indique a recorrência e a complexidade para a operação SiftUp cujo pseudo-código é fornecido abaixo.

```
SiftUp(A,n)
1  p ← ⌊n/2⌋
2  if n > 1 and A[p] < a[n]
3     then Swap(a[p], a[n])
4          SiftUp(A, p)
```

I.1.7 Encontre o menor *majorante assimptótico* (notação O) para a recursão

$$T(n) \leq \begin{cases} c & , \ n \leq n_0 \\ 8T(n/2) + 2n^3 + 3n & , \ n > n_0 \end{cases}$$

onde c e n_0 são constantes positivas.

I.1.8 Considere a função recursiva:

```
int sumxtoy(int x, int y) {
  int i, s;
  const c = 10;
  if (y - x < c) {
    s = 0;
    for (i = x; i <= y; i++)
      s += i;
    return s;
  }
  else {
    return sumxtoy(x,(x+y)/2) + sumxtoy((x+y)/2+1,y);
  }
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número de elementos $n = y - x + 1$, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.9 Considere a função recursiva:

```
int f(int n) {
  int r, i;

  r = n;
  i = 0;
  while(r > 0) {
    i++;
    r = r - i;
  }

  r += f(n/9);

  return r;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.10 Considere a função recursiva:

```
int f(int n) {
  int r = 1, i;

  if(n > 1) {
    i = 0;
    while(i*i < n)
      i++;

    r += i + f(n/4);
  }

  return r;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.11 Considere a função recursiva:

```
int f(int n) {
  int i = 0;

  while(i < n)
    i++;

  if(n > 1)
    i = 2*f(n/4)+f(n/4);

  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.12 Considere a função recursiva:

```
int f(int n) {
  int j, i;

  j = 0;
  i = 0;
  while(i < n) {
    j++;
    i += 2;
  }

  if(n > 1)
    i = 2*f(j) + f(j);

  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.13 Considere a função recursiva:

```
int f(int n)
{
  int i = 0;
  while(i*i < n)
    i++;
  if(n > 1)
    i = f(n/4) + f(n/4) + f(n/4);

  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.14 Considere a função recursiva:

```
int f(int n)
{
  int i = 0;
  while(i < n)
    i++;
  if(n > 1)
    i = f(n/8) + f(n/8) + f(n/8) + f(n/8);

  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.15 Considere a função recursiva:

```
int f(int n)
{
  int i = n*n*n;
  while(0 < i)
    i=i/2;
  if(n > 1)
    i = f(n/2) + f(n/2);

  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.16 Considere a função recursiva:

```
int f(int n)
{
  int i = 0;
  while(n > i)
    i=i+4;
  if(n > 1)
    i = f(n/4);

  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.17 Considere a função recursiva:

```
int f(int n)
{
  int i = 0, j = 0;
  while(n*n > i) {
    i = i + 2;
    j++;
  }

  if(n > 1)
    i = 5*f(n/2) + f(n/2) + f(n/2) + f(n/2);

  while (j > 0) {
    i = i + 2;
    j--;
  }
  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.18 Considere a função recursiva:

```
int f(int n)
{
  int i = 0, j = 0;
  while (j < 10) {
    i = i + 2;
    j++;
  }

  if(n > 1)
    i += f(n/2) + 3*f(n/2);

  while (j > 0) {
    i--;
    j = j - 2;
  }
  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.19 Considere a função recursiva:

```
int f(int n)
{
  int i = 0, j = 0;
  while(n > i*i) {
    i++;
    j = j + 2;
  }

  if(n > 1)
    i = f(n/3) + f(n/3) + n * f(n/3);

  while (j > 0) {
    i = i + 2;
    j--;
  }
  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.1.20 Considere a função recursiva:

```
int f(int n) {
  int i = n*n, j = 0;

  while(i > 0) {
    i = i / 2;
    j++;
  }

  if(n > 1)
    i = i * f(n/2) + f(n/2);

  while (j > 0) {
    i++;
    j--;
  }
  return i;
}
```

Indique a expressão (recursiva) que descreve o tempo de execução da função em termos do número n, e de seguida, utilizando os métodos que conhece, determine o menor majorante assimptótico.

I.2. Notação assimptótica - Algoritmos

I.2.1 Para cada uma das seguintes afirmações, indique se é verdadeira (V) ou falsa (F).

a. $n^2 \in \mathcal{O}(n^3)$.
b. $n^3 \in \mathcal{O}(n^2)$.
c. $(n+1)! \in \mathcal{O}(n!)$.
d. Para qualquer função não-negativa f, se $f(n) \in \mathcal{O}(n)$, então $(f(n))^2 \in \mathcal{O}(n^2)$.
e. Para qualquer função não-negativa f, se $f(n) \in \mathcal{O}(n)$, então $2^{f(n)} \in \mathcal{O}(2^n)$.

I.2.2 Seja ε um número real arbitrário tal que $0 < \varepsilon < 1$. Ordene os conjuntos seguintes usando as relações $\subset$ (subconjunto estrito) e $=$ (igualdade de conjuntos):

a. $O(n \log^2 n)$
b. $O(n^8)$
c. $O(n^{1+\varepsilon})$
d. $O((1+\varepsilon)^n)$
e. $O((n^2 + 8n + \log^3 n)^4)$
f. $O(n^2 / \log^2 n)$

I.2.3 Indique a menor complexidade assimptótica para determinar uma árvore abrangente de menor custo num grafo não dirigido e pesado com n vértices e m arcos. Indique também a menor complexidade assimptótica para determinar uma árvore abrangente no caso de um grafo não dirigido e não pesado.

I.2.4 Dada uma implementação de filas de prioridade sobre amontoados binários, indique o menor *majorante assimptótico* (notação O) para cada uma das seguintes operações em relação ao tempo de execução e em função do número de elementos n:

a. inserir um novo elemento;
b. determinar o elemento com maior prioridade;
c. extrair o elemento com maior prioridade;
d. eliminar um qualquer elemento;
e. aumentar a prioridade de um elemento;
f. fundir duas filas de prioridade.

I.2.5 Indique a menor complexidade de um algoritmo para testar a existência de ciclos negativos no algoritmo de Floyd-Warshall. Indique também qual a complexidade para determinar os vértices do ciclo negativo.

I.2.6 Indique as menores complexidades para determinar o corte mínimo após a execução do algoritmo de Edmonds-Karp e do algoritmo Relabel-To-Front, respectivamente.

I.2.7 Considere o algoritmo de Floyd-Warshall e a aplicação de sucessivas procuras em largura primeiro (BFS) a partir de cada vértice para determinar o fecho transitivo de um grafo $G = (V, E)$ orientado e não pesado. Qual é, respectivamente, o menor limite assimptótico para o tempo de cada uma destas abordagens?

I.2.8 Indique os valores dos menores majorantes assimptóticos, simplificados, para o seguintes algoritmos:

- Uma DFS sobre um grafo representado com uma matriz de adjacência.
- Encontrar um ciclo num grafo dirigido, representado em lista de adjacências, em que todos os vértices tem menos de 5 vizinhos.
- Calcular os caminhos mais curtos a partir uma única fonte, num grafo dirigido acíclico com pesos negativos.

I.2.9 Indique os valores dos menores majorantes assimptóticos para o seguintes algoritmos:

- Uma BFS sobre um grafo representado com uma matriz de adjacência.
- O algoritmo de Kruskal quando o peso do arcos varia entre 0 e V.
- Calcular os caminhos mais curtos entre todos os pares de vértices, num grafo dirigido acíclico.

I.2.10 Indique os valores dos menores majorantes assimptóticos para o seguintes algoritmos:

- O algoritmo de Tarjan sobre um grafo representado com uma matriz de adjacência.
- O algoritmo de Bellman-Ford sobre um grafo representado com uma matriz de adjacência.
- O algoritmo de Ford-Fulkerson quando a capacidade dos arcos varia entre 1 e 3.

I.2.11 Indique os valores dos menores majorantes assimptóticos para o seguintes algoritmos:

- O algoritmo de Dijkstra quando o peso dos arcos é um valor inteiro entre 1 e 3.
- O algoritmo de Prim quando o peso dos arcos é um valor inteiro entre 1 e 3.
- O algoritmo de Prim sobre um grafo representado com uma matriz de adjacência.

I.3. Estruturas de dados

I.3.1 Considere um grafo dirigido e não pesado $G = (V, E)$, com $|V| = 4$, no qual quaisquer dois vértices estão ligados por exactamente um arco, e portanto $|E| = 6$. Os vértices estão numerados de 1 a 4. Considere uma DFS com origem no vértice 1, e na qual os vértices são visitados por ordem crescente do seu número. Sabendo que a referida DFS produz 3 arcos de àrvore (*tree edges*), 1 arco para a frente (*forward edge*), 1 arco para trás (*back edge*) e 1 arco cruzado (*cross edge*), indique a matriz de adjacências do grafo.

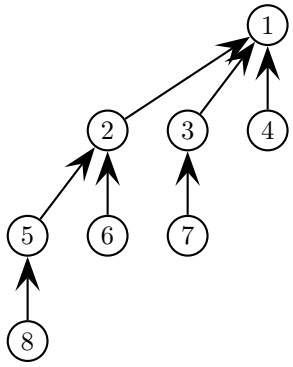
I.3.2 Fecho transitivo em grafos dinâmicos. Considere um grafo dinâmico, não dirigido, $G = (V, E)$, em que é possível acrescentar arcos a E, mas não retirar.

Um vértice v é atingível a partir, de outro vertice, v' quando existe um caminho, de um ou mais arcos, que começa em v e termina em v'. Nesse caso dizemos que o par (v, v') faz parte do fecho transitivo de G.

Descreva uma estrutura de dados que responda a questões de transitividade, ou seja dados v e v', devemos obter verdadeiro caso exista, pelo menos, um caminho entre v e v' e falso caso contrário. A estrutura precisa de ser actualizada quando é acrescentado um arco a E, da forma mais eficiente possível.

Apenas será atribuída a cotação máxima a soluções com a menor complexidade assimptótica.

I.3.3 Assuma que a seguinte árvore representa um conjunto de números. Desenhe a árvore que resulta de executar uma operação de `Find-Set(5)`, utilizando a heurística de compressão de caminhos.



I.3.4 Considere o seguinte conjunto de operações sobre conjuntos disjuntos:

```
UNION-FIND-USAGE()
1  for i ← 1 to 10
2        do Make-Set(x_i)
3  for i ← 1 to 5
4        do Union(x_2i, x_2i−1)
5  Union (x_2, x_7)
6  Union (x_9, x_5)
7  Union (x_4, x_8)
8  Union (x_10, x_2)
```

Use a estrutura em árvore para representação de conjuntos disjuntos com a aplicação das heuristicas de união por categoria e compressão de caminhos. Para cada elemento x_i, indique os valores de categoria ($rank[x_i]$) e o valor do seu pai na árvore ($p[x_i]$).

Nota: Na operação `Make-Set`(x), o valor da categoria de x é inicializado a 0. Na operação de `Union`(x, y), em caso de empate, considere que o representante de y é que fica na raíz.

I.3.5 Considere o seguinte conjunto de operações sobre conjuntos disjuntos:

```
UNION-FIND-USAGE()
 1  for i ← 1 to 10
 2        do Make-Set(x_i)
 3  for i ← 1 to 5
 4        do Union(x_2i, x_2i−1)
 5  Union (x_1, x_10)
 6  j ← Find-Set(x_2)
 7  k ← Find-Set(x_7)
 8  Union (x_7, x_4)
 9  Union (j, x_8)
10  Union (x_6, k)
```

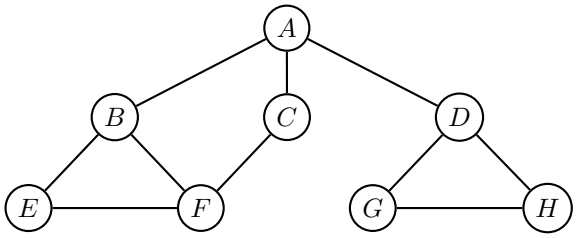
Use a estrutura em árvore para representação de conjuntos disjuntos com a aplicação das heurísticas de união por categoria e compressão de caminhos. Para cada elemento x_i, indique os valores de categoria ($rank[x_i]$) e o valor do seu pai na árvore ($p[x_i]$).

Nota: Na operação `Make-Set`(x), o valor da categoria de x é inicializado a 0. Na operação de `Union`(x, y), em caso de empate, considere que o representante de y é que fica na raíz.

Parte II.
Algoritmos em Grafos

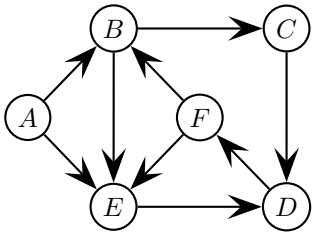
II.1. Algoritmos Elementares em Grafos - BFS e DFS

II.1.1 Considere o grafo não-dirigido:



Considere uma procura em largura (BFS) com início em D. Indique os valores de d e π para cada um dos vértices.

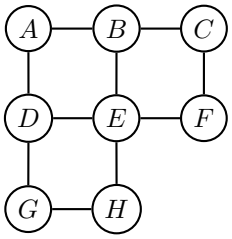
II.1.2 Considere o grafo não-dirigido:



Considere uma procura em largura (BFS) com início em A. Indique os valores de d e π para cada um dos vértices.

Nota: Os tempos de uma BFS começam em 0.

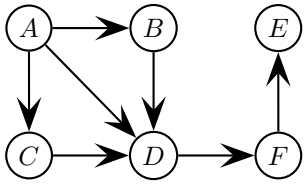
II.1.3 Considere o grafo não-dirigido:



Considere uma procura em largura (BSF) com início em A. Indique os valores de d e π para cada um dos vértices.

Nota: Os tempos de uma BFS começam em 0.

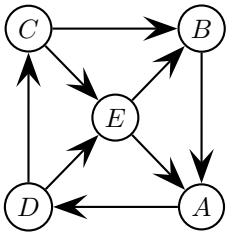
II.1.4 Considere o grafo dirigido:



Execute uma procura em largura (BFS) com início em A. Indique os valores de d e π para cada um dos vértices.

Nota: As distâncias de uma BFS começam em 0.

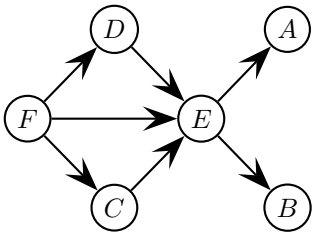
II.1.5 Considere o grafo dirigido:



Execute uma procura em largura (BFS) com início em D. Indique os valores de d e π para cada um dos vértices. Assuma que os vértices adjacentes são visitados por ordem lexicográfica.

Nota: As distâncias de uma BFS começam em 0.

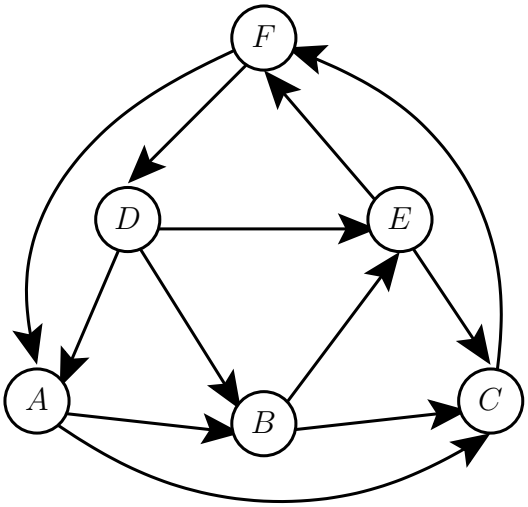
II.1.6 Considere o grafo dirigido:



Execute uma procura em largura primeiro (BFS), com início em F. Indique os valores de d e π para cada um dos vértices. Assuma que os vértices adjacentes são visitados por ordem lexicográfica.

Nota: As distâncias de uma BFS começam em 0.

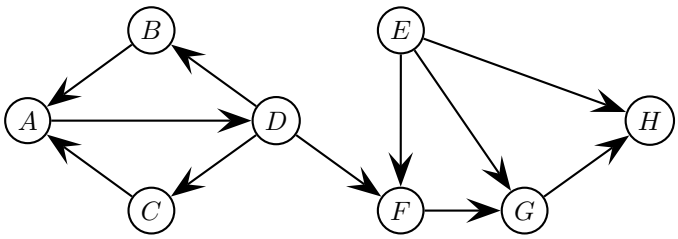
II.1.7 Considere o grafo dirigido:



Execute uma procura em largura (BFS) com início em D. Indique os valores de d e π para cada um dos vértices. Assuma que os vértices adjacentes são visitados por ordem lexicográfica.

Nota: As distâncias de uma BFS começam em 0.

II.1.8 Considere o grafo dirigido:



Indique os tempos de descoberta d e de finalização f, para cada um dos vértices, após a execução de uma procura em profundidade primeiro (DFS), onde os vértices são considerados por ordem lexicográfica (ou seja, $A, B, C, \ldots$).

II.1.9 Relativamente à execução do algoritmo DFS num grafo $G = (V, E)$, ligado, dirigido, com n vértices e m arcos, indique se cada uma das afirmações seguintes é verdadeira ou falsa.

a. O valor n representa um limite superior quer no número de arcos quer no número de árvores na floresta de DFS.

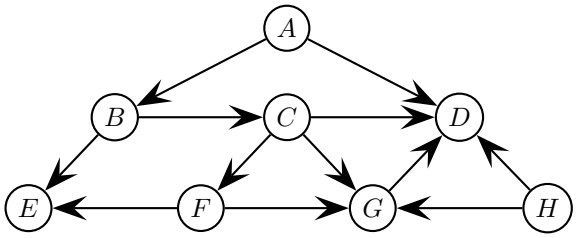
b. Não podem existir vértices u, com arcos de entrada e arcos de saída, tais que $f[u] = d[u] + 1$.

c. Para cada vértice v, os valores possíveis para os tempos de descoberta $d[v]$ e de fim $f[v]$ variam entre 1 e $2n$.

d. É possível identificar um arco para trás (u, v) tal que $[d[u], f[u]] \cap [d[v], f[v]] \neq \emptyset$.

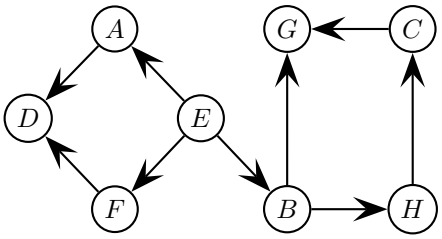
e. O número de valores distintos de $d[u]$ na DFS é diferente e inferior ao número de valores distintos de $d[u]$ na BFS, executada no mesmo grafo e relativa a um qualquer vértice inicial s.

II.1.10 Considere o grafo dirigido da figura seguinte.



Indique os tempo de descoberta d e de finalização f, para cada um dos vértices, após a execução de uma procura em profundidade primeiro (DFS), onde os vértices são considerados por ordem lexicográfica.

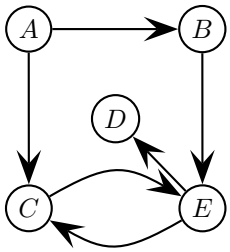
II.1.11 Considere o grafo dirigido aciclico DAG:



Aplique uma procura em profundidade primeiro (DFS) com início no vértice E e que visita os filhos por ordem lexicográfica. Indique os valores de descoberta (d) e fim (f), para cada um dos vértices. Indique também a ordenação topológica resultante.

Nota: Os tempos de uma DFS começam em 1.

II.1.12 Considere o grafo dirigido:

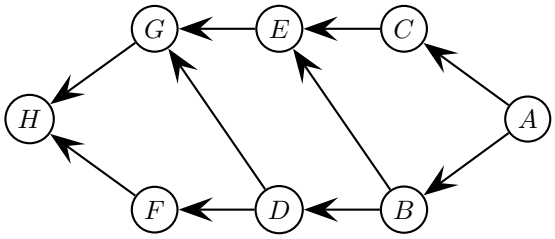


Indique os tempos de descoberta d e de finalização f, para cada um dos vértices, após a execução de uma procura em profundidade primeiro (DFS) onde os vértices são considerados por ordem lexicográfica (ou seja A, B, C,...) e os vizinhos de cada nó também são visitados por ordem lexicográfica.

Nota: Os tempos de uma DFS começam em 1.

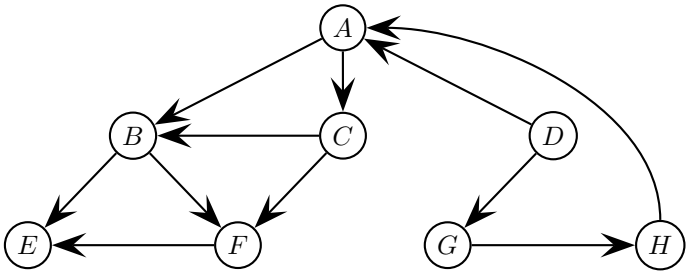
II.2. Ordenações Topológicas

II.2.1 Considere o grafo dirigido:



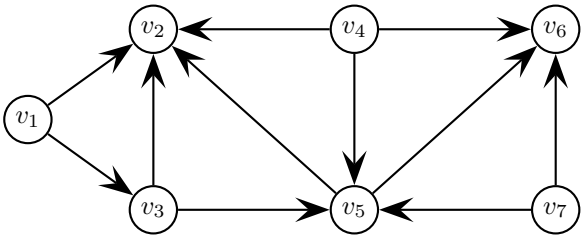
Indique duas das ordenações topológicas possíveis.

II.2.2 Considere o grafo dirigido:

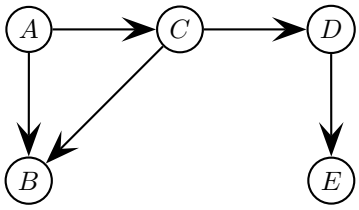


Indique os tempos de fim de uma DFS no grafo e a ordenação topológica resultante, considerando que os vértices são visitados por ordem lexicográfica.

II.2.3 Execute uma DFS no grafo abaixo e indique os tempos de início, de fim e a ordenação topológica resultante. Os vértices devem ser considerados por ordem crescente dos índices na execução da DFS.

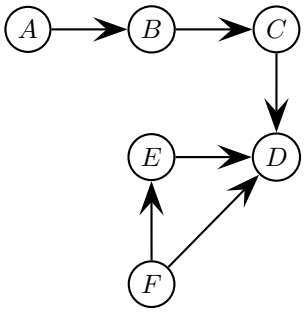


II.2.4 Considere o seguinte grafo orientado.



Indique os tempos de descoberta d e de finalização f, para cada um dos vértices, após a execução de uma procura em profundidade primeiro (DFS), onde os vértices são considerados por ordem lexicográfica (ou seja, A, B, C, ...). Qual é a ordenaçao topológica obtida pela DFS para o grafo acima?

II.2.5 Considere o grafo dirigido:

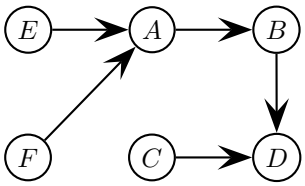


Indique quantas ordenações topológicas existem para este grafo.

Sugestão: Começe por resolver este problema sem o vértice E.

Indique a ordenação que resulta de uma procura em profundidade primeiro (DFS) com início no vértice A e que visita os filhos por ordem lexicográfica. Os recomeços da DFS escolhem o vértice não visitado com a menor letra, de acordo com a ordem lexicográfica.

II.2.6 Considere o grafo dirigido acíclico (DAG):

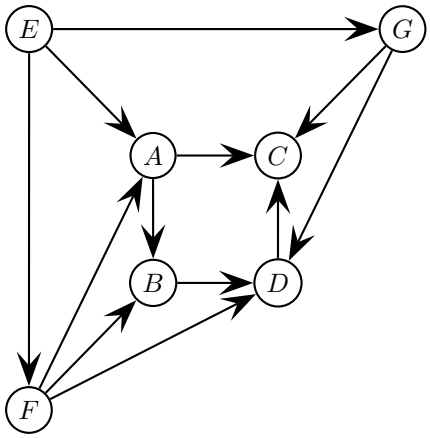


Aplique uma procura em profundidade primeiro (DFS) com início no vértice A e que visita os adjacentes por ordem lexicográfica. Os recomeços da DFS escolhem o vértice não visitado com a menor letra, de acordo com a ordem lexicográfica.

Indique os valores de descoberta (d) e fim (f), para cada um dos vértices. Apresente a ordenação topológica resultante da DFS.

Nota: Os tempos de uma DFS começam em 1.

II.2.7 Considere o grafo dirigido:

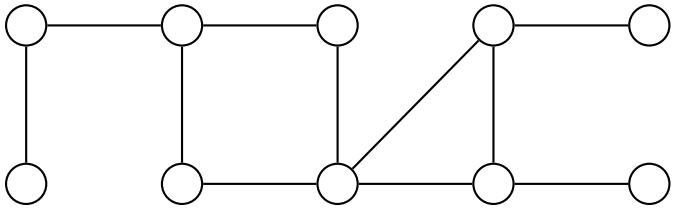


Indique uma ordenação topológica do grafo usando o algoritmo em profundidade primeiro (DFS) em que os vértices são explorados por ordem alfabética. Indique os tempos de descoberta d e de fim f de cada vértice.

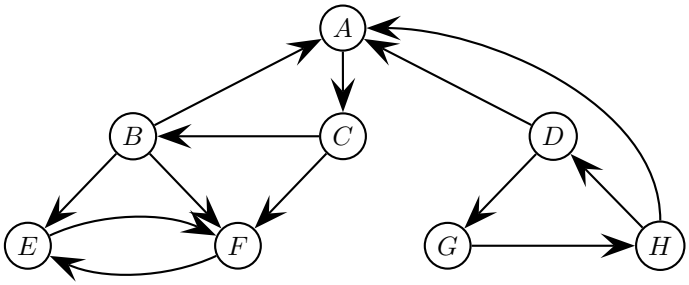
Nota: Neste algoritmo os valores de d começam em 1.

II.3. Componentes Fortemente Ligados

II.3.1 Considere o grafo dirigido representado abaixo, onde não foi indicado o sentido dos arcos. Dependendo do sentido dos arcos, o número de componentes fortemente ligados (SCCs) existentes no grafo poderá ser diferente. Indique todos os valores possíveis para o número de SCCs que podem existir no grafo.

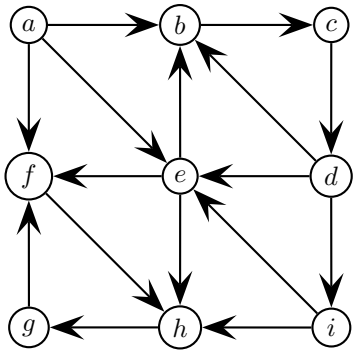


II.3.2 Considere o grafo dirigido:

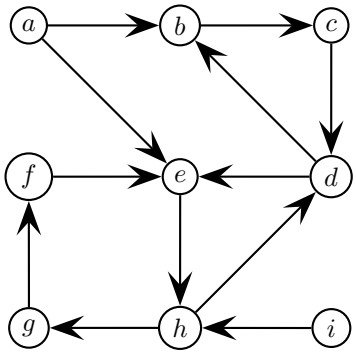


Quais são os componentes fortemente ligados? Admitindo que utilizou o algoritmo de Tarjan, indique o menor tempo de descoberta para cada componente fortemente ligada.

II.3.3 Para o grafo da figura, indique para cada vértice os valores d e l (low) após a execução do algoritmo de Tarjan. Considere que visita os vértices adjacentes por ordem lexicográfica.

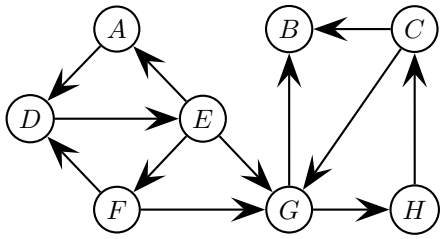


II.3.4 Considere o grafo dirigido da figura seguinte.



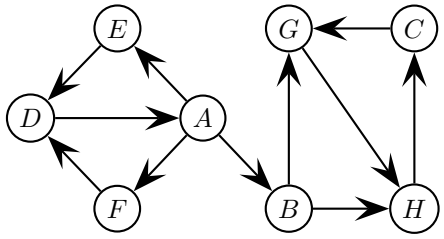
Para o grafo da figura, indique o número de componentes fortemente ligados (SCC) do grafo, e obtenha o grafo acíclico dos componentes fortemente ligados G^{SCC}.

II.3.5 Considere o grafo dirigido:



Considere o algoritmo de Tarjan e indique os valores de d e low, para cada um dos vértices. Indique também as componenetes fortemente ligadas. Assume que os vértices são considerados por ordem lexicográfica (ou seja A, B, C,...).

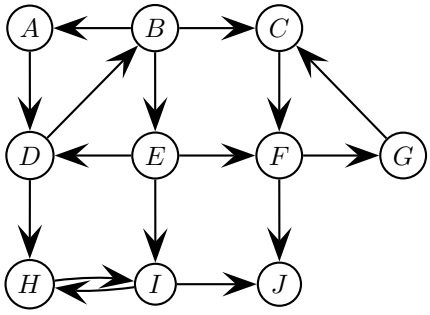
II.3.6 Considere o seguinte grafo dirigido :



Aplique o algoritmo de Tarjan para encontrar componentes fortemente ligadas. Assuma que o algoritmo começa no vértice A e que visita os vertices adjacents por ordem lexicográfica. Indique os valores de descoberta (d) e low, para cada um dos vértices. Indique o número de componentes resultantes.

Nota: Os tempos d começam em 1.

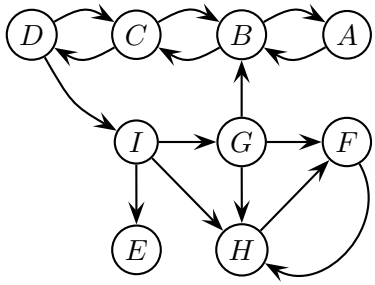
II.3.7 Considere o grafo dirigido:



Aplique ao algoritmo de Tarjan para determinar componentes fortemente ligados. Indique os respectivos valores de d, low obtidos pelo algoritmo. Apresente também a partição dos vértices em componentes. Inicie o algoritmo no vértice E e, em cada vértice, processe os vizinhos por ordem lexicográfica (ou seja A, B, C,...).

Nota: Neste algoritmo os valores de d começam em 0.

II.3.8 Considere o grafo dirigido:

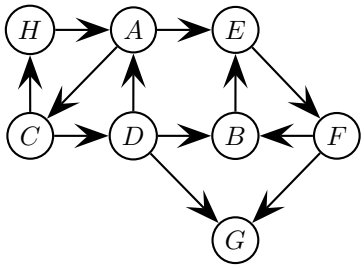


Aplique o algoritmo de Tarjan no grafo. Considere que inicia a sua travessia pelo vértice D e que os vértices são explorados por ordem alfabética.

Indique os vértices de cada componente fortemente ligado do grafo. Indique os tempos de descoberta d e de low de cada vértice.

Nota: Neste algoritmo os valores de d começam em 1.

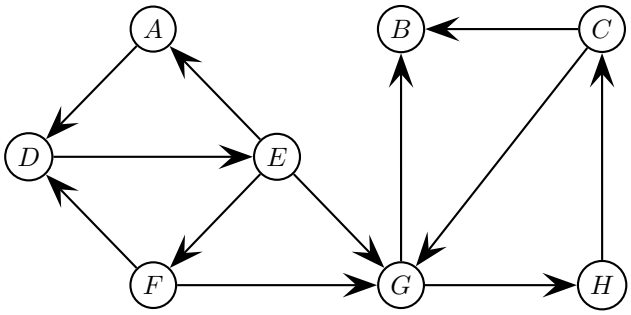
II.3.9 Considere o grafo dirigido:



Aplique o algoritmo de Tarjan, para encontrar componentes fortemente ligados, e indique os valores de d e low obtidos pelo mesmo. Assuma que o algoritmo começa no vértice H, mas que os vértices vizinhos e possiveis recomeços usam a ordem lexicográfica (ou seja, A, B, C,...).

Nota: Assuma que os valores de d começam em 0.

II.3.10 Considere o grafo dirigido:



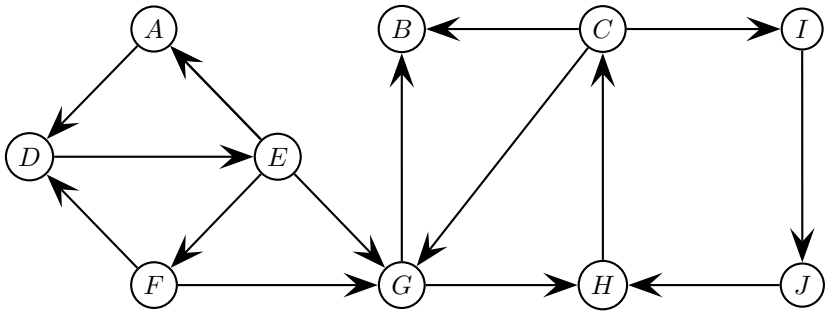
Aplique o algoritmo que utiliza duas travessias em profundidade primeiro (DFS) para encontrar os componentes fortemente ligados do grafo. Considere que na primeira DFS os vértices são considerados por ordem lexicográfica (ou seja, A, B, C...). Em ambas as DFS, os adjacentes são sempre considerados também por ordem lexicográfica.

Indique os tempos de descoberta d e de fim f de cada vértice na **segunda DFS do grafo**.

Indique os componentes fortemente ligados **pela ordem que são descobertos pelo algoritmo**.

Nota: Neste algoritmo os valores de d começam em 1.

II.3.11 Considere o grafo dirigido:



Aplique o algoritmo de Tarjan para encontrar os componentes fortemente ligados do grafo. Os vértices são sempre considerados por ordem lexicográfica (ou seja, A, B, C...). Os adjacentes também são sempre considerados por ordem lexicográfica.

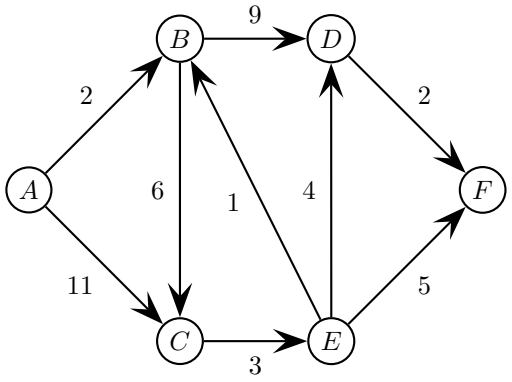
Para cada vértice indique os valores de descoberta d e low após a aplicação do algoritmo.

Indique os componentes fortemente ligados **pela ordem que são descobertos pelo algoritmo**.

Nota: Neste algoritmo os valores de d começam em 1.

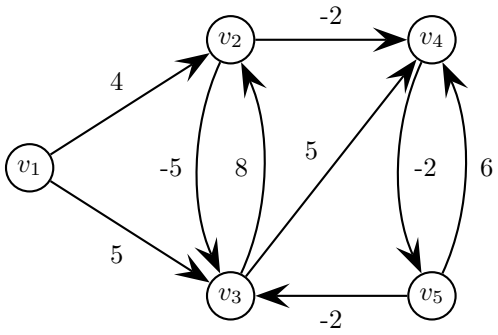
II.4. Caminhos mais Curtos Fonte Única

II.4.1 Considere o grafo dirigido e pesado da figura.



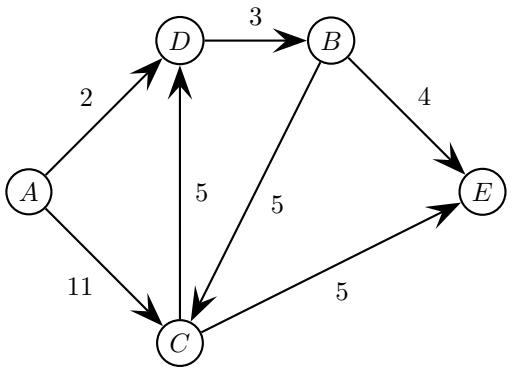
Aplique o algoritmo de Dijkstra para determinar os caminhos mais curtos do vértice A para cada um dos outros vértices. Indique o valor de d e de π para cada vértice.

II.4.2 Considere a execução do algoritmo de Bellman-Ford para o grafo da figura com início em v_1. Indique o predecessor π para cada vértice após a execução do algoritmo. Indique também o menor número de ierações de relaxação necessárias para poder terminar o algoritmo.



II.4.3 Considere o algoritmo de Dijkstra para determinar os caminhos de menor custo a partir de um dado vértice num grafo orientado e pesado. Proponha uma implementação para a fila de prioridade utilizada neste algoritmo por forma a que o mesmo corra em tempo $O(V^2)$, independentemente do número de arcos. Justifique.

II.4.4 Considere o grafo orientado e pesado da figura.



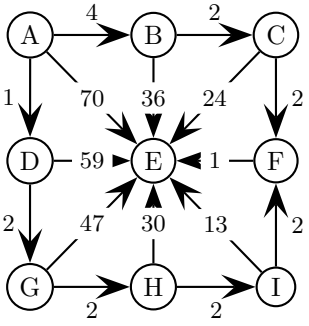
Aplique o algoritmo de Dijkstra para determinar os caminhos mais curtos de A para todos os vértices. Indique o predecessor e o custo do caminho mais curto para cada um dos vértices. Indique também a ordem pela qual o algoritmo de Dijkstra expande os vértices.

II.4.5 Consider o algoritmo de Bellman-Ford descrito no pseudo-código abaixo:

Bellman-Ford(G, w, s)
1 Initialize-Single-Source(G, s)
2 **for** $i \leftarrow 1$ **to** $|V[G]| - 1$
3 **do for** each $(u, v) \in E[G]$
4 **do** Relax(u, v, w)
5 **for** each $(u, v) \in E[G]$
6 **do if** $d[v] > d[u] + w(u, v)$
7 **then return** $FALSE$ ▷ Ciclo negativo
8 **return** $TRUE$ ▷ Sem ciclos negativos

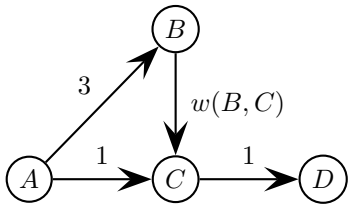
Explique como modificar o algoritmo por forma a que o **for** exterior execute menos do que $|V|$ vezes, para alguns grafos. Se no algoritmo modificado o ciclo exterior executar m vezes, que propriedade do grafo representa o valor m ?

II.4.6 Considere o grafo dirigido e pesado da figura.



Aplique o algoritmo de Dijkstra com origem no vértice A. Indique a ordem pela qual os vértices são retirados da fila de prioridade. Indique também a sequência de valores que a expressão `d[E]` assume ao longo da execução.

II.4.7 Considere o seguinte grafo dirigido e pesado:

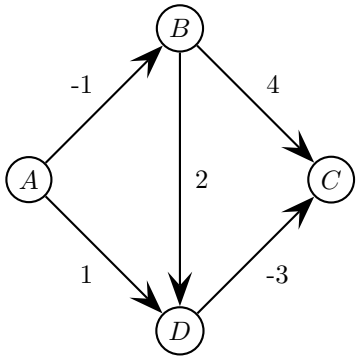


Considere a execução do algoritmo de Dijkstra, que determina o peso dos caminhos mais curtos, a partir do vértice A. Indique um valor de w para o arco (B, C) que faça com que o valor de $d[D]$ obtido não corresponda ao valor do caminho mais curto de A para D, i.e., $\delta(A, D)$.

II.5. Caminhos mais Curtos Entre Todos os Pares

II.5.1 Dado um grafo dirigido $G = (V, E)$, o seu fecho transitivo é um grafo $G' = (V, E')$ tal que, se existe um caminho de u para v em G, então $(u, v) \in E'$. Indique como é que o algoritmo de Floyd-Warshall pode ser utilizado para determinar o fecho transitivo de G.

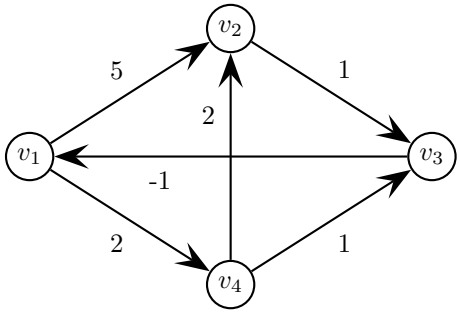
II.5.2 Considere a execução do algoritmo de Johnson, sobre o grafo dirigido e pesado da figura abaixo.



Quantas iterações do ciclo principal do algoritmo Bellman-Ford são necessárias no mínimo na primeira fase do algoritmo de Johnson (excluindo a iteração de confirmação)? Após a repesagem, começando no vértice B, apresente a ordem pela qual o algoritmo de Dijkstra extrai vértices da fila de prioridades. Começando no vértice B, quais são os pesos dos caminhos mais curtos, dados pelo algoritmo de Dijkstra, antes da re-repesagem (deve responder com $\hat{\delta}(B, v)$ e não $\hat{\delta}(B, v) + \delta(s, v) - \delta(s, B)$).

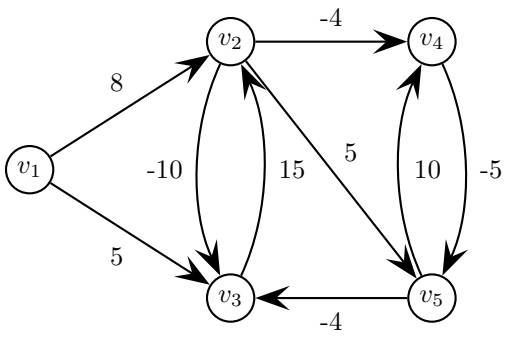
Nota: No que diz respeito à ordem de extracção dos vértices pelo algoritmo de Dijkstra, basta escrever 1 para o primeiro vértice, 2 para o segundo, etc. na grelha da folha de rosto.

II.5.3 Indique os valores da matriz Π^3 na execução do algoritmo de Floyd-Warshall para determinar os caminhos mais curtos entre todos os pares de vértices no grafo abaixo.

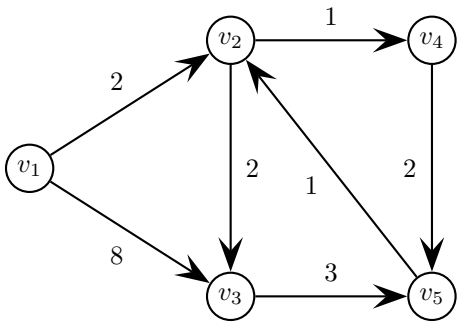


II.5.4

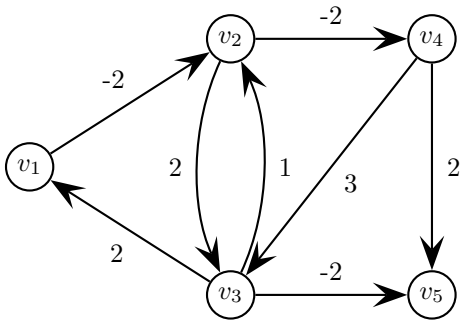
Considere a execução do algoritmo de Johnson para o grafo da figura. Indique quais os pesos dos arcos após a repesagem.



II.5.5 Indique os valores da matriz D^3 na execução do algoritmo de Floyd-Warshall para determinar os caminhos mais curtos entre todos os pares de vértices no grafo abaixo.



II.5.6 Indique qual o valor do caminho mais curto do vértice v_1 para o vértice v_3 após a repesagem dos arcos na execução do algoritmo de Johnson para o exemplo seguinte.



II.5.7 Indique se cada uma das seguintes afirmações é verdadeira (V) ou falsa (F).

a. O algoritmo de Bellman-Ford permite detectar ciclos negativos.

b. O tempo de execução do algoritmo de Floyd-Warshall é $O(VE)$.

c. Os caminhos mais curtos obedecem sempre à desigualdade triangular.

d. O tempo de execução do algoritmo de Bellman-Ford é $O(VE)$.

e. O algoritmo de Floyd-Warshall permite determinar sempre os caminhos de menor custo entre todos os pares de vértices, mesmo quando existem ciclos negativos.

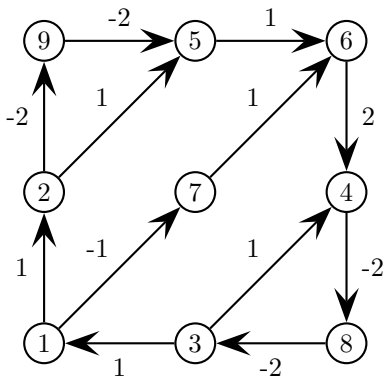
f. A repesagem dos arcos no algoritmo de Johnson conduz sempre a pesos diferentes dos originais.

II.5.8 Num dos algoritmos para determinar os caminhos mais curtos entre todos os pares de vértices, a entrada i, j da matriz de distância $D^{(2m)}$ é determinada da seguinte forma:

$$D_{ij}^{(2m)} = \min(D_{ij}^{(m)}, D_{ik}^{(m)} + D_{kj}^{(m)})$$

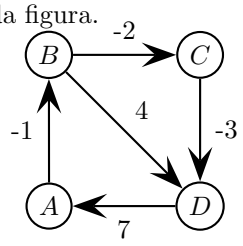
Indique como alterar/substituir esta linha para que a matriz $\Pi^{(2m)}$ seja também calculada. Assuma que a matriz $\Pi^{(m)}$ foi também calculada na iteração anterior.

II.5.9 Considere o grafo não dirigido e pesado da figura.



Considere o algoritmo Floyd-Warshall. Calcule os seguintes valores $d^{(4)}(1,6)$, $d^{(5)}(1,6)$, $d^{(7)}(1,6)$, $d^{(8)}(1,6)$, $d^{(9)}(1,6)$.

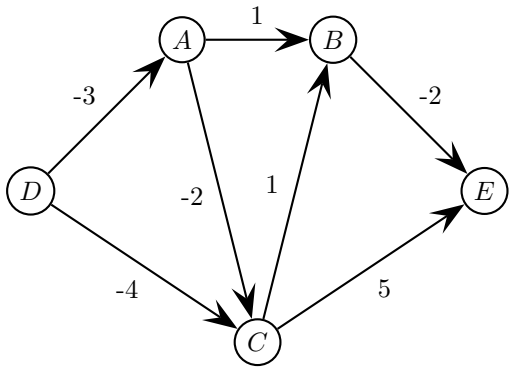
II.5.10 Considere o grafo pesado da figura.



Aplique o algoritmo de repesagem de `Jonhson` ao grafo. Indique o valor da função h e os novos pesos.

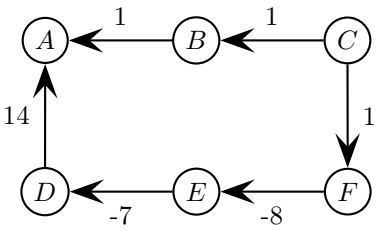
II.5.11 Dado um grafo pesado $G = (V, E, w)$ considere o seguinte procedimento de repesagem. Calcule $w^* = \min_{(u,v)\in E}\{w(u, v)\}$ e defina os novos pesos como $\hat{w}(u, v) = w(u, v) + w^*$.
Prove que este método funciona, ou mostre um contra-exemplo em como não funciona.

II.5.12 Considere o grafo pesado da figura.



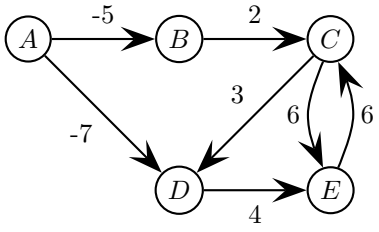
Aplique o algoritmo de repesagem de `Jonhson` ao grafo. Indique o valor da função h para cada vertice e o novo peso do arco (B, E).

II.5.13 Considere o grafo dirigido e pesado da figura.



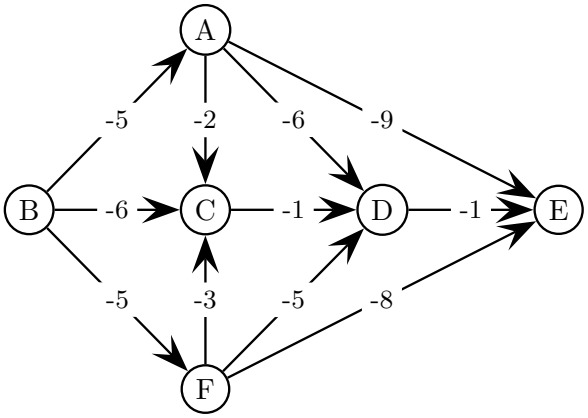
Considere a aplicação do algoritmo de Johnson ao grafo. Qual o peso dos arcos (B, A), (C, F) e (E, D) após a função de repesagem?

II.5.14 Considere o grafo pesado da figura.



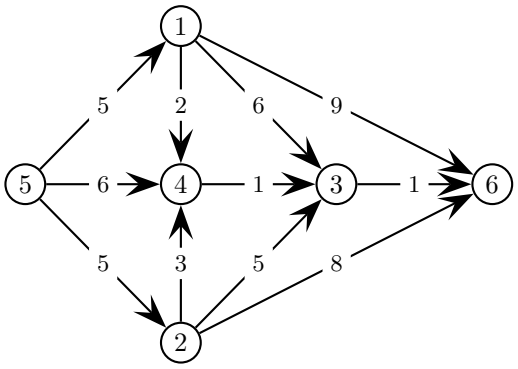
Indique os valores da função de repesagem de Johnson para os vértices e o valor dos novos pesos para os arcos (A,B), (A,D), (C,D), (C,E) e (E,C).

II.5.15 Considere o grafo dirigido e pesado da figura.



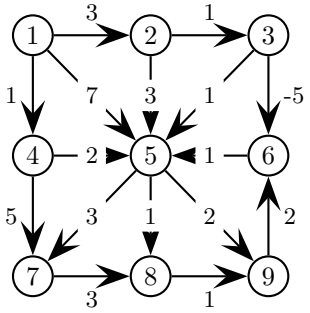
Indique os valores da função h do algoritmo de Johnson. Indique também os novos pesos dos arcos (A, C), (F, C),(C, D) e (D, E).

II.5.16 Considere o grafo dirigido e pesado da figura.



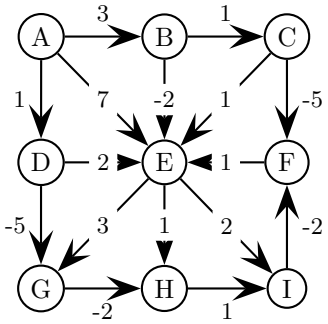
Considere o algoritmo Floyd-Warshall. Calcule os seguintes valores $d^{(1)}(5,6)$, $d^{(2)}(5,6)$, $d^{(3)}(5,6)$, $d^{(4)}(5,6)$.

II.5.17 Considere o grafo dirigido e pesado da figura.



Considere o algoritmo Floyd-Warshall. Calcule os seguintes valores $d^{(2)}(1,5)$, $d^{(2)}(1,9)$, $d^{(3)}(1,5)$, $d^{(4)}(1,5)$, $d^{(5)}(1,9)$, $d^{(6)}(1,9)$.

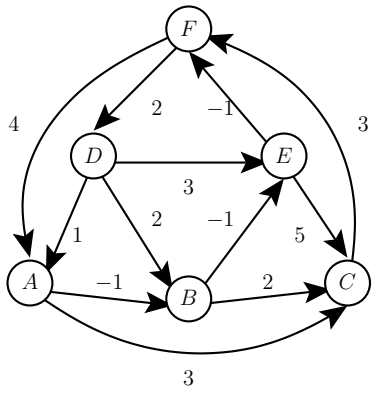
II.5.18 Considere o grafo dirigido e pesado da figura.



Considere o algoritmo de Johnson. Calcule os valores de $h(u)$ para todos os vértices $u \in V$ do grafo.

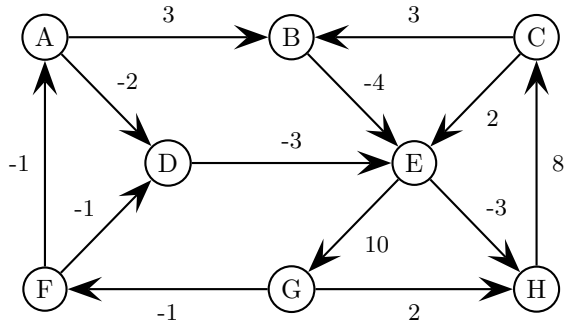
Calcule os pesos dos seguintes arcos após a repesagem: (B,E), (D,E), (E,G), (E,I), (H,I).

II.5.19 Considere o grafo pesado da figura.



Indique os valores da função de repesagem de Johnson para os vértices e o valor dos novos pesos para os arcos (B,C), (B,E), (D,E) e (E,F).

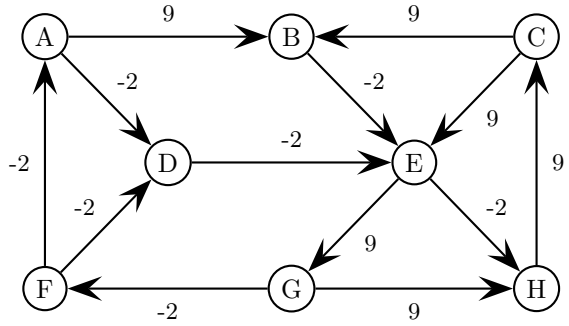
II.5.20 Considere o grafo dirigido e pesado da figura.



Considere o algoritmo de Johnson. Calcule os valores de $h(u)$ para todos os vértices $u \in V$ do grafo.

Calcule os pesos dos seguintes arcos após a repesagem: (A, B), (B, E), (C, E), (E, G), (G, H).

II.5.21 Considere o grafo dirigido e pesado da figura.

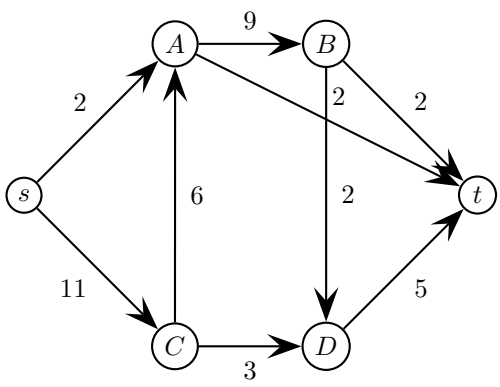


Considere a aplicação do algoritmo de Johnson ao grafo. Calcule os valores de $h(u)$ para todos os vértices $u \in V$ do grafo.

Indique, os pesos dos seguintes arcos após a repesagem: (A, B), (B, E), (C, E), (E, G), (G, H).

II.6. Fluxo Máximo

II.6.1 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



Considere o algoritmo de `RELABEL-TO-FRONT`. Qual é a altura e o excesso de cada vértice após três ciclos de descarga? Note que a descarga inicial do vértice s é ignorada na contagem e considere que a lista de vértices e as listas de vizinhos para cada vértice estão ordenadas lexicograficamente.

II.6.2 Indique se cada uma das seguintes afirmações é verdadeira (V) ou falsa (F).

a. Após a aplicação do algoritmo de Edmonds-Karp, a complexidade de identificar o corte minímo utilizando a rede residual é $O(V + E)$.

b. Após a aplicação do algoritmo de Edmonds-Karp, o valor do fluxo máximo é dado por $\Sigma_{v \in V} f(u, v)$ para qualquer vértice $u \in V$.

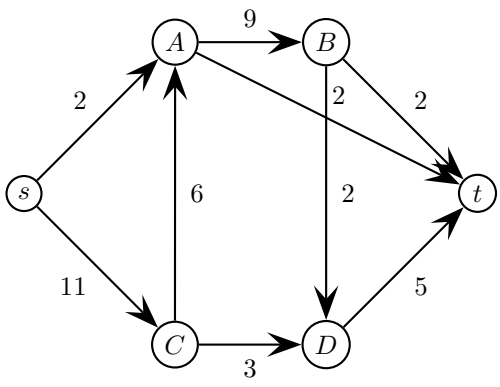
c. Na execução do algoritmo de Edmonds-Karp, o número de arcos nos caminhos de aumento cresce monotonicamente durante a execução do algoritmo.

d. Após a aplicação do algoritmo de Edmonds-Karp, é possível que exista um par de vértices (u, v) tal que $c(u, v) = 0$ e $f(u, v) > 0$.

e. A complexidade do algoritmo de Edmonds-Karp é $O(E^2V)$.

f. O método de Ford-Fulkerson permite detectar um emparelhamento bipartido máximo em tempo $O(VE)$.

II.6.3 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



Aplique o algoritmo de `EDMONDS-KARP` ao grafo. Quais são as capacidades residuais após três aumentos? Quais são os cortes mínimos?

II.6.4 Indique se cada uma das seguintes afirmações é verdadeira (V) ou falsa (F).

a. Seja $e[v]$ o excesso de fluxo do vértice v num determinado passo do algoritmo Relabel-To-Front e $|f^*|$ o valor do fluxo máximo, então verifica-se que:

$$\sum_{v \in V} e[v] \leq |f^*|$$

b. Numa rede de fluxo com $2|V| - 4$ arcos, cada um com capacidade 1, e onde não existem arcos a ligar a *source* à *sink*, o maior valor que o fluxo máximo pode assumir é $|f^*| = |V|$.

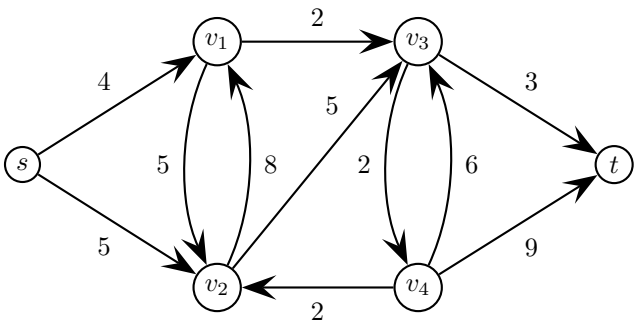
c. Durante a execução do algoritmo Relabel-To-Front, a altura de qualquer vértice nunca excede $2|V| - 1$.

d. Seja $G = (V, E)$ um grafo bipartido com partição de vértices $V = L \cup R$, e seja G' a rede de fluxo correspondente. Nestas condições, o comprimento de qualquer caminho de aumento em G', durante a execução do algoritmo de Ford-Fulkerson, é limitado superiormente por $2 \times \min(|L|, |R|) + 1$.

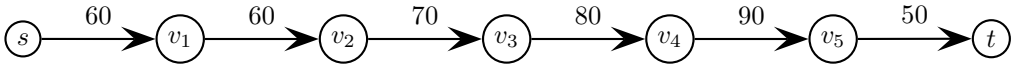
e. No algoritmo Relabel-To-Front só é possível enviar fluxo do vértice u para o vértice v se $h[u] > h[v] + 1$.

f. Após a execução do algoritmo Relabel-To-Front é possível detectar um corte mínimo em tempo $O(V)$.

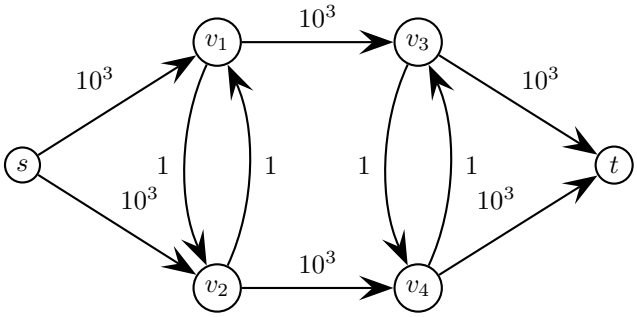
II.6.5 Indique os caminhos de aumento, a sua capacidade e o fluxo máximo após a execução do algoritmo Edmonds-Karp para o grafo da figura. Indique também o corte mínimo e a sua capacidade.



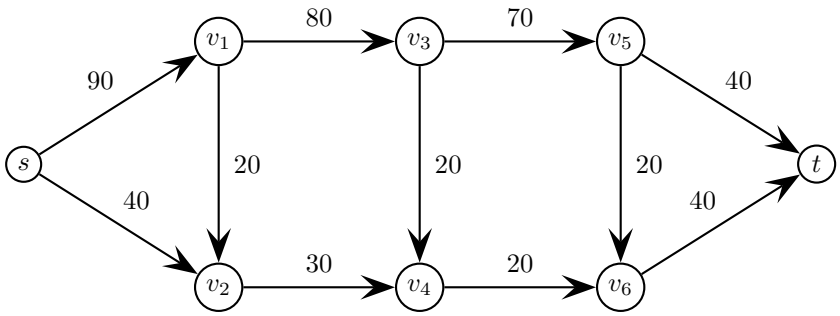
II.6.6 Indique a maior altura na execução do algoritmo Relabel-To-Front para o grafo da figura.



II.6.7 Indique o número máximo de iterações do método de Ford-Fulkerson para a rede de fluxo da figura seguinte:



II.6.8 Para a rede de fluxo da figura seguinte, indique qual a altura máxima para o vértice v_6 na execução do algoritmo Relabel-to-Front.

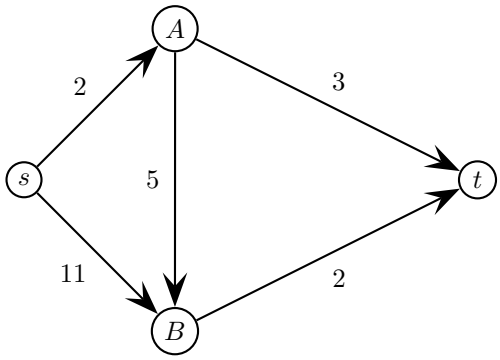


II.6.9 Seja $G = (V, E)$ uma rede de fluxo onde s e t são respectivamente os vértices fonte e destino na rede. Assuma que as capacidades são inteiras, i.e., $c(u, v) \in \mathbb{N}$ para qualquer $(u, v) \in E$. Suponha que após o cálculo do fluxo máximo em G, a capacidade do arco $(u, v) \in E$ é alterada. Proponha um algoritmo eficiente para actualizar o fluxo máximo em G após a alteração de capacidade do arco (u, v) em $k \in \mathbb{N}$ unidades. Note que deverá considerar o caso em que a capacidade do arco aumenta k e o caso em que a capacidade do arco diminui k. Apenas as soluções eficientes serão consideradas.

Sugestão: comece pelo caso de $k = 1$.

II.6.10 Dado um grafo $G = (V, E)$ não orientado e bipartido, indique qual é a menor complexidade assimptótica para determinar o emparelhamento máximo em G se utilizarmos o algoritmo de Edmonds-Karp.

II.6.11 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



Aplique o algoritmo `Relabel-To-Front` para determinar o fluxo máximo de s para t. Indique a altura h para cada vértice no final do algoritmo. Quais são os cortes mínimos?

II.6.12 Indique se cada uma das seguintes afirmações é verdadeira (V) ou falsa (F).

a. No método de Ford-Fulkerson, para um rede de fluxo com capacidades inteiras, um arco pode ser crítico no máximo $O(|f^*|)$ vezes.

b. Após a aplicação do método de Ford-Fulkerson é possível detectar um corte mínimo em tempo $O(V + E)$.

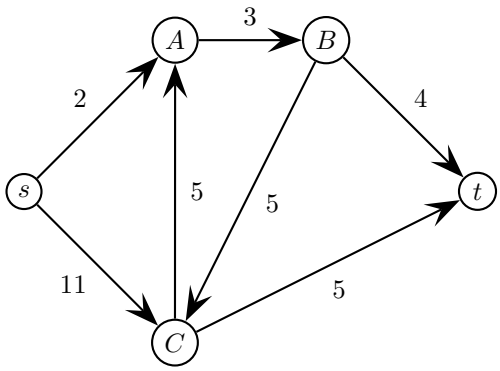
c. A complexidade do método de Ford-Fulkerson é $O(V^3)$.

d. Durante a execução do método de Ford-Fulkerson pode existir um vértice $u \in V \setminus \{s, t\}$ tal que $\sum_{v \in V} f(u, v) \neq 0$.

e. Após a execução do método de Ford-Fulkerson não pode existir um caminho na rede residual nem de s para t, nem de t para s.

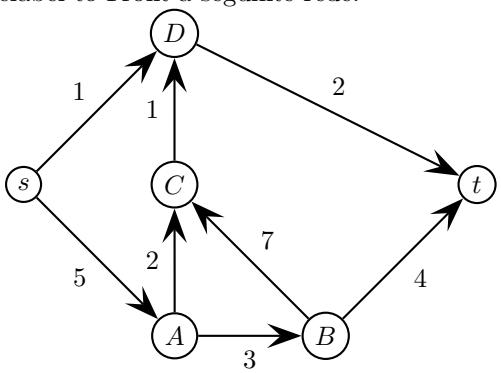
f. Após a execução do método de Ford-Fulkerson, podem existir mais do que um corte mínimo.

II.6.13 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



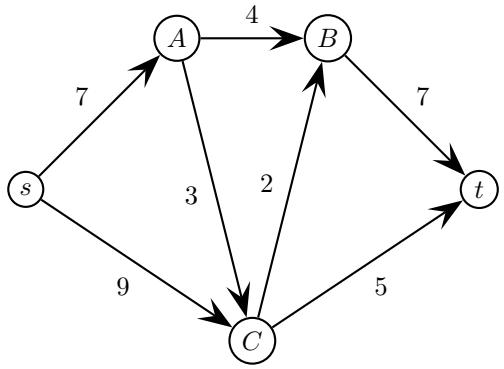
Aplique o método de `Ford-Fulkerson` ao grafo, utilizando procura em largura (BFS) para procurar os caminhos de aumento. Quais são as capacidades residuais após dois caminhos de aumento? Quais são os cortes mínimos?

II.6.14 Aplique o algoritmo Relabel to Front à seguinte rede:



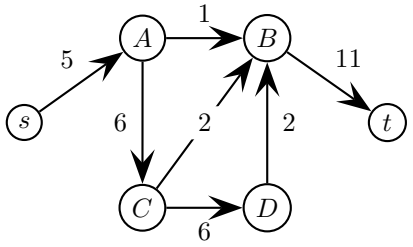
Indique o estado da lista L no final do algoritmo, assumindo que o estado inicial da lista L é $\{A, B, C, D\}$ e que as listas de vizinhos são as seguintes: $N[A] = \{s, C, B\}$, $N[B] = \{A, C, t\}$, $N[C] = \{D, B, A\}$, $N[D] = \{s, C, t\}$.

II.6.15 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



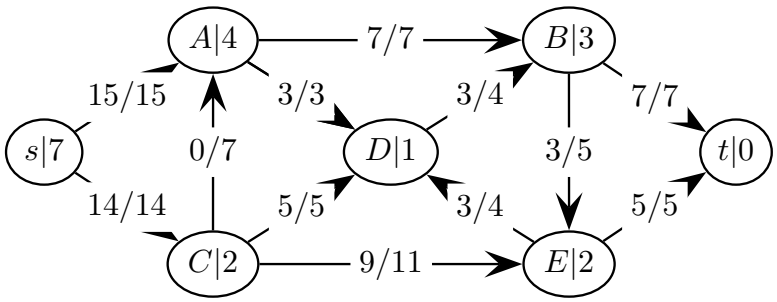
Aplique o método de `Edmonds-Karp` ao grafo. Indique os valores das capacidades residuais dos caminhos de aumento, encontrados durante a execução do algoritmo. Caso a BFS encontre dois caminhos escolha o de maior capacidade.
Indique o corte mínimo e o respectivo valor.

II.6.16 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



Aplique o algoritmo de `Edmonds-Karp` ao grafo. Indique os caminhos de aumento e as respetivas capacidades residuais.

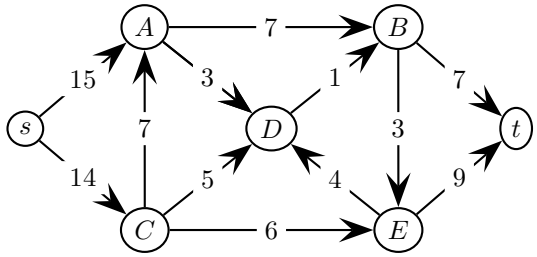
II.6.17 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede. Assuma que a configuração apresentada ocorre durante a execução de um algoritmo de Push-Relabel genérico. Em cada vértice são apresentados, respectivamente, a designação e a altura, separados por barras (|). Em cada arco está indicado o valor do pré-fluxo e a capacidade do arco, separados por uma barra (/).



Indique as operações que são aplicáveis a esta configuração. Indique também um corte com capacidade 16 e o valor do pré-fluxo que atravessa esse corte.

II.6.18 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.

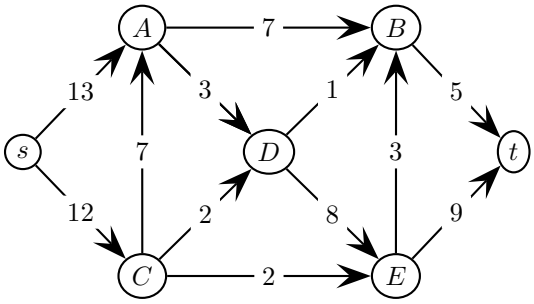
Aplique o algoritmo Relabel-To-Front na rede de fluxo. Considere que a lista de vértices é inicializada por ordem alfabética e que os vizinhos de cada vértice também estão ordenados alfabeticamente.



Indique a altura final de cada vértice. Indique ainda o corte mínimo da rede e o valor do fluxo máximo.

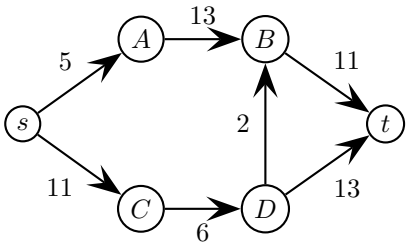
II.6.19 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.

Aplique o algoritmo Relabel-To-Front na rede de fluxo. Considere que a lista de vértices é inicializada por ordem alfabética e que os vizinhos de cada vértice também estão ordenados alfabeticamente.



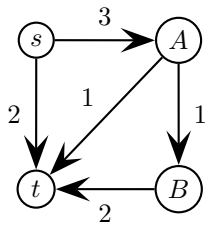
Indique a altura final de cada vértice. Indique ainda o corte mínimo da rede e o valor do fluxo máximo.

II.6.20 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



Aplique o método de `Ford-Fulkerson` ao grafo. Indique o valor do corte mínimo, o respectivo corte e o número minímo e máximo de caminhos de aumento que podem ser calculados pelo algoritmo.

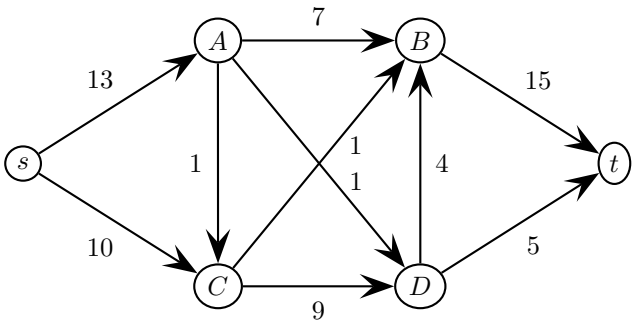
II.6.21 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



Aplique o algoritmo de `Edmonds-Karp`, que determina o valor do fluxo máximo. Indique os caminhos de aumento e respectivas capacidades residuais.

II.6.22 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.

Aplique o algoritmo Relabel-To-Front na rede de fluxo. Considere que a lista de vértices é inicializada por ordem alfabética e que os vizinhos de cada vértice também estão ordenados alfabeticamente. Assim, as listas de vizinhos dos vértices intermédios são as seguintes:
$N[A] =< B, C, D, s >$ $N[B] =< A, C, D, t >$ $N[C] =< A, B, D, s >$
$N[D] =< A, B, C, t >$

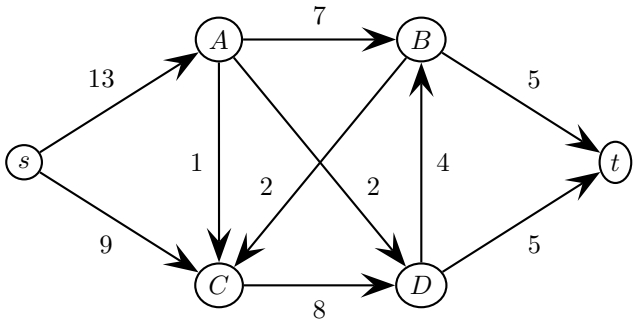


Indique a altura final de cada vértice. Indique ainda o corte mínimo da rede e o valor do fluxo máximo.

II.6.23 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.

Aplique o algoritmo Relabel-To-Front na rede de fluxo. Considere que a lista de vértices é inicializada por ordem alfabética e que os vizinhos de cada vértice também estão ordenados alfabeticamente. Assim, as listas de vizinhos dos vértices intermédios são as seguintes:

$N[A] =< B, C, D, s >$ $N[B] =< A, C, D, t >$ $N[C] =< A, B, D, s >$
$N[D] =< A, B, C, t >$



Indique a altura final de cada vértice. Indique ainda o corte mínimo da rede e o valor do fluxo máximo.

II.6.24 Considere uma rede de fluxo $G = (V, E)$. Suponha que foi calculada uma função de fluxo $f : E \to N$ que define o fluxo máximo na rede de fluxo G.

Considere que é adicionado um novo arco (u, v) à rede de fluxo com capacidade k. Ou seja, temos uma nova rede $G' = (V, E \cup \{(u, v)\})$ onde $u, v \in V$.

Proponha um algoritmo que calcule o fluxo máximo da rede G', actualizando a função de fluxo f calculada anteriormente para G.

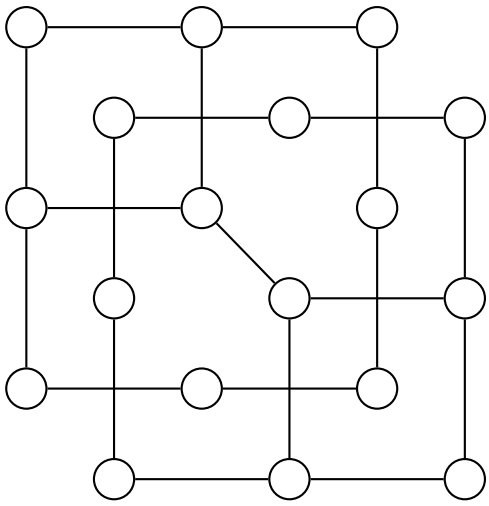
Indique a complexidade do algoritmo proposto.

II.7. Árvores Abrangentes de Menor Custo

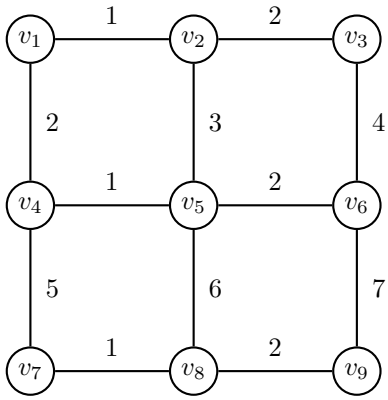
II.7.1 Indique qual das seguintes expressões corresponde ao número máximo de MSTs que é possível obter num grafo com $|V|$ vértices?

a. $|V| - 1$
b. $(|V| - 1)!$
c. $|V|^{|V|-2}$
d. $|V|$
e. $2|V|$
f. $\frac{|V|!}{2}$
g. $2^{|V|-1}$
h. $(|V| - 1)^{|V|-1}$

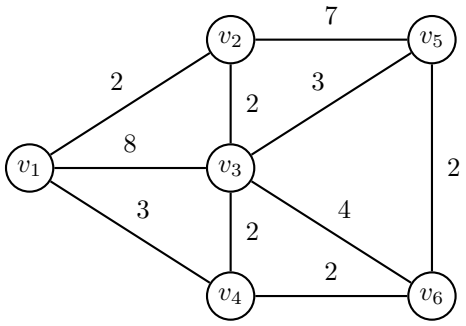
II.7.2 Considere o grafo não dirigido e não planar representado abaixo, onde todos os arcos têm peso igual a 1. Qual o número de MST's do grafo, e qual o seu custo ? Nota: Pode indicar o número de MST's como um produto.



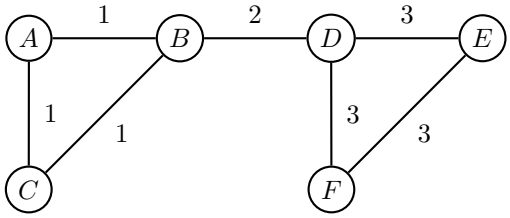
II.7.3 Indique quais os conjuntos disjuntos obtidos na execução do algoritmo de Kruskal para determinar a MST no grafo abaixo após o processamento de 8 arcos.



II.7.4 Indique a ordem de extracção, os predecessores, e a respectiva estimativa dos vértices na execução do algoritmo de Prim no grafo abaixo. Comece com o vértice v_1.

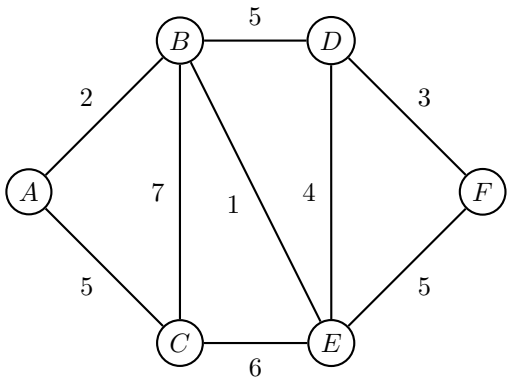


II.7.5 Considere o grafo não orientado e pesado da figura abaixo. Qual é o peso de uma árvore abrangente de menor custo (MST)? Quantas árvores abrangentes de menor custo (MST) existem neste grafo?



II.7.6 Considere o algoritmo de Kruskal para determinar uma MST num grafo ligado, não orientado e pesado $G = (V, E, w)$. Assumindo que a função de peso w toma valores inteiros em $\{0, ..., W\}$, com $W \leq |V|$, indique justificando que alteração faria ao algoritmo para que a complexidade do mesmo seja $O(E\,\alpha(V))$.

II.7.7 Considere o grafo não orientado e pesado da figura.



Aplique o algoritmo de Prim para determinar uma árvore abragente de menor custo (MST), com início em A. Indique a sequência de arcos leves seleccionados pelo algoritmo. Os arcos devem ser listados seguindo a ordem de selecção.

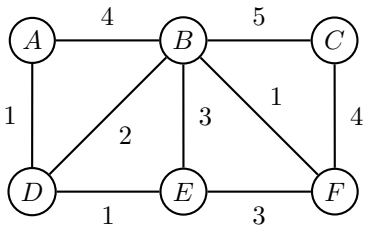
II.7.8 Consider o algoritmo de Prim descrito no pseudo-código abaixo:

```
MST-Prim(G,w,r)
 1  for each u ∈ V[G]                                  ▷ Inicialização
 2        do key[u] = ∞
 3           π[u] = NIL
 4  key[r] = 0
 5  Q = V[G]                                           ▷ Fila de Prioridade
 6  while (Q ≠ ∅)
 7        do u = Extract-Min(Q)
 8           for each v ∈ Adj[u]
 9                 do if (v ∈ Q and w(u,v) < key[v])
10                       then π[v] = u
11                            key[v] = w(u,v)          ▷ Actualização de Q
```

Assuma que é utilizado um Fibonacci Heap, em vez de um Heap binário. Para estas filas de prioridade a operação de `Extract-Min` tem o majorante de tempo $O(\log n)$ e a operação de `Decrease-Key` tem o majorante de tempo $O(1)$. Indique o menor limite assimptótico do algoritmo nesse caso.

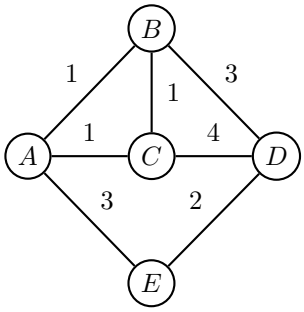
II.7.9 Suponha que um dado grafo $G = (V, E)$ denso é representado com uma matriz de adjacências. Num grafo denso $|E| = O(|V|^2)$.
Quanto tempo requer o algoritmo algoritmo de Prim para construir uma MST ?
Explique como melhorar este algoritmo por forma a que demore apenas $O(|V|^2)$.

II.7.10 Considere o grafo não dirigido e pesado da figura.



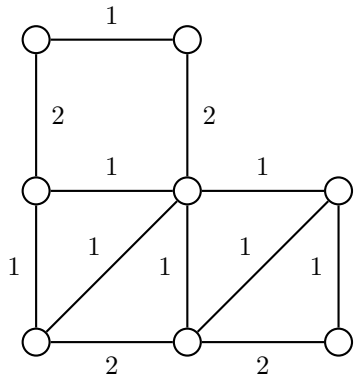
Quantos arcos contém uma árvore abragante de menor custo (MST)?
Qual o custo de uma MST neste grafo ?

II.7.11 Considere o grafo pesado da figura.



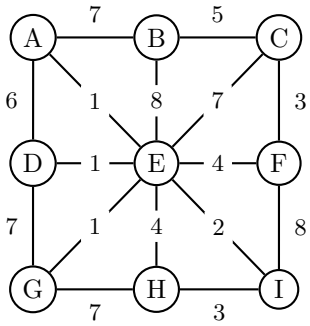
Considere a execução do algoritmo de Kruskal para determinar árvores abrangentes de menor custo, até processar arcos de peso 2, inclusivé. Indique a soma do peso dos arcos seguros seleccionados. Indique também os conjuntos disjuntos existentes nessa fase do algoritmo.

II.7.12 Considere o grafo pesado da figura.



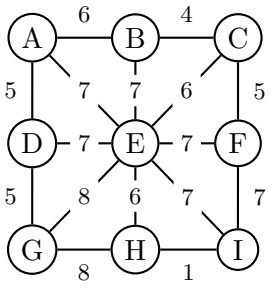
Quantos arcos contém uma árvore abrangente de menor custo no grafo anterior? Qual o custo de uma dessas árvores?

II.7.13 Considere o grafo não dirigido e pesado da figura.



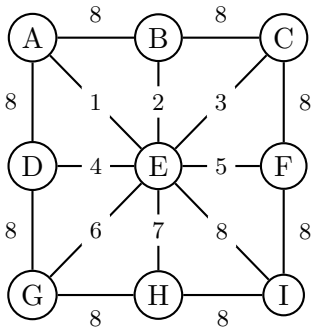
Quantos arcos contém uma árvore abragante de menor custo (MST)?
Qual o custo de uma MST neste grafo ?

II.7.14 Considere o grafo não dirigido e pesado da figura.



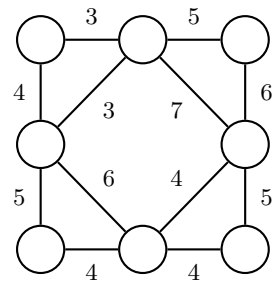
Aplique o algoritmo de Prim ao grafo, considerando o vértice A como origem. Indique a ordem pela qual os vértices são removidos da fila de prioridade durante a execução do algoritmo.
Qual o custo de uma MST neste grafo ?

II.7.15 Considere o grafo não dirigido e pesado da figura.



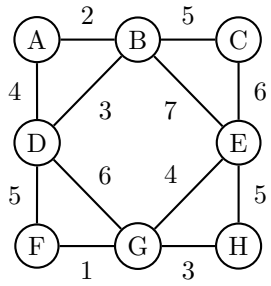
Aplique o algoritmo de Kruskal ao grafo. Indique um conjunto de arcos que definem uma árvore abrangente de menor custo (MST) no grafo.
Qual o custo de uma MST neste grafo ?
Qual o número de diferentes MSTs neste grafo?

II.7.16 Considere o grafo pesado da figura.



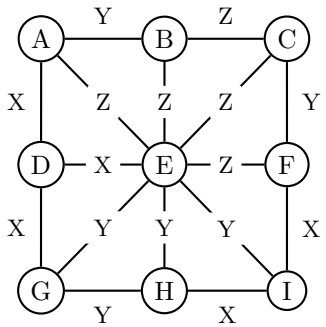
Quantos arcos contém uma árvore abrangente de menor custo no grafo anterior? Qual o custo de uma dessas árvores?

II.7.17 Considere o seguinte grafo não dirigido e pesado:



Indique quais os arcos selecionados pelo algoritmo de Kruskal, que determina árvores abrangentes de menor custo, antes de processar os arcos de peso 5.

II.7.18 Considere o grafo não dirigido e pesado da figura onde os pesos dos arcos são inteiros positivos X, Y e Z tal que $X < Y < Z$.



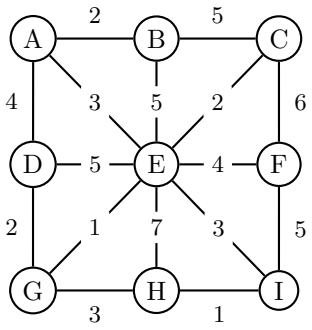
Aplique o algoritmo de Prim ao grafo, considerando o vértice A como origem. Indique a ordem pela qual os vértices são removidos da fila de prioridade durante a execução do algoritmo. Em caso de empate, considere os vértices por ordem lexicográfica.
Indique a expressão que corresponde ao custo de uma MST neste grafo em função de X, Y e Z.

II.7.19 Considere um grafo $G = (V, E)$ não dirigido e pesado e que $T \subseteq E$ define uma árvore abrangente de menor custo (MST) de G.

Supondo que é removido um arco $(u, v) \in T$ do grafo, proponha um algoritmo que actualiza o conjunto de arcos T tal que T é novamente uma árvore abrangente de menor custo.

Indique a complexidade do algoritmo proposto.

II.7.20 Considere o grafo não dirigido e pesado da figura.



Aplique o algoritmo de Kruskal ao grafo.

Indique a ordem pela qual os arcos são seleccionados para pertencer à árvore abrangente de menor custo (MST). Em caso de empate, considere os vértices por ordem lexicográfica.

Indique o custo de uma MST e quantas MST diferentes existem no grafo.

II.8. Resolução de Problemas usando Grafos

II.8.1 Descreva um algoritmo eficiente para identificar um ciclo (quando existente) num grafo dirigido. Analise a complexidade da solução proposta.

II.8.2 Proponha um algoritmo eficiente que, dado um grafo $G = (V, E)$ e dois vértices $u, v \in V$, determine o número de caminhos mais curtos de u para cada um dos outros vértices $v \in V$. Analise a complexidade da solução proposta.

II.8.3 Descreva um algoritmo que permita determinar se um grafo não dirigido é bipartido. Qual é a complexidade desse algoritmo?

II.8.4 Um vértice v num grafo $G = (V, E)$, ligado e não dirigido, diz-se *ponto de articulação* se a remoção de v e a remoção dos arcos incidentes em v resulta num grafo desligado. Proponha um algoritmo eficiente para determinar os pontos de articulação em G. Qual é a complexidade do algoritmo proposto?

II.8.5 Considere um grafo $G = (V, E)$, dirigido e acíclico (DAG), e pesado com função de pesos em $\mathbb{R}$. Nestas condições, proponha um algoritmo eficiente para calcular o peso dos caminhos de maior peso entre todos os vértices de G e um vértice destino t. Descreva justificadamente o algoritmo proposto, argumente a sua correcção, e analise a sua complexidade assimptótica.

II.8.6 Um grafo $G = (V, E)$, dirigido, diz-se *desligado* se existem $u, v \in V$ tal que $u \not\rightsquigarrow v$ e $v \not\rightsquigarrow u$, i.e. não existe caminho de u para v, nem de v para u. Proponha um algoritmo eficiente para decidir se um grafo é desligado.

II.8.7 Numa rede de distribuição europeia existem vários armazéns a_i e várias ligações l_{ij} entre os armazéns, neste caso entre o armazém a_i e o armazém a_j. Neste contexto, dada a cadeia de distribuição, as ligações apenas permitem a transferência de bens no sentido de a_i para a_j. A cada armazém a_i e a cada ligação l_{ij} estão também associadas capacidades $c_i > 0$ e $m_{ij} > 0$, respectivamente, que denotam as capacidades de processamento. Dada a actual situação económica, é importante determinar a quantidade máxima de bens que é possível distribuir através desta rede a partir das cidades portuguesas para as capitais europeias. Proponha um algoritmo que determine a quantidade máxima de bens. Qual a complexidade do algoritmo.

II.8.8 Proponha um algoritmo que, dado um grafo dirigido e não pesado $G = (V, E)$, determine se existem ciclos e, caso não existam, retorne o caminho mais longo em G. Qual é a complexidade da solução proposta? Nota: apenas as soluções eficientes serão consideradas.

II.8.9 Indique como é calculada a matriz Π dos predecessores na execução do algoritmo de Floyd-Warshall.

II.8.10 Considere uma MST T para um grafo não orientado e pesado $G = (V, E, w)$, determinada por um dos algoritmos estudados. Proponha um algoritmo que, dado um qualquer arco $(u, v) \in E$ não pertencente à MST, $(u, v) \notin T$, determine se (u, v) é um arco leve para algum corte e, em caso afirmativo, enumere todos os arcos leves em T para esse corte. Indique a complexidade do algoritmo proposto.

(Nota: Apenas as soluções mais eficientes terão a pontuação total.)

II.8.11 Descreva um algoritmo eficiente que determine se um grafo ligado é k-arco-ligado (k-edge-connected). Um grafo ligado é k-arco-ligado se, após a remoção de $k-1$ arcos arbitrárias, é ainda ligado. Qual é a complexidade desse algoritmo?

(Sugestão: considere uma abordagem baseada em fluxos máximos.)

II.8.12 Caminhos disjuntos. Considere um grafo não dirigido $G = (V, E)$. Um caminho neste grafo é uma sequencia de arcos $e_1, \ldots, e_k$ em que o vértice de destino de e_i coincide com o vértice de destino de e_{i+1}. Dado um par de vértices v e v' queremos determinar o número máximo de caminhos disjuntos entre v e v'. Dois caminhos são considerados disjuntos quando não partilham nenhum arco, embora possam partilhar vértices. Em particular vão partilhar o v e o v', mas também podem partilhar outros vértices.

Descreva um algoritmo que, dados v e v' calcula o número máximo de caminhos disjuntos entre v e v', indique e justifique a respectiva complexidade assimptótica.

Apenas será atribuída a cotação máxima a soluções com a menor complexidade assimptótica.

II.8.13 Nem sempre fazer o câmbio de moedas é uma tarefa simples, pode acontecer que o câmbio directo não seja o mais vantajoso. Suponhamos por exemplo que 1 Euro pode ser trocado por 1,28 dolars, mas que também pode ser trocado por 0,85 libras e que uma libra pode ser trocada por 1,52 dólares. Neste caso trocando os euros primeiro em libras e depois em dólares obtemos $0,85 \times 1,48 = 1,292$ dólares.

Note que o facto de podermos trocar 1 Euro por 1,28 dólares não significa que podemos trocar 1,28 dólares de volta por 1 Euro. Há perda de valor nestas trocas, pelo que 1,28 dólares podem ser trocado por menos de 1 Euro.
Modele este problema utilizando as estruturas estudadas. Descreva um algoritmo que permite determinar um sequência de trocas óptima entre duas moedas.
Pode acontecer que o valor da moeda se altere. Por exemplo se o euro valorizar 1% a nova taxa de conversão para dolar seria $1,28 \times 1,01 = 1,2928$ e para libras seria $0,85 \times 1,01 \approx 0.8584999$. Note que a mesma valorização foi aplicada em ambas a conversões.
Suponhamos finalmente que o dolar valoriza 2%, e o euro 1%, nesse caso a nova taxa de conversão seria $(0,85/1,02) \times 1,01 \approx 0.8416666$.
O que acontece a uma sequência de câmbios óptima quando há valorização de moedas?

II.8.14 Seja $G = (V, E)$ um grafo dirigido e $L : V \to \{1, 2, \ldots, |V|\}$ uma função que etiqueta os nós com números de 1 a $|V|$. Para um dado vértice u seja $R(u)$ o nó com a menor etiqueta que é atingivel a partir de u, i.e., $R(u) = v$ se v é atingivel a partir de u e $L(v)$ é o valor mínimo, de entre as etiquetas que é possível atingir a partir de u, $L(v) = \min\{L(v') \mid u \rightsquigarrow v'\}$.

Descreva um algoritmo que calcula $R(u)$ para todos os vertices u de G e indique e justifique a respectiva complexidade assimptótica.

Apenas será atribuída a cotação máxima a soluções com a menor complexidade assimptótica.

II.8.15 O peso máximo de uma árvore abragente T, de um grafo não orientado e pesado $G = (V, E, w)$, é o maior peso de entre os arcos de T, ou seja $\max(T) = \max\{w(e)|e \in T\}$. Uma árvore abrangente é minimax, se o seu máximo é o menor máximo de entre todas as árvores abrangentes. Formalmente, uma árvore abragente T é minimax se qualquer seja a árvore abrangente T' temos $\max(T) \leq \max(T')$.
Prove, ou mostre um contra-exemplo, que as árvores abragentes de menor custo (MSTs) são árvores minimax.
Dado um grafo G e um valor m descreva um algoritmo eficiente para determinar se G contém uma árvore abragente T' com peso máximo menor ou igual a m, i.e., $\max(T') \leq m$.

II.8.16 Considere o problema de determinar o fluxo máximo de uma rede. Para além das capacidades associadas aos arcos da rede assuma que os nós também são limitados por um valor de capacidade, que é um número real, dado pela função $c' : V \to R$. Neste caso o fluxo que passa através de um nó v não pode ser superior a $c'(v)$.
Descreva um algoritmo eficiente para calcular o fluxo máximo que pode passar pela rede.

II.8.17 Considere que vamos utilizar um computador para analizar o crescimento de bactérias numa caixa de Petri. O estado da caixa é monitorizado regularmente. Em cada instante uma imagem da caixa é digitalizada e processada. As imagens processadas consistem em grelhas de pixeis. A cor de cada pixel indica o tipo de bactéria presente numa determinada região da caixa. Os pixeis transparentes indicam a ausência de bactérias. Pixeis adjacentes, da mesma cor, formam manchas. De uma imagem para a seguinte os pixeis transparentes podem ganhar uma cor, mas um pixel que tenha cor nunca muda de cor. Em cada instante queremos determinar o tamanho da maior mancha existente na imagem.

Este problema pode ser modelado com as seguintes operações:

- CriaImagem(), cria uma grelha de pixeis, com todos os pixeis transparentes.
- PintaPixel(p, c), atribui uma cor a um pixel. Onde p é a posição do pixel e c a nova cor.
- Maior(), devolve o tamanho da maior mancha.

Proponha uma implementação eficiente para as mesmas e indique a respectiva complexidade temporal e espacial.

II.8.18 Considere um parque de diversões, no qual um visitante pode utilizar rodas gigantes, carrosséis, montanhas russas, etc. Em cada local do parque existem várias atrações acessíveis. Logo, cada visitante pode escolher apenas as atracções acessíveis no local onde se encontra. Para além disso, após usar uma atracção, o visitante sai para outro local do parque onde pode seleccionar outra diversão.

Para aceder a uma determinada atracção é preciso entrar na respectiva fila de espera. Cada fila tem uma lotação máxima. Caso uma fila esteja esgotada o visitante tem de escolher outra. Os visitantes podem utilizar as atracções que desejarem, desde que não passem duas vezes pelo mesmo local. Em particular, isto impossibilita usar duas vezes a mesma atracção.

Assuma que o parque funciona por turnos. Todas as atracções demoram 15 minutos e estão todas sincronizadas, i.e., começam todas ao mesmo tempo e terminam todas ao mesmo tempo. Quando uma diversão começa, todas as pessoas da respectiva fila entram.

Utilizando apenas o critério de escolha de filas acima pode acontecer que num determinado local um visitante não possa escolher nenhuma fila porque estão todas cheias.

1) Modele este problema, utilizando estruturas que conhece.
2) Proponha um algoritmo que determina o número máximo de pessoas que podem entrar no parque por forma a garantir, que caso todos escolham um caminho acertado, não haverá locais onde um visitante tenha que ficar à espera porque todas as filas estão cheias.
3) Utilizando o algoritmo anterior descreva como reduzir o tamanho das filas por forma a garantir que todos os visitante irão escolher um caminho acertado.

II.8.19 Neste problema queremos optimizar a eficiência de uma corporação de bombeiros. Numa determinada região há um conjunto de quartéis de bombeiros. A região é representada por um conjunto de locais, nos quais podem existir quartéis, florestas ou serem apenas locais de passagem. O objectivo é fazer chegar um carro de bombeiros até uma floresta quando há um incêndio na mesma.

O carro de bombeiros pode mover-se entre alguns pares de regiões e em ambos os sentidos. O tempo que demora é o mesmo em qualquer um dos os sentidos.

Este problema consiste em determinar, para cada floresta, qual é o quartel de bombeiros que está mais próximo, i.e., aquele que mais rapidamente consegue fazer chegar um carro à floresta.

1) Modele este problema, utilizando estruturas que conhece.

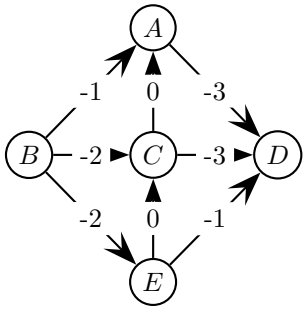
2) Proponha um algoritmo que resolva este problema, simultaneamente para todas as florestas, da forma mais eficiente possível. Indique a respectiva complexidade temporal.

II.8.20 Considere um grafo não dirigido dinâmico. O problema consiste em manter informação sobre o numero de arcos de um vértice. O objectivo é encontrar uma estrutura de dados que mantenha esta informação de forma eficiente. Em cada instante podemos fazer qualquer umas das seguintes operações:

- `Inicializa(V)`, inicializa a estrutura de dados, que deve representar um grafo com `V` vértices. Quando o grafo é inicializado não existem arcos, i.e., existe um componente ligado por cada vértice.
- `Popular(v)`, considera todos os vértices do componente ligado a que `v` pertence e devolve o vértice que tem o maior número de vizinhos, ou seja o vértice que tem mais arcos.
- `Ligar(u,v)`, criar um arco que liga os vértices `u` e `v`.

Note que os arcos nunca são removidos, apenas acrescentados. Assuma que em qualquer sequência de operações uma operação de `Ligar(u,v)` só é invocada quando o arco (`u`, `v`) ainda não faz parte do grafo, pelo que não é preciso verificar casos de duplicação.

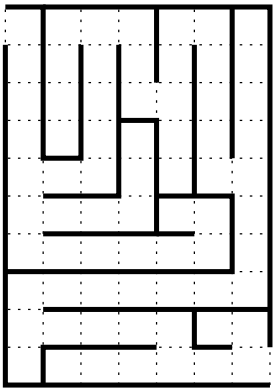
II.8.21 Considere o seguinte grafo dirigido acíclico e pesado:



Neste exemplo o valor do caminho mais curto de B para D é -5 e existem 4 alternativas para esse caminho. Proponha um algoritmo eficiente, que dado um grafo acíclico pesado e um par de vértices indique quantas alternativas existem para o caminho mais curto. Indique a complexidade do algoritmo proposto.

II.8.22 Considere o seguinte labirinto. O objetivo é entrar pela canto superior esquerdo e encontrar um percurso que saia pelo canto inferior direito. Esse percurso não deve cruzar as linhas pretas, que representam as paredes. Existem também linhas tracejadas que realçam a estrutura em grelha do problema.

Neste problema queremos encontrar o menor percurso que atravessa, com sucesso, o labirinto. O tamanho do percurso consiste no numero de quadrados da grelha que são atravessados.



a. Modele este problema, utilizando estruturas que conhece. Note que o respetivo modelo irá servir apenas para encontrar o percurso e não para ser uma representação das propriedades geométricas do labirinto. Indique a menor complexidade espacial desta representação, com uma expressão simplificada.

b. Proponha um algoritmo que encontre o menor percurso. Indique a menor complexidade temporal deste algoritmo, com uma expressão simplificada.

II.8.23 Um grafo dirigido é ligado por arcos quando entre qualquer par de vértices existe um caminho que os liga. Um grafo é dois ligado por arcos quando mesmo que lhe seja retirado 1 arco, continua ligado. Em geral um grafo é k ligado por arcos quando mesmo que lhe sejam retirados $k-1$ arcos continua ligado.

Proponha um algoritmo eficiente que, dado um grafo, determina o maior valor de k para o qual o grafo é k ligado por arcos.

II.8.24 O Sr. João Caracol encarregou um dos seus directores de logística de propor um sistema de transporte interno entre os diversos edifícios da empresa. Note que cada ligação entre edifícios apenas funciona num dos sentidos porque o transporte pode continuar para outro edifício.

Seja $E = \{e_1, e_2 \dots e_n\}$ o conjunto de n edifícios ocupados pela empresa. O Sr. João Caracol determinou que entre cada par de edifícios (e_i, e_j) tem que existir um percurso interno de e_i para e_j **ou** então existir um percurso de e_j para e_i. Note que basta haver o percurso num dos sentidos. Os percursos não precisam ser directos. Podem passar por outros edifícios.

O director tem uma proposta para o Sr. João Caracol validar que consiste num conjunto L de ligações directas entre edifícios. Cada ligação é um par $(e_a, e_b) \in L$ que determina a existência de um transporte do edifício e_a para o edifício e_b.

a. Dado um conjunto de edifícios E e um conjunto de ligações L, modele este problema utilizando estruturas que conhece.

b. Proponha um algoritmo eficiente que determina se o conjunto de ligações L satisfaz os requisitos do Sr. João Caracol.

c. Indique a complexidade da solução proposta em função do número de edifícios e do número de ligações.

II.8.25 O Sr. João Caracol tem uma empresa que ocupa áreas enormes em cada um dos seus polos. O Sr. João definiu uma hora de almoço comum para todas as pessoas e preparou várias cantinas onde as pessoas podem almoçar. No entanto, existem dois problemas: (1) as cantinas têm uma capacidade limitada e (2) cada empregado de um edifício apenas se consegue deslocar em tempo útil a um subconjunto de cantinas, dependendo do edifício em que trabalha. Pode considerar que as cantinas estão em anexos diferentes dos edifícios.

Seja $E = \{e_1, e_2 \dots e_n\}$ o conjunto de n edifícios ocupados pela empresa e sejam $C = \{c_1, c_2 \dots c_m\}$ o conjunto de m cantinas onde as pessoas podem almoçar. Sejam ainda $num(e_i)$ o número de pessoas no edifício e_i e $cap(c_j)$ a capacidade da cantina c_j. Finalmente, para cada edifício, as cantinas vizinhas onde as pessoas se podem deslocar é dado por $V(e_i)$. Ou seja, $V(e_i) \subseteq C$ denota o conjunto de cantinas onde as pessoas do edifício e_i podem almoçar.

a. Dado um conjunto de edifícios E onde cada edifício e_i tem $num(e_i)$ empregados que podem almoçar em qualquer cantina do conjunto $V(e_i)$, e um conjunto de cantinas C onde cada cantina c_j tem capacidade $cap(c_j)$, defina um algoritmo eficiente que determine quantos empregados de cada edifício podem ir a cada uma das cantinas.

b. Indique a complexidade da solução proposta.

II.8.26 O Sr. João Caracol encarregou um dos seus directores de logística de propor um sistema de transporte interno entre os diversos edifícios da empresa. Note que cada ligação entre edifícios apenas funciona num dos sentidos porque o transporte pode continuar para outro edifício.

Seja $E = \{e_1, e_2 \dots e_n\}$ o conjunto de n edifícios ocupados pela empresa. O Sr. João Caracol determinou que entre qualquer par de edifícios (e_i, e_j) tem que existir um percurso interno de e_i para e_j **e** existir um percurso de e_j para e_i. Os percursos não precisam ser directos. Podem passar por outros edifícios. No máximo, cada edifício apenas tem ligação directa a outros 2 edifícios.

Cada ligação tem associado um tempo de deslocação, mas o tempo mínimo do percurso entre qualquer par de edifícios não pode ser superior ao valor K.

O director tem uma proposta para o Sr. João Caracol validar que consiste num conjunto L de ligações directas entre edifícios. Cada ligação é um par $(e_a, e_b) \in L$ que determina a existência de um transporte do edifício e_a para o edifício e_b. Associada a cada ligação (e_a, e_b) há um tempo de deslocação $t_{ab} > 0$.

a. Dado um conjunto de edifícios E e um conjunto de ligações L, modele este problema utilizando estruturas que conhece.

b. Proponha um algoritmo eficiente que determina se o conjunto de ligações L satisfaz os requisitos do Sr. João Caracol.

c. Indique a complexidade da solução proposta em função do número de edifícios e do número de ligações.

II.8.27 Uma das fábricas do Sr. João Caracol está dividida em secções especializadas. Apenas os trabalhadores com determinadas competências é que podem operar em cada secção. Por outro lado, os trabalhadores estão divididos em categorias. Cada categoria está associada a um conjunto de competências por forma a poder distribuir os trabalhadores por cada secção.

Seja $S = \{s_1, s_2 \dots s_n\}$ o conjunto de n secções e seja $C = \{c_1, c_2 \dots c_m\}$ o conjunto de m categorias de trabalhadores. Seja ainda $num(c_j)$ o número de trabalhadores na categoria c_i. Note que um trabalhador apenas pertence a uma categoria. Finalmente, o conjunto $T(s_i)$ define o conjunto de categorias dos trabalhadores que podem trabalhar na secção s_i. Por exemplo, se $T(s_1) = \{c_1, c_3, c_4\}$, então qualquer trabalhador dessas categorias pode trabalhar na secção s_1.

Para cada secção operar, precisa de um número específico de trabalhadores. Seja $trab(s_i)$ o número de trabalhadores para a secção s_i operar. Defina um algoritmo que determina a distribuição dos trabalhadores por forma a cada secção poder operar.

a. Utilizando estruturas que conhece, modele o problema de distribuição dos trabalhadores pelas secções tal como descrito acima.

b. Proponha um algoritmo eficiente que distribua os trabalhadores pelas secções por forma a que estas possam operar.

c. Indique a complexidade da solução proposta.

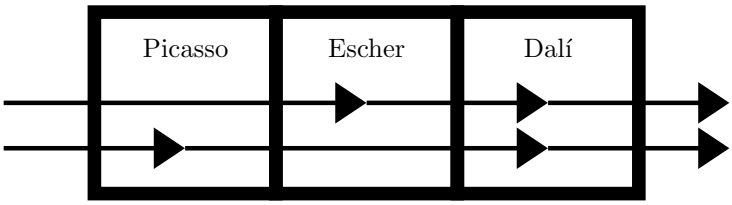
II.8.28 Considere o problema da organização de uma exposição de pintura num museu. O museu consiste numa sequência de salas, todas umas a seguir às outras, em linha. Se as numerarmos a partir da primeira sala podemos avançar da sala 1 para a sala 2, desta para a sala 3, e assim sucessivamente. Suponha que cada sala tem quadros de um único pintor.

Esta arquitectura não é a ideal para organizar uma exposição com múltiplos percursos temáticos, uma vez que o visitante tem de passar pelas salas todas, da primeira à última. Um percursos temático consiste numa sequência ordenada de pintores.

Considere a organização das salas conforme a figura onde temos Pablo Picasso (sala 1), M. C. Escher (sala 2) e Salvador Dalí (sala 3). Os visitantes que pretendem ver as obras de M. C. Escher seguidas das de Salvador Dalí, passam pela sala com as obras de Pablo Picasso sem as observar. Outro percurso temático aceitável para esta organização do museu seria ver as obras de Pablo Picasso e de seguida as obras de Salvador Dalí.

Note que cada percurso temático tem uma ordem. O percurso Pablo Picasso seguido de Salvador Dalí é considerado diferente do percurso que tem Salvador Dalí seguido de Pablo Picasso. O percurso Salvador Dalí seguido de Pablo Picasso não seria possível na organização das salas da figura.

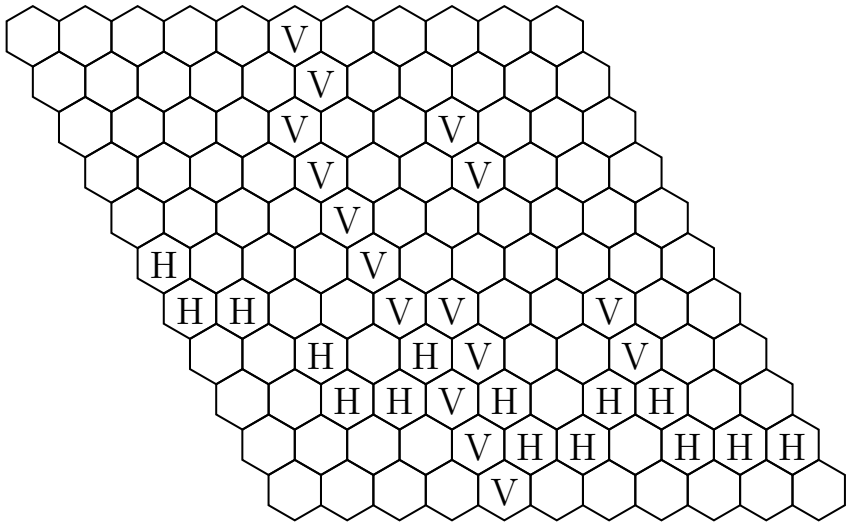
Dado um conjunto de percursos temáticos, o objectivo é encontrar uma organização para as salas do museu que permita a realização dos vários percursos temáticos, *sem que o visitante tenha de voltar para trás.*



a. Modele este problema, utilizando estruturas que conhece.

b. Proponha um algoritmo que resolva este problema, i.e., dado um conjunto de percursos temáticos, indica se existe uma organização possível que permite a realização dos percursos. Caso exista, o algoritmo devolve a organização das salas do museu. Indique a respectiva complexidade temporal da solução proposta.

II.8.29 Neste problema queremos encontrar uma estrutura de dados para implementar o jogo Hex.



Este jogo utiliza um tabuleiro constituído por 11×11 hexágonos. Os dois jogadores, o vertical V e o horizontal H, jogam alternadamente. Em cada jogada colocam uma peça num qualquer hexágono, que esteja vazio, i.e., que ainda não contenha uma peça. Na figura, as casas que têm peças estão identificadas com a letra do respectivo jogador. As peças nunca podem ser retiradas do tabuleiro, ou movidas para outra casa.

Duas peças, do mesmo jogador, em casas adjacentes consideram-se ligadas. Um jogador ganha quando consegue criar um conjunto de peças ligadas que estabeleçam um caminho entre lados opostos do tabuleiro: lado de cima/baixo para o jogador vertical (V), e lado esquerdo/direito para o jogador horizontal (H). Na figura o jogador vertical (V) ganhou, porque criou um caminho entre o lado de cima do tabuleiro e o lado de baixo. Para o jogador horizontal ganhar teria de establecer um caminho entre o lado esquerdo e o lado direito, o que já não é possivel devido ao caminho criado pelo jogador V.

a. Modele este problema, utilizando estruturas que conhece.

b. Proponha um algoritmo que, após uma jogada, permita determinar eficientemente se um jogador já ganhou o jogo. Indique a complexidade da solução assumindo que a dimensão de um tabuleiro genérico é n por n.

II.8.30 O funcionamento de uma comunidade depende da sua coerência interna. Esta coerência pode existir ou não existir, dependendo da forma como a informação circula na comunidade. Quando existe uma afinidade entre elementos da comunidade, um deles comunica ao outro toda a informação que recebe. Esta interação não tem que ser reciproca. Por exemplo, se o João comunica ao Pedro toda a informação que recebe, o Pedro não tem que necessariamente fazer o mesmo com o João.

Define-se que há coerência na comunidade quando a informação recebida por um qualquer elemento da comunidade é transmitida (directa ou indirectamente) a todos os outros.

Considere que tem acesso às afinidades entre elementos de uma comunidade que determinam como a informação é transmitida entre os seus elementos. O seu objectivo é determinar se essa comunidade é coerente.

a. Modele este problema, utilizando estruturas que conhece.

b. Proponha um algoritmo que determine se uma dada comunidade é coerente, i.e., o algoritmo indica se existem (ou não) elementos que não conseguem fazer chegar a informação que recebem a todos os outros.

II.8.31 A empresa Coelho e Caracol Lda faz transporte de mercadorias entre localidades. Uma rota consiste numa sequência de localidades. O transporte de mercadorias gera receita, enquanto que a deslocação tem um custo, combustível, portagens, etc.

O Sr. Caracol simplifica esta complexidade associando a cada par de localidades um valor de perda, que resulta de subtrair a receita ao custo. Portanto, se a receita for maior que o custo, o valor será negativo. Se o valor for positivo significa que o custo supera a receita. Naturalmente, o Sr. Caracol prefere ter valores de perda negativos. Entre cada par de localidades há no máximo um valor de perda.

Considere que existe um conjunto S de localidades onde pode ser estabelecida a sede da empresa. Todas a rotas têm inicio na sede.

Determine qual a melhor localização para a sede, escolhendo a localidade a partir da qual se consegue obter a rota com menor valor de perda. Note que esta rota pode ou não incluir todas as localidades. Caso exista mais do que uma localização com um mesmo valor de perda mínimo qualquer uma delas é uma resposta válida.

Assuma que não exite nenhuma localidade a partir da qual seja possivel percorrer um caminho, e regressar á mesma localidade, com um valor de perda negativo.

a. Modele este problema, utilizando estruturas que conhece.

b. Proponha um algoritmo que determine qual a localidade de S a escolher como sede. Indique a complexidade da solução. Apenas a solução mais eficiente terá cotação máxima.

II.8.32 Suponha a situação em que há duas concentrações de manifestantes: uma a favor da saída do Reino Unido da União Europeia e outra contra. Após a concentração, os manifestantes irão percorrer algumas ruas previamente definidas de forma a que os percursos não se cruzam.

No entanto, o Sr. Caracol, comandante da polícia, quer certificar-se que não existem desvios nos percursos já definidos tal que os manifestantes das duas concentrações se encontrem numa mesma rua.

O Sr. Caracol tem um mapa da cidade sob a forma de um grafo $G = (V, E)$ não dirigido e pesado. Cada arco (u, v) do grafo representa uma rua e o peso $w(u, v)$ o número de polícias necessários para impedir os manifestantes de usarem a rua (u, v).

Considere que $p_1 =< u_1, u_2 \dots u_k >$ define o percurso de uma das manifestações e que $p_2 =< v_1, v_2 \dots v_m >$ define o percurso da outra manifestação. O Sr. Caracol pretende colocar polícias a bloquear algumas ruas de forma a que se os manifestantes se desviarem dos percursos definidos, então não se irão encontrar com os manifestantes opostos.

Defina um algoritmo que ajude o Sr. Caracol a definir quais as ruas a bloquear tal que o número de polícias necessário para esta operação seja mínimo. Indique a complexidade da solução proposta.

II.8.33 Considere um grafo $G = (V, E)$ dirigido e sem pesos. Um vértice $u \in V$ é denominado como vértice raíz se para todos os outros vértices $v \in V$ existe um caminho de u para v. Ou seja, todos os vértices do grafo são atingíveis a partir de um vértice raíz.

Defina um algoritmo eficiente que permite identificar **todos** os vértices raíz de um grafo G. Note que o grafo G pode não ter um vértice raíz, pelo que o seu algoritmo deve identificar essa situação.

Indique a complexidade da solução proposta.

Parte III.
Síntese de Algoritmos & Tópicos Adicionais

III.1. Algoritmos Greedy

III.1.1 Considere que tem uma mochila com capacidade 22 e o seguinte conjunto de objectos $\{(6, 32), (8, 64), (12, 56), (4, 32), (10, 48), (8, 40)\}$ onde cada par de valores (w_i, v_i) denota o peso e valor do objecto i. Indique o valor óptimo da mochila se pudesse transportar fracções de objectos.

III.1.2 Uma associação humanitária recebeu um carregamento de contentores com bens de durabilidade limitada, para distribuir junto de uma comunidade necessitada. Cada contentor contém bens com um prazo de entrega comum e tem um valor associado. Contudo, a comunidade em questão fica a um dia de viagem (ida e volta) e só é possível transportar um contentor de cada vez. Por cada contentor não entregue dentro do seu prazo, perde-se o seu valor total. O objectivo é determinar uma sequência óptima de distribuição de contentores, isto é, uma sequência que minimize a soma dos custos dos contentores cuja entrega ocorra após o prazo dos bens contidos.

Este problema é uma concretização do problema de escalonamento de tarefas de tempo constante. Formalmente, uma instância do problema é descrita por um conjunto $T = \{t_1, \ldots, t_n\}$ de $n \in IN$ tarefas identificadas por $t_i \in IN$, com $1 \leq i \leq n$, uma sequência de n prazos $d_i \in IN$, com $1 \leq d_i \leq n$ e tal que d_i é o prazo da tarefa t_i, e uma sequência de n penalizações $p_i \in IN$, tal que p_i é a penalização associada à tarefa t_i, caso esta não seja concluída dentro do prazo d_i. Dado um conjunto de tarefas T, um escalonamento S é uma permutação do conjunto T. O objectivo deste problema é a obtenção de escalonamentos que minimizem a soma das penalizações das tarefas *em atraso*, i.e., das tarefas cuja conclusão ocorra após o seu prazo.

Proponha um algoritmo eficiente para este problema. Analise a sua complexidade.

III.1.3 Considere o problema de compressão de dados de um ficheiro usando a codificação de Huffman. Para cada caracter, indique o código livre de prefixo óptimo para um ficheiro com as seguintes frequências: $f(a) = 6$, $f(b) = 10$, $f(c) = 11$, $f(d) = 5$, $f(e) = 8$, $f(f) = 45$, $f(g) = 7$, $f(h) = 8$.

III.1.4 Considere um conjunto de n tarefas onde cada tarefa i requer exactamente uma hora para ser realizada e garante um retorno de v_i euros, com $v_i > 0$, se começada antes ou no instante de tempo t_i. Se uma tarefa for escalonada para começar após o instante t_i, não existe qualquer interesse em executá-la uma vez que o retorno será 0. Proponha um algoritmo para determinar o escalonamento de tarefas que permite maximizar o lucro. Qual a complexidade do algoritmo.

III.1.5 Considere o seguinte conjunto de caracteres e as respectivas frequências.

`a`:10 `b`:13 `c`:8 `d`:6 `e`:3 `f`:21 `g`:39

Atendendo às frequências indicadas, utilize o algoritmo de Huffman para determinar a codificação de Huffman, indicando para cada caracter a palavra binária resultante. Quando constrói a árvore, considere o bit a 0 no ramo para o nó com menor frequência.

III.1.6 Considere o problema de compressão de ficheiros usando a codificação de Huffman. Dado um ficheiro com 5000 caracteres com as seguintes frequências relativas, indique o tamanho (em bits) do ficheiro comprimido.

	a	b	c	d	e	f	g
Frequência	7	11	14	3	26	19	20

III.1.7 Considere o seguinte conjunto de caracteres e as respectivas frequências.

`a`:8 `b`:10 `c`:7 `d`:11 `e`:45 `f`:14 `g`:5

Atendendo às frequências indicadas, utilize o algoritmo de Huffman para determinar a codificação de Huffman, indicando para cada caracter a palavra binária resultante. Ao construir a árvore, considere o bit a 0 no ramo para o nó com menor frequência.

III.1.8 Considere o problema de compressão de dados de um ficheiro usando a codificação de Huffman. Indique o código livre de prefixo óptimo para cada caracter num ficheiro com as seguintes ocorrências: $f(a) = 20, f(b) = 5, f(c) = 3, f(d) = 50, f(e) = 1, f(f) = 80, f(g) = 10$. Quando constrói a árvore, considere o bit 0 para o nó com menor frequência.

III.1.9 Considere o problema de compressão de dados de um ficheiro usando a codificação de Huffman. Indique o número de bits para cada caracter do código livre de prefixo óptimo para um ficheiro com as seguintes ocorrências: $f(a) = 10, f(b) = 20, f(c) = 15, f(d) = 45, f(e) = 30, f(f) = 40, f(g) = 1$.

III.1.10 Considere que trabalha numa loja para pequenas reparações de electrodomésticos e a sua tarefa é fazer as reparações.

Nos últimos minutos entraram n pessoas com material para reparar. Para cada electrodoméstico i, o seu colega mais experiente estimou um tempo t_i para a sua reparação. Ou seja, você sabe quanto tempo deve demorar cada reparação.

Como a sua loja fica longe, todos os clientes resolveram esperar até que o respectivo material esteja reparado. Indique um algoritmo para estabelecer uma ordem dos electrodomésticos a reparar por forma a minimizar a soma do tempo de espera dos clientes todos.

Note que enquanto repara um electrodoméstico, todos os clientes cujo material ainda não foi reparado estão a esperar. O objectivo é minimizar o tempo total de espera.

III.1.11 Considere que é caixa num hipermercado. Quando os clientes pagam em numerário, tem que usar notas e moedas para perfazer o troco a dar ao cliente.

Suponha que tem um conjunto de notas e moedas denominadas $1 \ldots n$ cujos valores são $d_1 \ldots d_n$ tal que $d_1 = 1$ e $d_i \geq 2 * d_{i-1}$ onde $2 \leq i \leq n$.

Considere que tem acesso a um número infinito de cada uma das n denominações. Indique um algoritmo greedy que use as n denominações para perfazer um troco de K por forma a que o número de notas e moedas usadas no troco seja mínimo.

Se a condição $d_i \geq 2 * d_{i-1}$ onde $2 \leq i \leq n$ não se verificar, o seu algoritmo ainda produz o número mínimo de moedas? Justifique ou indique um contraexemplo.

III.1.12 Considere o seguinte conjunto de actividades, onde são indicados o tempo de inicio t_i e tempo de fim f_i. Indique quantas e quais são as actividades seleccionadas pelo algoritmo greedy que maximiza o número de actividades executadas.

i	1	2	3	4	5	6	7	8	9
t_i	4	2	7	6	11	9	17	14	20
f_i	5	6	10	11	14	11	25	16	23

III.1.13 Considere o problema de compressão de dados de um ficheiro usando a codificação de Huffman para um ficheiro com as seguintes as ocorrências indicadas na seguinte tabela.

	a	b	c	d	e
f	21	20	19	18	22

Indique para cada caracter a palavra binária resultante. Quando constroi a árvore considere o bit 0 associado ao ramo com menor frequência.

III.1.14 Considere o problema do servidor com os seguintes 7 clientes. O objectivo é minimizar o tempo dispendido pelos clientes no sistema. Cada cliente necessita de um tempo de serviço t_i, indicado na seguinte tabela:

i	1	2	3	4	5	6	7
t_i	5	2	4	3	1	7	6

Indique a ordem pela qual devem ser servidos os clientes, por forma a minimizar o tempo total dispendido no sistema. Indique também o tempo dispendido por cada cliente. Note que enquanto processa um cliente, todos os clientes cuja tarefa ainda não foi processada estão a esperar, i.e. a dispender tempo no sistema.

III.1.15 Considere que tem duas caixas e n objectos onde cada objecto i ($1 \leq i \leq n$) tem peso p_i e valor v_i. Considere ainda que cada caixa tem capacidade para suportar até K unidades de peso, sendo que o somatório do peso dos n objectos é superior a $2K$. Ou seja, $\sum_{i=1}^{n} p_i > 2K$.

Supondo que consegue seleccionar qualquer fracção dos n objectos, elabore um algoritmo que permita seleccionar os objectos a colocar em cada uma das duas caixas tal que o valor colocado em cada uma das caixas seja maximizado e o peso dos objectos seja o mesmo em ambas as caixas.

Indique a complexidade do algoritmo proposto.

III.1.16 Considere o problema de compressão de dados de um ficheiro usando a codificação de Huffman. Indique o código livre de prefixo óptimo para cada caractere num ficheiro com 10000 caracteres com as seguintes frequências de ocorrência: $f(a) = 1, f(b) = 15, f(c) = 8, f(d) = 20, f(e) = 2, f(f) = 24, f(g) = 30$. Quando constrói a árvore, considere o bit 1 para o nó com menor frequência.

Indique o total de bits no ficheiro codificado.

III.1.17 Considere o problema de compressão de dados de um ficheiro usando a codificação de Huffman. Indique o código livre de prefixo ótimo para cada caractere num ficheiro com 100 caracteres com o seguinte número de ocorrências: $f(a) = 12, f(b) = 11, f(c) = 16, f(d) = 18, f(e) = 21, f(f) = 22$. Quando constrói a árvore, considere o bit 0 para o nó com menor frequência.

Indique também o total de bits no ficheiro codificado.

III.1.18 Considere o problema de compressão de dados de um ficheiro usando a codificação de Huffman. Indique o código livre de prefixo ótimo para cada caractere num ficheiro com 100 caracteres com o seguinte número de ocorrências: $f(a) = 1, f(b) = 2, f(c) = 5, f(d) = 13, f(e) = 34, f(f) = 45$. Quando constrói a árvore, considere o bit 0 para o nó com menor frequência.

Indique também o total de bits no ficheiro codificado.

III.1.19 Considere o seguinte conjunto de clientes que tem tarefas para ser processadas por um servidor. Cada tarefa de demora s_i unidades de tempo a processar. O objetivo é minimizar o tempo total de espera dos clientes, indique a ordem porque devem ser processadas as tarefas para atingir este objectivo.

i	1	2	3	4	5	6	7	8
s_i	10	11	5	9	7	15	13	12

III.2. Programação Dinâmica

III.2.1 Dada uma sequência de n números reais $a_1, \ldots, a_n$, proponha um algoritmo que determine uma subsequência (não necessariamente contígua) de tamanho máximo tal que os valores da mesma formam uma sequência estritamente crescente. Qual é a complexidade desse algoritmo?

III.2.2

Considere o problema SUBSET-SUM: dado um inteiro k e conjunto S de números inteiros $s_1, \ldots, s_n$, existe algum subconjunto de S não vazio cuja soma seja k?

a. Proponha um algoritmo baseado em Programação Dinâmica para este problema.

b. Analise a complexidade do algoritmo proposto. O algoritmo é pseudo-polinomial?

c. Dado um conjunto S, uma partição em dois subconjuntos S_1 e S_2 diz-se balanceada se a diferença entre a soma dos elementos em cada conjunto é mínima. Explique como é que o algoritmo proposto pode ser utilizado para determinar uma partição balanceada para S.

III.2.3 Considere as sequências de caracteres `AABCBDEB` e `ABCBDEBC`. Determine a LCS apresentando a matriz de programação dinâmica e o traço correspondente à solução encontrada.

III.2.4 Considere uma sequência de números $A = \langle a_1, a_2, \ldots, a_n \rangle$. $B = \langle b_1, b_2, \ldots, b_k \rangle$ é uma sub-sequência de A se existir uma sequência estritamente crescente $i_1, i_2, \ldots, i_k$ de índices de A, tal que para $j = 1, 2, \ldots, k$, $a_{i_j} = b_j$. B é *crescente* se $b_1 < b_2 < \ldots < b_k$. O problema da maior sub-sequência crescente (LIS) consiste em encontrar a sequência crescente de maior tamanho. Proponha justificadamente um algoritmo eficiente para o problema LIS.

III.2.5 A empresa VeganHaven pretende abrir um conjunto de restaurantes ao longo da EN125. As localizações possíveis para os restaurantes são $1, 2, \ldots, n$, situadas respectivamente aos km $k_1, k_2, \ldots, k_n$. As restrições são as seguintes:

a. Para cada localização i, a empresa VeganHaven pode abrir não mais do que 1 restaurante, existindo um lucro estimado de p_i Euro.

b. Restrições orçamentais obrigam a que cada par de restaurantes esteja afastado de pelo menos Q km.

Proponha um algoritmo para calcular o lucro estimado máximo total para a empresa VeganHaven.

III.2.6 Considere o problema da mochila não fraccionário. Assuma que, numa formulação do problema em termos de programação dinâmica, dados n objectos com valor v_i e peso w_i, com $1 \leq i \leq n$, o valor óptimo da mochila é definido por:

$$v[i,j] = \begin{cases} 0 & \text{se } i = 0 \vee j = 0 \\ -\infty & \text{se } j < 0 \\ \langle \text{ XXX } \rangle & \text{c.c.} \end{cases}$$

Indique a expressão que deve aparecer em vez de $\langle$ XXX $\rangle$.

III.2.7 Seja S a string *volume* e T a string *violent*. Apresente a matriz de programação dinâmica para determinar a maior subsequência comum (LCS) entre S e T. Indique também na matriz o traço que lhe permite reconstruir a LCS.

III.2.8 Uma sub-sequência contínua de uma sequência S é uma sub-sequência constituída por elementos consecutivos de S. Por exemplo, com $S = \langle 5, -10, 7, -3, 40, -20, -10, 5\rangle$, $\langle 5, -10, 7\rangle$ é uma sub-sequência contínua, mas $\langle 5, 7, -3\rangle$ não é. Desenvolva um algoritmo eficiente para o problema seguinte:

Entrada: uma sequência de números $a_1, a_2, \ldots, a_n$.

Saída: sub-sequência contínua com soma máxima.

(Nota: uma sub-sequência com comprimento 0 tem soma 0).

III.2.9 Considere o problema da identificação da maior subsequência comum (LCS) entre duas sequências, S e T. Admita que, numa formulação do problema em termos de programação dinâmica, o comprimento da maior subsequência comum entre os prefixos $S_i = \langle s_1, s_2, \ldots, s_i\rangle$ e $T_j = \langle t_1, t_2, \ldots, t_j\rangle$ é definido por:

$$c[i,j] = \begin{cases} 0 & \text{se } i = 0 \vee j = 0 \\ c[i-1, j-1] + 1 & \text{se } i, j > 0 \wedge s_i = t_j \\ \langle\ \text{XXX}\ \rangle & \text{se } i, j > 0 \wedge s_i \neq t_j \end{cases}$$

Indique a expressão que deve aparecer em vez de $\langle$ XXX $\rangle$.

III.2.10 Considere a seguinte instância do problema da mochila não fraccionário com 4 objectos:

$$\begin{aligned} M &= 11 \\ p &= \langle 4, 5, 5, 7\rangle \\ v &= \langle 8, 12, 15, 24\rangle \end{aligned}$$

M é o peso máximo da mochila, p_i é o peso do objecto i e v_i é o valor do objecto i. Calcule os valores máximos conseguidos utilizando:

- uma estratégia *greedy* com base na ordenação dos valores v_i/p_i;
- um algoritmo baseado em programação dinâmica.

III.2.11 Considere o problema dos trocos utilizando programação dinâmica, com as moedas com as denominações seguintes: $\langle d_1, d_2, d_3, d_4, d_5\rangle = \langle 1, 2, 4, 7, 8\rangle$. Para fazer o troco de $N = 12$, indique qual o menor número de moedas que é necessário utilizar. Indique uma solução. Para a matriz $c[i, j]$, qual o valor de $c[7, 10]$?

III.2.12 Suponha que gere uma empresa de produção de azeite e que existe um conjunto de n encomendas. Cada encomenda i ($1 \leq i \leq n$) permite uma receita de v_i euros na compra de a_i quilolitros de azeite, onde $a_i \geq 1$ e $a_i \in \mathbb{N}$.

Este ano a produção foi mais reduzida do que em anos anteriores, tendo sido produzidos apenas K quilolitros. Como consequência, não será possível satisfazer todas as n encomendas porque $\sum_{i=1}^{n} a_i > K$.

Considerando que as encomendas não podem ser parcialmente satisfeitas, indique um modelo de programação dinâmica que permite decidir quais as encomendas a satisfazer por forma a maximizar a receita total. Indique a complexidade da solução proposta.

III.2.13 Neste problema o objectivo é contar de quantas maneiras diferentes podemos obter um determinado número. Os dados de entrada consistem numa sequência de n números inteiros positivos, e um número m. O objectivo é determinar de quantas formas diferentes podemos escolher elementos da sequência, tal que a soma dos elementos escolhidos seja m.

Considere, por exemplo, a sequência `4, 6, 6, 6, 4`. Suponhamos que $m = 10$. Neste caso existem 6 maneiras de obter este valor. Utilizando um número 4 e um número 6.

Escreva a formula da recursão para a resolução deste problema em termos de programação dinâmica. Indique a complexidade da solução proposta.

III.2.14 Considere uma sequência `A` de n números inteiros positivos. O objectivo é determinar uma subsequência dos n números tal que a soma da subsequência seja maximizada, mas sem escolher dois números consecutivos na sequência.

Por exemplo, considere a sequência `A = < 2, 9, 14, 11, 3, 4, 8>`. Para este caso particular, a melhor solução seria seleccionar os elementos `9, 11, 8` cuja soma é 28. Note que não pode seleccionar dois elementos consecutivos da sequência.

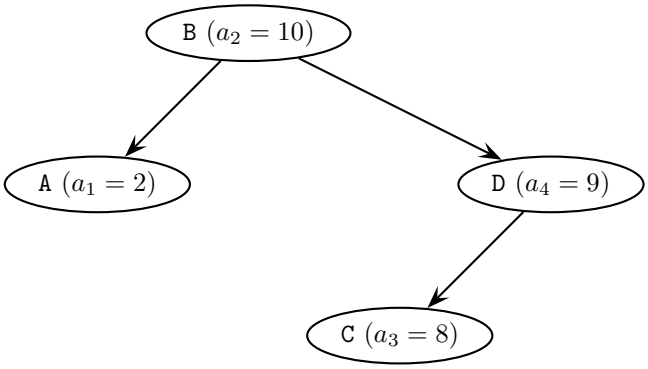
Escreva a fórmula da recursão para a resolução deste problema em termos de programação dinâmica. Indique a complexidade da solução proposta.

III.2.15 Considere uma sequência `A` de n números inteiros positivos e um número M. O objectivo é determinar uma subsequência dos n números tal que a sua soma seja o mais próxima possível de M, mas sem ultrapassar M.

Por exemplo, considere a sequência `A = < 3, 9, 4, 3, 3>` e que $M = 11$. Para este caso particular, a melhor solução seria seleccionar os elementos `3, 4, 3` cuja soma é 10. Observe que a soma dos elementos é menor ou igual a M e a diferença é apenas de 1 unidade.

Escreva a fórmula da recursão para a resolução deste problema em termos de programação dinâmica. Indique a complexidade da solução proposta.

III.2.16 Considere uma árvore binária de pesquisa, em que os nós contêm chaves de procura, tal que travessia *inorder* corresponde a uma sequência ordenada de chaves. No exemplo abaixo essas chaves são as letras `A` $<$ `B` $<$ `C` $<$ `D`. Em geral as chaves de procura $c_1 < \ldots < c_i < \ldots < c_n$ formam uma sequência ordenada.



Consideremos então uma sequência de m acessos a essas chaves, sendo que a chave i é consultada a_i vezes, tal que temos obrigatoriamente $m = \sum_{i=1}^{n} a_i$. Assumindo que a árvore binária de pesquisa é estática, queremos determinar qual seria a estrutura da árvore óptima para uma dada sequência de acessos. A árvore óptima é a que minimiza o custo total que é necessário para executar a sequência de acessos.

Note que o custo de aceder a uma chave corresponde ao número de nós que é preciso percorrer desde a raíz até chegar ao nó que contém a chave. Por exemplo, o custo de aceder à chave `B` é 1 e o custo para aceder à chave `C` é 3. Para a árvore acima, usando a informação sobre a sequência de acessos dada na própria árvore, temos o seguinte custo total $2 \times 2 + 10 \times 1 + 8 \times 3 + 9 \times 2 = 56$.

Sabendo que a árvore apresentada acima não é óptima, indique a árvore óptima para a sequência de acessos considerada. Indique também o custo total para a árvore óptima.

Complete a fórmula da recursão para a resolução deste problema em termos de programação dinâmica.

$$\texttt{B[i,j]} = \begin{cases} 0 & \text{, se } j < i \\ \min_{i \le k \le j} \left\{ \right. & \left. \right\} \text{, caso contrário.} \end{cases}$$

III.2.17 Considere que tem uma mochila com capacidade 12 e o seguinte conjunto de objectos $\{(6,32),(8,64),(12,56),(4,32),(10,48),(8,40)\}$ onde cada par de valores (w_i, v_i) denota o peso e valor do objecto i. Indique o valor óptimo da mochila se pudesse transportar apenas objectos integrais. Indique quais os objectos transportados.

Considerando o modelo de programação dinâmica estudado para este problema, indique também os seguintes valores da respectiva tabela de programação dinâmica: $v[1,6]$, $v[3,12]$, $v[4,10]$, $v[5,10]$.

III.2.18 Considere o problema de multiplicar cadeias de matrizes. O objectivo é determinar por que ordem devem ser feitas as multiplicações por forma a minimizar o numero total de produtos de números que precisam de ser calculados.

Considere uma sequência com 5 matrizes $A_1(1 \times 2)$, $A_2(2 \times 3)$, $A_3(3 \times 1)$, $A_4(1 \times 4)$, $A_5(4 \times 1)$, com as respetivas dimensões entre parênteses. Resolva este problema preenchendo a matriz `m[i,j]` que guarda o menor número de multiplicações que precisam de ser executadas para obter o produto das matrizes de A_i a A_j. Indique os valores de `m[1,4]`, `m[2,5]` e `m[1,5]`. Indique também a colocação de parênteses que obtém o valor indicado em `m[1,5]`.

III.2.19 Nesta questão iremos considerar o problema de distribuir palavras por linhas, por forma a que o resultado final seja o mais equilibrado possivel. Consequentemente o resultado final deverá ser apelativo.

Consideremos a sequência de palavras `aaa bb cc ddddd`, que podem, por exemplo, ser distríbuidas em 3 linhas da seguintes maneiras:

```
aaa bb
cc
ddddd

aaa
bb cc
ddddd
```

Considere que para identificar esta diferença, é dado um vetor `L[i,j]` que representa o custo de guardar as palavras da `i` à `j` numa linha. Quanto menor for este custo melhor. Caso as palavras excedam o tamanho limite da linha o valor `L[i,j]` será $+\infty$. No exemplo acima $\texttt{L}[1,2] + \texttt{L}[3,3]$ é maior do que $\texttt{L}[1,1] + \texttt{L}[2,3]$, indicando assim que a segunda distribuição é considerada melhor. Note que as palavras começam a ser numeradas em 1.

Considerando que o vetor `L[i,j]` já foi previamente calculado, complete a fórmula da recursão para a resolução deste problema em termos de programação dinâmica. Assumindo que `C[j]` representa o custo da melhor distribuição das primeiras j palavras em frases, tantas quantas as que forem necessárias.

$$\texttt{C}[\texttt{j}] = \begin{cases} 0 & \text{, se } \texttt{j} = 0 \\ \min_{1 \le \texttt{i} \le \texttt{j}} \left\{ \right. & \left. \right\} \text{, caso contrário.} \end{cases}$$

III.2.20 Considere a seguinte instância do problema da mochila não fraccionário com 6 objectos, onde p_i denota o peso do objecto i e v_i o valor do objecto i conforme a seguinte tabela:

p_i	2	4	6	7	9	10
v_i	7	9	18	13	17	20

Considerando uma mochila de capacidade $M = 18$, calcule os valores máximos conseguidos utilizando:

- uma estratégia *greedy* com base na ordenação dos objectos por $\frac{v_i}{p_i}$.
- um algoritmo baseado em programação dinâmica

III.2.21 Considere uma sequência `A` de n números inteiros positivos e um número M. O objectivo é determinar qual a maior subsequência dos n números tal que a sua soma é o número M.

Por exemplo, considere a sequência `A = < 1, 3, 2, 3, 4, 3, 6 >` e que $M = 10$. Para este caso particular, a maior subsequência teria tamanho 4 correspondendo a seleccionar os elementos `1, 3, 3, 3`.

Escreva a fórmula da recursão para a resolução deste problema em termos de programação dinâmica. Indique a complexidade da solução proposta.

III.2.22 Considere a seguinte instância do problema da mochila com 5 objectos:

$$\begin{aligned} M &= 7 \\ p &= \langle 1, 2, 3, 4, 5 \rangle \\ v &= \langle 12, 16, 30, 16, 30 \rangle \end{aligned}$$

M é o peso máximo da mochila, p_i é o peso do objecto i e v_i é o valor do objecto i. Calcule os valores máximos obtidos para os seguintes problemas:

- Para o problema fracionário, utilizando uma estratégia *greedy* com base na ordenação dos valores v_i/p_i.
- Para o problema não fracionário, utilizando um algoritmo baseado em programação dinâmica.

Indique a lista dos objectos selecionados, pelo respectivo índice, ou peso. Na solução greedy o objecto que é partido deve ser o último da lista.

III.2.23 Considere o problema de multiplicar cadeias de matrizes. O objetivo é determinar por que ordem devem ser feitas as multiplicações por forma a minimizar o número total de operações escalares que precisam ser efetuadas.

Considere uma sequência com 5 matrizes $A_1(2 \times 1)$, $A_2(1 \times 3)$, $A_3(3 \times 2)$, $A_4(2 \times 1)$, $A_5(1 \times 4)$, com as respetivas dimensões entre parênteses. Resolva este problema preenchendo a matriz `m[i,j]` que guarda o menor número de multiplicações que precisam de ser executadas para obter o produto das matrizes de A_i a A_j. Indique os valores de `m[1,4]`, `m[2,5]` e `m[1,5]`. Indique também a colocação de parênteses que obtém o valor indicado em `m[1,5]`.

III.2.24 Recorde o problem SubSet-Sum que considera um conjunto de números inteiros positivos S e um inteiro alvo $t > 0$. O problema consiste em determinar se existe um sub-conjunto de inteiros $S' \subseteq S$ tal que a sua soma seja t. Por conveniência assumimos que os valores de S estão ordenados e que podem ser referênciados como S_i para algum índice i a começar em 1. Por exemplo, para o conjunto $S = \{3, 6, 7\}$ temos que $S_2 = 6$.

Nesta questão pretendemos determinar uma algoritmo pseudo-polinomial para este problema, utilizando programação dinâmica. Utilizaremos uma matriz `B[i,j]` em que `i` varia entre 0 e o tamanho de S e `j` varia entre 0 e t. Cada valor `B[i,j]` deverá indicar o maior valor que pode ser obtido somando um sub-conjunto S' dos primeiros i elementos de S de forma a não exceder j, i.e., $\texttt{B[i, j]} \leq \texttt{j}$.

Complete a fórmula da recursão para a resolução deste problema.

$$\texttt{B[i, j]} = \begin{cases} -\infty & \text{, se } j < 0 \\ \boxed{} & \text{, se } i = 0 \text{ e } j \geq 0 \\ \boxed{} & \text{, se } i > 0 \text{ e } j \geq 0 \end{cases}$$

III.2.25 Considere o problema dos trocos utilizando programação dinâmica. Assuma que apenas existem moedas com dois valores, a moeda d_1 com valor 4 e a moeda d_2 com valor 5. É possivel utilizar várias moedas com o mesmo valor, por exemplo para fazer um troco de 16 é possível utilizar quatro moedas d_1. Existem quantias que não podem ser obtidas com estas moedas, por exemplo não é possível obter um troco de valor 3.

Utilizando programação dinâmica identifique o maior troco que não pode ser obtido com estas moedas. Indentifique também qual o menor número de moedas necessárias para obter um troco de valor 17 e quais são as moedas necessárias.

III.2.26 Nesta questão iremos utilizar um algoritmo de programação dinâmica para encontrar uma sub-sequência alternada que maximiza a soma total. Os dados consistem numa sequência de números não negativos S que podem ser referênciados como S_i para algum índice i a começar em 1. Por exemplo, para a sequência $S = (5, 2, 1, 7)$, temos $S_2 = 2$.

O objectivo é obter uma sub-sequência com a maior soma possível, respeitando a regra que a sub-sequência não pode ter índices consecutivos, por exemplo se $S_1 = 5$ for selecionado então $S_2 = 2$ não pode fazer parte da sub-sequência. Neste caso a solucão seria $S_1 + S_4 = 5 + 7 = 12$.

Utilizaremos um vector `D[i]` em que `i` varia entre 0 e o tamanho de S. Cada valor `D[i]` deverá indicar a maior soma que pode ser obtida a partir de uma sub-sequência alternada S' dos primeiros i elementos de S.

Complete a fórmula da recursão para a resolução deste problema.

$$\texttt{D[i]} = \begin{cases} \boxed{} & \text{, se } i = 0 \\ \boxed{} & \text{, se } i = 1 \\ \boxed{} & \text{, se } i > 1 \end{cases}$$

III.3. Programação Linear

III.3.1 De modo a assegurar saúde óptima, um técnico de laboratório tem de alimentar coelhos com uma dieta diária contendo um mínimo de 24g de gordura, 36g de hidratos de carbono, e 4g de proteína. Além disso, não deve ser oferecido aos coelhos mais do que 150g de comida por dia.

O técnico de laboratório deve encomendar dois tipos de alimentos, X e Y, e liquidificá-los de modo a obter uma mistura óptima. O alimento X contém 32g de gordura, 48g de hidratos de carbono, e 8g de proteína por cada 100g, e custa 0.65 Euro por 100g. O alimento Y contém 48g de gordura, 48g de hidratos de carbono, e 4g de proteína por cada 100g, e custa 0.53 Euro por 100g.

A mistura é óptima se é tão barata quanto possível, mas garante a saúde óptima dos coelhos. Formule o problema como um problema de programação linear. (Considere a quantidade de comida que um coelho necessita por dia.)

III.3.2 Uma fábrica de comida congelada labora de segunda a sexta e tem 3 fornecedores de produtos frescos. Todos os dias de laboração é necessário comprar um lote de produtos frescos, o qual é comprado ao fornecedor que fizer o preço mais baixo. No entanto, os produtos de segunda, terça e quarta são sempre comprados a fornecedores diferentes, garantindo assim que é efectuada pelo menos uma compra semanal a cada um dos 3 fornecedores.

No início de cada semana a fábrica sabe os preços que cada um dos fornecedores fará em cada dia dessa semana para o lote de produtos frescos, os quais têm em conta os ciclos de produção de cada fornecedor e a procura estimada para cada dia da semana. Considere que os preços de cada fornecedor para o lote de produtos frescos, para cada dia de uma dada semana, são dados pela tabela abaixo.

fornecedor	segunda	terça	quarta	quinta	sexta
A	a_1	a_2	a_3	a_4	a_5
B	b_1	b_2	b_3	b_4	b_5
C	c_1	c_2	c_3	c_4	c_5

Tabela semanal de preços do lote de produtos frescos

Apresente uma formulação em termos de programação linear que, dados os valores de $a_1, \ldots, a_5, b_1, \ldots, b_5, c_1 \ldots, c_5$ para uma dada semana, permita determinar qual o custo total que a fabrica terá com a aquisição dos lotes de produtos frescos durante essa semana. A sua formulação deve ser apresentada na forma *standard* e deve incluír no máximo 12 restrições, para além das restrições de não negatividade das variáveis.

III.3.3 Uma fábrica produz 4 produtos em arame: tecido industrial; redes para insectos; revestimento de telhados; e vedações. Para cada produto, o arame em alumínio é esticado até atingir uma espessura apropriada e depois moldado de modo a formar uma rede. Os requisitos de produção e as margens de lucro variam para cada produto de acordo com o quadro abaixo. Existem 600 horas disponíveis na máquina que produz o arame, e 1000 horas na máquina que produz os diferentes tipos de rede. Existem também 15 toneladas de arame em alumínio disponíveis. Que quantidade de cada produto deverá ser produzido de modo a maximizar o lucro? Assuma que a companhia pode vender tudo o que produz.

Valores por 1000 m^2 de cada produto:	Arame em alumínio (ton.)	Produção de arame (100 hrs.)	Produção de rede (100 hrs.)	Margem de lucro ($ 100)
Tecido industrial	1	1	2	6
Rede para insectos	3	1	1	5
Revestimento de telhados	3	2	1.5	3.8
Vedações	2.5	1.5	2	4

Formule o programa linear que permite resolver o problema.

III.3.4 Considere uma empresa de consultoria que tem os seus colaboradores divididos por quatro níveis de experiência (E1, E2, E3 e E4), sendo que o salário é também diferente para cada nível. Cada colaborador pode ser alocado a um qualquer projecto, mas com diferente desempenho, o qual depende do seu nível de experiência. No caso desta empresa, dada a sua área de negócio, apenas é alocado um colaborador a cada projecto.

A tabela seguinte indica, para cada nível de experiência, o número de colaboradores, o seu salário mensal e o indice de desempenho.

Categorias	E1	E2	E3	E4
#Colaboradores	6	8	10	4
Salário (euros)	900	1200	1500	2000
Desempenho	3	4	6	5

Suponha que a empresa tem 14 projectos a decorrer neste momento. Formule o programa linear que permite resolver o problema da alocação dos colaboradores a esse conjunto de projectos, maximizando o desempenho, mas de tal forma que o somatório total dos salários dos colaboradores seleccionados não é superior a 20000 euros.

III.3.5 Uma empresa utiliza quatro fábricas para produzir automóveis. A produção de cada automóvel em cada fábrica requere recursos, os quais se encontram divididos entre horas de mão de obra, unidades de materiais utilizados, e unidades de poluição produzida:

Fábrica	Mão de obra	Materiais	Poluição
1	2	3	15
2	3	4	10
3	4	5	9
4	5	6	7

As restrições de produção são as seguintes:

a. Por acordo com os sindicatos, a fábrica 3 produz pelo menos 400 automóveis.

b. A mão de obra total (por mês) não pode exceder as 3300 horas.

c. Os materiais disponíveis (por mês) não podem exceder as 4000 unidades.

d. A poluição produzida (por mês) não pode exceder as 12000 unidades.

O objectivo é maximizar a produção de automóveis, tendo em conta as restrições existentes.
Formule o programa linear que permite resolver este problema.

III.3.6 Considerando a industrialização da confecção de comida e o estilo de vida actual das sociedades ocidentais, a manutenção de uma dieta equilibrada tornou-se um problema para a maior parte das pessoas. Os Valores Diários de Referência (VDR) constituem uma base referencial para adultos saudáveis com uma constituição média.

A seguinte tabela contém os VDR para alguns componentes da dieta:

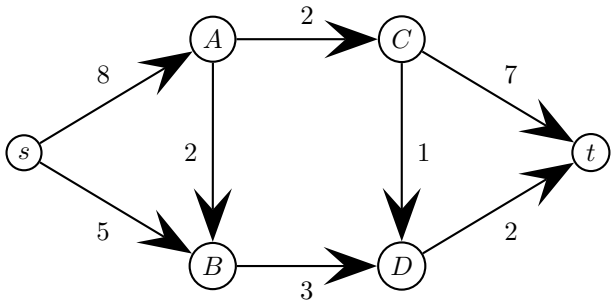
	VDR
Energia	2000 kcal.
Proteínas	50 g.
Hidratos de Carbono	270 g.
Fibra	25 g.

Considere-se uma dieta equilibrada a ingestão de alimentos que permita obter entre 90% a 110% dos VDR para cada componente da dieta. Considere ainda que pode comprar vários produtos alimentares cuja composição e preço unitário estão descritos na tabela seguinte.

Alimento	Energia	Proteínas	Hidratos de Carbono	Fibra	Preço Unit.
A_1	300	20	30	5	4.5
A_2	900	5	90	5	6
A_3	200	10	50	10	2
A_4	500	15	70	10	3

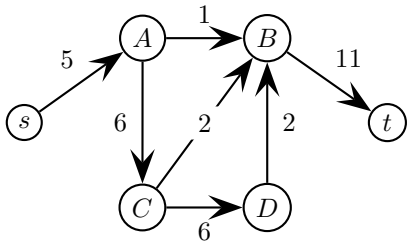
Supondo que pode comprar qualquer proporção de cada alimento, formule o programa linear que permita obter uma dieta equilibrada minimizando o custo total dos alimentos a comprar.

III.3.7 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



Formule o programa linear que permita calcular o fluxo máximo da rede.

III.3.8 Considere a rede de fluxo da figura onde s e t são respectivamente os vértices fonte e destino na rede.



Explicite o programa linear para resolução do problema do cálculo do fluxo máximo na rede de fluxo.

III.3.9 Explicite o programa linear para resolução do problema da mochila fraccionário considerando uma mochila com capacidade 18 e um conjunto de 6 objectos conforme a seguinte tabela:

p_i	2	4	6	7	9	10
v_i	7	9	18	13	17	20

Recorde que p_i denota o peso do objecto i e v_i o valor do objecto i.

III.3.10 Resolva o programa linear:

$$\begin{array}{lrcr} \text{Maximizar} & x_1 - x_2 - 2x_3 & & \\ \text{Sujeito a} & -x_1 - 2x_2 - 3x_3 & \leq & 1 \\ & x_2 - x_3 & \leq & 2 \\ & x_1 - x_3 & \geq & -3 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

III.3.11 Escreva o programa dual para o programa linear do exercício anterior. Qual é a solução óptima do programa dual?

III.3.12 Suponha que temos um programa linear com n variáveis e m restrições. Indique o número máximo de variáveis e de restrições após convertermos o programa linear para a forma standard.

III.3.13 Indique se cada uma das seguintes afirmações é verdadeira ou falsa.

a. A redução de um programa linear para a forma slack tem complexidade polinomial no número de variáveis e de restrições do problema original.

b. Qualquer problema de programacao linear é resolúvel em tempo polinomial pelo algoritmo Simplex.

c. O número de formas slack possíveis é exponencial no número de variáveis básicas e não básicas.

d. Se um problema linear primal é impossível, então o problema linear dual respectivo também é impossível.

e. Se na função objectivo de um dado problema linear todos os coeficientes forem negativos, então a solução óptima é a solução básica inicial.

III.3.14 Considere o programa linear

$$\begin{array}{lrcl} \text{Maximizar} & 2x_1 + x_2 & & \\ \text{Sujeito a} & x_1 + x_2 & \leq & 10 \\ & x_1 - x_2 & \leq & 4 \\ & x_1 & \geq & 3 \\ & x_1, x_2 & \geq & 0 \end{array}$$

Formule e resolva o programa auxiliar. (Resolva apenas o programa auxiliar).

III.3.15 Calcule o valor óptimo da função objectivo e o respectivo valor das variáveis x_1, x_2 e x_3 para o programa linear abaixo utilizando o algoritmo *Simplex*.

$$\begin{array}{lrcl} \text{Maximizar} & x_1 + 3x_2 + x_3 & & \\ \text{Sujeito a} & x_1 + x_2 + x_3 & \leq & 4 \\ & 2x_1 + x_2 & \leq & 10 \\ & x_1 + 2x_2 & \leq & 6 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

Indique também o valor óptimo e a solução do problema dual.

III.3.16 Resolva o seguinte programa linear.

$$\begin{array}{lrcl} \text{Minimizar} & -2x_1 - x_2 & & \\ \text{Sujeito a} & x_1 + x_2 & \leq & 10 \\ & x_1 - x_2 & \leq & 4 \\ & x_1 & \leq & 3 \\ & x_1, x_2 & \geq & 0 \end{array}$$

Indique a forma *slack* final, a solução e o valor óptimo.

III.3.17 Indique se a solução básica inicial é exequível para o o programa linear seguinte. Caso não seja, calcule uma solução básica inicial exequível, utilizando o método do programa linear auxiliar.

$$\begin{array}{lrcl} \text{Maximizar} & x_1 + x_2 + x_3 & & \\ \text{Sujeito a} & x_1 + x_2 & \leq & 2 \\ & x_1 - x_3 & \leq & 4 \\ & -x_3 & \leq & -2 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

III.3.18 Utilize o algoritmo *Simplex* para calcular o valor óptimo da função objectivo e o respectivo valor das variáveis x_1 e x_2 para o programa linear seguinte.

$$\begin{array}{lrcl} \text{Maximizar} & 30x_1 + 20x_2 & & \\ \text{Sujeito a} & 2x_1 + x_2 & \leq & 100 \\ & x_1 + x_2 & \leq & 80 \\ & x_1 & \leq & 40 \\ & x_1, x_2 & \geq & 0 \end{array}$$

III.3.19 Calcule o valor óptimo da função objectivo e o respectivo valor das variáveis x_1 e x_2 para o seguinte programa linear:

$$\begin{array}{lrcl} \text{Minimizar} & 3x_1 + 2x_2 & & \\ \text{Sujeito a} & -x_1 + x_2 & \leq & -3 \\ & 2x_1 - x_2 & \leq & 6 \\ & -x_1 - x_2 & \leq & -1 \\ & x_1, x_2 & \geq & 0 \end{array}$$

III.3.20 Calcule o valor óptimo da função objectivo e o respectivo valor das variáveis x_1 e x_2 para o seguinte programa linear:

$$\begin{array}{lrcl} \text{Maximizar} & 2x_1 + x_2 & & \\ \text{Sujeito a} & -x_1 + x_2 & \leq & 3 \\ & x_1 + x_2 & \leq & 6 \\ & x_1 & \leq & 5 \\ & x_1, x_2 & \geq & 0 \end{array}$$

III.3.21 Indique o valor das variáveis x_1, x_2 e x_3 na solução básica inicial do seguinte programa linear. Indique ainda se a solução básica inicial corresponde a uma solução óptima.

$$\begin{array}{lrcl} \text{Maximizar} & x_1 - x_2 - 3x_3 & & \\ \text{Sujeito a} & -x_1 + x_2 + 2x_3 & \leq & -4 \\ & -2x_1 + x_2 + 2x_3 & \leq & 2 \\ & -x_2 - x_3 & \leq & 1 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

III.3.22 Indique a forma standard para o seguinte programa linear:

$$\begin{array}{lrcl} \text{Minimizar} & -2x_1 - x_2 + 3x_3 & & \\ \text{Sujeito a} & x_1 + 4x_2 - 4x_3 & \leq & -5 \\ & -2x_1 + x_2 + x_3 & = & 10 \\ & 4x_2 - 5x_3 & \geq & 6 \\ & x_1, x_3 & \geq & 0 \end{array}$$

III.3.23 Indique o valor da função objectivo e o respectivo valor das variáveis x_1, x_2 e x_3 na **primeira solução exequível** encontrada pelo algoritmo Simplex para o seguinte programa linear.

Em caso de empate em algum critério de aplicação do algoritmo, aplique a regra de Bland. Ou seja, escolha a variável de menor indice.

$$\begin{array}{lrcl} \text{Maximizar} & -5x_1 + 2x_2 + x_3 & & \\ \text{Sujeito a} & x_1 - x_2 & \leq & -6 \\ & x_2 + x_3 & \leq & 8 \\ & -x_1 - x_3 & \leq & -2 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

III.3.24 Indique a formulação dual para o seguinte programa linear:

$$\begin{array}{lrcl} \text{Maximizar} & 3x_1 - x_2 - 3x_3 & & \\ \text{Sujeito a} & x_1 + 4x_2 - 4x_3 & \leq & -2 \\ & -2x_1 + x_2 + x_3 & \leq & 10 \\ & x_1 - x_2 & \leq & -5 \\ & 4x_2 - 6x_3 & \leq & 6 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

III.3.25 Calcule o valor óptimo da função objectivo e o respectivo valor das variáveis x_1, x_2 e x_3 para o seguinte programa linear:

$$\begin{array}{lrcl} \text{Maximizar} & 4x_1 + 2x_2 + x_3 & & \\ \text{Sujeito a} & x_1 + x_2 & \leq & 4 \\ & x_2 + x_3 & \leq & 8 \\ & x_1 - x_3 & \leq & 2 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

III.3.26 Considere o seguinte programa linear. Determine uma solução óptima para o programa e indique o respectivo valor de objectivo.

$$\begin{array}{ll} \text{Maximizar} & 2x_1 + x_2 + x_3 \\ \text{Sujeito a} & x_1 - x_2 - 2x_3 \leq 4 \\ & x_1 - 2x_2 - x_3 \leq 6 \\ & x_1 + x_2 + x_3 \leq 20 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

III.3.27 Indique a formulação dual para o programa linear da pergunta anterior. Verifique também se a atribuição $y_1 = 1/5$, $y_2 = 1/5$, $y_3 = 8/5$ é exequível para o sistema dual. Assuma que a variável y_1 está associada à primeira restrição, a variável y_2 à segunda restrição e a variável y_3 à terceira.

III.3.28 Considere o seguinte programa linear:

$$\begin{array}{lrcl} \text{Maximizar} & -4x_1+5x_2+3x_3 & & \\ \text{Sujeito a} & -\ x_1+\ x_2+\ x_3 & \leq & -2 \\ & -3x_1+4x_2+4x_3 & \leq & 1 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

Indique o valor da função objectivo e o respectivo valor das variáveis x_1, x_2 e x_3 na **primeira solução exequível** encontrada pelo algoritmo Simplex. Em caso de empate em algum critério de aplicação do algoritmo, aplique a regra de Bland. Ou seja, escolha a variável de menor índice.

Indique também o valor da função objectivo e o respectivo valor das variáveis para a solução ótima e para a solução do sistema dual, com a variável y_1 associada à primeira restrição e a variável y_2 associada à segunda restrição.

III.3.29 Calcule o valor óptimo da função objectivo e o respectivo valor das variáveis x_1, x_2 e x_3 para o seguinte programa linear:

$$\begin{array}{lrcl} \text{Maximizar} & 3x_1 + x_2 + x_3 & & \\ \text{Sujeito a} & x_1 + x_2 & \leq & 4 \\ & -x_2 + x_3 & \leq & 5 \\ & x_1 + x_3 & \leq & 2 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

III.3.30 Indique o valor da função objectivo e o respectivo valor das variáveis x_1, x_2 e x_3 na primeira solução exequível encontrada pelo algoritmo Simplex (i.e., a solução do programa linear auxiliar) para o seguinte programa linear:

$$\begin{array}{lrcl} \text{Maximizar} & 3x_1 + x_2 + x_3 & & \\ \text{Sujeito a} & -x_2 - x_3 & \leq & -2 \\ & -2x_1 - x_3 & \leq & -4 \\ & x_1 + x_2 + x_3 & \leq & 6 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

III.3.31 Calcule o valor óptimo da função objectivo e o respectivo valor das variáveis x_1 e x_2 para o seguinte programa linear:

$$\begin{array}{lrcl} \text{Minimizar} & 4x_1 + x_2 & & \\ \text{Sujeito a} & x_1 - x_2 & \leq & 2 \\ & 2x_1 + x_2 & \leq & 4 \\ & -2x_1 + x_2 & \leq & 0 \end{array}$$

III.3.32 Considere o programa linear

$$\begin{array}{lrcl} \text{Maximizar} & 3x_1 + x_2 + 2x_3 & & \\ \text{Sujeito a} & x_1 + x_2 + 3x_3 & \leq & 30 \\ & 2x_1 + 2x_2 + 5x_3 & \leq & 24 \\ & 4x_1 + x_2 + 2x_3 & \leq & 36 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

Suponha que a forma slack seguinte é equivalente ao programa linear acima, onde x_4, x_5 e x_6 são as variáveis de slack correspondentes às três restrições.

$$\begin{array}{rcl} z & = & 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3} \\ x_1 & = & 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3} \\ x_2 & = & 4 - \frac{8x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3} \\ x_4 & = & 18 - \frac{x_3}{2} + \frac{x_5}{2} \\ x_1, x_2, x_3, x_4, x_5, x_6 & \geq & 0 \end{array}$$

Escreva o programa dual, para o programa linear acima. Qual é a solução óptima do programa dual?

III.3.33 Indique a formulação dual do seguinte programa linear.

$$\begin{array}{lrcl} \text{Maximizar} & 3x_1 - 2x_2 + x_3 - 4x_5 & & \\ \text{Sujeito a} & x_1 + x_2 + x_4 + x_5 & \leq & 10 \\ & -2x_1 + 4x_3 + 3x_4 - x_5 & \leq & -5 \\ & x_2 - 3x_3 & \leq & 4 \\ & 3x_1 - 5x_2 + 4x_3 - 3x_4 + x_5 & \leq & -3 \\ & x_1, x_2, x_3, x_4, x_5 & \geq & 0 \end{array}$$

III.3.34 Considere o seguinte programa linear:

$$\begin{array}{lrcl} \text{Maximizar} & -4x_1 + 5x_2 + x_3 & & \\ \text{Sujeito a} & -x_1 + x_2 + x_3 & \leq & -4 \\ & -3x_1 + 4x_2 + 4x_3 & \leq & 2 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

Indique o valor da função objectivo e o respectivo valor das variáveis x_1, x_2 e x_3 na **primeira solução exequível** encontrada pelo algoritmo Simplex. Em caso de empate em algum critério de aplicação do algoritmo, aplique a regra de Bland. Ou seja, escolha a variável de menor índice.

Indique também o valor da função objectivo e o respectivo valor das variáveis para a solução ótima e para a solução do sistema dual, com a variável y_1 associada à primeira restrição e a variável y_2 associada à segunda restrição.

III.3.35 Considere o seguinte programa linear:

$$\begin{array}{lrl} \text{Maximizar} & -4x_1+5x_2-4x_3 & \\ \text{Sujeito a} & -\ x_1+\ x_2+\ x_3 \leq & -2 \\ & -3x_1+4x_2+3x_3 \leq & 1 \\ & x_1, x_2, x_3\ \geq & 0 \end{array}$$

Indique o valor da função objectivo e o respectivo valor das variáveis x_1, x_2 e x_3 na **primeira solução exequível** encontrada pelo algoritmo Simplex. Em caso de empate em algum critério de aplicação do algoritmo, aplique a regra de Bland. Ou seja, escolha a variável de menor índice.

Indique também o valor da função objectivo e o respectivo valor das variáveis para a solução ótima e para a solução do sistema dual, com a variável y_1 associada à primeira restrição e a variável y_2 associada à segunda restrição.

III.4. Emparelhamento de Cadeias de Caracteres

III.4.1 Considere o algoritmo de Knuth-Morris-Pratt (KMP) para emparelhamento de cadeias de caracteres. Dado o padrão $P =$`ababbababbababba`, determine os valores de π.

III.4.2 Considere o algoritmo `FINITE-AUTOMATON-MATCHER` e o alfabeto $\Sigma = \{a, b\}$. Dado o padrão $P =$`aabaaabba`, o autómato para P e o texto $T =$`aaabaaabbaabaaa`, indique o estado do autómato para cada caracter do texto T.

III.4.3 Proponha uma adaptação do algoritmo de Rabin-Karp para o problema de procurar k padrões num texto. Assuma que os padrões têm o mesmo tamanho.

III.4.4 Dado o padrão $P =$`ababbababbababba`, determine o autómato finito que o permite identificar. Indique o estado resultante das seguintes transições: $\delta(3, \texttt{a})$, $\delta(6, \texttt{b})$, $\delta(9, \texttt{a})$, $\delta(12, \texttt{b})$, $\delta(15, \texttt{a})$.

III.4.5 Considere o algoritmo de Knuth-Morris-Pratt (KMP) para emparelhamento de cadeias de caracteres, cujo pseudo-código é apresentado abaixo.

```
KMP-Matcher(T, P)
 1  n ← length[T]
 2  m ← length[P]
 3  π ← Compute-Prefix-Function(P)
 4  q ← 0
 5  for i ← 1 to n
 6        do while q > 0 and P[q + 1] ≠ T[i]
 7              do q ← π[q]
 8           if P[q + 1] = T[i]
 9              then q ← q + 1
10           if q = m
11              then Print "Pattern occurs with shift" i − m
12                   q ← π[q]
```

Sabendo que $\pi = \langle 0, 0, 1, 2, 3, 1, 2, 0\rangle$, e que $\Sigma = \{\texttt{a}, \texttt{b}\}$, indique qual o padrão P. Se o texto a ser analizado for $T =$`abababab`, indique para cada valor de i do ciclo **for** da linha 5, qual o valor de q observado na linha 10.

III.4.6 Indique um algoritmo que permita determinar em tempo linear se uma sequência A é uma rotação cíclica de outra sequência B. Por exemplo, `vala` e `lava` são rotações cíclicas uma da outra. Justifique a sua resposta.

III.4.7 Considere o algoritmo Finite-Automaton-Matcher, o alfabeto $\Sigma = \{a, b\}$ e o padrão $P = ababaaababa$. Determine a função de transição δ para o automato construído pelo algoritmo para o padrão P.

III.4.8 Considere o algoritmo de Rabin-Karp, o alfabeto $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ e $q = 7$. Indique o número de falsos positivos detectados na execução do algoritmo para o padrão $P = 21$ e o texto $T = 2772149565$.

III.4.9 Assuma que queremos procurar k padrões P_i, com $1 \leq i \leq k$ e $k \in \mathbb{N}$, num texto T. Assuma que os padrões P_i têm todos os mesmo comprimento m e que o texto tem comprimento n. Indique a menor complexidade assimptótica em função de k, n e m no caso de utilizamos o algoritmo de Knuth-Morris-Pratt para resolver este problema.

III.4.10 Considere o algoritmo de Knuth-Morris-Pratt. Determine a função de prefixo π para o padrão abaixo.

`aababaabaababababaabb`

III.4.11 Considere o algoritmo Finite-Automaton-Matcher, o alfabeto $\Sigma = \{a, b\}$, o padrão $P = abaaaba$, e o texto $T = ababaaabaaabaa$. Indique o estado do autómato para cada caracter do texto T.

III.4.12 Considere o algoritmo de Rabin-Karp. A complexidade pior caso deste algoritmo é $\Theta((n - m + 1)m)$, com preprocessamento em $\Theta(m)$. Indique um exemplo de um padrão e de um texto que exercitem o pior caso do algoritmo de Rabin-Karp.

III.4.13 Assuma que queremos procurar k padrões P_i, com $1 \leq i \leq k$ e $k \in \mathbb{N}$, num texto T. Assuma que os padrões P_i têm todos o mesmo comprimento m e que o texto tem comprimento n. Indique a menor complexidade assimptótica em função de k, n e m para indicar para cada posição do texto a possível ocorrência de um qualquer padrão utilizando uma abordagem baseado no algoritmo de Rabin-Karp. Note que não deve considerar o custo de verificar a ocorrência de cada padrão, apenas o custo de identificar a possível ocorrência de um padrão.

III.4.14 Considere o algoritmo baseado em autómatos para o emparelhamento de cadeias de caracteres. Dado o padrão `abbab`, indique para cada posição do texto `aababbaabbabbab` o estado do autómato na excução do algoritmo.

III.4.15 Considere o algoritmo baseado em autómatos para o emparelhamento de cadeias de caracteres. Seja $n \in \mathbb{N}$ e P o padrão $ab^nab^{n-1}ab^{n-2}...ab^2aba$ tal que $a \neq b$ e $a, b \in \Sigma$. Calcule o número de transições para estados diferentes do inicial em função de n e $|\Sigma|$.

III.4.16 Considere o algoritmo de Knuth-Morris-Pratt. Calcule a função de prefixo π para o padrão `dcbadbcdcbadbdcb`.

III.4.17 Considere o algoritmo para emparelhamento de caracteres baseados em autómatos finitos. Para o padrão $P = abaaba$, com $\Sigma = \{a, b\}$, indique o número de transições para cada estado.

III.4.18 Considere o algoritmo de Knuth-Morris-Pratt (KMP). Para um padrão com m caracteres, qual o valor máximo que a função de prefixo pode tomar? Indique um exemplo de padrão para o qual o valor máximo possível para a função de prefixo é atingido.

III.4.19 Considere o algoritmo de Rabin-Karp, com $P = 2829$, $T = 5656727328293940$, com $q = 11$. Indique o número de emparelhamentos válidos e de *spurious hits*.

III.4.20 Considere o algoritmo de Knuth-Morris-Pratt. Calcule a função de prefixo $\pi[k]$ para o padrão $P =$ `bbaababbaababa`. Indique os valores de $\pi[2]$, $\pi[6]$, $\pi[10]$, $\pi[13]$ e $\pi[14]$.

III.4.21 Considere o algoritmo de autómatos finitos para o emparelhamento de caracteres. Seja $n \in \mathbb{N}$ e P o padrão $ab^nab^{n-1}ab^{n-2}\ldots ab^3ab^2ab$ tal que $a \neq b$, $a, b \in \Sigma$ e $n \geq 2$. Sendo $\delta : Q \times \Sigma \to Q$ a função de transição do autómato, indique, o número de transições para o estado inicial.

III.4.22 Dado o padrão `bbaababba`, determine o autómato finito que emparelha este padrão numa cadeia de caracteres. Indique o estado resultante das seguintes transições: $\delta(2, b)$, $\delta(4, a)$, $\delta(5, a)$, $\delta(9, a)$, $\delta(9, b)$.

III.4.23 Considere o algoritmo de Knuth-Morris-Pratt para o emparelhamento de caracteres. Seja $n \in \mathbb{N}$ e P o padrão $a^nba^{n-1}ba^{n-2}b\ldots a^3ba^2bab$ tal que $a \neq b$, $a, b \in \Sigma$ e $n \geq 2$. Considere o cálculo da função de prefixo $\pi[k]$ para o padrão P. Indique, em função de n, para quantos valores diferentes de k é que temos $\pi[k] = 0$.

III.4.24 Considere o algoritmo de autómatos finitos para o emparelhamento de caracteres. Dado o padrão $P =$ `cbaacbabcbaacb`, indique os valores para as seguintes transições de estado: $\delta(5, c)$, $\delta(7, a)$, $\delta(11, b)$, $\delta(13, c)$ e $\delta(14, a)$.

Nota: O estado inicial do autómato é o estado 0.

III.4.25 Considere o algoritmo de Knuth-Morris-Pratt para o emparelhamento de caracteres. Seja $n \in \mathbb{N}$ e P o padrão $(ab)c(ab)^2c^2(ab)^3c^3\ldots(ab)^{n-1}c^{n-1}(ab)^nc^n$ tal que $\Sigma = \{a, b, c\}$ e $n \geq 2$.

Considere o cálculo da função de prefixo $\pi[k]$ para o padrão P. Indique, em função de n, para quantos valores diferentes de k é que temos $\pi[k] = 0$.

III.4.26 Considere o algoritmo de autómatos finitos para o emparelhamento de caracteres. Se o padrão tiver k caracteres, os estados do autómato são $Q = \{0, 1, 2, \ldots, k-1, k\}$.

Seja $n \in \mathbb{N}$ e P o padrão $(ab)^nc^n$ tal que $\Sigma = \{a, b, c\}$ e $n \geq 2$. Sendo $\delta : Q \times \Sigma \to Q$ a função de transição do autómato, indique em função de n o número de transições para o estado 1. Note que o estado inicial é o estado 0.

III.4.27 Considere o algoritmo de Knuth-Morris-Pratt. Calcule a função de prefixo $\pi[k]$ para o padrão $P =$ `cbaabcbabcbaac`. Indique os valores de $\pi[5]$, $\pi[8]$, $\pi[10]$, $\pi[13]$ e $\pi[14]$.

III.4.28 Dada uma string $P = P_pP_s$, onde P_p é um prefixo e P_s é um sufixo, a string P_sP_p é uma rotação cíclica. Por exemplo dada a string $P = ABACTD$ as strings $TDABAC$ e $ACTDAB$, entre outras, são rotações cíclicas. Indique um algoritmo que dadas duas strings P e T verifica se T é uma rotação cíclica de P. Note que apenas será dada a cotação máxima ao algoritmo mais eficiente possível. Indique a complexidade do algoritmo proposto.

III.4.29 Considere o algoritmo de Rabin-Karp utilizando `mod` 9. Dado um texto $T =$ `444040635` e um padrão com 4 letras, calcule o valores de hashing t_i do respectivo texto.

III.4.30 Considere o algoritmo de autómatos finitos para o emparelhamento de caracteres. Dado o padrão $P =$ `aababaaabaa`, indique os valores para as seguintes transições de estado: $\delta(4, b)$, $\delta(4, a)$, $\delta(7, b)$, $\delta(10, b)$ e $\delta(11, b)$.

Nota: O estado inicial do autómato é o estado 0.

III.4.31 Considere o algoritmo de Knuth-Morris-Pratt para o emparelhamento de caracteres. Dado o padrão $P =$`aabaaaabaabaaaa`, determine os valores de π.

III.4.32 Dado o padrão $P =$ `bbaababbaababa`, determine o autómato finito que emparelha este padrão numa cadeia de caracteres. Indique o estado resultante das seguintes transições: $\delta(2, b)$, $\delta(5, b)$, $\delta(6, a)$, $\delta(13, b)$ e $\delta(14, b)$.

III.4.33 Considere o algoritmo de autómatos finitos para o emparelhamento de caracteres. Seja $n \in \mathbb{N}$ e P o padrão $(ab)^n c^n$ tal que $\Sigma = \{a, b, c\}$ e $n \geq 2$.

Sendo $\delta : Q \times \Sigma \to Q$ a função de transição do autómato, indique em função de n o número de transições para o estado inicial.

III.4.34 Considere o algoritmo de Knuth-Morris-Pratt. Calcule a função de prefixo $\pi[k]$ para o padrão $P =$ `abcababcabacabcabc`. Indique os valores de $\pi[6]$, $\pi[11]$, $\pi[15]$, $\pi[17]$ e $\pi[18]$.

III.4.35 Considere o algoritmo de Knuth-Morris-Pratt para emparelhamento de caracteres onde π denota a função de prefixo. Seja $n \in \mathbb{N}$ e P o padrão $(ab)^n c^n$ tal que $\Sigma = \{a, b, c\}$ e $n \geq 2$. Indique, em função de n, para quantos valores de k é que temos $\pi[k] \neq 0$.

III.4.36 Considere a maior sub-sequência comum entre as duas strings $AAACTATT$ e $ACATACT$ e calcule a respectiva matriz de programação dinâmica $c[i, j]$ para este problema, em que o índice i está associado à string $AAACTATT$. Indique os seguintes valores: $c[2, 3]$, $c[2, 6]$, $c[3, 7]$, $c[4, 5]$, $c[8, 7]$.

III.4.37 Considere o autómato finito determinista para o padrão $P = ABABCABABAB$, indique os seguintes valores de transição: $\delta(2, A)$, $\delta(3, A)$, $\delta(4, A)$, $\delta(5, B)$, $\delta(9, C)$.

III.4.38 Considere o algoritmo de Knuth-Morris-Pratt. Dado o padrão $P = \texttt{aabaaabaab}$, calcule a função de prefixo $\pi[k]$ para o padrão P. Indique os valores de $\pi[4]$, $\pi[6]$, $\pi[8]$, $\pi[9]$ e $\pi[10]$.

III.4.39 Considere o autómato finito determinista para o padrão $P = aabaaabaab$, indique os seguintes valores de transição: $\delta(2, a)$, $\delta(5, b)$, $\delta(6, a)$, $\delta(9, a)$, $\delta(10, a)$.

Parte IV.
Classes Complexidade & Algoritmos de Aproximação

IV.1. Classes Complexidade

IV.1.1 Para cada uma das afirmações seguintes indique se é verdadeira (**V**), se é falsa (**F**) ou se não se sabe (**D**).

a. $NPC \subseteq NP$.

b. Existe $X \in P$ tal que $X \not\leq_p Y$, qualquer que seja $Y \in NPC$.

c. $NPC \cap P = \emptyset$.

d. Se SAT for resolúvel em tempo polinomial, então $P = NP$.

e. $P \neq NP \cap coNP$.

IV.1.2 Para cada uma das afirmações seguintes indique se é verdadeira (**V**), se é falsa (**F**) ou se não se sabe (**D**).

a. Dado um problema $X \in NP$, se $Y \leq_P X$ para qualquer $Y \in NP$, então $X \in NPC$.

b. Se existe $X \in NP \cap coNP$ tal que $X \notin P$, então $P \neq NP$.

c. Dado um problema X, se existe $Y \in NPC$ tal que $Y \leq_P X$, então $X \in$ *NP-Difícil*.

d. Se $X \in NPC$ e $Y \in P$, então $X \leq_P Y$.

e. Se $X \in NP$, é possível verificar respostas para instâncias positivas de X em tempo polinomial no tamanho da instância.

f. Basta que exista um algoritmo com complexidade polinomial que resolva $X \in NP$ para que $NPC \subseteq P$.

IV.1.3 Responda a cada uma das seguintes questões, justificando a sua resposta.

a. Suponha que sabemos que o problema X é NP-completo e que descobrimos que existe um algoritmo que resolve qualquer instância de X em tempo polinomial no tamanho da mesma. Este facto implica que o problema SAT pode ser resolvido em tempo polinomail?

b. Suponha que esse algoritmo tem complexidade $O(n^3)$, em que n denota o tamanho das instâncias de X. Neste caso, qualquer problema Y na classe de problemas NP pode ser resolvido em tempo $O(m^3)$, em que m denota o tamanho das instâncias de Y?

c. Dado um inteiro $n > 1$, o problema da factorização consiste em descobrir se existem dois números inteiros p e q tais que $n = p \times q$. Considere o seguinte algoritmo para o problema da factorização:

```
PRIME(n)
1  for k ← 2 to √n
2        do if k divides n
3              then return false
4  return true
```

Este é um algoritmo com tempo polinomial para o problema da factorização?

IV.1.4 Indique para cada uma das seguintes afirmações se é verdadeira (V), se é falsa (F) ou se não se sabe (D).

a. As instâncias de programação linear cujas variáveis só podem tomar valores inteiros são resolúveis em tempo polinomial.

b. NP = NPC.

c. Existem subproblemas do problema SAT, ou seja, conjuntos de instâncias de SAT, que são resolúveis em tempo polinomial.

d. 2-CNFSAT $\leq_p$ 3-CNFSAT.

e. Dado $X \in$ NPC, tem-se que $X \leq_p Y$ para qualquer $Y \in$ NP-difícil.

f. Basta que exista um algoritmo com complexidade polinomial que resolva um problema $X \in NP$ para que $P = NP$.

IV.1.5 Indique para cada uma das seguintes afirmações se é verdadeira (V), se é falsa (F) ou se não se sabe (D).

a. Se existem $X \in$ NPC e $Y \in$ NP tais que $X \leq_p Y$, então $Y \in$ NP-difícil.

b. P = NP.

c. É possível verificar em tempo polinomial respostas para instâncias positivas de qualquer $X \in$ NPC.

d. P = co-NP $\cap$ NP.

e. Dado $X \in$ NP-difícil, qualquer $Y \in$ NPC verifica $X \leq_p Y$.

f. Dados dois problemas X e Y tais que $X \leq_p Y$ e $Y \in$ P, então $X \in$ P.

IV.1.6 Indique para cada uma das seguintes afirmações se é verdadeira (V), se é falsa (F) ou se não se sabe (D).

a. NP $\cup$ coNP $= \emptyset$.

b. NP $-$ P $\neq \emptyset$.

c. Assuma P $\neq$ NP, $X \in$ NP, tal que $\forall_{Z \in \text{NP}}, Z \leq_p X$. Então não existe algoritmo the tempo polinomial para X.

d. Assuma P $\neq$ NP. Então P = coNP.

IV.1.7 Para cada uma das afirmações seguintes, indique se é verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**).

a. NP $\cap$ co-NP $= \emptyset$

b. P $\cap$ NP-Completo $= \emptyset$

c. P = NP

d. NP-Completo $\subseteq$ NP

e. Dado $X \in$ NP-Difícil, qualquer $Y \in$ NP-Completo verifica $Y \leq_p X$

IV.1.8 Para cada uma das afirmações seguintes, indique se é verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**).

a. P $\subseteq$ NP

b. NP-Completo = NP

c. O problema de programação linear (PL) pertence a P.

d. O algoritmo Simplex usado para resolver programação linear (PL) é exponencial no pior caso.

e. P $\cap$ NP = $\emptyset$

IV.1.9 Suponha que o Prof. Martelo provou que não existe nenhum algoritmo polinomial para resolver o problema CNF-SAT. Considerando esta descoberta do Prof. Martelo, para cada uma das afirmações seguintes, indique se é verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**)

a. P $\cap$ NP-Completo = $\emptyset$

b. P = NP

c. NP $\cap$ Co-NP = $\emptyset$

d. P $\subseteq$ NP $\cap$ Co-NP

e. NP-Completo = NP

IV.1.10 Suponha que o Prof. Martelo descobriu um algoritmo polinomial para resolver o problema CNF-SAT. Considerando esta descoberta do Prof. Martelo, para cada uma das afirmações seguintes, indique se é verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**)

a. P = NP

b. NP $\cap$ Co-NP = $\emptyset$

c. P = NP $\cap$ Co-NP

d. P $\cap$ NP-Completo = $\emptyset$

e. NP-Completo = NP

IV.1.11 O problema 2CNF-SAT consiste em verificar se uma fórmula proposicional na forma normal conjuntiva (CNF) com n variáveis e m cláusulas onde todas as cláusulas têm dois literais pode ser satisfeita. Ou seja, se existe uma atribuição de verdade às n variáveis da fórmula que a tornem verdadeira.

Por exemplo, a fórmula $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_3)$ é uma instância de 2CNF-SAT cuja solução será atribuir x_1 e x_3 a falso e x_2 a verdadeiro.

Indique se 2CNF-SAT $\in P$. Se considerar que 2CNF-SAT $\in P$, indique um algoritmo polinomial que determine se uma fórmula 2CNF-SAT pode ser satisfeita. Caso contrário, prove que 2CNF-SAT é NP-Completo.

IV.1.12 Suponha que o Prof. Paulo descobriu um algoritmo polinomial para resolver o problema 2CNF-SAT. Considerando esta descoberta do Prof. Paulo, para cada uma das afirmações seguintes, indique se é verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**).

a. SAT $\in$ NP
b. NP $\cap$ P $= \emptyset$
c. 2CNF-SAT $\in$ NP
d. SubSet-Sum $\in$ P
e. CLIQUE $\leq_P$ VERTEX-COVER

IV.1.13 Classifique as seguintes afirmações como verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**)

a. P $\subseteq$ NP
b. P $\cap$ NP $= \emptyset$
c. 3CNFSAT $\leq_{\mathrm{P}}$ 3COLOR
d. CNFSAT $\leq_{\mathrm{P}}$ VERTEX $-$ COVER
e. NP $\subseteq$ P

IV.1.14 Para cada uma das afirmações seguintes, indique se é verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**).

a. P $=$ NP
b. P $\subseteq$ NP
c. P $\cap$ NP-Completo $= \emptyset$
d. NP-Completo $\subseteq$ NP
e. NP $\cap$ Co-NP $= \emptyset$

IV.1.15 Para cada uma das afirmações seguintes, indique se é verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**).

a. Se existir um algoritmo polinomial que resolve o problema da mochila não-fraccionário, então P $=$ NP.
b. Dado $X \in$ NP-Difícil, qualquer $Y \in$ NP-Completo verifica $Y \leq_p X$
c. Dado $X \in$ NP-Difícil, qualquer $Y \in$ P verifica $Y \leq_p X$
d. O problema de programação linear não pode ser resolvido em tempo polinomial.
e. NP-Completo $\subseteq$ NP

IV.1.16 Suponha que o Prof. Carlos descobriu um algoritmo polinomial para resolver o problema 2-COLORING. Considerando esta descoberta do Prof. Carlos, para cada uma das afirmações seguintes, indique se é verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**).

a. 2-COLORING $\in$ NP

b. P = NP

c. PARTITION $\in$ NP-HARD

d. HORN-SAT $\notin$ P

e. 3-CNFSAT $\leq_P$ 3-COLORING

IV.1.17 Suponha que o Prof. Caracol descobriu um algoritmo polinomial para resolver o problema 3-COLORING. Considerando esta descoberta do Prof. Caracol, para cada uma das afirmações seguintes, indique se é verdadeira (**V**), falsa (**F**) ou se não se sabe (**D**).

a. 3-COLORING $\in$ NP

b. P = NP

c. CLIQUE $\notin$ NP-HARD

d. 2-COLORING $\in$ P

e. 3-CNFSAT $\notin$ NP-HARD

IV.2. Redução de Problemas e Algoritmos de Aproximação

IV.2.1 Dado um grafo não dirigido $G = (V, E)$, uma clique é um sub-grafo completo de G, ou seja, um sub-grafo de G em que todos os vértices são adjacentes uns dos outros. O problema *CLIQUE* consiste em decidir se existe uma clique de G com k vértices.

Dado um grafo não dirigido $G = (V, E)$, um conjunto independente de vértices (*ISV*), é qualquer conjunto de vértices V' que não contenha vértices adjacentes. O problema *ISV* consiste em decidir se existe um conjunto independente de vértices de G com k vértices.

Sabendo que o problema *CLIQUE* é NP-completo, prove que o problema *ISV* é NP-completo, utilizando uma redução a partir do problema *CLIQUE*.

IV.2.2 O problema de optimização *MIN-MAKESPAN-SCHEDULING* é NP-Difícil e consiste em: dadas n tarefas para realizar em m máquinas, em que cada tarefa i tem duração d_i, efectuar o agendamento de cada tarefa por forma a que o instante em que todas a máquinas já terminaram as respectivas tarefas ocorra o mais cedo possível. Descreva um algoritmo capaz de obter uma solução aproximada para o problema *MIN-MAKESPAN-SCHEDULING* em tempo polinomial. Caracterize a qualidade da solução aproximada indicando, justificadamente, o respectivo limite da razão.

IV.2.3 Dado um conjunto de números inteiros S, o problema *SET-PARTITION* consiste em verificar se S pode ser dividido em dois conjuntos B e $\overline{B} = S \setminus B$ tal que $\sum_{x \in B} x = \sum_{x \in \overline{B}} x$.

Dado um conjunto de n objectos, com volumes $v_1, \ldots, v_n$, onde $0 < v_i \leq 1$, e um número inteiro k, o problema *BIN-PACKING* consiste em verificar se é possível guardar os n objectos em k contentores de capacidade unitária. Cada contentor pode guardar qualquer conjunto de objectos cuja soma dos respectivos volumes não exceda 1.

Sabendo que o problema *SET-PARTITION* é NP-completo, prove que o problema *BIN-PACKING* é NP-completo, utilizando uma redução a partir do problema *SET-PARTITION*.

IV.2.4 O problema *BIN-PACKING*$_{opt}$ é um problema de optimização relacionado com o problema *BIN-PACKING*, que consiste em determinar o número mínimo de contentores de capacidade unitária que são necessários para guardar os n objectos. Indique um algoritmo capaz de obter uma solução aproximada para o problema *BIN-PACKING*$_{opt}$ em tempo polinomial. Caracterize a qualidade da solução aproximada indicando, justificadamente, o limite da razão $\rho(n)$.

IV.2.5 Considere a variante do problema de determinar o fluxo máximo num grafo digido e pesado $G = (V, E)$ estudado nas aulas mas em que queremos encontrar o fluxo máximo tal que, para qualquer arco $(u, v) \in E$, temos ou $f(u, v) = 0$ ou $f(u, v) = c(u, v)$ (Nota: no problema original apenas exigimos que $f(u, v) \leq c(u, v)$ para qualquer arco $(u, v) \in E$). Formule o problema de decisão associado a este problema de optimização e mostre que o mesmo é NP-completo. Sugestão: tenha em conta que o problema SUBSET-SUM é NP-completo.

IV.2.6 Considere a restrição de fórmulas CNF a fórmulas CHorn. Uma cláusula diz-se CHorn se contiver não mais do que um literal complementado. Uma fórmula diz-se CHorn se todas as suas cláusulas forem CHorn. O problema de decisão CHornSAT consiste em decidir a satisfação de fórmulas CHorn e pode ser resolvido eficientemente. Nestas condições proponha um algoritmo eficiente para o problema CHornSAT. Qual é a complexidade do algoritmo?

IV.2.7 Considere o problema AL2LC que, dada uma fórmula proposicional em forma normal conjuntiva (CNF), com 3 literals por cláusula, decide se existe uma atribuição de valores que satisfaz pelo menos dois literais por cláusula. Indique se o problema AL2LC é NP-Completo. Em caso afirmativo, inclua uma demonstração de que AL2LC é NP-Completo. Caso contrário, proponha caso exista um algoritmo polinomial para o problema.

IV.2.8 Conside o problema de decisão `Grafo-Comum-Máximo` (GCM): dados dois grafos $G_1 = (V_1, E_2)$ e $G_2 = (V_2, E_2)$, e um valor b, decidir se existem $V_1' \subseteq V_1$ e $V_2' \subseteq V_2$ cuja remoção deixa pelo menos b vértices em cada grafo, e torna os dois grafos idênticos. Prove que o problema de decisão GCM é NP-completo.

(Sugestão: considere utilizar um dos problemas estudados em ASA.)

IV.2.9 Considere o problema de decisão *3MaxSAT*: dada uma fórmula na forma normal conjuntiva, com 3 literais por cláusula, e um inteiro k, existe uma atribuição de valores que satisfaz pelo menos k cláusulas? Prove que este problema de decisão é NP-completo.

(Sugestão: este problema pode ser visto como uma generalização de um dos problemas de decisão estudados em ASA.)

IV.2.10 Considere o problema de decisão P-CNFSAT, das fórmulas proposicionais em forma normal conjuntiva (CNF) em que as cláusulas apenas têm literais positivos. Indique se o problema P-CNFSAT está em NP. Indique se o problema P-CNFSAT está em P ou se é NP-completo. Se o problema estiver em P apresente um algoritmo, caso contrário prove que o problema é NP-completo. Justifique todas as respostas apresentadas.

IV.2.11 Considere o problema de decisão N-CNFSAT, das fórmulas proposicionais em forma normal conjuntiva (CNF) em que as cláusulas apenas têm literais negativos. Indique se o problema N-CNFSAT está em NP. Indique se o problema N-CNFSAT está em P ou se é NP-completo. Se o problema estiver em P apresente um algoritmo, caso contrário prove que o problema é NP-completo. Justifique todas as respostas apresentadas.

IV.2.12 Considere o problema de decisão PN-CNFSAT, das fórmulas proposicionais em forma normal conjuntiva (CNF) em que para cada cláusula todos os literais são positivos ou todos os literais são negativos. Indique se o problema PN-CNFSAT está em NP. Indique se o problema PN-CNFSAT está em P ou se é NP-completo. Se o problema estiver em P apresente um algoritmo, caso contrário prove que o problema é NP-completo. Justifique as respostas apresentadas.

IV.2.13 Dada uma matriz A de $m \times n$ valores inteiros e um vector de inteiros b de dimensão m, o problema de Programação Linear Inteira 0-1 (ILP 0-1) consiste em verificar se existe um vector x de dimensão n tal que os elementos de x pertencem a $\{0, 1\}$ e $Ax \leq b$.

Dado um conjunto S de n objectos onde cada item i ($1 \leq i \leq n$) tem um valor v_i e um peso w_i, o problema 01_KNAPSACK consiste em verificar se existe um subconjunto S' tal que $S' \subseteq S$, a soma dos valores dos objectos de S' é maior ou igual a K e a soma dos pesos dos objectos de S' é menor ou igual a W.

Sabendo que o problema 01_KNAPSACK é NP-Completo, prove que o problema ILP 0-1 é NP-Completo usando uma redução a partir de 01_KNAPSACK.

(Nota: Prove primeiro que ILP 0-1 $\in$ NP.)

IV.2.14 Dada uma sequência S de n objectos onde cada item i ($1 \leq i \leq n$) tem um valor v_i e um peso w_i, o problema 01_KNAPSACK consiste em verificar se existe uma subsequência S' tal que $S' \subseteq S$, a soma dos valores dos objectos de S' é maior ou igual a K ($\sum_{i \in S'} v_i \geq K$) e a soma dos pesos dos objectos de S' é menor ou igual a W ($\sum_{i \in S'} w_i \leq W$).

Dado um conjunto de números inteiros S e um número inteiro t, o problema SUBSET-SUM consiste em verificar se existe um subconjunto S' tal que $S' \subseteq S$ e a soma dos elementos de S' seja t.

Sabendo que o problema SUBSET-SUM é NP-Completo, prove que o problema 01_KNAPSACK é NP-Completo usando uma redução a partir de SUBSET-SUM.

(Nota: Prove primeiro que 01_KNAPSACK $\in NP$.)

IV.2.15 Dada uma sequência S de n objectos onde cada item i ($1 \leq i \leq n$) tem um valor v_i e um peso w_i, o problema da mochila não fraccionário 01_KNAPSACK consiste em verificar se existe um subconjunto de objectos S' tal que $S' \subseteq S$, a soma dos valores dos objectos de S' é maior ou igual a K ($\sum_{i \in S'} v_i \geq K$) e a soma dos pesos dos objectos de S' é menor ou igual a W ($\sum_{i \in S'} w_i \leq W$).

Dado um conjunto de números inteiros S, o problema SET-PARTITION consiste em verificar se o conjunto S pode ser dividido em dois subconjuntos disjuntos S_1 e S_2 tal que $S_1 \cup S_2 = S$ e a soma dos elementos de S_1 é igual à soma dos elementos de S_2. Ou seja, $\sum_{x \in S_1} x = \sum_{y \in S_2} y$.

Sabendo que o problema SET-PARTITION é NP-Completo, prove que o problema da mochila não fraccionário 01_KNAPSACK é NP-Completo usando uma redução a partir de SET-PARTITION.

(Nota: Prove primeiro que 01_KNAPSACK $\in$ NP.)

IV.2.16 No problema NÃO TODOS IGUAIS 3CNF-SAT (NAE 3CNF-SAT) o input consiste numa formula, em forma normal conjuntiva, em que as cláusulas contém exactamente 3 literais (variáveis ou negações de variáveis). A solução do problema consiste em determinar uma atribuição dos valores de verdadeiro e falso para as variáveis por forma a que todas as cláusulas contenham pelo menos um literal que avalia para verdadeiro e pelo menos um literal que avalia para falso.

Indique se o NAE 3CNF-SAT $\in$ P ou se NAE 3CNF-SAT $\in$ NPC. Se considerar que $\in$ P, indique um algoritmo polinomial que determina se uma formula pode ser satisfeita. Caso contrário, prove que o NAE 3CNF-SAT é NP-Completo. Sugestão: Recorde que 3CNF-SAT é NP Completo e assuma que uma instância de NAE 3CNF-SAT pode utilizar a constante F, que avalia sempre para falso.

IV.2.17 Nesta questão consideramos o problema do conjunto certeiro (HITTING-SET). Uma instância consiste num conjunto S, numa família C de sub-conjuntos de S e num inteiro k.

Um exemplo seria $S = \{1, 2, 3, 4, 5, 6\}, C = \{\{1, 2, 3, 6\}, \{3, 4, 5\}, \{4, 5, 6\}, \{5, 6\}, \{6\}\}$ e $k = 2$.

O problema consiste em determinar um conjunto certeiro X, tal que $X \subseteq S$, com tamanho k, i.e., $|X| = k$. Um conjunto certeiro tem de acertar em todos os elementos c da família, mais precisamente $c \cap X \neq \emptyset$ para todos os conjuntos $c \in C$. Neste exemplo podiamos ter $X = \{3, 6\}$. Caso a instância tivesse $k = 1$ então não tinha solução.

Indique se HITTING-SET $\in$ P ou se HITTING-SET $\in$ NP-Completo. Se considerar que HITTING-SET $\in$ P indique um algoritmo polinomial que calcula X. Caso contrário prove que HITTING-SET $\in$ NP-Completo.

Recorde que o problema VERTEX-COVER $\in$ NP-Completo. Uma instância deste problema consiste num grafo não dirigido e num inteiro k. Uma solução consiste num conjunto A de vértices, tal que para qualquer arco (u, v) do grafo temos que $u \in A$ ou $v \in A$.

IV.2.18 Dado um conjunto de n objectos onde cada objecto i ($1 \leq i \leq n$) tem um peso p_i e um conjunto de m caixas de capacidade K, o problema BIN-PACKING consiste em verificar se existe uma forma de colocar todos os n objectos nas m caixas tal que o peso total dos objectos em cada caixa não seja superior a K. Note que os objectos não podem ser fraccionados.

Dada uma matriz A de $m \times n$ valores inteiros e um vector de inteiros b de dimensão m, o problema de Programação Linear Inteira 0-1 (ILP 0-1) consiste em verificar se existe um vector x de dimensão n tal que os elementos de x pertencem a $\{0, 1\}$ e $Ax \leq b$.

Sabendo que o problema BIN-PACKING é NP-Completo, prove que o problema ILP 0-1 é NP-Completo usando uma redução a partir de BIN-PACKING.

(Nota: Prove primeiro que ILP 0-1 $\in$ NP.)

IV.2.19 Dada uma matriz A de $m \times n$ valores inteiros e um vector de inteiros b de dimensão m, o problema de Programação Linear Inteira 0-1 (ILP 0-1) consiste em verificar se existe um vector x de dimensão n tal que os elementos de x pertencem a $\{0, 1\}$ e $Ax \leq b$.

Dado um grafo não dirigido $G = (V, E)$ e um conjunto de K cores (onde $K \geq 3$), o problema K-COLORING consiste em verificar se é possível colorir os vértices do grafo usando apenas K cores tal que vértices adjacentes não podem ter a mesma cor (se o arco (u, v) existir no grafo, então os vértices u e v têm obrigatoriamente cores diferentes).

Sabendo que o problema K-COLORING é NP-Completo, prove que o problema ILP 0-1 é NP-Completo usando uma redução a partir de K-COLORING.

(Nota: Prove primeiro que ILP 0-1 $\in$ NP.)

IV.2.20 O problema da divisão de soma quadrada limitada **DSQL** consiste em dividir um conjunto A de valores inteiros positivos em K subconjuntos disjuntos $A_i \subseteq A$ por forma a que a soma dos quadrados das somas dos elementos de cada A_i seja menor do que um valor J dado. As seguintes propriedades definem matematicamente o DSQL:

$$0 \leq i \neq j \leq K \Rightarrow A_i \cap A_j = \emptyset$$
$$A = \bigcup_{i=1}^{K} A_i$$
$$\sum_{i=1}^{K} \left(\sum_{e \in A_i} e \right)^2 \leq J$$

A instância com $A = \{1, 2, 4\}$, $K = 2$ e $J = 26$ do **DSQL** tem solução com $A_1 = \{1, 2\}$ e $A_2 = \{4\}$, dado que $(1 + 2)^2 + 4^2 = 9 + 16 = 25 \leq 26$. Esta divisão obtem a soma mínima, pelo que para $J = 24.5$ a instância não tem solução.

Recorde o problema **PARTITION** que dado um conjunto S de inteiros determina se existe uma partição de S por forma que o total de ambos os subconjuntos sejam iguais, i.e., um subconjunto $S' \subseteq S$ tal que $\sum_{e \in S'} e = \sum_{e \in S \setminus S'} e$.

Sabendo que o problema **PARTITION** é NP-Completo, prove que o problema **DSQL** é NP-Completo. Prove primeiro que **DSQL** $\in$ NP. Para a prova que **DSQL** $\in$ NP-Hard pode precisar da função $f(x) = x^2 + (t - x)^2$, para alguma constante t. Esta função atinge um mínimo global em $t/2$, pelo que $f(x) \leq t^2/2$ para qualquer valor de x e se $f(x_0) = t^2/2$ então $x_0 = t/2$.

IV.2.21 O problema **2-CLIQUE** de um determinado grafo não dirigido G_2 com um conjunto vértices V e arcos E consiste em encontrar dois cliques C_1 e C_2 disjuntos e possivelmente vazios que no total contenham pelo menos k_2 vértices. As seguintes propriedades definem matematicamente o **2-CLIQUE**:

$$C_1 \subseteq V$$
$$C_1 \times C_1 \subseteq E$$
$$C_2 \subseteq V$$
$$C_2 \times C_2 \subseteq E$$
$$C_1 \cap C_2 = \emptyset$$
$$|C_1 \cup C_2| \geq k_2$$

Recorde o problema **CLIQUE** que encontra um clique num grafo não dirigido G_1. Sabendo que o problema **CLIQUE** é NP-Completo, prove que o problema **2-CLIQUE** é NP-Completo. Prove primeiro que **2-CLIQUE** $\in$ NP.

Parte I.
Revisão de Introdução aos Algoritmos e Estruturas de Dados

I.1. Notação assimptótica - Recorrências

I.1.1
$T(n) = \Theta(n^{\log_2 3})$. Caso 1 do teorema Mestre.

I.1.2 $O(n)$

I.1.3 A recorrência neste caso é dada por $T(n) = 2\,T(n-1)+O(1)$, com $T(n) = \Theta(2^n)$.

I.1.4 Pelo primeiro caso do Teorema Mestre, $\mathcal{O}(n^{\log_3 4})$.

I.1.5 A recorrência neste caso é dada por $T(n) = 2\,T(\frac{n}{2}) + O(1)$. Pelo primeiro caso do Teorema Mestre, tem-se que $T(n) = \Theta(n)$.

I.1.6
Recorrência: $T(n) \leq T(\frac{n}{2}) + 1$
Complexidade: $O(\log(n))$

I.1.7
Pelo segundo caso do Teorema Mestre, temos que $T(n) = O(n^3 \log n)$.

I.1.8 $T(n) = 2T(n/2)+O(1)$ e, pelo primeiro caso do Teorema Mestre, $T(n) \in O(n)$.

I.1.9

Expressão	$T(n/9) + \sqrt{n}$
Majorante	$O(\sqrt{n})$

I.1.10
$T(n) = T(n/4)+O(\sqrt{n})$. Esta expressão satisfaz a condições do caso III do Teorema mestre, visto que $f(n) = \sqrt{n} = \Omega(n^{0+1/2}) = \Omega(n^{(\log_4 1)+\epsilon})$ com $\epsilon = 1/2$ e ainda $\sqrt{n/4} = \sqrt{n}/2 < c\sqrt{n}$, com $c = 0.6 < 1$. Portanto $T(n) = O(\sqrt{n})$.

I.1.11

Expressão	$2 * T(n/4) + n$
Majorante	$O(n)$

I.1.12

Expressão:	$2 * T(n/2) + n$
Majorante:	$O(n \log n)$

I.1.13

Expressão	$3 \times T(n/4) + \sqrt{n}$
Majorante	$O(n^{\log_4(3)})$

I.1.14

Expressão	$4 \times T(n/8) + n$
Majorante	$O(n)$

I.1.15

Expressão	$T(n) = 2T(n/2) + 3\log_2 n$
Majorante	$O(n^1)$

I.1.16

Expressão	$T(n) = T(n/4) + n/4$
Majorante	$O(n)$

I.1.17

Expressão	$T(n) = 4 * T(n/2) + O(n^2)$
Majorante	$O(n^2\ lg\ n)$

I.1.18

Expressão	$T(n) = 2 * T(n/2) + O(1)$
Majorante	$O(n)$

I.1.19

Expressão	$T(n) = 3 * T(n/3) + O(\sqrt{n})$
Majorante	$O(n)$

I.1.20

Expressão	$T(n) = 2 * T(n/2) + O(lg(n))$
Majorante	$O(n)$

I.2. Notação assimptótica - Algoritmos

I.2.1

a)	b)	c)	d)	e)
V	F	F	V	F

I.2.2

a	$\subset$	c	$\subset$	f	$\subset$	b	$=$	e	$\subset$	d

I.2.3 $O(n \log n + m)$ e $O(n + m)$, respectivamente.

I.2.4

a.	b.	c.	d.	e.	f.
$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$

I.2.5

A menor complexidade é $O(V)$ em ambos os casos.

I.2.6

Edmonds-Karp: $O(V + E)$
Relabel-To-Front: $O(V)$

I.2.7 $O(V^3)$ e $O(V(V + E))$, respectivamente.

I.2.8

DFS	$O(V^2)$
$Ciclo$	$O(V)$
$SSSP$+DAG	$O(V + E)$

I.2.9

BFS	$O(V^2)$
$Kruskal$	$O(E\alpha(V))$
$APSP$+DAG	$O(V(V + E))$

I.2.10

Tarjan	$O(V^2)$
Bellman-Ford	$O(V^3)$
Ford-Fulkerson	$O(EV)$

I.2.11

Dijkstra com pesos entre 1 e 3	$O(E)$
Prim com pesos entre 1 e 3	$O(E)$
Prim com matriz de adjacência	$O(V^2 + E\ lg\ V)$

I.3. Estruturas de dados

I.3.1

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

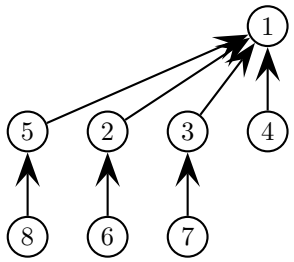
I.3.2

Podemos utilizar uma estrutura de conjuntos disjuntos. Existe um caminho entre os vértices u e v se e só se u e v estão no mesmo componente fortemente ligado. Logo a estrutura guarda 1 conjunto por cada componente fortemente ligado. Portanto dado um par de vertices u e v existe um caminho entre eles se e só se `Find-Set(`u`) = Find-Set(`v`)`.

Quando um arco (v, v') é adicionado e o par (v, v') não faz parte do fecho transitivo de G então fazemos a `Union(`u`,`v`)`, caso contrário nada é feito.

A complexidade resultante depende da estrutura de conjuntos disjuntos utilizada. Caso seja a estrutura baseada em árvores o custo de n operações num grafo com $|V|$ vertices o tempo total é $O(n\alpha|V|)$, onde α é o inverso da função de Ackermann.

I.3.3



I.3.4

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
$rank[x_i]$	1	0	1	0	2	0	3	0	1	0
$p[x_i]$	x_7	x_7	x_7	x_3	x_7	x_5	x_7	x_7	x_5	x_5

I.3.5

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
$rank[x_i]$	1	0	3	0	1	0	1	0	2	0
$p[x_i]$	x_9	x_9	x_3	x_3	x_3	x_5	x_3	x_3	x_3	x_9

Parte II.
Algoritmos em Grafos

II.1. Algoritmos Elementares em Grafos - BFS e DFS

II.1.1

	A	B	C	D	E	F	G	H
d	1	2	2	0	3	3	1	1
π	D	A	A	$-$	B	B, C	D	D

II.1.2

	A	B	C	D	E	F
d	0	1	2	2	1	3
π	-	A	B	E	A	D

II.1.3

	A	B	C	D	E	F	G	H
d	0	1	2	1	2	3	2	3
π	$-$	A	B	A	B, D	C, E	D	E, G

II.1.4

	A	B	C	D	E	F
d	0	1	1	1	3	2
π	-	A	A	A	F	D

II.1.5

	A	B	C	D	E
d	2	2	1	0	1
π	E	C	D	-	D

II.1.6

	A	B	C	D	E	F
d	2	2	1	1	1	0
π	E	E	F	F	F	NIL

II.1.7

	A	B	C	D	E	F
d	1	1	2	0	1	2
π	D	D	A	NIL	D	E

II.1.8

	A	B	C	D	E	F	G	H
d	1	3	5	2	15	7	8	9
f	14	4	6	13	16	12	11	10

II.1.9

a.	b.	c.	d.	e.
V	F	V	V	F

II.1.10

	A	B	C	D	E	F	G	H
d	1	2	3	4	7	6	9	15
f	14	13	12	5	8	11	10	16

II.1.11

	A	B	C	D	E	F	G	H
d	2	6	10	3	1	14	7	9
f	5	13	11	4	16	15	8	12

Ordem: E, F, B, H, C, G, A, D

II.1.12

	A	B	C	D	E
d	1	2	4	6	3
f	10	9	5	7	8

II.2. Ordenações Topológicas

II.2.1

A	B	C	D	E	F	G	H
A	C	B	E	D	G	F	H

II.2.2

v	A	B	C	D	E	F	G	H
$f[v]$	10	7	9	16	4	6	15	14
$ord.t.$	4	6	5	1	8	7	2	3

II.2.3

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
d	1	2	4	11	5	6	13
f	10	3	9	12	8	7	14
O.T.	3^o	7^o	4^o	2^o	5^o	6^o	1^o

II.2.4

	A	B	C	D	E
d	1	2	4	5	6
f	10	3	9	8	7
O.T.	1^o	5^o	2^o	3^o	4^o

II.2.5

Número de ordenações:	10
Ordenação:	F, E, A, B, C, D

II.2.6

	A	B	C	D	E	F
d	1	2	7	3	9	11
f	6	5	8	4	10	12
Ordenação:			F,E,C,A,B,D			

II.2.7

Ord. Topológica	E, G, F, A, B, D, C

	A	B	C	D	E	F	G
d/f	1 / 8	2 / 7	4 / 5	3 / 6	9 / 14	10 / 11	12 / 13

II.3. Componentes Fortemente Ligados

II.3.1

5, 7, 8, 10

II.3.2 Existem várias soluções possíveis, dependendo da forma como é efectuada a travessia em profundidade primeiro (DFS). Usando uma ordem lexicográfica e considerando que o primeiro tempo é 0, então os tempos de descoberta são os seguintes: $\{A, B, C\} : 0$, $\{E, F\} : 3$ e $\{D, G, H\} : 5$

II.3.3

	a	b	c	d	e	f	g	h	i
d	0	1	2	3	4	5	7	6	8
l	0	1	1	1	1	5	5	5	1

II.3.4 G tem 3 SCCs. $G^{SCC} = a \rightarrow bcdefgh \leftarrow i$.

II.3.5

	A	B	C	D	E	F	G	H
d	0	5	7	1	2	3	4	6
low	0	5	4	0	0	1	4	4

SSCs: $\{A, D, E, F\}$, $\{C, G, H\}$, e $\{B\}$

II.3.6

	A	B	C	D	E	F
d	1	2	5	7	6	8
low	1	2	3	1	1	1
$componentes$: 3						

II.3.7

	A	B	C	D	E	F	G	H	I	J
d	3	2	4	1	0	5	6	8	9	7
low	1	0	4	0	0	4	4	8	8	7
$componentes$	$(A, B, D, E); (C, F, G); (H, I); (J)$									

II.3.8

Componentes Fortemente Ligados	$\{A, B, C, D, I, G\}, \{E\}, \{F, H\}$

	A	B	C	D	E	F	G	H	I
d	4	3	2	1	6	8	7	9	5
low	3	2	1	1	6	8	2	8	2

II.3.9

	A	B	C	D	E	F	G	H
d	1	4	2	3	5	6	7	0
low	0	4	0	1	4	4	7	0

II.3.10

	A	B	C	D	E	F	G	H
d/f	1 / 8	15 / 16	10 / 13	3 / 6	2 / 7	4 / 5	9 / 14	11 / 12

$SSCs$:	$\{A, D, E, F\}$	$\{C, G, H\}$	$\{B\}$		

II.3.11

	A	B	C	D	E	F	G	H	I	J
d/low	1 / 1	6 / 6	8 / 5	2 / 1	3 / 1	4 / 2	5 / 5	7 / 5	9 / 7	10 / 7

$SSCs$:	$\{B\}$	$\{C, G, H, I, J\}$	$\{A, D, E, F\}$	

II.4. Caminhos mais Curtos Fonte Única

II.4.1

v	A	B	C	D	E	F
$d[v]$	0	2	8	11	11	13
$\pi[v]$	nil	A	B	B	C	D

II.4.2

	v_1	v_2	v_3	v_4	v_5
π	NIL	v_1	v_5	v_2	v_4

Iterações:
1 + 1 p/ verificação

II.4.3

Assumindo que existem n vértices e que estes estão identificados de 0 a $n-1$, basta utilizar um vector d em que $d[i]$ é o valor actual para o caminho com menor custo da fonte até ao vértice i. Neste caso a actualização ocorre em tempo constante, mas a pesquisa do mínimo sobre o vector d não ordenado ocorre em tempo linear $O(V)$. Dado que cada vértice é extraído da fila uma e uma só vez, temos que o algoritmo corre neste caso em tempo $O(V^2)$.

II.4.4

	A	B	C	D	E
π	NIL	D	B	A	B
d	0	5	10	2	9
Ordem:	1º	3º	5º	2º	4º

II.4.5

Fast-BF(G, w, s)

```
 1  Initialize-Single-Source(G, s)
 2  c ← TRUE
 3  i ← 1
 4  while c and i < |V[G]|
 5        do c ← FALSE
 6           i ← i + 1
 7           for each (u, v) ∈ E[G]
 8                 do if d[v] > d[u] + w(u, v)
 9                       then Relax(u, v, w)
10                            c ← TRUE
11  return c
```

O valor m consiste no tamanho (em numero de arcos) do maior caminho de menor custo entre um par de vértices u e v.

II.4.6

Vértices		A	D	G	B	H	C	I	F
d[E]	$+\infty$	70	60	50	40	35	30	20	9

II.4.7

w(B,C) =	-3

II.5. Caminhos mais Curtos Entre Todos os Pares

II.5.1 No fecho transitivo $G' = (V, E')$, E' é tal que $(u, v) \in E'$ se e só se existe um caminho de u para v em G. Para determinar o fecho transitivo de G em tempo $O(|V|^3)$ com o algoritmo de Floyd-Warshall podemos atribuir peso 1 a cada arco em G e, depois de executar o algoritmo, verificar se $D_{uv} < |V|$. Nesse caso existe um caminho de u para v e, portanto, adicionamos (u, v) a E'. Caso contrário $D_{uv} = \infty$ e, portanto, não existe caminho de u para v.

II.5.2

Bellman-Ford (# iter.)	1			
v	A	B	C	D
Dijkstra (ordem de extr.)	4	1	3	2
$\hat{\delta}(B, v)$	∞	0	1	1

II.5.3

$\Pi^3 =$

NIL	1	2	1
3	NIL	2	1
3	1	NIL	1
3	4	4	NIL

II.5.4

$\hat{w}(v_1, v_2)$	$\hat{w}(v_1, v_3)$	$\hat{w}(v_3, v_2)$	$\hat{w}(v_2, v_3)$	$\hat{w}(v_2, v_5)$	$\hat{w}(v_2, v_4)$	$\hat{w}(v_5, v_3)$	$\hat{w}(v_4, v_5)$	$\hat{w}(v_5, v_4)$
8	18	2	3	14	0	0	0	5

II.5.5

$$D^0 = \begin{pmatrix} 0 & 2 & 8 & \infty & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & 0 & 2 \\ \infty & 1 & \infty & \infty & 0 \end{pmatrix}$$

$$D^1 = \begin{pmatrix} 0 & 2 & 8 & \infty & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & 0 & 2 \\ \infty & 1 & \infty & \infty & 0 \end{pmatrix}$$

$$D^2 = \begin{pmatrix} 0 & 2 & 4 & 3 & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & 0 & 2 \\ \infty & 1 & 3 & 2 & 0 \end{pmatrix}$$

$$D^3 = \begin{pmatrix} 0 & 2 & 4 & 3 & 7 \\ \infty & 0 & 2 & 1 & 5 \\ \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & 0 & 2 \\ \infty & 1 & 3 & 2 & 0 \end{pmatrix}$$

II.5.6

O valor ou custo do caminho mais curto de v_1 para v_3 após a repesagem é 0.

II.5.7

V	F	V	V	F	F

II.5.8

1 **if** $D_{ij}^{(m)} > D_{ik}^{(m)} + D_{kj}^{(m)}$
2 **then** $D_{ij}^{(2m)} = D_{ik}^{(m)} + D_{kj}^{(m)}$
3 $\Pi_{ij}^{(2m)} = \Pi_{kj}^{(m)}$
4 **else** $D_{ij}^{(2m)} = D_{ij}^{(m)}$
5 $\Pi_{ij}^{(2m)} = \Pi_{ij}^{(m)}$

II.5.9

	$d^{(4)}(1,6)$	$d^{(5)}(1,6)$	$d^{(7)}$	$d^{(8)}$	$d^{(9)}$
$d(1,6)$	$+\infty$	3	0	-1	-2

II.5.10

	A	B	C	D
h	0	-1	-3	-6

	(A,B)	(B,C)	(B,D)	(C,D)	(D,A)
$\hat{w}$	0	9	0	0	1

II.5.11

Não funciona, porque caso haja arcos com peso negativo o arcos permanecem negativos após a repesagem.

II.5.12

	A	B	C	D	E
h	-3	-4	-5	0	-6

$\hat{w}(B,E) = 0$

II.5.13

$\hat{w}(B,A) =$	2	$\hat{w}(C,F) =$	1	$\hat{w}(E,D) =$	0

II.5.14

h(A) = 0	h(B) = -5	h(C) = -3	h(D) = -7	h(E) = -3
ŵ(A,B) = 0	ŵ(A,D) = 0	ŵ(C,D) = 7	ŵ(C,E) = 6	ŵ(E,C) = 6

II.5.15

	A	B	C	D	E	F
h	-5	0	-8	-11	-14	-5

	(A,C)	(F,C)	(C,D)	(D,E)
w'	1	0	2	2

II.5.16

	$d^{(1)}$	$d^{(2)}$	$d^{(3)}$	$d^{(4)}$
$d(5,6)$	14	13	11	8

II.5.17

$d^{(2)}(1,5)$	$d^{(2)}(1,9)$	$d^{(3)}(1,5)$	$d^{(4)}(1,5)$	$d^{(5)}(1,9)$	$d^{(6)}(1,9)$
6	∞	5	3	5	2

II.5.18

	A	B	C	D	E	F	G	H	I
$h()$	0	0	0	0	-7	-8	-5	-7	-6

$\widehat{w}(B,E)$	$\widehat{w}(D,E)$	$\widehat{w}(E,G)$	$\widehat{w}(E,I)$	$\widehat{w}(H,I)$
5	9	1	1	0

II.5.19

h(A) = 0	h(B) = -1	h(C) = 0	h(D) = -1	h(E) = -2	h(F) = -3

ŵ(B,C) = 1	ŵ(B,E) = 0	ŵ(D,E) = 4	ŵ(E,F) = 0

II.5.20

	A	B	C	D	E	F	G	H
$h()$	-2	0	-2	-4	-7	-1	0	-10

$\widehat{w}(A,B)$	$\widehat{w}(B,E)$	$\widehat{w}(C,E)$	$\widehat{w}(E,G)$	$\widehat{w}(G,H)$
1	3	7	3	12

II.5.21

	A	B	C	D	E	F	G	H
$h()$	-4	0	-1	-6	-8	-2	0	-10

$\widehat{w}(A,B)$	$\widehat{w}(B,E)$	$\widehat{w}(C,E)$	$\widehat{w}(E,G)$	$\widehat{w}(G,H)$
5	6	16	1	19

II.6. Fluxo Máximo

II.6.1

v	s	A	B	C	D	t
$h[v]$	6	1	1	7	0	0
$e[v]$	-11	6	0	0	5	0

II.6.2

a.	b.	c.	d.	e.	f.
V	F	V	F	V	V

II.6.3

	(s,A)	(s,C)	(C,A)	(A,B)	(A,t)	(C,D)	(B,D)	(B,t)	(D,t)
c_f	0	6	4	7	0	0	2	0	2
	(A,s)	(C,s)	(A,C)	(B,A)	(t,A)	(D,C)	(D,B)	(t,B)	(t,D)
c_f	2	5	2	2	2	3	0	2	3
Cortes	$(\{s,A,B,C,D\},\{t\})$ e $(\{s,A,B,C\},\{D,t\})$								

II.6.4
F, F, V, V, F, V

II.6.5 Caminhos de aumento:
$s \to v_1 \to v_3 \to t$, $c_f = 2$
$s \to v_2 \to v_3 \to t$, $c_f = 1$
$s \to v_2 \to v_3 \to v_4 \to t$, $c_f = 2$
Corte mínimo: $\{s, v_1, v_2, v_3\}, \{v_4, t\}$ com capacidade 5.

II.6.6
A maior altura, para qualquer nó em $v_1, \ldots, v_5$ é 8.

II.6.7 O número máximos de iterações, i.e., de caminhos de aumento, é 2×10^3.

II.6.8 A altura máxima para o vértice v_6 é 1.

II.6.9 Em primeiro lugar, apenas é necessário actualizar o valor do fluxo máximo (i) caso a alteração na capacidade do arco (u, v) seja positiva e o arco esteja saturado, ou (ii) caso a alteração na capacidade do arco (u, v) seja negativa e a nova capacidade seja inferior à capacidade residual na rede residual actual. No caso (i) basta então verificar se existem novos caminhos de aumento na rede residual e, em caso afirmativo, aumentar o fluxo ao longo desses caminhos. No caso (ii) é necessário identificar pares de caminhos de s para u e de v para t na rede residual, usando apenas arcos com fluxo positivo, e diminuir o fluxo ao longo desses caminhos. No caso (ii), após terem sido identificados os pares caminhos suficientes para corrigir o fluxo máximo, é necessário verificar se existem novos caminhos de aumento na rede residual e, em caso afirmativo, aumentar o fluxo. Dado que em ambos os casos podemos aumentar ou diminuir o fluxo ao longo de um caminho em apenas uma unidade de fluxo, no pior caso esta actualização ocorrerá em tempo $O(k(V + E))$, o que é vantajoso para valores de k pequenos.

II.6.10
$O(VE)$

II.6.11

	s	A	B	t
h	4	1	5	0
Cortes:	$(\{s, B\}, \{A, t\})$			

II.6.12

V	V	F	F	F	V

II.6.13

	(s, A)	(s, C)	(A, B)	(B, C)	(B, t)	(C, A)	(C, t)
c_f	0	6	1	5	2	5	0
	(A, s)	(C, s)	(B, A)	(C, B)	(t, B)	(A, C)	(t, C)
c_f	2	5	2	0	2	0	5
Cortes	$\{s, A, C\}$, $\{B, t\}$						

II.6.14

L	D, A, C, B

II.6.15

Capacidades :	c_f(s,C,t)= 5; c_f(s,A,B,t) = 4; c_f(s,C,B,t) = 2
Corte :	({s,A,C}, {B,t})
Valor :	11

II.6.16

Caminhos :	s,A,B,t	s,A,C,B,t	s,A,C,D,B,t
$c_f(p)$:	1	2	2

II.6.17

Operações :	Push(E,D)	Relabel(A)	Relabel(D)
Corte :	s,A,C,D,E / B,t	$f(S, T) =$	12

II.6.18

h[A]	h[B]	h[C]	h[D]	h[E]
8	1	8	9	1
Corte :	$\{s, A, C, D\}/\{B, E, t\}$		$f(S, T) =$	14

II.6.19

h[A]	h[B]	h[C]	h[D]	h[E]
8	9	8	1	1
Corte :	$\{s, A, B, C\}/\{D, E, t\}$		$f(S, T) =$	12

II.6.20

Corte :	$(\{s, C\}, \{A, B, D, t\})$		
Valor :	11		
Min :	2	Max :	6

II.6.21

	p	c_p
1^o	s,t	2
2^o	s,A,t	1
3^o	s,A,B,t	1

II.6.22

	A	B	C	D
$h()$	7	1	7	8
Corte :	$\{s, A, C, D\}/\{B, t\}$		$f(S, T) =$	17

II.6.23

	A	B	C	D
$h()$	7	8	7	8
Corte :	$\{s, A, B, C, D\}/\{t\}$		$f(S, T) =$	10

II.6.24

Ao adicionar um novo arco (u, v), o valor do fluxo poderá aumentar se (u, v) aumentar a capacidade do corte mínimo. Se o arco não atravessar o corte mínimo de G, então o valor do fluxo mantém-se porque não existe nenhum caminho de s para t na rede residual.

Supondo que (u, v) atravessa o corte mínimo de G, então todos os novos caminhos de aumento de s para t precisam passar por (u, v). Caso contrário, o fluxo máximo ainda não teria sido calculado para G. Desta forma, o aumento do valor do fluxo é limitado pela capcidade do arco (u, v). Se usarmos o algoritmo de Ford-Fulkerson, então a actualização da rede pode ser feita em $O(E * c(u, v))$.

II.7. Árvores Abrangentes de Menor Custo

II.7.1 O número máximo de MSTs é obtido quando o grafo é completo (todos os vértices estão ligados entre si), e todos os arcos têm o mesmo peso. A resposta correcta é $|V|^{|V|-2}$, que é conhecida como *formula de Cayley*. A derivação da mesma não é trivial, mas não é necessária para a resolução do problema. Basta determinar por inspecção o número máximo de MSTs possíveis em grafos completos, com $|V|$ igual a 2, 3 e 4, que seria 1, 3 e 16, respectivamente. A expressão indicada é a única que verifica estes valores.

II.7.2

Número de MST's: $28 \times 28 = 784$. Custo de cada MST: 17.

II.7.3

Após o processamento de 8 arcos, temos dois conjuntos disjuntos:

$$\{v_1, v_2, v_3, v_4, v_5, v_6\}$$
$$\{v_7, v_8, v_9\}$$

II.7.4

	v_1	v_2	v_3	v_4	v_5	v_6
O.E.	1^o	2^o	3^o	4^o	6^o	5^o
d	0	2	2	2	2	2
π	NIL	1	2	3	6	4

II.7.5

Peso da MST: 10	Nº de MSTs: 9

II.7.6

Basta utilizar o *counting sort* para ordenar os arcos.

II.7.7 (A, B),(B, E),(D, E),(D, F),(A, C)

II.7.8

Majorante	$O(E + V \log V)$

II.7.9 O tempo do algoritmo seria $O(|V|^2 \log |V|)$.

Basta não utilizar um heap. Utilizamos apenas um array com os valores $key[v]$. Para encontrar o mínimo percorremos todo o array, tempo $O(|V|)$. Modificar uma chave é feito em tempo $O(1)$ com um acesso directo.

II.7.10

5 arcos. Custo 9.

II.7.11

Total :	4
Conjuntos :	{A,B,C},{D,E}

II.7.12

Arcos :	7
Custo :	8

II.7.13

arcos	8
custo	20

II.7.14

Ordem vértices	A	D	G	B	C	F	E	H	I
Custo MST	38								

II.7.15

Arcos	(A,E)	(B,E)	(C,E)	(D,E)	(E,F)	(E,G)	(E,H)	(E,I), (F,I), (H,I)	
Custo MST	36								
Número MST	3								

II.7.16

Arcos :	7
Custo :	28

II.7.17

Arcos :	(A,B) (B,D) (F,G) (E,G) (G,H)

II.7.18

Ordem vértices	A	D	E	G	B	H	I	F	C
Custo MST	$5X + 3Y$								

II.7.19 Seja $T' = T \backslash \{(u, v)\}$. Considerando um qualquer vértice s do grafo como origem, aplica-se uma pesquisa em profundidade primeiro (DFS) para identificar os vértices atingíveis a partir de s usando apenas os arcos de T'.

Seja S o conjunto dos vértices atingíveis a partir de s e $R = V \backslash S$. Seja $(x, y) \in E$ o arco de menor peso tal que $x \in S$ e $y \in R$. Nesse caso, podemos formar uma nova árvore abrangente de menor custo adicionando (x, y) a T'. A complexidade do algoritmo proposto é $O(E)$.

II.7.20

Ordem arcos	(E,G)	(H,I)	(A,B)	(C,E)	(D,G)	(A,E)	(E,I)	(E,F)
Custo MST	18							
Nº MST	2							

II.8. Resolução de Problemas usando Grafos

II.8.1 Um grafo G, dirigido ou não, contém um ciclo sse uma DFS em G produz um arco para trás (*back edge*). Basta portanto verificar a existência de arcos para trás durante a DFS. A complexidade é $O(V + E)$.

II.8.2 Manter em cada vértice v um contador $s[v]$ com o número de caminhos mais curtos desde u até v, que será inicializado a 0 para todos os vértices, exceptuando para u, que será inicializado a 1. Executar uma BFS com origem em u e, para cada arco (x, v), verificar adicionalmente se v é branco ou se $d[v] = d[x] + 1$. No caso de alguma das condições ser verdadeira, fazer $s[v] = s[v] + s[x]$. A complexidade é a mesma de uma BFS, ou seja $O(V + E)$, em termos de tempo e espaço.

Esta solução funciona tanto no caso em que queremos o número de caminhos mais curtos entre dois vértices u e v, como no caso em que queremos o número de caminhos mais curtos de um vértice u para todos os outros vértices. A única diferença é que no primeiro caso podemos parar após expandirmos todos os vértices w com $d[w] = d[v] - 1$. Mesmo assim, a complexidade mantém-se $O(V + E)$.

II.8.3 Um grafo diz-se bipartido se é possível dividir o conjunto de vértices em dois sub-conjuntos X e Y, tal que cada arco tenha um vértice em X e outro em Y. Verificar se um grafo é bipartido é equivalente a verificar se pode ser colorido com 2 cores.

Existem várias soluções para este problema, e qualquer solução correcta e razoavelmente eficiente seria aceite. Assumindo que o grafo é ligado, uma solução simples seria efectuar uma BFS, e considerar em X os vértices x com $d[x]$ par e em Y os vértices y com $d[y]$ ímpar. Sempre que ao considerarmos um arco (u, v) encontrarmos um vértice v já visitado anteriormente (de côr cinzenta ou preta), devemos verificar se $d[v]$ e $d[u]$ são ambos pares ou ambos ímpares. Nesse caso, o procedimento deverá terminar indicando que o grafo não é bipartido. Se a BFS terminar normalmanete sem esta condição se verificar, então o grafo é bipartido. Uma vez que esta verificação não introduz complexidade adicional na BFS, o tempo de execução será $O(V + E)$.

II.8.4 Uma solução em $O(V + E)$ consiste em executar uma DFS a partir de um qualquer vértice e, para cada vértice v visitado, guardar o menor d, designado por $low(v)$, que se atinge entre os descentendes de v na árvore DFS através de arcos de árvore ou de arcos para trás. Um vértice v é ponto de articulação se tiver um filho u na árvore DFS tal que $low(u) \geq low(v)$. A raiz é um caso especial e é ponto de articulação se tiver mais do que um filho na árvore DFS.

II.8.5 Uma solução em $O(V + E)$ é:

a. Trocar cada arco (u, v) por um arco (v, u).

b. Multiplicar por (-1) o peso de cada arco (v, u).

c. Designar o vértice t como s.

d. Calcular os caminhos mais curtos de s para todos os restantes vértices (num DAG) em $O(V + E)$.

e. Utilizar cada peso $\delta(s, u)$ calculado pelo ponto anterior, para obter o caminho de maior peso entre u e t, $\gamma(u, t) = -\delta(s, u)$.

II.8.6

Um grafo é *desligado* se não for semi-ligado, i.e. para quaisquer $u, v \in V$, $u \rightsquigarrow v$ ou $v \rightsquigarrow u$, i.e. existe caminho de u para v, ou de v para u. Basta resolver este segundo problema, para o qual existe um algoritmo em $O(V + E)$.

O algoritmo seguinte permite decidir, em $O(V + E)$ se um grafo é semi-ligado:

a. Calcular os SCCs do grafo, e representar o grafo de SCCs.

b. Realizar uma ordenação topológica no grafo de SCCs.

c. Verificar se os vértices $v_1, \ldots, v_n$ da ordenação topológica formam uma cadeia, i.e. $(v_i, v_{i+1}) \in E$, $i = 1, \ldots, n-1$. Em caso afirmativo, o grafo é semi-ligado; caso contrário, não é.

II.8.7

Este problema pode ser resolvido como um problema de determinar o fluxo máximo. Para tal, dada a rede distribuição, uma vez que quer os vértices quer os arcos têm capacidades, constrói-se uma rede de fluxo em que é necessário criar dois vértices por

cada armazém a_i, a_i^{in} e a_i^{out}, ligados por um arco (a_i^{in}, a_i^{out}) com capacidade c_i. Para cada ligação l_{ij} é adicionado um arco (a_i^{out}, a_j^{in}) à rede de fluxo com capacidade m_{ij}. Por último, uma vez que temos várias fontes e vários destinos, é necessário adicionar à rede de fluxo dois vértices s e t, arcos (s, a_i) para cada armazém a_i^{in} numa cidade portuguesa, e arcos (a_i^{out}, t) para cada armazém a_i numa capital europeia. Os arcos de s e para t têm capacidade ∞. A solução pode então ser obtida com o algoritmo de Edmonds-Karp ou com o algoritmo Relabel-To-Front. Uma vez que a construção da rede de fluxo é feita em tempo linear no tamanho da rede original, a complexidade é dominada pelo algoritmo escolhido para determinar o fluxo máximo, i.e., $O(VE^2)$ ou $O(V^3)$.

II.8.8

Basta executar uma DFS modificada em que guardamos para cada vértice o comprimento do caminho mais longo atingido a partir do mesmo, o qual é actualizado após o retorno de cada DFS-Visit. A actualização consiste em verificar se o comprimento do caminho mais longo a partir do vértice acabado de visitar, mais um, é maior do que o comprimento actual. Caso não existam arcos para trás, i.e., ciclos, retornamos o maior dos comprimentos calculados. Complexidade: $O(V + E)$.

II.8.9

Para calcular a matriz Π durante a execução do algoritmo é necessáro introduzir as linhas 3 a 7 (inicialização) e substituir o cálculo usual do mínimo no ciclo interno pelas linhas 11 a 13:

```
FLOYD-WARSHALL(W)
 1  n ← ROWS(W)
 2  D ← W
 3  for i ← 1 to n
 4        do for j ← 1 to n
 5              do if i ≠ j ∧ W_ij < ∞
 6                    then Π_ij = i
 7                    else  Π_ij = NIL
 8  for k ← 1 to n
 9        do for i ← 1 to n
10              do for j ← 1 to n
11                    do if D_ij > D_ik + D_kj
12                          then D_ij = D_ik + D_kj)
13                               Π_ij = Π_kj
14  return D
```

II.8.10

A solução consiste em determinar o caminho na MST T entre os extremos do arco (u, v) com uma BFS e enumerar todos os arcos na árvore cujo peso seja igual a $w(u, v)$. Caso não exista nenhum, então (u, v) não é arco leve para nenhum corte. A complexidade deste algoritmo é $O(V)$, em relação ao tempo de execução.

II.8.11 Basta calcular o fluxo máximo entre cada par de vértices, associando capacidade unitária a todos os arcos, e garantir que o fluxo máximo é sempre maior do que k. Se utilizarmos o algoritmo de Edmonds-Karp para determinar o fluxo máximo, a complexidade deste algoritmo é $O(V^3E^2)$.

II.8.12 Resolvemos este problema com algoritmos de fluxo máximo. Associamos a cada arco a capacidade de 1. Criamos uma origem S com um único arco para v e um

destino T com um único arco de entrada com origem em v'. O valor do fluxo máximo de S para T é o número máximo de caminhos disjuntos.

II.8.13

Este problema pode ser modelado como um grafo pesado e dirigido em que as moedas são os vertices. O peso entre duas moedas é dado como $\log(1/t)$ onde t é a taxa de conversão entre as moedas. Neste caso um caminho que maximiza o valor recebido corresponde a um caminho mais curto neste grafo.

A valorização da moeda comporta-se como uma repesagem pelo que os caminhos mais curtos não são alterados. A valorização define uma função h que associa a cada moeda um valor. No caso do Euro valorizar 1% teriamos $h(\text{Euro}) = log(1/1,01)$. Para obter o novo valor de um arco (u, v) fazemos o seguinte cálculo $\hat{w}(u, v) = h(u)+w(u, v)-h(v)$. Nestas condições os caminhos mais curtos não são alterados.

II.8.14

Primeiro criamos o grafo transposto G^T, que consiste em trocar o sentido dos arcos, este procedimento demora $O(|V| + |E|)$. De seguida ordenamos os vértices por ordem crescente das etiquetas L, este procedimento demora $O(|V| \log |V|)$ com um algoritmo de ordenação eficiente. Para finalizar executamos uma DFS em G^T, fazendo com que os recomeços respeitem a ordem crescente das etiquetas L. Para cada vértice u visitado por um recomeço que começou em v temos que $R(u) = v$. Estes valores podem ser guardados num array. O tempo deste procedimento é $O(|V| + |E|)$.

Logo o tempo total do procedimento é $O(|E| + |V| \log |V|)$.

II.8.15

Todas as MST's são minimax. Suponhamos por contradição que para uma dada MST T temos $\max T > \max T'$, onde T' é uma árvore abragente qualquer, ambas sobre o grafo G.

Seja e o arco com peso máximo de T, i.e., $w(e) = \max(T)$ e $e \in T$. Considere-se o corte c obtido removendo e a T, ou seja se $e = (u, v)$ temos $c = (U, V - U)$, onde U é o conjunto dos vértices atíngiveis a partir de u em $T - \{e\}$. Seja e' o arco de T' que atravessa c temos que $w(e') < w(e)$, porque e' é arco de T' e e é o arco de peso máximo de T. Logo e faz parte de uma MST, mas não é arco leve. Podemos portanto trocar e por e' em T e obter uma árvore abragente com custo menor que T. O que é uma contradição porque T é uma árvore de menor custo.

O algoritmo é uma DFS que utiliza apenas os arcos e de G cujo peso é menor ou igual a m, i.e., $w(e) \leq m$.

II.8.16 Expandir os nós da rede original v para um par de nós v_i, v_o ligados por um arco de capacidade $c'(v)$, (v_i, v_o). Os arcos de entrada originais no nó ligam a v_i e os arcos de saída tem v_o como origem. A capacidade desta nova rede é a capacidade da rede original, mas esta rede já tem o formato standard.

O número de arcos desta rede é $|E| + |V|$ e o número de vértices é $2|V|$, pelo que os algoritmos de fluxo nesta nova rede tem o comportamento assimptótico habitual, $O(|V||E|^2)$ para o Edmonds-Karp e $O(|V|^3)$ para o Relabel to Front.

II.8.17

A forma mais eficiente de modelar este problema é utilizar uma estrutura de dados para conjuntos disjuntos. Nomeadamente a representação em árvore, com as heurísticas de compressão de caminhos e união por categoria. Para uma imagem com n pixeis o espaço ocupado pela representação proposta é $O(n)$.

Para executar CriaImagem fazemos MakeSet para cada pixel. Uma mancha vai corresponder a um conjunto. Guardamos para cada conjunto o respectivo tamanho e o tamanho da maior mancha numa variável auxiliar. Inicialmente estes valores são todos postos a 0.

Para calcular a função Maior devolvemos o valor guardado na variável auxiliar.

Para calcular a função PintaPixel fazemos Union entre p e os pixeis nas 4 posições adjacentes que tiverem a cor c. Em cada união somamos o tamanho dos conjuntos envolvidos e guardamos esse valor como sendo o tamanho do novo conjunto.

Consideramos o tempo amortizado das operações descritas. A operação PintaPixel demora $O(\alpha(n))$, i.e., o inverso da função de Ackermann.

A operação Maior demora $O(1)$. A operação CriaImagem demora $O(n \times \alpha(n))$.

II.8.18

1) O problema pode ser modelado como uma rede de fluxo, os locais são os vértices e as diversões os arcos. A capacidade dos arcos é o tamanho da fila da respectiva diversão. Caso haja mais do que uma diversão a ligar o mesmo par de locais, a capacidade é a soma das respectivas capacidades.

2) O número máximo de pessoas que pode entrar no parque, num dado turno, é dado pelo valor do fluxo máximo da rede. Portanto o problema pode ser resolvido em tempo $O(V^3)$ com o algoritmo Relabel-to-Front.

3) Uma forma de evitar esperas é reduzir o tamanho das filas para ficar com o valor do fluxo atribuído a cada arco pelo algoritmo anterior.

II.8.19 O problema pode ser modelado como um grafo não dirigido. Os locais são representados pelos vértices. Quando há caminho entre um determinado par de vértices significa que há um arco entre esses vértices. O valor associado a esse arco é o numero, não negativo, que representa o tempo que o carro demora a mover-se de uma localidade para outra.

Podemos resolver este problema em $O(|E| \log |V|)$, utilizando o algoritmo de Dijkstra. Para tal acrescenta-se um vértice artifical, origem, com arcos de peso 0 a todos os quarteis. O quartel mais perto de uma determinada floresta é o único quartel que pode ser encontrado percorrendo a árvore dos caminhos mais curtos, partindo do vértice em questão.

Percorrer todos estes caminhos (utilizando o valor do predecessor na array π) irá, potencialmente, aumentar o tempo de execução do algoritmo. Para calcular este valor eficientemente utilizaremos a tabela χ, que a cada vértice associa o vértice correspondente ao quartel mais próximo, i.e., o quartel que o antecede na árvore dos caminhos mais curtos.

Esta tabela é preenchida durante a execução do algoritmo de Dijkstra. Inicialmente todos os valores da χ são postos a `NIL`. No primeiro passo, i.e., quando o valor `d` de um quartel passa a ser 0, seu valor de χ passa a ser o próprio quartel. Para todos os outros vértices, sempre que o valor de `d` é alterado, i.e., quando $d[v]$ é diminui devido ao relaxamento do arco (u, v), então o valor de $\chi[v]$ passa a ser igual a $\chi[u]$. Este procedimento garante que no fim do algoritmo os valores de χ das florestas estão correctamente atribuidos. A complexidade temporal do procedimento completo é também $O(|E| \log |V|)$.

II.8.20

Vamos assumir que os vértices estão numerados e que portanto os argumentos u e v das funções `Popular(v)` e `Ligar(u,v)` recebem números entre 0 e $V - 1$.

A estrutura de dados utilizada é constituida por duas arrays de inteiros `P[]` e `E[]`, com tamanho V, e uma estrutura para manter conjuntos disjuntos, também com tamanho V. Assumimos que a estrutura para conjuntos disjuntos suporta as operações de `Union(u,v)` e `FindSet(v)`. A primeira operação une os conjuntos, em que os vértices u e v estão contidos. A segunda operação devolve um outro vértice r que é o representante do conjunto a que v pertence.

O valor `E[v]` irá indicar o número de arcos ligados ao vértice v. Os valores `P[]` são mais complexos, mas irão ser utilizados para a operação `Popular(v)`.

A estrutura é alterada pela operações da seguinte forma:

- `Inicializa(V)`, Inicializa a estrutura de conjuntos disjuntos e as arrays, com `P[v] = v` e `E[v] = 0`, para todos os vértices. Demora tempo $O(V)$.
- `Popular(v)`, devolve o valor `P[FindSet(v)]`. Esta operação demora $O(\alpha(V))$, onde α é o inverso da função de Ackermann.
- `Ligar(u,v)`, incrementa os valores `E[u]` e `E[v]`, por uma unidade. De seguida invoca `Union(u,v)`. Compara `E[P[FindSet(v)]]` com `E[v]`, caso o segundo valor seja maior atríbui v a `P[FindSet(v)]`. Finalmente compara `E[P[FindSet(v)]]` com `E[u]`, caso o segundo valor seja maior atríbui u a `P[FindSet(v)]`. Esta operação demora $O(\alpha(V))$.

II.8.21

Começamos por ordenar os vértices do grafo por ordem topológica. De seguida executamos um algoritmo de caminhos mais curtos para DAGs, a partir da origem s. Este processo tem complexidade $O(V + E)$, assumindo uma representação do grafo em lista de adjacência.

Os valores de $d[v]$ são guardados para a próxima fase. São retirados do grafo todos os arcos (u, v) para os quais se verifica $d[v] < d[u] + w(u, v)$. Este processo requer tempo $O(V + E)$.

Inicializamos os contadores do número de alternativas a 0, para todos os vértices, exepto a origem, i.e., para todo vértice $u \in V \backslash \{s\}, c[u] = 0$ e $c[s] = 1$.

Finalmente, percorremos os vértices do grafo, pela ordem topológica. Para processar um determinado vértice u consideramos os seus arcos (u, v) e executamos a operação $c[v] = c[v] + c[u]$. No fim do algoritmo o número de caminhos mais curtos alternativos de s para t é dado pelo valor $c[t]$. Este último passo também tem complexidade $O(V + E)$, pelo que esta expressão é um majorante da complexidade assimptótica total do algoritmo.

II.8.22

a. O problema pode ser modelado com um grafo não dirigido. Os quadrados correspondem aos vértices. Existe um arco entre dois vértices exactamente quando correspondem a quadrados adjacentes que não estão separados por uma parede. O vértice que corresponde ao canto superior direito é considerado o vértice de partida e o vértice do canto inferior esquerdo o de destino. Como cada vértice tem no máximo 4 arcos associados temos que $E \leq 4V$, pelo que uma representação em listas de adjacência ocupa espaço $O(V)$.

b. Podemos executar uma BFS com origem na partida. A resposta desejada é o valor de `d` do vértice de destino. O tempo deste algoritmo é $O(V)$, usando o argumento da alínea anterior para simplificar a expressão.

II.8.23 Para determinar quantos arcos de um grafo podem ser removidos até que para um determinado par de vértices s e t deixe de haver caminho que os ligue basta considerar o grafo como uma rede de fluxo em que a capacidade dos arcos é sempre 1.

Executar o algoritmo de Ford-Fulkerson entre com origem em s e destino em t demora $O(EV)$, visto que o fluxo máximo nunca pode ser maior do que $V-1$. Note que podemos sempre considerar o corte que isola um vértice do resto do grafo e capadidade desse corte é no máximo $V - 1$ porque o vértice pode ter no máximo esse número da arcos associados.

Para encontrar o minimo global podemos repetir este algoritmo para todos os pares de vértices, o que demoraria $O(V^3E)$, mas é possivel melhorar este algoritmo para $O(V^2E)$. Note que pelo teorema do fluxo máximo corte minimo o valor que pretendemos calcular é o corte minimo global. Considere uma enumeração dos vértices do grafo $v_1, v_2, \ldots, v_i, \ldots, v_V$. Note que o corte minimo irá necessáriamente separar um par de

vértices (v_i, v_{i+1}), pelo que basta executar o algoritmo Ford-Fulkerson apenas para estes pares.

II.8.24 O problema pode ser modelado usando um grafo onde para cada edifício $e_i \in E$ existe um vértice correspondente no grafo. Para cada ligação $(e_a, e_b) \in L$ na proposta do director, existe um arco dirigido do vértice e_a para o vértice e_b.

Para satisfazer os requisitos do Sr. João Caracol, basta verificar se o grafo que modela a proposta do director é semi-ligado. Esse problema pode ser resolvido da seguinte forma:

- Construir o grafo dos componentes fortemente ligados (G_{SCC})
- Encontrar uma ordenação topológica de G_{SCC}
- Validar que a ordenação topológica encontrada no passo anterior é única. Se existir mais do que uma ordenação topológica, então existe um par de edifícios e_i, e_j tal que não existe caminho entre eles (em qualquer dos sentidos). Caso contrário, o grafo é semi-ligado e a proposta do director é válida.

Cada um dos passos descritos pode ser realizado em $O(V + E)$, definindo assim a complexidade da solução proposta.

II.8.25 Este problema pode ser resolvido usando uma rede de fluxo construida da seguinte forma:

- Para cada edifício $e_i \in E$ e cada cantina $c_j \in C$ criamos um vértice na rede de fluxo. Adicionamos ainda dois vértices auxiliares s e t para representar a source e sink da rede
- O vértice s tem um arco para cada edifício $e_i \in E$ com capacidade $num(e_i)$.
- Cada cantina $c_j \in C$ tem um arco para o vértice t com capacidade $cap(c_j)$
- Cada edifício $e_i \in E$ tem um arco para as cantinas do respectivo conjunto $V(e_i)$ com capacidade $+\infty$

O fluxo máximo da rede irá produzir a afectação dos empregados de cada edifício às cantinas. A capacidade dos arcos de s para cada edifício $e_i \in E$ limita o número de empregados do edíficio. A capacidade dos arcos de cada cantina $c_j \in C$ para t limita o número de pessoas na cantina. Como cada edifício e_i apenas se liga às cantinas no conjunto $V(e_i)$, então os empregados de cada edifício apenas podem ir a cantinas desse conjunto.

O fluxo máximo da rede pode ser calculado usando o algoritmo Relabel-To-Front com complexidade $O(V^3)$, onde V indica o número de vértices da rede de fluxo.

II.8.26 O problema pode ser modelado usando um grafo onde para cada edifício $e_i \in E$ existe um vértice correspondente no grafo. Para cada ligação $(e_a, e_b) \in L$ na proposta do director, existe um arco dirigido do vértice e_a para o vértice e_b com peso t_{ab}.

Para satisfazer os requisitos do Sr. João Caracol, temos que calcular o caminho mais curto entre todos os pares de edifícios. Como os pesos são todos não negativos e $|L| \leq 2 * |E|$, podemos aplicar o algoritmo de Dijkstra a partir de cada edifício com complexidade $O(E\ (E + L)\ lg\ E)$.

A proposta não é válida se entre um determinado par de edifícios não existir percurso ou então se o peso do percurso mais curto for superior a K. Caso contrário, a proposta é válida.

II.8.27 O problema pode ser modelado através de uma rede de fluxo da seguinte forma:

- Por cada secção do conjunto $S = \{s_1, s_2 \ldots s_n\}$ adicionamos um vértice na rede.
- Por cada categoria de trabalhadores $C = \{c_1, c_2 \ldots c_m\}$ adicionamos um vértice na rede.

- O vértice fonte s tem um arco para cada secç ao s_i com capacidade $trab(s_i)$.
- Cada categoria c_j tem um arco para o vértice destino t com capacidade $num(c_j)$.
- Cada secção s_i tem um arco para cada categoria do conjunto $T(s_i)$ com capacidade $trab(s_i)$.

O fluxo máximo desta rede irá resultar numa associaç ao dos trabalhadores às secç oes, sendo que cada secç ao nunca terá mais do que $trab(s_i)$ e cada categoria nunca irá associar mais do que $num(c_j)$ trabalhadores às secç oes.

Para resolver o problema de forma eficiente podemos aplicar o algoritmo de Relabel-To-Front com complexidade $O((|S| + |C|)^3)$.

II.8.28

a. O problema pode ser modelado com um grafo. Os pintores correspondem aos vértices. Há um arco do pintor A para o pintor B se existe um percurso temático que contém os dois e no qual são imediatamente consecutivos.

b. O problema tem solução caso exista uma ordenação topológica para o grafo. Encontrar a ordenação topológica demora $O(V + E)$, para um grafo com V vértices e E arcos representado com listas de adjacência. Caso o grafo contenha ciclos não é possível encontrar uma ordenação topológica. Este caso pode ser detectado se for encontrado um arco para trás na DFS.

   Para construir a representação do grafo assume-se que os percursos são dados. Para cada percurso geram-se todos os arcos induzidos pelos caminhos em tempo $O(V + E)$ (sem ser necessário eliminar arcos repetidos).

II.8.29

a. O tabuleiro pode ser modelado como um grafo não dirigido. As casas são os vértices e há arestas que ligam as casas adjacentes. Existem ainda 4 vértices especiais, que representam os extremos, de cima, de baixo, da esquerda e da direita. Cada um destes vértices está ligado às 11 casas do respectivo extremo. Os agrupamentos de peças podem ser modeladas com conjuntos.

b. Para determinar se um jogador conseguiu establecer um caminho entre os extremos utilizamos estruturas para conjuntos disjuntos para os agrupamentos de peças. Quando um jogador coloca uma peça numa casa, fazemos a união dessa peça com todas as peças do mesmo jogador, que estejam em casas adjacentes. Assume-se que as casas que representam os extremos em cima e baixo contém à partida peças do jogador V e as casas dos extremos esquerda e direita contém peças do jogador H. Podemos verificar que o jogador V ganhou quando os extremos de cima e de baixo pertencem ao mesmo conjunto. De forma similar o jogador H ganha quando os extremos da esquerda e da direita pertencem ao mesmo conjunto. Todas as complexidades são $O(\alpha(n^2))$, amortizado, por operação, i.e., reunir conjuntos e verificar se são iguais. Inicializar a estrutura de dados demora $O(n^2\alpha(n^2))$, onde α é a função inversa da função de Ackermann.

II.8.30

a. O problema pode ser modelado com um grafo dirigido. Os elementos correspondem aos vértices. Há um arco do elemento u para o elemento v se existe comunicação do elemento associado a u para o elemento associado a v.

b. O problema consiste em determinar se o grafo tem apenas um componente fortemente ligado. Utilizando ao algoritmo de Tarjan este procedimento demora $O(V + E)$.

Se existir mais do que um componente fortemente ligado, então a comunidade não é coerente porque existe pelo menos um par de vértices entre os quais não se consegue estabelecer comunicação. Caso contrário, se existir apenas um componente, então a comunidade é coerente.

II.8.31

a. O problema pode ser modelado como um grafo dirigido e pesado. As localidades são os vértices. Há uma arco entre dois vértices sempre que possa ser feito um transporte entre essas localidades. O peso de um arco é o respectivo valor de perda.

b. Visto que o conjunto S pode ser V, este problema pode ser resolvido com um algoritmo de caminhos mais curtos entre todos os pares. Vamos utilizar o algoritmo de Jonhson. Assumimos que não existem ciclos de peso negativo. Após o algoritmo de Jonhson calculamos para cada $s \in S$ o valor $x[s]$ que minimiza $\delta(s, v)$, entre todos os vértices $v \in V$. De seguida determinamos o vértice s de S que minimiza $x[s]$. Esse vértice corresponde à localidade que queremos determinar. O tempo total do algoritmo neste caso, executando o algoritmo de Dijktra apenas para os vertices de S, é $O(EV + S(V + E) \log V)$.

II.8.32

O problema apresentado corresponde a encontrar um corte mínimo num grafo não dirigido que pode ser resolvido usando uma rede de fluxo $G' = (V', E')$ da seguinte forma:

- Definem-se os vértices da rede de fluxo como sendo $V' = V \cup \{s, t\}$;
- Para cada arco $(u, v) \in E$ adicionam-se os arcos (u, v) e (v, u) a E' com capacidade $w(u, v)$;
- Para cada vértice $u_i \in p_1$ adiciona-se um arco (s, u_i) a E' com capacidade $+\infty$;
- Para cada vértice $v_j \in p_2$ adiciona-se um arco (v_j, t) a E' com capacidade $+\infty$;

Os arcos que definem o corte mínimo da rede de fluxo correspondem às ruas a bloquear, minimizando o número de polícias necessários para separar os conjuntos de manifestantes. A complexidade do algoritmo proposto é $O(V^3)$ se aplicarmos o algoritmo Relabel-To-Front.

II.8.33 A solução deste problema é semelhante a identificar se o grafo G é semi-ligado.

Suponha que aplica o algoritmo de Tarjan para identificar os componentes fortemente ligados (SCC) do grafo G.

Seja u o último vértice a ser fechado pelo algoritmo. Se existir um vértice raíz em G, então u é vértice raíz porque pertence ao último SCC identificado e este SCC não tem arcos de entrada. Caso contrário, não seria o último SCC a ser identificado.

Para testar se u é vértice raíz, basta verificar se todos os outros vértices do grafo são atingíveis a partir de u. Isto pode ser feito usando uma DFS. Se u não for vértice raíz, então o grafo não tem vértice raíz. Caso contrário, todos os vértices do mesmo SCC de u são vértices raíz.

A complexidade deste algoritmo é $O(V + E)$ devido à aplicação do algoritmo de Tarjan e DFS.

Parte III.
Síntese de Algoritmos & Tópicos Adicionais

III.1. Algoritmos Greedy

III.1.1
148

III.1.2
Um algoritmo ganancioso que obtém uma sequência S óptima:
Entrada: um conjunto de tarefas $T = \{t_1, \ldots, t_n\}$ em que cada tarefa t_i tem associada um prazo d_i e um peso p_i.
Saída: uma permutação do conjunto $\{t_1, \ldots, t_n\}$, ou seja uma sequência de execução.
Sejam S e R duas sequências.

a. Inicializar as sequências S e R vazias.
b. Afectar Q com a sequência de tarefas resultante da ordenação do conjunto T, por ordem decrescente dos pesos p_i.
c. Enquanto Q não estiver vazio:
   (a) Retirar o primeiro elemento t de Q (t é a tarefa de maior peso em Q).
   (b) Se for possível inserir t em S de forma a continuar a não existirem tarefas em atraso em S, então inserir t em S por ordem crescente de prazo. Caso contrário, inserir t em R.
d. Inserir no final de S as tarefas em R.
e. Retornar S.

Este algoritmo corre em tempo quadrático no pior caso.

É possível obter um algoritmo mais eficiente se utilizarmos a estrutura de dados para *conjuntos disjuntos*. Neste caso um algoritmo ganancioso que também obtém uma sequência S óptima define-se da seguinte forma:
Entrada: um conjunto de tarefas $T = \{t_1, \ldots, t_n\}$ em que cada tarefa t_i tem associada um prazo d_i e um peso p_i.
Saída: uma permutação do conjunto $\{t_1, \ldots, t_n\}$, ou seja uma sequência de execução.
Sejam S e R dois vectores.

a. Inicializar o vector S vazio. Cada posição de S irá estar vazia ou conter uma tarefa. Inicialmente todas as posições de S estão vazias.
b. Inicializar a estrutura de dados para conjuntos disjuntos DS com as tarefas, ou seja, $\textsc{makeSet}(t_i)$ para $1 \leq i \leq n$.
c. Para cada tarefa $t \in T$, por ordem decrescente das penalizações p_i, onde d é o prazo de t:
   (a) Se $S[d]$ estiver livre, afectar $S[d] \leftarrow t$ e $pos \leftarrow d$.
   (b) Caso contrário, afectar $s \leftarrow \textsc{findSet}(S[d])$ com o conjunto em DS que contém a tarefa $S[d]$; afectar $pos \leftarrow \textsc{largest}(s) - \textsc{size}(s)$, ou seja, afectar pos com a diferença entre o maior índice em S onde está inserido um elemento de s e o tamanho de s, e:
      i. Se $pos > 0$, afectar $S[pos] \leftarrow t$.
      ii. Caso contrário, a tarefa t fica em atraso e é colocada na última posição ainda livre de S. Afectar pos com a posição em que ficou t.
   (c) É necessário garantir que não existem dois conjuntos em DS cujas tarefas estejam contíguas em S:

i. Se $S[pos-1]$ não está livre, então UNION(FINDSET($S[pos]$), FINDSET($S[pos-1]$)).
ii. Se $S[pos+1]$ não está livre, então UNION(FINDSET($S[pos]$), FINDSET($S[pos+1]$)).

d. Retornar S.

O array S é tal que cada posição i corresponde à i-ésima unidade de tempo e $S[i]$ corresponde, após o término do algoritmo, à tarefa que na sequência de execução óptima é efectuada na i-ésima unidade de tempo.

A ideia do algoritmo é atribuir tarefas às posições de S, inicialmente vazias, por forma a minimizar a soma do peso das tarefas que não possam ser efectuadas a tempo. Para cada tarefa t_i, por ordem decrescente do valor, verifica-se se podemos ainda efectuar t_i na unidade de tempo d_i, a última unidade possível sem ultrapassar o prazo d_i. Caso não seja possível, tentamos colocar a tarefa t_i na maior posição livre antes da posição d_i. Se não for possível, a tarefa t_i ficará em atraso e colocamo-lo na última posição ainda livre de S. Na descrição acima utiliza-se uma estrutura de dados para conjuntos disjuntos para determinar as posições livres. Cada conjunto contém tarefas em posições contíguas em S. Neste caso o algoritmo corre em tempo $O(n \log n)$.

III.1.3 Esta é uma das várias soluções possíveis:

a	b	c	d	e	f	g	h
0010	0110	000	0011	0100	1	0101	0111

III.1.4 Um algoritmo ganancioso que obtém o escalonamento S óptimo pode ser obtido utilizando uma estrutura de dados para *conjuntos disjuntos*. Note-se que também é possível obter uma solução quadrática sem utilizar conjuntos disjuntos. O algoritmo ganancioso proposto define-se da seguinte forma:
Entrada: um conjunto de tarefas $T = \{1, \ldots, n\}$ em que cada tarefa i tem associado um valor de retorno v_i e um prazo t_i.
Saída: uma permutação do conjunto $\{1, \ldots, n\}$, ou seja uma sequência de execução.
Seja S um vector.

a. Inicializar o vector S vazio. Cada posição de S irá estar vazia ou conter uma tarefa. Inicialmente todas as posições de S estão vazias.
b. Inicializar a estrutura de dados para conjuntos disjuntos DS com as tarefas, ou seja, MAKESET(i) para $1 \leq i \leq n$.
c. Para cada tarefa $i \in T$, por ordem decrescente dos valores v_i, onde t_i é o prazo de i:
   (a) Se $S[t_i]$ estiver livre, afectar $S[t_i] \leftarrow i$ e $pos \leftarrow t_i$.
   (b) Caso contrário, afectar $s \leftarrow$ FINDSET($S[t_i]$) com o conjunto em DS que contém a tarefa $S[t_i]$; afectar $pos \leftarrow$ LARGEST(s) $-$ SIZE(s), ou seja, afectar pos com a diferença entre o maior índice em S onde está inserido um elemento de s e o tamanho de s, e:
      i. Se $pos > 0$, afectar $S[pos] \leftarrow i$.
      ii. Caso contrário, a tarefa i fica em atraso e é colocada na última posição ainda livre de S. Afectar pos com a posição em que ficou i.
   (c) É necessário garantir que não existem dois conjuntos em DS cujas tarefas estejam contíguas em S:
      i. Se $S[pos-1]$ não está livre, então UNION(FINDSET($S[pos]$), FINDSET($S[pos-1]$)).
      ii. Se $S[pos+1]$ não está livre, então UNION(FINDSET($S[pos]$), FINDSET($S[pos+1]$)).

d. Retornar S.

O array S é tal que cada posição i corresponde à i-ésima unidade de tempo e $S[i]$ corresponde, após o término do algoritmo, à tarefa que na sequência de execução óptima é efectuada na i-ésima unidade de tempo. A ideia do algoritmo é atribuir tarefas às posições de S, inicialmente vazias, por forma a minimizar a soma do valor de retorno das tarefas que não possam ser efectuadas a tempo. Para cada tarefa i, por ordem decrescente do valor, verifica-se se podemos ainda efectuar i na unidade de tempo t_i, a última unidade possível sem ultrapassar o prazo t_i. Caso não seja possível, tentamos colocar a tarefa i na maior posição livre antes da posição t_i. Se não for possível, a tarefa i ficará em atraso e colocamo-lo na última posição ainda livre de S. Na descrição acima utiliza-se uma estrutura de dados para conjuntos disjuntos para determinar as posições livres. Cada conjunto contém tarefas em posições contíguas em S. Neste caso o algoritmo corre em tempo $O(n \log n)$.

III.1.5

a	b	c	d	e	f	g
100	101	1100	11011	11010	111	0

III.1.6 13200 bits.

III.1.7

a	b	c	d	e	f	g
1110	1111	1011	100	0	110	1010

III.1.8

	a	b	c	d	e	f	g
Codificação	101	10001	100001	11	100000	0	1001

III.1.9

	a	b	c	d	e	f	g
#bits	5	3	4	2	2	2	5

III.1.10 O algoritmo para resolver este problema começa por ordenar os electrodomésticos de forma não decrescente pelo tempo de reparação. Desta forma, o tempo total de espera dos clientes é minimizado. Este problema é em tudo semelhante ao problema do tempo de espera dos processos num servidor que foi leccionado nas aulas. A prova de optimalidade é análoga.

III.1.11

Supondo que desejamos perfazer um troco de K, iteramos sobre as diferentes denominações começando na denominação n até à primeira. Para cada denominação i, escolhemos usar m_i moedas onde $m_i = K \; div \; d_i$ (onde div denota o operador para a divisão inteira) e actualizamos K para $K - m_i * d_i$. O número total de moedas seria $\sum_{i=1}^{n} m_i$

O algoritmo greedy pode não produzir a solução óptima se a condição não se verificar.

Se $n = 3$ e $d_1 = 1, d_2 = 4, d_3 = 6$, o algoritmo greedy indica 3 moedas para perfazer um troco de 8 quando a solução óptima seria apenas 2 moedas.

III.1.12

Numero de Actividades:	5
Actividades	1, 3, 5, 8, 9

III.1.13

	a	b	c	d	e
Palavra	01	00	111	110	10

III.1.14

Ordem	5	2	4	3	1	7	6
Tempo	1	3	6	10	15	21	28

III.1.15

Este problema é equivalente ao problema da mochila fraccionário, pelo que tem uma solução *greedy*.

Começamos por fazer a ordenação dos n objectos de forma não crescente do valor por unidade de peso. Seguindo essa ordem, vamos colocando sucessivamente 50% de cada objecto em ambas as caixas. Quando atingirmos um objecto i tal que $p_i/2$ excede a capacidade residual das caixas, seleccionamos apenas a percentagem suficiente para encher cada uma das duas caixas e o algoritmo termina.

Este algoritmo tem complexidade $O(n\ lg\ n)$ devido ao procedimento de ordenação dos n objectos. A selecção dos objectos a colocar nas caixas é $O(n)$.

III.1.16

	a	b	c	d	e	f	g
Codificação	01111	010	0110	11	01110	10	00

Total Bits	24000

III.1.17

	a	b	c	d	e	f
Codificação	101	100	110	111	00	01

Total Bits	257

III.1.18

	a	b	c	d	e	f
Codificação	10000	10001	1001	101	11	0

Total Bits	187

III.1.19

Ordem	3, 5, 4, 1, 2, 8, 7, 6

III.2. Programação Dinâmica

III.2.1

Seja $L[j]$ o comprimento da maior subsequência estritamente crescente que termina em j. Podemos definir $L[j]$ recursivamente da seguinte forma:

$$L[j] = \max\{L[i] + 1 \mid i < j \text{ e } a_i < a_j\}. \tag{1}$$

A solução óptima é dada por $\max_j\{L[j]\}$. Os valores $L[j]$ podem ser determinados em tempo $O(n^2)$ ou, com recurso a filas de prioridade, em tempo $O(n \log n)$.

III.2.2

a) Seja $V(i, x)$ tal que:

$$V(i,x) = \begin{cases} 1 & \text{se algum subconjunto de } S \text{ tem soma } x \\ 0 & \text{c.c.} \end{cases}$$

Podemos escrever $V(i, x)$ como: $V(i, x) = 1$ se $P(i-1, x) = 1$ ou $P(i-1, x-s_i) = 1$, ou ainda $V(i, x) = max(P(i-1, x), P(i-1, x-s_i))$, em que $P(0, x) = 0$. Basta calcular $P(n, k)$ para determinar a resposta ao problema.

b) Complexidade: $O(nk)$. i varia entre 1 e n e x é no máximo k. O algoritmo é pseudo-polinomial porque depende do tamanho da representação de k.

c) Queremos minimizar $\left| \sum_{x \in S_1} x - \sum_{x \in S_2} x \right|$. Seja $t = \left(\sum_{i=1}^{n} s_i \right) /2$ e calculemos

$$\min_{y \leq t}\{t - y : P(n, y) = 1\}$$

Dado y mínimo, temos que existe S_1 tal que $\sum_{x \in S_1} x = y$ e $S_2 = S \setminus S_1$, tal que $\sum_{x \in S_2} x = 2t - y$. Estes conjuntos produzem a menor diferença porque:

$$2 \min_{y \leq t}\{t - y : P(n, y) = 1\} = 2(t - y) = (2t - y) - y = \sum_{x \in S_2} x - \sum_{x \in S_1} x$$

Logo, dado que y minimiza $2 \min_{y \leq t}\{t - y : P(n, y) = 1\}$, tem-se que a última diferença é mínima.

III.2.3

	-	A	**A**	**B**	**C**	**B**	**D**	**E**	**B**
-	0	0	0	0	0	0	0	0	0
A	0	1	**1**	1	1	1	1	1	1
B	0	1	1	**2**	2	2	2	2	2
C	0	1	1	2	**3**	3	3	3	3
B	0	1	1	2	3	**4**	4	4	4
D	0	1	1	2	3	4	**5**	5	5
E	0	1	1	2	3	4	5	**6**	6
B	0	1	1	2	3	4	5	6	**7**
C	0	1	1	2	3	4	5	6	7

III.2.4

Uma solução possível consiste em reduzir o problema LIS ao problema LCS. Seja $A = \langle a_1, a_2, \ldots, a_n \rangle$ a sequência que é dada. Ordenar a sequência A, obtendo uma nova sequência $S = \langle s_1, s_2, \ldots, s_n \rangle$. A LCS entre A e S representa uma LIS de A. O problema LCS pode ser resolvido com uma abordagem de programação dinâmica, em $O(n^2)$. Assim, a complexidade desta solução tem complexidade em $O(n \log n + n^2) = O(n^2)$.

Uma outra solução consiste em desenvolver uma solução de programação dinâmica para o problema LIS. Definir $L[i]$ como o comprimento da LIS que termina na posição i.

$$L[i] = \begin{cases} 1 + \max_{j=0,\ldots,i-1}\{L[j] \,|\, a_j < a_i\} & \text{if } i > 0 \\ 0 & \text{if } i = 0 \end{cases}$$

Uma implementação básica de $L[i]$ resulta num algoritmo em $O(n^2)$.

Nota: A partir da 2^a solução é possível obter um algoritmo em $O(n \log n)$, observando que basta guardar apenas o menor valor para cada um dos comprimentos das sequências crescentes.

III.2.5

Nota-se que uma solução gananciosa em que se escolhe em primeiro lugar a localização com maior lucro não funciona, podendo conduzir a uma solução subóptima. Com vista a uma abordagem baseada em programação dinâmica, assuma-se sem perda de generalidade que $k_1 < k_2 < \ldots k_n$ e seja $V[i, p]$ o lucro óptimo quando estamos a considerar as primeiras i localizações e existe um restaurante no km p, com $p > k_i$. O valor óptimo $V[i, p]$ pode ser determinado por

$$V[i, p] = \begin{cases} 0 & \text{se } i = 0 \\ V[i-1, p] & \text{se } i \neq 0 \text{ e } p - k_i \leq Q \\ \max(V[i-1, p], V[i-1, k_i] + p_i) & \text{se } i \neq 0 \text{ e } p - k_i > Q \end{cases}$$

O valor óptimo para as n localizações é dado por $V[n, \infty]$.

Uma vez que a segunda condição na formulação acima serve apenas para iterar até determinarmos a maior posição compatível com a última posição ocupada, podemos

melhorar a complexidade do algoritmo calculando essas posições a priori. Deste modo, seja $c(i) = \min\{j \mid k_i - k_j \leq Q\} - 1$ e $V[i]$ o lucro óptimo quando estamos a considerar as primeiras i localizações. O valor óptimo $V[i]$ pode agora ser determinado por

$$V[i] = \begin{cases} p_1 & \text{se } i = 1 \\ \max(V[i-1], V[c(i)] + p_i) & \text{se } i > 1 \text{ e } c(i) > 0 \\ \max(V[i-1], p_i) & \text{se } i > 1 \text{ e } c(i) = 0 \end{cases}$$

em tempo linear. O valor óptimo para as n localizações é dado por $V[n]$.

III.2.6 $\max(v[i-1, j], v[i-1, j-w_i] + v_i)$

III.2.7

	-	**v**	i	**o**	**l**	**e**	n	t
-	**0**	0	0	0	0	0	0	0
v	0	**1**	**1**	1	1	1	1	1
o	0	1	1	**2**	2	2	2	2
l	0	1	1	2	**3**	3	3	3
u	0	1	1	2	**3**	3	3	3
m	0	1	1	2	**3**	3	3	3
e	0	1	1	2	3	**4**	**4**	**4**

III.2.8 Seja $S[i]$ o valor da subsequência na posição i, com $i = 1, \ldots, |S|$. Seja mx[k] o valor da sub-sequência contínua com soma máxima que termina em k, com mx[0] = 0, e mx[i] = $max(0, \text{mx}[i-1] + S[i])$, para $i = 1, \ldots, |S|$. Se mx[i] = 0 para algum i, então a soma máxima não inclui elementos com indices não superiores a i.

III.2.9
$\max(c[i-1, j], c[i, j-1])$

III.2.10
Estratégia *Greedy*: 32.
Programação dinâmica Dinâmica: 32.

III.2.11

número de moedas	Solução	$c[7, 10]$
2	de valor 4 e 8	3

III.2.12
A abordagem usando programação dinâmica tem complexidade $O(n \times K)$.

Considere-se a tabela $\texttt{r}[\texttt{i}, \texttt{j}]$, em que `i` varia de 0 a n e que itera sobre todas as encomendas, e `j` varia de 1 a K. O tamanho desta tabela é portanto $O(n \times K)$.

O valor $\texttt{r}[\texttt{i}, \texttt{j}]$ representa a receita que é possivel obter seleccionando apenas entre as primeiras i encomendas e com uma produção de j quilolitros para escoar. Portanto, temos a seguinte relação:

$$\texttt{r}[\texttt{i}, \texttt{j}] = \begin{cases} 0 & \text{, se } \texttt{j} = 0 \text{ ou } \texttt{i} = 0. \\ -\infty & \text{, se } \texttt{j} < 0 \\ \max(\texttt{v}_\texttt{i} + \texttt{r}[\texttt{i}-1, \texttt{j}-\texttt{a}_\texttt{i}], \texttt{r}[\texttt{i}-1, \texttt{j}]) & \text{, caso contrário} \end{cases}$$

III.2.13
Um algoritmo eficiente demora $O(m \times n)$ e consiste numa abordagem de programação dinâmica. Considere-se a tabela $\texttt{D}[\texttt{i}, \texttt{j}]$, em que `i` varia de 0 a n e que itera sobre os elementos da sequência, e `j` varia de 1 a m. O tamanho desta tabela é portanto $O(m \times n)$.

O valor $\texttt{D[i, j]}$ representa o número de formas que é possivel obter j utilizando apenas alguns dos i primeiros elementos da sequência. Portanto, temos a seguinte relação:

$$\texttt{D[i, j]} = \begin{cases} 1 & \text{, se } \texttt{j} = 0 \text{ e } \texttt{i} > 0. \\ 0 & \text{, se } \texttt{j} < 0, \text{ ou } \texttt{j} > 0 \text{ e } \texttt{i} = 0 \\ \texttt{D[i} - 1, \texttt{j]} + \texttt{D[i} - 1, \texttt{j} - \texttt{S[i]]} & \text{, caso contrário} \end{cases}$$

Note-se que na última expressão o valor $\texttt{D[i, j} - \texttt{S[i]]}$ é 0 se $\texttt{j} < \texttt{S[i]}$. A expressão $\texttt{S[i]}$ representa o valor do $\texttt{i}$-ésimo valor na sequência, que é representada por S.

III.2.14 A abordagem usando programação dinâmica tem complexidade $O(n)$.

Considere-se a tabela $\texttt{s[i, j]}$, em que $\texttt{i}$ varia de 0 a n, e $\texttt{j}$ varia de 0 a 1. O valor de j representa se o valor que está na posição i seja seleccionado. O tamanho desta tabela é portanto $O(n)$.

A posição $\texttt{s[i, j]}$ representa o melhor valor que conseguimos obter considerando apenas as i primeiras posições da sequência.

Seja a_i o i-ésimo valor da sequência de entrada. Assim, a fórmula de recursão seria:

$$\texttt{s[i, j]} = \begin{cases} 0 & \text{, se } \texttt{i} = 0 \\ \texttt{a}_\texttt{i} + \texttt{s[i} - 1, 0\texttt{]} & \text{, se } \texttt{j} = 1 \\ \max(\texttt{a}_\texttt{i} + \texttt{s[i} - 1, 0\texttt{]}, \texttt{s[i} - 1, 1\texttt{]}) & \text{, caso contrário} \end{cases}$$

III.2.15 A abordagem usando programação dinâmica tem complexidade $O(n \times M)$.

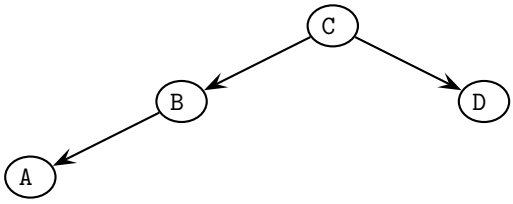
Considere-se a tabela $\texttt{s[i, j]}$, em que $\texttt{i}$ varia de 0 a n, e $\texttt{j}$ varia de 0 a M. O tamanho desta tabela é portanto $O(n \times M)$.

A posição $\texttt{s[i, j]}$ representa o valor da soma mais próxima de j que conseguimos obter considerando apenas os primeiros i elementos da sequência de entrada.

Seja a_i o i-ésimo valor da sequência de entrada. Assim, a fórmula de recursão seria:

$$\texttt{s[i, j]} = \begin{cases} 0 & \text{, se } \texttt{i} = 0 \\ -\infty & \text{, se } \texttt{j} < 0 \\ \max(\texttt{a}_\texttt{i} + \texttt{s[i} - 1, \texttt{j} - \texttt{a}_\texttt{i}\texttt{]}, \texttt{s[i} - 1, \texttt{j]}) & \text{, caso contrário} \end{cases}$$

III.2.16 A árvore optima é a seguinte com custo total de $2\times3+10\times2+8\times1+9\times2 = 52$.



Para a recurrência de programação dinâmica utilizamos uma tabela $\texttt{B[i, j]}$, em que $\texttt{i}$ e $\texttt{j}$ variam de 1 a n. Esta tabela deve guardar o valor do custo total da árvore óptima, quando a sequência de acesso é restringida a apenas conter chaves entre a_i e a_j. Note que se $j < i$ então o conjunto das chaves é vazio. Assim, a fórmula de recursão seria:

$$\texttt{B[i, j]} = \begin{cases} 0 & \text{, se } j < i \\ \min_{i \le k \le j} \left\{ \texttt{B[i, k} - 1\texttt{]} + \texttt{B[k} + 1, \texttt{j]} + \sum_{k=i}^{j} a_k \right\} & \text{, caso contrário.} \end{cases}$$

III.2.17

Valor óptimo :	96
Objectos	(8,64), (4,32)

$v[1,6]$	$v[3,12]$	$v[4,10]$	$v[5,10]$
32	64	64	64

III.2.18

`m[1,4]`	`m[2,5]`	`m[1,5]`	Parênteses
12	12	13	$(A_1 \times (A_2 \times A_3)) \times (A_4 \times A_5)$

III.2.19 A fórmula de recursão é:

$$\texttt{C[j]} = \begin{cases} 0 & \text{, se } \texttt{j} = 0 \\ \min_{1 \le \texttt{i} \le \texttt{j}} \{\texttt{C[i} - 1\texttt{]} + \texttt{L[i, j]}\} & \text{, caso contrário.} \end{cases}$$

III.2.20

Greedy	34
Prog. Dinâmica	45

III.2.21

A abordagem usando programação dinâmica tem complexidade $O(n \times M)$.

Considere-se a tabela `s[i, j]`, em que `i` varia de 0 a n, e `j` varia de 0 a M. O tamanho desta tabela é portanto $O(n \times M)$.

O valor `s[i, j]` representa o tamanho da maior subsequência cuja soma é j, considerando apenas os primeiros i elementos da sequência de entrada.

Seja a_i o i-ésimo valor da sequência de entrada. Assim, a fórmula de recusão seria:

$$\texttt{s[i, j]} = \begin{cases} 0 & \text{, se } \texttt{j} = 0 \\ -\infty & \text{, se } \texttt{j} < 0 \text{ ou } \texttt{i} = 0 \\ \max(1 + \texttt{s[i} - 1, \texttt{j} - \texttt{a}_\texttt{i}\texttt{]}, \texttt{s[i} - 1, \texttt{j]}) & \text{, caso contrário} \end{cases}$$

III.2.22

	Valor	Lista de índices
Greedy	64	1, 3, 2, 5
PD	58	1,2,3

III.2.23

`m[1,4]`	`m[2,5]`	`m[1,5]`	Parênteses
10	12	18	$(A_1 \times ((A_2 \times A_3) \times A_4)) \times A_5$

III.2.24

$$\texttt{B[i, j]} = \begin{cases} -\infty & \text{, se } j < 0 \\ \boxed{0} & \text{, se } i = 0 \text{ e } j \ge 0 \\ \boxed{\max\{B[i-1, j-S_i] + S_i,\ B[i-1, j]\}} & \text{, se } i > 0 \text{ e } j \ge 0 \end{cases}$$

III.2.25

maior	número de moedas	Solução
11	4	$3 \times d_1 + d_2$

III.2.26

$$\texttt{D[i]} = \begin{cases} \boxed{0} & \text{, se } i = 0 \\ \boxed{S_1} & \text{, se } i = 1 \\ \boxed{\max\{D[i-2] + S_i, D[i-1]\}} & \text{, se } i > 1 \end{cases}$$

III.3. Programação Linear

III.3.1

$$\begin{array}{lllr} \textit{minimize} & 0.0065x + 0.0053y & & \\ \textit{subject to} & 0.32x + 0.48y & \geq & 24 \\ & 0.48x + 0.48y & \geq & 36 \\ & 0.08x + 0.04y & \geq & 4 \\ & x + y & \leq & 150 \\ & x, y & \geq & 0 \end{array}$$

III.3.2

$$\begin{array}{rc} \max & x_{123} + x_4 + x_5 \\ \text{s.t.} & x_{123} \leq a_1 + b_2 + c_3 \\ & x_{123} \leq a_1 + c_2 + b_3 \\ & x_{123} \leq b_1 + a_2 + c_3 \\ & x_{123} \leq b_1 + c_2 + a_3 \\ & x_{123} \leq c_1 + a_2 + b_3 \\ & x_{123} \leq c_1 + b_2 + a_3 \\ & x_4 \leq a_4 \\ & x_4 \leq b_4 \\ & x_4 \leq c_4 \\ & x_5 \leq a_5 \\ & x_5 \leq b_5 \\ & x_5 \leq c_5 \\ & x_{123}, x_4, x_5 \geq 0 \end{array}$$

III.3.3 Seja x_1, x_2, x_3 e x_4 a quantidade (em milhares de m^2) de tecido industrial, rede para insectos, revestimento para telhados e vedações, respectivamente. O programa linear pode ser formulado como se segue:

$$\begin{array}{lrcl} \textit{max} & 6x_1 + 5x_2 + 3.8x_3 + 4x_4 & & \\ \textit{s.a.} & x_1 + 3x_2 + 3x_3 + 2.5x_4 & \leq & 15 \\ & x_1 + x_2 + 2x_3 + 1.5x_4 & \leq & 6 \\ & 2x_1 + x_2 + 1.5x_3 + 2x_4 & \leq & 10 \\ & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}$$

III.3.4

$$\begin{array}{lrcl} \textit{max} & 3x_1 + 4x_2 + 6x_3 + 5x_4 & & \\ \textit{s.a.} & 900x_1 + 1200x_2 + 1500x_3 + 2000x_4 & \leq & 20000 \\ & x_1 + x_2 + x_3 + x_4 & \leq & 14 \\ & x_1 & \leq & 6 \\ & x_2 & \leq & 8 \\ & x_3 & \leq & 10 \\ & x_4 & \leq & 4 \\ & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}$$

III.3.5

max	$x_1 + x_2 + x_3 + x_4$		
s.a.	x_3	$\geq$	400
	$2x_1 + 3x_2 + 4x_3 + 5x_4$	$\leq$	3300
	$3x_1 + 4x_2 + 5x_3 + 6x_4$	$\leq$	4000
	$15x_1 + 10x_2 + 9x_3 + 7x_4$	$\leq$	12000
	x_1, x_2, x_3, x_4	$\geq$	0

III.3.6

Seja x_i a proporção do alimento A_i a comprar.

$$\begin{array}{rrcl}
min & 4.5x_1 + 6x_2 + 2x_3 + 3x_4 & & \\
s.a. & 300x_1 + 900x_2 + 200x_3 + 500x_4 & \leq & 2000 \times 1.1 \\
 & 300x_1 + 900x_2 + 200x_3 + 500x_4 & \geq & 2000 \times 0.9 \\
 & 20x_1 + 5x_2 + 10x_3 + 15x_4 & \leq & 50 \times 1.1 \\
 & 20x_1 + 5x_2 + 10x_3 + 15x_4 & \geq & 50 \times 0.9 \\
 & 30x_1 + 90x_2 + 50x_3 + 70x_4 & \leq & 270 \times 1.1 \\
 & 30x_1 + 90x_2 + 50x_3 + 70x_4 & \geq & 270 \times 0.9 \\
 & 5x_1 + 5x_2 + 10x_3 + 10x_4 & \leq & 25 \times 1.1 \\
 & 5x_1 + 5x_2 + 10x_3 + 10x_4 & \geq & 25 \times 0.9 \\
 & x_1, x_2, x_3, x_4 & \geq & 0
\end{array}$$

III.3.7 Seja f_{xy} o fluxo do vértice x para o vértice y da rede.

$$\begin{array}{rrcl}
max & f_{sa} + f_{sb} & & \\
s.a. & f_{sa} & \leq & 8 \\
 & f_{sb} & \leq & 5 \\
 & f_{ab} & \leq & 2 \\
 & f_{ac} & \leq & 2 \\
 & f_{bd} & \leq & 3 \\
 & f_{cd} & \leq & 1 \\
 & f_{ct} & \leq & 7 \\
 & f_{dt} & \leq & 2 \\
 & f_{sa} - f_{ab} - f_{ac} & = & 0 \\
 & f_{sb} + f_{ab} - f_{bd} & = & 0 \\
 & f_{ac} - f_{cd} - f_{ct} & = & 0 \\
 & f_{bd} + f_{cd} - f_{dt} & = & 0 \\
 & f_{sa}, f_{sb}, f_{ab}, f_{ac}, f_{bd}, f_{cd}, f_{ct}, f_{dt} & \geq & 0
\end{array}$$

III.3.8 Considere que f_{ij} denota o valor do fluxo do vértice i para o vértice j.

$$\begin{array}{rrcl}
Maximizar & f_{sA} & & \\
Sujeito\ a & f_{sA} & \leq & 5 \\
 & f_{AB} & \leq & 1 \\
 & f_{AC} & \leq & 6 \\
 & f_{Bt} & \leq & 11 \\
 & f_{CB} & \leq & 2 \\
 & f_{CD} & \leq & 6 \\
 & f_{DB} & \leq & 2 \\
 & f_{sA} - f_{AB} - f_{AC} & = & 0 \\
 & f_{AB} + f_{CB} + f_{DB} - f_{Bt} & = & 0 \\
 & f_{AC} - f_{CB} - f_{CD} & = & 0 \\
 & f_{CD} - f_{DB} & = & 0 \\
 & f_{sA}, f_{AB}, f_{AC}, f_{Bt}, f_{CB}, f_{CD}, f_{DB} & \geq & 0
\end{array}$$

III.3.9 Considere que x_i denota a percentagem do objecto i a colocar na mochila.

$$\begin{array}{lrcl}
Maximizar & 7x_1 + 9x_2 + 18x_3 + 13x_4 + 17x_5 + 20x_6 & & \\
Sujeito\ a & 2x_1 + 4x_2 + 6x_3 + 7x_4 + 9x_5 + 10x_6 & \leq & 18 \\
& x_1 & \leq & 1 \\
& x_2 & \leq & 1 \\
& x_3 & \leq & 1 \\
& x_4 & \leq & 1 \\
& x_5 & \leq & 1 \\
& x_6 & \leq & 1 \\
& x_1, x_2, x_3, x_4, x_5, x_6 & \geq & 0
\end{array}$$

III.3.10 Este programa linear á ilimitado.

III.3.11 O programa dual, definido abaixo, é impossível.

$$\begin{array}{lrcl}
min & y_1 + 2y_2 + 3y_3 & & \\
s.a & -y_1 - y_3 & \geq & 1 \\
& -2y_1 + y_2 & \geq & -1 \\
& -3y_1 - y_2 + y_3 & \geq & -2 \\
& x_1, x_2, x_3 & \geq & 0
\end{array}$$

III.3.12 $2n$ e $2m$, respectivamente.

III.3.13
As afirmações 1 e 3 são verdadeiras, e as afirmações 2, 4 e 5 são falsas.

III.3.14
Valor óptimo: 0. $x_1 = 3, x_2 = 0$.

III.3.15 O valor óptimo do primal é 10, com $x_1 = 0$, $x_2 = 3$ e $x_3 = 1$. O valor óptimo do dual é também 10, com $y_1 = 1$, $y_2 = 0$ e $y_3 = 1$.

III.3.16
Valor óptimo: -13. Solução: $x_1 = 3$, $x_2 = 7$. Forma *slack* final:

$$\begin{array}{rcl}
z & = & 13 - x_3 - x_5 \\
x_1 & = & 3 - x_5 \\
x_2 & = & 7 - x_3 + x_5 \\
x_4 & = & 8 - x_3 + 2x_5
\end{array}$$

III.3.17

Exequível ?	x_1	x_2	x_3
Não	0	0	2

III.3.18

z	x_1	x_2
1800	20	60

III.3.19

Objectivo	x_1	x_2
9	3	0

III.3.20

Objectivo	x_1	x_2
11	5	1

III.3.21

Óptimo?	x_1	x_2	x_3
Sim	4	0	0

III.3.22

$$\begin{array}{lrcl} \text{Maximizar} & 2x_1 + x_2^{'} - x_2^{''} - 3x_3 & & \\ \text{Sujeito a} & x_1 + 4x_2^{'} - 4x_2^{''} - 4x_3 & \leq & -5 \\ & -2x_1 + x_2^{'} - x_2^{''} + x_3 & \leq & 10 \\ & 2x_1 - x_2^{'} + x_2^{''} - x_3 & \leq & -10 \\ & -4x_2^{'} + 4x_2^{''} + 5x_3 & \leq & -6 \\ & x_1, x_2^{'}, x_2^{''}, x_3 & \geq & 0 \end{array}$$

III.3.23

Objectivo	x_1	x_2	x_3
6	2	8	0

III.3.24

$$\begin{array}{lrcl} \text{Minimizar} & -2y_1 + 10y_2 - 5y_3 + 6y_4 & & \\ \text{Sujeito a} & y_1 - 2y_2 + y_3 & \geq & 3 \\ & 4y_1 + y_2 - y_3 + 4y_4 & \geq & -1 \\ & -4y_1 + y_2 - 6y_4 & \geq & -3 \\ & y_1, y_2, y_3, y_4 & \geq & 0 \end{array}$$

III.3.25

Objectivo	x_1	x_2	x_3
24	4	0	8

III.3.26

	Objectivo	x_1	x_2	x_3
Óptima	34	14	2	4

III.3.27

Exequível?	Sim
Objectivo	$\min : 4y_1 + 6y_2 + 20y_3$
Sujeito a	$y_1 + y_2 + y_3 \geq 2$ $-y_1 - 2y_2 + y_3 \geq 1$ $-2y_1 - y_2 + y_3 \geq 1$ $y_1, y_2, y_3 \geq 0$

III.3.28

	Objectivo	x_1	x_2	x_3
Primeira	-8	2	0	0
Ótima	-1	9	7	0

	Objectivo	y_1	y_2
Dual Ótima	-1	1	1

III.3.29

Objectivo	x_1	x_2	x_3
8	2	2	0

III.3.30

São aceites duas soluções possíveis para este problema:

Objectivo	x_1	x_2	x_3
8	2	2	0

Objectivo	x_1	x_2	x_3
5	1	0	2

III.3.31

Objectivo	x_1	x_2
-12	-2	-4

III.3.32

Valor óptimo: 28. $y_1 = 0, y_2 = 1/6, y_3 = 2/3$.

III.3.33

$$\begin{array}{lrcl} \text{Minimizar} & 10y_1 - 5y_2 + 4y_3 - 3y_4 & & \\ \text{Sujeito a} & y_1 - 2y_2 + 3y_4 & \geq & 3 \\ & y_1 + y_3 - 5y_4 & \geq & -2 \\ & 4y_2 - 3y_3 + 4y_4 & \geq & 1 \\ & y_1 + 3y_2 - 3y_4 & \geq & 0 \\ & y_1 - y_2 + y_4 & \geq & -4 \\ & y_1, y_2, y_3, y_4 & \geq & 0 \end{array}$$

III.3.34

Primeira	Z =-16	x_1 =4	x_2 =0	x_3 =0
Ótima	Z = -2	x_1 =18	x_2 =14	x_3 =0
Dual Ótima	Z' =-2		$y_1 = 1$	$y_2 = 1$

III.3.35

Primeira	Z =-8	x_1 =2	x_2 =0	x_3 =0
Ótima	Z = -1	x_1 =9	x_2 =7	x_3 =0
Dual Ótima	Z' =-1		$y_1 = 1$	$y_2 = 1$

III.4. Emparelhamento de Cadeias de Caracteres

III.4.1

P	a	b	a	b	b	a	b	a	b	b	a	b	a	b	b	a
π	0	0	1	2	0	1	2	3	4	5	6	7	8	9	10	11

III.4.2

T	a	a	a	b	a	a	a	b	b	a	a	b	a	a	a
q	1	2	2	3	4	5	6	7	8	9	2	3	4	5	6

III.4.3

Dado que os k padrões têm o mesmo comprimento, basta calcular a chave para cada padrão e, para cada caracter do texto, procurar a chave actual na lista de chaves dos padrões. O processo de procura de uma chave na lista de chaves dos padrões pode ser feita em tempo $O(\log k)$ através de pesquisa binária.

III.4.4

$\delta(3, \mathtt{a})$	$\delta(6, \mathtt{b})$	$\delta(9, \mathtt{a})$	$\delta(12, \mathtt{b})$	$\delta(15, \mathtt{a})$
1	7	3	0	16

III.4.5

P	a	b	a	b	a	a	b	b

i	1	2	3	4	5	6	7	8
q	1	2	3	4	5	4	5	4

P	b	a	b	a	b	b	a	a

i	1	2	3	4	5	6	7	8
q	0	1	2	3	4	5	4	5

III.4.6

Verificar que o comprimento da sequência A é igual ao comprimento da sequência B. Aplicar o algoritmo de Knuth-Morris-Pratt, onde o texto é AA e o padrão é B. O padrão B é encontrado em AA sse B é uma rotação cíclica de A. O algoritmo de Knuth-Morris-Pratt termina em tempo linear.

III.4.7

q	0	1	2	3	4	5	6	7	8	9	10	11
$\delta(q,a)$	1	1	3	1	5	6	7	1	9	1	11	6
$\delta(q,b)$	0	2	0	4	0	4	2	8	0	10	0	4

III.4.8 Existem 4 falso positivos.

III.4.9 $O(k(n+m))$

III.4.10

P	a	a	b	a	b	a	a	b	a	a	b	a	b	a	b	a	a	b	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
π	0	1	0	1	0	1	2	3	4	2	3	4	5	6	0	1	2	3	0

III.4.11

i	$-$	a	b	a	b	a	a	a	b	a	a	a	b	a	a
$estado$	0	1	2	3	2	3	4	5	6	7	4	5	6	7	4

III.4.12 Texto: $aaaa \ldots aaaa$, com n caracteres.
Padrão: $aaaa \ldots aaaa$, com m caracteres.

III.4.13 Uma vez que apenas queremos identificar possíveis ocorrências, basta calcular os chaves para os padrões em tempo $O(km)$, guardar estas chaves num vector ordenado em tempo $O(k \log k)$, e executar uma pesquisa binária nesse vector em tempo $O(\log k)$ para $n - m + 1$ posições do texto, i.e., $O(n)$ posições. Portanto, a menor complexidade assimptótica é neste caso $O((n+k)\log k + mk)$.

III.4.14

	a	a	b	a	b	b	a	a	b	b	a	b	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
q	1	1	2	1	2	3	4	1	2	3	4	5	3	4	5

III.4.15 $n^2 + 3n + 2$

III.4.16

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\pi[i]$	0	0	0	0	1	0	0	1	2	3	4	5	6	1	2	3

III.4.17

$Estado$	0	1	2	3	4	5	6	Total
$Numero$	3	3	3	1	2	1	1	14

III.4.18

Máximo	Exemplo
m-1	$P = aa \cdots a$

III.4.19

Válidos	Spurious
1	5

III.4.20

	$\pi[2]$	$\pi[6]$	$\pi[10]$	$\pi[13]$	$\pi[14]$
Valor	1	0	4	7	0

III.4.21

Expressão	2

III.4.22

	$\delta(2,b)$	$\delta(4,a)$	$\delta(5,a)$	$\delta(9,a)$	$\delta(9,b)$
Estado	2	0	6	4	1

III.4.23

Expressão	$n+1$

III.4.24

	$\delta(5,c)$	$\delta(7,a)$	$\delta(11,b)$	$\delta(13,c)$	$\delta(14,a)$
Valor	1	4	0	1	7

III.4.25

Expressão	$\sum_{i=1}^{n-1} i+3 = \frac{n^2-n}{2}+3$

III.4.26

Expressão	$2n+1$

III.4.27

	$\pi[5]$	$\pi[8]$	$\pi[10]$	$\pi[13]$	$\pi[14]$
Valor	0	3	1	4	1

III.4.28 Se P e T tiverem tamanhos diferentes o algoritmo retorna que não são rotações cíclicas. Caso P e T tenham o mesmo tamanho executar Knuth-Morris-Pratt com P contra a string TT, i.e., T repetido duas vezes. Se for encontrada um ocorrência de P em TT então T é uma rotação cíclica de P, caso contrário não é. O tempo deste algoritmo é $O(n+m)$, onde n é o tamanho de P e m é o tamanho de T.

III.4.29

	t_0	t_1	t_2	t_3	t_4	t_5
Valor	3	3	8	1	4	5

III.4.30

	$\delta(7,b)$	$\delta(10,b)$	$\delta(11,b)$	$\delta(4,b)$	$\delta(4,a)$
Valor	3	5	3	5	2

III.4.31

P	a	a	b	a	a	a	a	b	a	a	b	a	a	a	a
π	0	1	0	1	2	2	2	3	4	5	3	4	5	6	7

III.4.32

	$\delta(2,b)$	$\delta(5,b)$	$\delta(6,a)$	$\delta(13,b)$	$\delta(14,b)$
Valor	2	2	0	8	1

III.4.33

Expressão	$4n+2$

III.4.34

	$\pi[6]$	$\pi[11]$	$\pi[15]$	$\pi[17]$	$\pi[18]$
Valor	1	6	3	5	3

III.4.35

Expressão	$2n-2$

III.4.36

c[2,3]	c[2,6]	c[3,7]	c[4,5]	c[8,7]
2	2	3	3	5

III.4.37

$\delta(2, A)$	$\delta(3, A)$	$\delta(4, A)$	$\delta(5, B)$	$\delta(9, C)$
3	1	3	0	5

III.4.38

	$\pi[4]$	$\pi[6]$	$\pi[8]$	$\pi[9]$	$\pi[10]$
Valor	1	2	4	5	3

III.4.39

$\delta(2, a)$	$\delta(5, b)$	$\delta(6, a)$	$\delta(9, a)$	$\delta(10, a)$
2	3	2	6	4

Parte IV.
Classes Complexidade & Algoritmos de Aproximação

IV.1. Classes Complexidade

IV.1.1

a.	b.	c.	d.	e.
V	F	D	V	D

IV.1.2

a	b	c	d	e	f
V	V	V	D	V	F

IV.1.3

a) Sim. Uma vez que X é NP-completo, temos que qualquer problema NP se reduz polinomialmente a X, i.e., qualquer instância de um problema NP Y pode ser transformada numa instância de X em tempo polinomial e a resposta obtida para a instância de X é válida para a instância original de Y. Portanto, se existir um algoritmo que resolve qualquer instância de X em tempo polinomial, tem-se que também conseguimos resolver qualquer instância de um qualquer problema NP em tempo polinomial, basta transformar essa instância para uma instância de X e utilizar o algoritmo, em tempo polinomial. Como SAT é um problema NP, SAT poderia ser igualmente resolvido em tempo polinomial.

b) Não. A transformação das instâncias de Y para instâncias de X pode aumentar o tamanho da instância não linearmente. Por exemplo, se a transformação das instâncias de Y para instâncias de X aumentar o tamanho em $O(m^2)$, a complexidade do algoritmo para resolver as instâncias de Y seria $O(m^6)$.

c) Não. A complexidade do algoritmo é dada em termos do tamanho da instância, que neste caso é o inteiro n, cujo tamanho é dado em termos do número de bits necessários para representar n, i.e., o tamanho da instância é $O(\log n)$. Uma vez que a complexidade deste algoritmo é $O(\sqrt{n})$ e que o tamanho da instância é $|n| = \log n$, tem-se que a complexidade do algoritmo em termos do tamanho da instância é $O(2^{\frac{|n|}{2}})$, o que não é polinomial em $|n|$.

IV.1.4

D	D	V	V	V	F

IV.1.5

V	D	V	D	F	V

IV.1.6

a	b	c	d
F	D	V	F

IV.1.7

	a)	b)	c)	d)	e)
Resposta	F	D	D	V	V

IV.1.8

	a)	b)	c)	d)	e)
Resposta	V	D	V	V	F

IV.1.9

	a)	b)	c)	d)	e)
Resposta	V	F	F	V	F

IV.1.10

	a)	b)	c)	d)	e)
Resposta	V	F	V	F	V

IV.1.11 O problema 2CNF-SAT $\in P$. A fórmula pode ser resolvida recorrendo a uma representação de grafos.

- Começamos por criar um grafo dirigido da seguinte forma:
  - Para cada variável x_i da fórmula criamos dois vértices para representar x_i e $\neg x_i$.
  - Para cada cláusula binária $(l_i \vee l_j)$ criamos dois arcos no grafo: $(\neg l_i, l_j)$ e $(\neg l_j, l_i)$
- Identificamos os componentes fortemente ligados (SCC) no grafo
- Se existe um SCC com x_i e $\neg x_i$, então a instância não pode ser satisfeita. Caso contrário, é satisfazível.

O algoritmo é linear no número de variáveis e cláusulas da fórmula.

IV.1.12

	a)	b)	c)	d)	e)
Resposta	V	F	V	D	V

IV.1.13

	a)	b)	c)	d)	e)
Resposta	V	F	V	V	D

IV.1.14

	a)	b)	c)	d)	e)
Resposta	D	V	D	V	F

IV.1.15

	a)	b)	c)	d)	e)
Resposta	V	V	V	F	V

IV.1.16

	a)	b)	c)	d)	e)
Resposta	V	D	V	F	V

IV.1.17

	a)	b)	c)	d)	e)
Resposta	V	V	F	V	F

IV.2. Redução de Problemas e Algoritmos de Aproximação

IV.2.1

Em primeiro lugar é necessário provar que *ISV*$\in NP$, ou seja, que o certificado do problema é verificável em tempo polinomial. Considere-se como certificado um conjunto independente de vértices. O algoritmo de verificação determina se todos os vértices são independentes, ou seja, se não são adjacentes a nenhum outro vértice do conjunto, e se são pelo menos em número k. Esta verificação pode ser efectuada em tempo $O(V+E)$, sendo portanto polinomial na dimensão do problema.

Resta provar que *ISV*$\in$ *NP-Difícil*. Para tal, vamos provar que $CLIQUE \leq_P ISV$. A transformação pode ser a seguinte:

- seja G o grafo do problema *CLIQUE*, determinar $\overline{G} = (V, \overline{E})$, ou seja, o grafo complementar de G, que contém todos os arcos que não estão em G;
- utilizar $\overline{G}$ como o grafo para o problema *ISV*;

- o valor de k para o problema de *ISV* é igual ao valor de k para a instância do problema de *CLIQUE*.

A solução da instância do problema de *ISV* resultante define a solução da instância do problema original de *CLIQUE*, dado que o complemento de uma clique é um conjunto independente de vértices. A transformação é feita em tempo $O(V+E)$, sendo portanto polinomial.

IV.2.2 Existem vários algoritmos de aproximação para este problema, com limites da razão distintos. Um dos algoritmos mais simples, consiste em:

- considerar as tarefas por uma ordem qualquer;
- agendar uma dada tarefa para a máquina que, atendendo às tarefas que já foram agendadas, estará disponível mais cedo.

O custo de uma solução é o tempo decorrente desde o início até a última máquina ter terminado o processamento das respectivas tarefas. O custo de qualquer solução óptima é limitado inferiormente pelo melhor caso, que corresponde a todas as máquinas terminarem as respectivas tarefas ao mesmo tempo, ou seja,

$$C^* \geq \frac{\sum_{i=1}^{n} d_i}{m} \tag{2}$$

Considerando que a tarefa j, com início em t_j, é a última tarefa a ser terminada, então,

$$C = t_j + d_j \tag{3}$$

Atendendo ao algoritmo de aproximação descrito, t_j corresponde ao primeiro instante de tempo em que uma das máquinas terminou as respectivas tarefas, considerando que todas as tarefas foram agendadas, com excepção da tarefa j. O valor máximo de t_j corresponde ao caso em que todas as máquinas terminaram a execução das respectivas tarefas ao mesmo tempo, ou seja,

$$t_j \leq \frac{(\sum_{i=1}^{n} d_i) - d_j}{m} \tag{4}$$

Assim, considerando as Eqs. (3) e (4), obtém-se

$$C \leq \frac{(\sum_{i=1}^{n} d_i) - d_j}{m} + d_j \quad \Leftrightarrow \quad C \leq \frac{\sum_{i=1}^{n} d_i}{m} + (1 - \frac{1}{m})d_j \tag{5}$$

A partir das Eqs. (2) e (4), e sabendo que $d_j \leq C^*$ e que $\left(1 - \frac{1}{m}\right) \leq 1$, podemos concluir que

$$C \leq 2\, C^* \tag{6}$$

IV.2.3

Em primeiro lugar é necessário provar que *BIN-PACKING*$\in NP$, ou seja, que o certificado do problema é verificável em tempo polinomial. Considere-se como certificado a atribuição de cada um dos n objectos ao respectivo contentor, representado por um índice. Os índices dos contentores são números inteiros sequenciais, começando em 1. O algoritmo de verificação determina se o índice do contentor a que corresponde cada objecto não excede k e se a soma dos volumes dos objectos contidos dentro de cada contentor não excede 1. O tempo total necessário para realizar ambas as operações é $O(n)$, sendo portanto polinomial na dimensão do problema.

É ainda necessário provar que *BIN-PACKING*$\in$ *NP-Difícil*. Para tal, vamos provar que *SET-PARTITION*$\leq_P$*BIN-PACKING*. A transformação pode ser a seguinte:

- seja $S = \{s_1, s_2, \ldots, s_m\}$ o conjunto de números inteiros do problema de *SET-PARTITION*;

- calcular $\alpha = \sum_{i=1}^{m} s_i/2$;
- o conjunto dos volumes dos objectos do problema de *BIN-PACKING* será então dado por $V = \{v_1, v_2, \ldots, v_n\}$, onde $v_i = s_i/\alpha$, para $i = 1, 2, \ldots, n$ e $n = m$; observar que $\sum_{i=1}^{n} v_i = 2$;
- o valor de k para o problema de *BIN-PACKING* será $k = 2$.

A solução da instância do problema de *BIN-PACKING* resultante define a solução da instância do problema original de *SET-PARTITION*, dado que se for possível guardar em $k = 2$ contentores de capacidade unitária os n objectos do problema de *BIN-PACKING*, cuja soma dos volumes é exactamente 2, então existe uma partição $(B, \overline{B})$ dos elementos de S do problema de *SET-PARTITION* tal que $\sum_{x \in B} x = \sum_{x \in \overline{B}} x$. A transformação é feita em tempo $O(m)$, sendo portanto polinomial.

IV.2.4

Existem vários algoritmos possíveis para obter uma solução aproximada para o problema de *BIN-PACKING*$_{opt}$. Um dos algoritmos mais simples consiste em utilizar uma estratégia *greedy*, em que os objectos são processados por qualquer ordem, e cada objecto é colocado no primeiro contentor com capacidade para o guardar. Se não existir nenhum contentor nestas condições, é adicionado um novo contentor, sendo o objecto nele guardado. Seguindo esta estratégia, é fácil provar que durante o processamento dos objectos não existe mais do que um contentor com capacidade disponível igual ou superior a 1/2.

Seja C^* o número óptimo de contentores, produzido pelo algoritmo *BIN-PACKING*$_{opt}$, então $C^* \geq \sum_{i=1}^{n} v_i$, dado que no melhor caso é possível encher completamente todos os contentores (que têm capacidade unitária). Como foi referido, para o algoritmo de aproximação proposto, existe no máximo 1 contentor com capacidade disponível igual ou superior a 1/2, pelo que todos os restantes contentores têm que guardar um volume superior a 1/2. Assim, seja C o número de contentores obtido pelo algoritmo de aproximação proposto, então $\sum_{i=1}^{n} v_i > \frac{C-1}{2}$, uma vez que se $C - 1$ contentores têm cada um que guardar um volume superior a 1/2, então $\frac{C-1}{2}$ tem que limitar inferiormente o volume total dos objectos guardados. De $C^* \geq \sum_{i=1}^{n} v_i$ e $\sum_{i=1}^{n} v_i > \frac{C-1}{2}$ conclui-se que $C^* > \frac{C-1}{2}$ e logo que $C \leq 2C^*$, pelo que $\rho(n) = 2$.

IV.2.5

O problema de decisão associado a este problema de optimização define-se da seguinte forma: dado um grafo dirigido e pesado $G = (V, E, w)$ com fonte s e destino t, existe um fluxo f para G tal que cada arco (u, v) ou está vazio ou está saturado, e o fluxo total f é maior ou igual a k? Designamos este problema por 01-FLOW e mostramos que é NP-completo.

Primeiro temos de verificar que está em NP. Dada uma instância positiva (G, s, t, k), um certificado é uma lista de arcos saturados, com no máximo tamanho $|E|$. Portanto podemos verificar que, para qualquer vértice v excepto s e t, o fluxo que entra em v é igual ao fluxo que sai de v, e que o fluxo que sai de s é pelo menos k, em tempo $O(|V||E|)$. Logo, 01-FLOW está em NP.

Para mostrar que 01-FLOW é NP-difícil basta mostrar que SUBSET-SUM $\leq_p$ FLOW-01. Seja (X, t) uma instância de SUBSET-SUM. Definimos G com vértices a, b, c e um vértice v_i para cada $x_i \in X$. Para cada $1 \leq i \leq n$, adicionamos os arcos (a, v_i) e (v_i, b) com capacidade x_i. Adicionamos também um arco (b, c) com capacidade t. Portanto, a instância de 01-FLOW é dada por (G, a, c, t). Falta mostrar que, (1) se $X_0 \subseteq X$ é tal que $\Sigma_{x \in X_0} x = t$, então G tem um fluxo válido de pelo menos t, e (2) se G tem um fluxo válido de pelo menos t, então existe $X_0 \subseteq X$ tal que $\Sigma_{x \in X_0} x = t$.

Prova de (1): para cada $x_i \in X_0$, saturamos os arcos de a para v_i e de v_i para b com fluxo x_i; logo o fluxo para b soma precisamente t e podemos transferi-lo para c

pelo arco (b, c) com capacidade t; portanto temos um fluxo válido de a para c em G de precisamente t unidades.

Prova de (2): qualquer fluxo válido em G pode ter no máximo t unidades, uma vez que não podemos passar mais do que t unidades pelo arco (b, c); consideremos um fluxo válido com, portanto, t unidades de fluxo; como não existe ganho ou perda de fluxo nos vértices v_i e b, tem de existir um conjunto de arcos (v_i, b) com fluxo diferente de zero (e saturados por definição do problema 01-FLOW), tal que a soma das capacidades é t; seja X_0 o conjunto constituído pelos x_i tais que o arco (v_i, b) está saturado, que por construcção verifica $\Sigma_{x \in X_0} x = t$.

IV.2.6 A solução para este problema consiste em trocar as polaridades atribuições de valores para o algoritmo descrito nas aulas teóricas para o problema Horn-SAT, Cap. 34. Nomeadamente:

```
CHornSAT(φ)
1  while (∃ cláusulas com literal negativo x_i)
2        do atribuir x_i = 0
3           satisfazer clásulas com ¬x_i
4           reduzir cláusulas com x_i
5           if (existe cláusula vazia)
6              then eliminar atribuições
7                   return FALSE
8  atribuir x_j = 1 às variáveis ainda não atribuídas
9  return TRUE
```

IV.2.7

O problema está em P. Considere uma cláusula $(l_1 \vee l_2 \vee l_3)$. A definição do problema requer:

$$\begin{aligned} & \neg(\neg l_1 \wedge \neg l_2) \wedge \neg(\neg l_1 \wedge \neg l_3) \wedge \neg(\neg l_2 \wedge \neg l_3) \\ \equiv \quad & (l_1 \vee l_2) \wedge (l_1 \vee l_3) \wedge (l_2 \vee l_3) \end{aligned}$$

Assim, cada instância do problema AL2LC pode ser representada como uma fórmula 2CNFSAT, que está em P. A redução é realizada em tempo polinomial, pelo que AL2LC está em P.

IV.2.8 Prova de que GCM é NP-completo, por redução de CLIQUE:

a. GCM $\in$ NP. Dada uma escolha de V_1', V_2', e de um mapa entre $V_1 \setminus V_1'$ e $V_2 \setminus V_2'$, i.e., um certificado, podemos verificar se os grafos obtidos têm pelo menos b vértices e se são idênticos em tempo polinomial.

b. CLIQUE $\leq_P$ GCM. Dada um grafo $G = (V, E)$ e um valor k, construir uma instância do problema de decisão GCM da forma seguinte: $G_1 = G$ e G_2 é um sub-grafo completo com dimensão k. O valor b deve ser igual a k. G têm um sub-grafo completo com tamanho k se e só se removendo $|V| - k$ vértices de G_1 (i.e., $|V_1'| = |V| - k$, restando k vértices em G_1) o grafo obtido é idêntico a G_2, i.e. um subgrafo completo com tamanho k.

IV.2.9 Prova de que 3MaxSAT é NP-completo, por redução de 3CNFSAT.

a. 3MaxSAT $\in$ NP. Dada uma atribuição de valores às variáveis, verificar quais as clásulas satisfeitas e não satisfeitas. Aceitar se número de clásulas satisfeitas não inferior a k; caso contrário rejeitar. O algoritmo de verificação corre em tempo linear no tamanho da fórmula.

b. 3CNFSAT $\leq_P$ 3MaxSAT. Dada uma fórmula φ de 3CNFSAT, com m cláusulas, criar uma instância de 3MaxSAT, em que $k = m$. A redução é executada em tempo polinomial. A correcção da redução é imediata.

IV.2.10 P-CNFSAT $\in$ P: basta atribuir valor 1 a todas as variáveis. Dado que P $\subseteq$ NP, então P-CNFSAT $\in$ NP. Dado que P-CNFSAT $\in$ P, o problema P-CNFSAT não é NP-completo, assumindo P $\neq$ NP.

IV.2.11 N-CNFSAT $\in$ P: basta atribuir valor 0 a todas as variáveis. Dado que N $\subseteq$ NP, então N-CNFSAT $\in$ NP. Dado que N-CNFSAT $\in$ P, o problema N-CNFSAT não é NP-completo, assumindo P $\neq$ NP.

IV.2.12

O problema PN-CNFSAT é NP-completo. É imeditato concluir que o problema PN-CNFSAT está em NP por analogia com CNFSAT ou qualquer das suas variantes. Para concluir a prova de que PN-CNFSAT é NP-completo, reduzimos CNFSAT a PN-CNFSAT:

a. Seja F uma fórmula de CNFSAT.

b. Construir uma fórmula F' de PN-CNFSAT a partir de F, da forma seguinte:

   (a) Para cada cláusula $c_i \in F$, sejam $p(c_i)$ e $n(c_i)$ os literais positivos e negativos de c_i, respectivamente.

   (b) Criar duas cláusulas $c_{i1} = p(c_i) \vee y_i$ e $c_{i2} = n(c_i) \vee \neg y_i$, em que y_i é uma nova variável.

   (c) Podemos concluir que c_{i1} é uma cláusula de N-CNFSAT, e c_{i2} é uma cláusula de P-CNFSAT.

c. É imediato concluir que $F' \in$ PN-CNFSAT e que a redução é de tempo polinomial (aliás é de tempo linear).

d. A prova de que F é satisfeita se e só se F' é satisfeita é simples; basta adaptar as demonstrações relativas a 3CNFSAT dadas nas aulas.

IV.2.13

Em primeiro lugar provamos que ILP 0-1 $\in$ NP. Consideramos como certificado a atribuição de valores 0 ou 1 aos elementos de x. O algoritmo de verificação valida se essa atribuição satisfaz todas as restrições do problema ($Ax \leq b$). O algoritmo é polinomial ($O(nm)$), pelo que ILP 0-1 $\in$ NP.

De seguida provamos que ILP 0-1 $\in$ NP-Dificil usando uma redução a partir de 01_KNAPSACK, ou seja, 01_KNAPSACK $\leq_p$ ILP 0-1. A redução é a seguinte:

- Para cada objecto i do conjunto S criamos uma variável x_i na instância ILP 0-1. x_i toma valor 1 se o objecto é incluido em S' e 0 caso contrário.
- Definimos uma restrição $\sum_{i=1}^{n} w_i x_i \leq W$ por forma a garantir que a soma dos pesos dos objectos seleccionados é menor ou igual a W.
- Definimos ainda uma restrição $\sum_{i=1}^{n} -v_i x_i \leq -K$ para garantir que a soma dos valores dos objectos seleccionados é maior ou igual a K.

Esta redução tem complexidade linear. As variáveis x_i com valor 1 na solução da instância ILP 0-1 definem os objectos do subconjunto S' que é solução da instância 01_KNAPSACK.

IV.2.14 A prova tem duas partes. Primeiro é necessário provar que 01_KNAPSACK $\in NP$. Considere-se como certificado uma sequência de objectos S'. O algoritmo de verificação determina se S' é um subconjunto de S, se a soma do valor dos objectos de

S' é maior ou igual a K e se a soma dos pesos dos objectos de S' é menor ou igual a W. Facilmente podemos idealizar um algoritmo de verificação polinomial no tamanho das sequências. Logo, 01_KNAPSACK $\in NP$.

É ainda necessário provar que 01_KNAPSACK $\in NP - Hard$. Neste caso, vamos provar que SUBSET-SUM $\leq_p$ 01_KNAPSACK. A transformação pode ser a seguinte:

- Considere-se que o conjunto S do problema SUBSET-SUM tem n elementos.
- Para cada elemento i tal que $i \in S$, temos um objecto de valor i e peso i no problema 01_KNAPSACK
- Definimos que K e W no problema 01_KNAPSACK são iguais a t

Esta redução tem complexidade linear no tamanho do conjunto S. Para além disso, a solução da instância 01_KNAPSACK produzirá uma sequência de objectos S' tal que a soma dos pesos e dos valores dos elementos de S' são iguais a t (porque K e W são iguais a t). Como a cada objecto de S' corresponde um valor no conjunto S, então estamos a seleccionar uma subsequência de S cuja soma é igual a t.

IV.2.15 A prova tem duas partes. Primeiro é necessário provar que 01_KNAPSACK $\in NP$. Considere-se como certificado uma sequência de objectos S'. O algoritmo de verificação determina se S' é um subconjunto de S, se a soma do valor dos objectos de S' é maior ou igual a K e se a soma dos pesos dos objectos de S' é menor ou igual a W. Facilmente podemos idealizar um algoritmo de verificação polinomial no tamanho das sequências. Logo, 01_KNAPSACK $\in NP$.

É ainda necessário provar que 01_KNAPSACK $\in NP - Hard$. Neste caso, vamos provar que SET-PARTITION $\leq_p$ 01_KNAPSACK. A transformação pode ser a seguinte:

- Considere-se que o conjunto S do problema SET-PARTITION tem n elementos.
- Para cada elemento i tal que $i \in S$, temos um objecto de valor i e peso i no problema 01_KNAPSACK.
- Definimos que K e W no problema 01_KNAPSACK são iguais a metade da soma dos valores em S, ou seja, $K = W = \sum_{x \in S} x/2$.

Esta redução tem complexidade linear no tamanho do conjunto S. Para além disso, a solução da instância 01_KNAPSACK produzirá uma sequência de objectos S' tal que a soma dos pesos e dos valores dos elementos de S' são iguais a metade dos valores em S (porque K e W foram definidos dessa forma). Assim, cada objecto de S' corresponde a um valor no conjunto S_1 e os objectos excluidos da mochila correspondem aos valores no conjunto S_2, definindo a solução do problem de SET-PARTITION.

IV.2.16

O problema NAE 3CNF-SAT pertence a NPC.

Para verificar que o problema está em NP é preciso verificar em tempo polinomial se uma solução para o problema é válida. Para tal, dada uma atribuição, é necessário avaliar cada cláusula e contando o número de literais que avaliam para verdadeiro, caso este número seja 1 ou 2 então a cláusula é considerada satisfeita. Caso todas as m cláusulas estejam satisfeitas então a formula é declarada verdadeira, caso contrário é declarada falsa. Cada cláusula é avaliada em tempo $O(1)$, dado que cada literal tambem é avaliado em tempo $O(1)$ e cada cláusula contém exactamente 3 literais. Logo o tempo total deste algoritmo é $O(n+m)$, pelo que é polinomial no tamanho do input, que tem de especificar as m cláusulas e n variáveis da atribuição dada. A parcela $O(n)$ conta o tempo de ler a atribuição.

Para verificar que o NAE 3CNF-SAT é NP-difícil vamos apresentar uma redução a partir de 3CNF-SAT. Dada uma cláusula (l_1, l_2, l_3) é escolhida uma variaável nova x e são criadas duas cláusulas de NAE 3CNF-SAT a (l_1, l_2, x) e a $(l_3, \neg x, F)$.

Queremos mostrar que se a formula original é satisfeita por alguma atribuição então as novas cláusulas também podem ser satisfeitas. Usamos a essencialmente a mesma

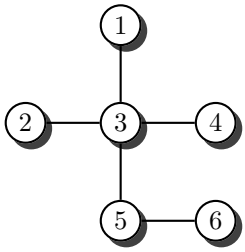
atribuição para ambos os problemas. Suponhamos que l_1 avalia para verdadeiro, nesse caso escolhemos que x avalia para falso e portanto a cláusula (l_1, l_2, x) fica satisfeita em NAE 3CNF-SAT, por outro lado $\neg x$ avalia para verdadeiro e F para falso, pelo que a cláusula $(l_3, \neg x, F)$ também tambem fica satisfeita. O caso em que l_2 avalia para verdadeiro é semelhante. Caso seja l_3 que avalia para verdadeiro a cláusula $(l_3, \neg x, F)$ fica satisfeita, porque F avalia para falso. Neste caso escolhemos para x o contrário do valor que l_1 avalia e portanto a primeira cláusula também fica satisfeita.

Falta também mostrar que se as cláusulas de NAE 3CNF-SAT são satisfeitas por alguma atribuição então tambem é possivel satisfazer a cláusula de 3CNF-SAT. A cláusula $(l_3, \neg x, F)$ está satisfeita e F avalia para falso, pelo que ou l_3 avalia para verdadeiro (e portanto a cláusula (l_1, l_2, l_3) fica satisfeita) ou $\neg x$ avalia para verdadeiro. No segundo caso x avalia para falso, pelo que ou l_1 ou l_2 avalia para verdadeiro, dado que estamos a assumir que (l_1, l_2, x) está satisfeita, em ambos os casos (l_1, l_2, l_3) fica satisfeita.

IV.2.17

Temos que HITTING-SET $\in$ NPC. Para provar que HITTING-SET $\in$ NP assumimos que o certificado é guardado numa array em que é usado o valor de $1 = \texttt{A}[i]$ quando $i \in X$ e o valor $0 = \texttt{A}[i]$ quando $i \notin X$. Para cada conjunto $c \in C$ fazemos a soma do valores $\texttt{A}[i]\texttt{x}$ para os elementos $i \in c$, caso esta soma seja 0, para algum conjunto c então X não é solução da instância. Caso contrário todas a somas são positivas e X é solução da instância. Este algorimo é polinomial, visto que o tempo é majorado por $O(|C| \times |S|)$. Para o exemplo acima estas somas teriam os seguintes valores: $2, 1, 1, 1, 1$.

Para provar que HITTING-SET $\in$ NP-HARD fazemos uma redução a partir do problema de VERTEX-COVER, i.e., VERTEX-COVER $\leq_{\text{P}}$ HITTING-SET. Dado um grafo não dirigido (V, E), escolhemos $S = V$ e k igual ao tamanho da cobertura de vértices pretendida. Para cada arco $(u, v) \in E$ criamos o conjunto $\{u, v\} \in C$. Estes são os únicos conjuntos que a família C contém. Consideremos o seguinte grafo:



Para este grafo a instância gerada teria $C = \{\{1, 3\}, \{2, 3\}, \{3, 4\}, \{3, 5\}, \{5, 6\}\}$.

Falta mostrar que se existe uma cobertura de vértices então existe um conjunto que acerta na instância gerada. Se $A \subseteq V$ é uma cobertura de vértices escolhemos como conjunto que acerta $X = A$. Seja $e \in C$ um elemento da famfilia C, devido á construção da família existe um arco (u, v) de E tal que $e = \{u, v\}$, como A é uma cobertura de vértices temos que ou $u \in A = X$ ou $v \in A = X$. Portanto em ambos os casos o cojunto X acerta no elemento da família e, portanto X é um conjunto certeiro.

Por último queremos mostrar que se existe um conjunto X certeiro então existe uma cobertura de vértices para o grafo. Escolhemos como cobertura $A = X$. Seja (u, v) um arco do grafo. Pela construção da família temos que existe um conjunto $\{u, v\} \in C$. Como X é um conjunto certeiro temos que ou $u \in X = A$ ou $v \in X = A$, pelo que A é uma cobertura de vértices.

IV.2.18

Em primeiro lugar é necessário provar que ILP 0-1 $\in$ NP. Considere-se como certificado uma atribuição às variáveis do problema (vector x). O algoritmo de verificação determina se os valores das variáveis são inteiros 0 ou 1 e se todas as restrições do problema (definidas por $Ax \leq b$) são satisfeitas. O algoritmo de verificação é linear no tamanho das restrições do problema.

De seguida provamos que ILP 0-1 $\in$ NP-Hard a partir de uma redução do problema de BIN-PACKING.

Para cada objecto i ($1 \leq i \leq n$) do problema de BIN-PACKING definimos um conjunto de variáveis x_{ij} onde ($1 \leq j \leq m$). Se a variável x_{ij} tomar valor 1, então o objecto i fica colocado na caixa j.

Como os objectos não podem ser fraccionados, têm que ficar numa única caixa. Logo, para cada objecto i definimos a seguinte restrição: $\sum_{j=1}^{m} x_{ij} = 1$. Note que a restrição de igualdade pode ser trivialmente transformada em desigualdades tal como definido para o problema ILP 0-1.

Finalmente, o peso total dos objectos dentro de cada caixa não pode ser superior a K. Assim, para cada caixa j definimos a seguinte restrição: $\sum_{i=1}^{n} p_i x_{ij} \leq K$

Se a instância do problema ILP 0-1 tiver solução, então a instância do problema de BIN-PACKING também tem solução e a colocação dos objectos nas caixas é dada pelas variáveis x_{ij} que estiverem a 1. Se a instância ILP 0-1 não tem solução, então o problema original de BIN-PACKING também não é satisfazível.

IV.2.19

Em primeiro lugar provamos que ILP 0-1 $\in$ NP. Consideramos como certificado a atribuição de valores 0 ou 1 aos elementos de x. O algoritmo de verificação valida se essa atribuição satisfaz todas as restrições do problema ($Ax \leq b$). O algoritmo é polinomial ($O(nm)$), pelo que ILP 0-1 $\in$ NP.

De seguida provamos que ILP 0-1 $\in$ NP-Dificil usando uma redução a partir de K-COLORING, ou seja, K-COLORING $\leq_p$ ILP 0-1. A redução é a seguinte:

- Para cada vértice i ($1 \leq i \leq |V|$) do grafo e para cada cor j ($1 \leq j \leq K$), criamos uma variável x_{ij} que toma valor 1 se o vértice i for colorido com a cor j. Caso contrário, x_{ij} deverá ter valor 0.
- Como todos os vértices têm que ser coloridos com uma e só uma cor, definimos a seguinte restrição para cada vértice i: $\sum_{j=1}^{K} x_{ij} = 1$.
- Como os vértices adjacentes não podem ter a mesma cor, então para cada arco (u, v) do grafo e para cada cor j adicionamos a seguinte restrição: $x_{uj} + x_{vj} \leq 1$.

Se a instância do problema ILP 0-1 tiver solução, então a instância do problema de K-COLORING também tem solução e a coloração dos vértices é dada pelas variáveis x_{ij} que estiverem a 1. Se a instância ILP 0-1 não tem solução, então o problema original de K-COLORING também não é satisfazível.

IV.2.20

Em primeiro lugar é necessário provar que **DSQL** $\in$ NP. Vamos assumir que `A` é fornecido em forma da array tal que `A`$[i] \in A$. Assumimos que a solução a testar é fornecida na forma de um array `P` tal que `P`$[j] = i$ significa que `A`$[j] \in A_i$. O seguinte algoritmo pode ser utilizado para a soma dos quadrados:

```
j = 0;
while(j < K){
  Q[j] = 0;
  j++;
}
j = 0;
while(j < n){
  Q[P[j]] += A[j];
  j++;
}
s = 0;
j = 0;
```

```
while(j < K){
  s += Q[j]*Q[j];
  j++;
}
```

Caso $\texttt{s} \leq J$ então a solução é aceite. Este algoritmo demora tempo $O(n + K)$ onde n é o tamanho do conjunto A, pelo que é polinomial no tamanho do input.

Para provar que **DSQL** $\in$ NP-Hard iremos reduzir o **PARTITION** a **DSQL**. Dado S criamos uma instância do **DSQL** com $A = S$, $K = 2$ e $J = t^2/2$, onde $t = \sum_{e \in S} e$. Esta redução demora $O(1 + |S|)$, o que é polinomial no tamanho do input.

Se a instância do **PARTITION** tiver solução S' então escolhemos $A_1 = S'$ e $A_2 = S \setminus S'$ como solução do **DSQL**. Para esta escolha as condições do **DSQL** reduzem-se às seguintes:

$$\begin{aligned} &S' \cap (S \setminus S') = \emptyset \\ &S = S' \cup (S \setminus S') \\ &(t/2)^2 + (t/2)^2 \leq t^2/2 \end{aligned}$$

Na última condição utilizamos a seguinte propriedade $\sum_{e \in S'} e = \sum_{e \in S \setminus S'} e = t/2$, que uma solução S' do **PARTITION** verifica.

Para a prova no outro sentido assumimos que A_1 e $A_2 = A \setminus A_1$ é uma solução do **DSQL**, então escolhemos $S' = A_1$ como solução do **PARTITION**. Consideramos $x_0 = \sum_{e \in S'} e$, como por hipótese (A_1, A_2) é solução do **DSQL** temos que $f(x_0) \leq J = t^2/2$. Usamos a propriedade que $f(x_0) \geq t^2/2$ para concluir que $f(x_0) = t^2/2$, o que pela segunda propriedade implica que $x_0 = t/2$. Combinamos estas relações para obter a igualdade final:

$$\sum_{e \in S'} e = (t/2) = t - (t/2) = \sum_{e \in S \setminus S'} e$$

IV.2.21 Em primeiro lugar é necessário provar que **2-CLIQUE** $\in$ NP. Vamos assumir os elementos de V são representados por números de 1 a $|V|$. Uma solução do **2-CLIQUE** é fornecida em forma de array em que $\texttt{A}[i] = 1$ se $i \in C_1$, $\texttt{A}[i] = 2$ se $i \in C_2$ e $\texttt{A}[i] = 0$ caso contrário. É utilizado um contador c que conta o número de entradas de `A` não nulas. Caso $c < k_2$ o certificado é rejeitado. Caso $c \geq k_2$ vamos testar para todos os pares $\texttt{i}, \texttt{j}$ se $\texttt{A[i]} \neq \texttt{A[j]}$ ou $\texttt{A[i]} \neq 0$ ou $(\texttt{i}, \texttt{j}) \in E$. Caso esta condição falhe para algum par o certificado é rejeitado. Este algoritmo demora tempo $O(|V|^2)$, o que é polinomial no tamanho do input.

Para provar que **2-CLIQUE** $\in$ NP-Hard iremos reduzir o **CLIQUE** a **2-CLIQUE**. Dado $G_1 = (V_1, E_1)$ grafo não dirigido e k_1 um inteiro, que formam uma instância do `CLIQUE`, criamos $G_2 = (V_2, E_2)$ também um grafo não dirigido e escolhemos $k_2 = 2k_1$, como instância do `2-CLIQUE`. O grafo G_2 consiste em duas cópias independentes do G_1, onde não existem arcos entre os vértices de cópias diferentes, em cada cópia são mantidos os arcos que existiam no G_1 original. Esta redução demora $O(|V| + |E|)$, o que é polinomial no tamanho do input.

Se a instância do **CLIQUE** tiver solução C então escolhemos C_1 como os vértices de C na primeira cópia de G_1 e C_2 como os vértices de C na segunda cópia de G_1, esta será a solução do `2-CLIQUE`. Como C é um clique temos que C_1 e C_2 também são cliques, dado que C_1 e C_2 foram escolhidos de cópias distintas o C_1 e o C_2 são disjuntos. Finalmente temos que $|C| \geq k$, pelo que $|C_1 \cup C_2| = |C_1| + |C_2| = |C| + |C| \geq k_1 + k_1 = k_2$.

Se a instância do **2-CLIQUE** tiver solução C então escolhemos temos $|C_1 \cup C_2| = |C_1| + |C_2| \geq k_2 = 2k_1$, onde C_1 e C_2 são cliques disjuntos. Nesse caso pelo menos um deles deve conter pelo menos k_1 vértices, vamos assumir, sem perda de generalidade, que $|C_1| \geq k_1$. Nesse caso C_1 é solução do problema `CLIQUE` original.