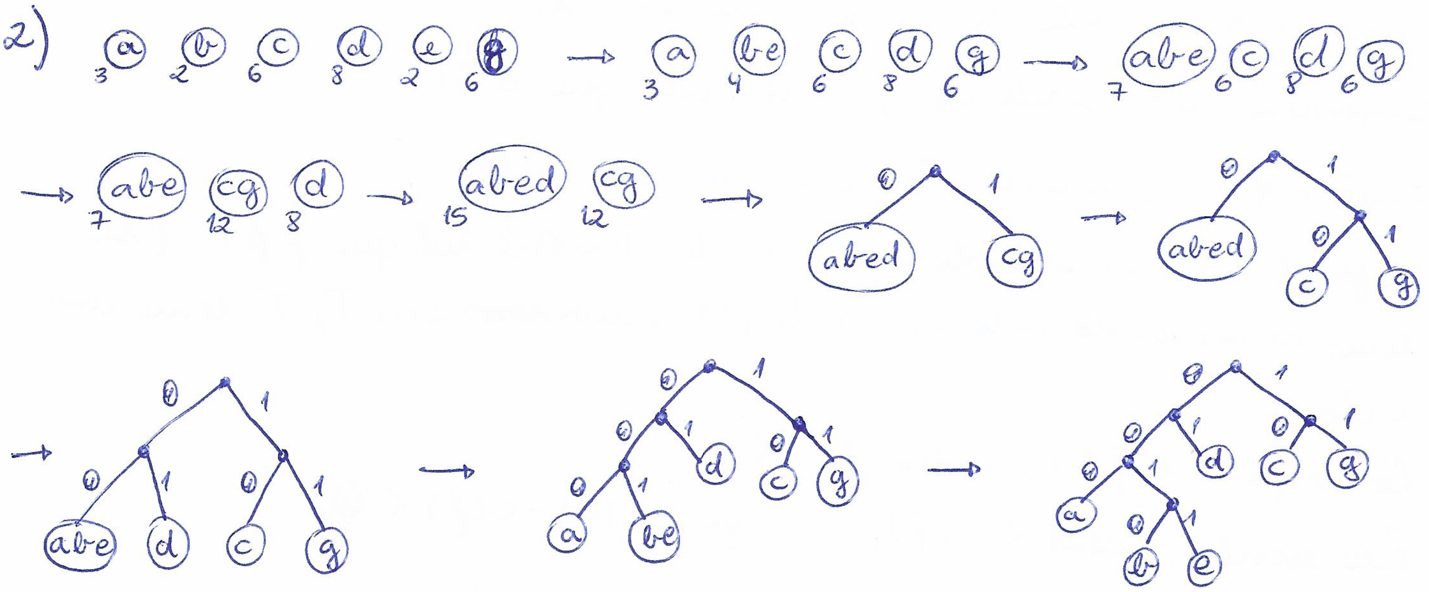
Lista 3

1) Não é ótimo. Por exemplo, se $v_1 = 2, w_1 = 1, v_2 = 10, w_2 = 10$ e $W = 10$, o algoritmo escolhe o item 1 apenas, com valor 2, sendo que a solução ótima é o item 2.

2)



3)

```
KRUSKAL (G,c)
  ordene as arestas de forma crescente no custo
  renomeie-as 1,2,...,m = |E| tal que c(1) ≤ ... ≤ c(m)
  T = ∅
  sejam LIDER, TAM e COMP vetores de tamanho n globais e indexados por vértices
  para todo v ∈ V
      LIDER[v] = v
      TAM[v] = 1
      COMP[v] = lista contendo v
  para i = 1 a m
      seja i = uv
      se FIND(u) ≠ FIND(v)
          T = T ∪ {i}
          UNION(FIND(u), FIND(v))
  retorne T
```

```
FIND (v)
   retorne LIDER[v]
```

```
UNION (x, y)
   se TAM[x] < TAM[y]:
      para todo v em COMP[x]
          LIDER[v] = y
      TAM[y] = TAM[x] + TAM[y]
      COMP[y] = concatena COMP[x] e COMP[y]
   senão:
      para todo v em COMP[y]
          LIDER[v] = x
      TAM[x] = TAM[x] + TAM[y]
      COMP[x] = concatena COMP[x] e COMP[y]
```

4) Suponha por contradição que existem duas árvores distintas T e T' que são geradoras de custo mínimo $(c(T) = c(T'))$.

Sendo diferentes, elas possuem algumas arestas diferentes.

Seja $X = \{e : e \in T \text{ e } e \notin T' \text{ ou } e \notin T \text{ e } e \in T'\}$.

Seja $e \in X$ a aresta de menor custo em X.

Suponha, sem perda de generalidade, que $e \in T$.

Então $e \notin T'$ e, portanto $T' \cup \{e\}$ tem um ciclo.

Seja $f \in T'$ uma aresta do ciclo de $T' \cup \{e\}$ tal que $f \notin T$ (se todas as arestas do ciclo em $T' \cup \{e\}$ estivessem em T, T teria um ciclo).

Como $f \in T'$ e $f \notin T$, então $f \in X$.

Pela escolha, $c(e) < c(f)$, ou seja, $c(e) - c(f) < 0$.

Note que $T' \cup \{e\} \setminus \{f\}$ é uma árvore geradora.

Além disso, $c(T' \cup \{e\} \setminus \{f\}) = c(T') + c(e) - c(f) < c(T')$.

mas então construímos uma árvore com custo menor do que o custo de uma árvore mínima ← contradição!!

Então não pode haver duas árvores distintas se os pesos das arestas forem distintos.

5)

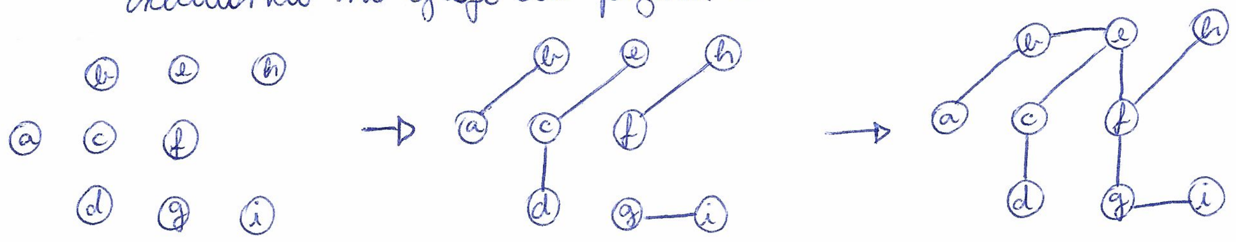
```
DECODIFICA (raiz, sequencia)
    no ← raiz        texto = vetor
    k ← 0
    j ← 0
    enquanto sequencia[k] tem bit
        se no.representa letra
            texto [j] = letra
            j = j+1
            no = raiz
        senão se sequencia [k] == 0
            no = no. esquerda
        senão
            no = no. direita

        k = k+1
    retorna texto
```

6) As arestas de custo 1 não formam ciclo entre si e portanto todas sempre estarão em uma MST.
A aresta fg também sempre deve estar.
Com isso temos os componentes abefg, cd e hi.
Para ligar o primeiro ao segundo temos 3 possibilidades e para ligar o primeiro ao terceiro temos 2.
Logo, temos 6 MSTs diferentes ao todo.

7) Ideia geral do Boruvka:

crie uma componente para cada vértice
T = ∅
enquanto houver mais de uma componente:
  para cada componente C:
    seja e a aresta de menor custo que liga C a uma outra componente
    T = T ∪ {e} // unimos C a essa outra componente.

Executando no grafo da figura 2:



A seguir um pseudocódigo para Boruvka que usa a estrutura Union-Find:

BORUVKA(G, c)

```
1  sejam MENOR, LIDER, TAM e COMP vetores de tamanho n globais
   // MENOR[u] vai dizer qual aresta de menor custo que liga a comp. de u a outra
2  T = ∅
3  NCOMP = n
4  para todo v ∈ V:
5        LIDER[v] = v
6        TAM[v] = 1
7        COMP[v] = lista contendo v
8        MENOR[v] = ∅
9  enquanto NCOMP > 1:
10       para toda uv ∈ E:
11             se FIND(u) ≠ FIND(v):
12                   se MENOR[FIND(u)] == ∅ ou c(MENOR[FIND(u)]) > c(uv):
13                         MENOR[FIND(u)] = uv
14                   se MENOR[FIND(v)] == ∅ ou c(MENOR[FIND(v)]) > c(uv):
15                         MENOR[FIND(v)] = uv
16       para todo v ∈ V:
17             se MENOR[v] ≠ ∅:
18                   seja xy = MENOR[v]
19                   UNION(FIND(x), FIND(y))
20                   T = T ∪ {xy}
21                   NCOMP = NCOMP − 1
22                   MENOR[v] = ∅
23 retorne T
```

Análise do tempo:

$\Theta(n)$ na inicialização (linhas 4 a 8) $= O(m)$ (pois $m > n$)

$+$

$O(m \log n)$ para encontrar as menores arestas (linhas 10 a 15)
(são no máx. $O(\log n)$ iterações do enquanto na linha 9 pois o número de componentes é pelo menos reduzido pela metade a cada iteração)

$+$

$O(n \log n)$ execuções das linhas 16 a 22

$+$

$O(n \log n)$ atualizações de líderes (UNION)

$= O(m \log n)$

8) A intuição é escolher tarefas de maior peso que ao mesmo tempo tenham comprimento pequeno: a razão w_i/l_i parece imitar isso.

O algoritmo ordena o conjunto de tarefas por ordem decrescente da razão w_i/l_i e as renomeia para que

$$\frac{w_1}{l_1} \geq \frac{w_2}{l_2} \geq \dots \geq \frac{w_n}{l_n}$$

A solução devolvida pelo algoritmo é o escalonamento $\sigma = (1, 2, \dots, n)$. Claramente ele é uma solução válida pois é uma sequência de todas as tarefas.

O algoritmo leva tempo $O(n \log n)$ devido à ordenação.

Vamos provar que σ gerada pelo alg. é ótima.

Suponha σ^* um escalonamento ótimo e suponha $\sigma^* \neq \sigma$.

Então em σ^* devem existir tarefas consecutivas i e j tais que $i > j$ (pois se todas as tarefas consecutivas k e l fossem tais que $k < l$, teríamos o escalonamento σ).

Vamos criar uma sequência nova σ' que é construída a partir de σ^* mas com a ordem de i e j trocada.

Assim, se A é a sequência de tarefas em σ^* que foram escalonadas antes de i e B é a sequência das que foram escalonadas depois de j, podemos escrever $\sigma^* = (A, i, j, B)$ e $\sigma' = (A, j, i, B)$.

Note que $C^*_k = C'_k$ para todo $k \in A \cup B$, $C'_i = C^*_i + l_j$ e $C'_j = C^*_j - l_i$.

Então por definição temos:

$$\text{custo}(\sigma^*) = \sum_{k \in A} w_k C^*_k + w_i C^*_i + w_j C^*_j + \sum_{k \in B} w_k C^*_k$$

$$\begin{aligned}
\text{custo}(\sigma') &= \sum_{k \in A} w_k C'_k + w_j C'_j + w_i C'_i + \sum_{k \in B} w_k C'_k \\
&= \sum_{k \in A} w_k C^*_k + w_j (C^*_j - l\ ) + w_i (C^*_i + l_j) + \sum_{k \in B} w_k C^*_k \\
&= \sum_{k \in A} w_k C^*_k + w_j C^*_j - w_j l_i + w_i C^*_i + w_i l_j + \sum_{k \in B} w_k C^*_k \\
&= \text{custo}(\sigma^*) + w_i l_j - w_j l_i
\end{aligned}$$

Como $w_i/l_i \leq w_j/l_j$ pela escolha de i e j, temos $w_i l_j \leq w_j l_i$ e, portanto, temos $\text{custo}(\sigma') \leq \text{custo}(\sigma^*)$.

Como $\text{custo}(\sigma')$ não é maior do que $\text{custo}(\sigma^*)$, podemos repetidamente inverter tarefas consecutivas como i e j até chegar em σ sem piorar o custo. Então σ deve ser ótima também.