EXAME 13-14 normal

1)

a)

Inicializar um vector de vectores de inteiros (v_retorno) para guardar as subsequencias;
Inicializar um inteiro (tamanho) a 0 que guarda o tamanho da subsequencia de maior valor;
Inicializar um inteiro (tamanho_agora) a 0 para guardar o tamanho da sequencia atual;

```
PARA(cada i de D)
    v.clear();
    v.push(D[i]);
    tamanho_agora = 1;
    PARA(cada a = i + 1; a < D.tamanho(); a++)
        SE(D[i] < D[a])
            v.push(D[a]);
            tamanho_agora++;
        FIM SE
    FIM PARA

    SE(tamanho_agora > tamanho)
        tamanho = tamanho_agora;
    FIM SE

    v_retorno.push(v);

FIM PARA
```

O(n^2)

b)

Declarar um vetor de vetores de inteiros L com tamanho da sequencia D;

```
L[0].push_back(D[0]);

PARA( i = 1; i < D.tamanho(); i++)
    PARA(cada j nulo ou positivo menor que i)
        SE(D[j] < D[i])
            L[i] = L[j];
        FIM SE
    FIM PARA

    L[i].push_back(D[i]);

FIM PARA
```

2)

a)

Analisar (4->2), Atualizar dist de vértice 2 para 5, e path para 4, adicionas 2 à priority queue;
Analisar (4->1), Atualizar dist de vértice 1 para 20, e path para 4, adicionas 1 à priority queue;
Analisar (4->6), Atualizar dist de vértice 6 para 10, e path para 4, adicionar 6 à priority queue;
Analisar (4->5), Atualizar dist de vértice 5 para 11, e path para 4, adicionar 5 à

priority queue;
Retira-se o vértice 2 da priority queue, pois é aquele que tem menor valor de dist;
Analisar (2->1), Atualizar dist de vértice 1 para 15, e path para 2, atualizar 1 na priority queue;
Analisar (2->3), Atualizar dist de vértice 3 para 8, e path para 2, adicionar 3 para a priority queue;
Retiramos 3 da priority queue para analisar as suas arestas;
Analisar (3->5), não atualizamos o seu valor porque o peso a atualizar iria ser superior ao peso atual;
Retiramos 6 da priority queue, para analisar as suas arestas;
Analisar (6->5), não atualizamos o seu valor porque o peso a atualizar iria ser superior ao peso atual;
Não faz sentido atualizar o vértice inicial
Analisar (6->1), atualizar o dist de vértice 1 para 12, o path para 6 e atualizar 1 na priority queue;
Nenhum outro valor vai ser atualizado depois

b) Aplicamos o algoritmo de Dijkstra com uma ligeira modificação, atualizamos o dist do vértice incrementado-o com uma unidade cada vez que analisarmos uma aresta cujo destino é esse mesmo vértice.

3)

a)

Algoritmo de Prim

b)

Retirar todas as arestas de peso maior que 'w'.
Fazer uma pesquisa em profundidade / largura e contar os nós.
Se o número de nós for igual ao número de nós inicial então o grafo continua conexo e verifica-se que existe uma árvore de expansão de bottleneck mínimo em que a aresta de bottleneck possui peso ≤ w.

c)

Não pois é possível que uma árvore de bottleneck mínimo tenha arestas que mesmo tendo peso < bottleneck, não fazem parte da árvore de expansão mínima.

4)

a)

12

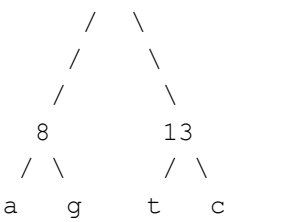
b)

{t1, t2, t3} e {s1, s2, s3, a, c, b, d}

c)

Diminuirá para 9 pois há um "minimum cut" mais pequeno que 12.

5)



Seriam necessários no máximo 2 bits para representar um caracter.
tendo em conta as primeiras duas letras 'aa' elas seriam representadas por 6 '0', o que não é possível.
Logo não é possível que a mensagem esteja codificada por Huffman.

6)

a)

Cada antena é um vértice e tem associado um conjunto de frequências. As arestas significam que os vértice que ligam estão suficientemente próximos para criar interferências.
O objetivo do problema é garantir que dois vértices que estejam ligados por arestas não tenham o mesmo conjunto de frequência.

b)

Se fosse possível resolver o problema da coloração em tempo polinomial então este seria um P probem.
Ao encontrar uma solução para este problema também era possível encontrar solução para o problema em questão, tendo em conta que se trata do mesmo problema pois as cores significam os conjuntos de frequências, e as arestas simbolizam a adjacência.