Prova com consulta. Duração: 2h00m. Exame Época Normal

RESOLUÇÃO

1.a) Um algoritmo ganancioso consiste em encher ao máximo cada prateleira, o que tenderá a minimizar o número de prateleiras e, por conseguinte, a altura total (custo). Pode ser feito com um simples processamento sequencial dos livros, em tempo *O(n)*. O ótimo não é no entanto garantido, como demonstra o seguinte exemplo: com 3 livros de alturas 1, 2 e 2, largura 1 e *LP*=2, esta estratégia dá um custo de 4 (com 2 livros na 1ª prateleira), enquanto que o custo ótimo é 3 (com 1 livro na 1ª prateleira).

Código em C++:

```
custo = 0;
alt = larg = 0;    // altura e largura dos livros na prateleira corrente
for (int i = 0; i < n; i++)
  if (larg + L[i] <= LP) { // usa prateleira corrente
    larg += L[i];
    if (A[i] > alt) {
      custo +=  A[i] - alt;
      alt = A[i];
    }
  }
  else { // usa nova prateleira
    custo += A[i];
    alt = A[i];
    larg = L[i];
  }
return custo;
```

1.b) Formulação recursiva (assumindo índices a começar em 1, 1≤i≤n):

Custo[i] =
| A[i], se i=n
| max(A[i], …, A[n]), se i<n e L[i]+ ….+ L[n] ≤ LP
| min{max(A[i], …, A[j]) + Custo[j+1] | j = i+1, …, n-1 e L[i]+ …+L[j] ≤ LP}, se i<n e L[i]+….+L[n] > LP

Efetuando os cálculos *backwards* e usando variáveis auxiliares para calcular incrementalmente mínimos, máximos e somas, haverá apenas o ciclo exterior para *i* e o ciclo interior para *j*, logo a complexidade é $O(n^2)$.

Código em C++ (assumindo índices a começar em 1):

```
Custo[n] = A[n];
soma_L_i_n = L[n];
max_A_i_n = A[n];
int i = n-1;
```

```
for ( ; i > 0 && soma_L_i_n + L[i] <= LP; i--) {
   soma_L_i_n += L[i];
   max_A_i_n = max(A[i], max_A_i_n);
   Custo[i] = max_A_i_n;
}
for ( ; i > 0; i--) {
   soma_L_i_j = L[i];
   max_A_i_j = A[i];
   Custo[i] = A[i] + Custo[i+1];
   for (int j = i+1; j < n && soma_L_i_j + L[j] <= LP; j++) {
      soma_L_i_j += L[j];
      max_A_i_j = max(A[j], max_A_i_j);
      Custo[i] = min(Custo[i], max_A_i_j + Custo[j+1]);
   }
}
return Custo[1];
```

2.a)

	A	B	C	D	E	F	G
Init	0	∞	∞	∞	∞	∞	∞
Proc. A	0	1	3	∞	∞	10	∞
Proc. B	0	1	2	8	6	10	3
Proc. C	0	1	2	8	5	10	3
Proc. G	0	1	2	8	5	10	3
Proc. E	0	1	2	7	5	7	3
Proc. D	0	1	2	7	5	7	3
Proc. F	0	1	2	7	5	7	3

Caminho mais curto de A a F: A - B - C - E - F

2.b) Basta encontrar o caminho mais curto de v_i a v_k, encontrar o caminho mais curto de v_k a v_f, e concatenar. Para evitar passar 2 vezes no mesmo vértice, ignora-se v_f quando se procura o caminho de v_i a v_k, e ignora-se v_i quando se procura o caminho de v_k a v_f. Tratando-se de um DAG, o algoritmo baseado em ordenação topológica permite resolver o problema em tempo linear no tamanho do grafo O(|V|+|E|).

3.a) Pontos de articulação: d, pois a sua remoção torna o grafo desconexo.

3.b) Solução: a-d-g-e-b-d-e-d-f-c-a

Passo 1) Achar vértices de grau ímpar: d, e

Passo 2) Achar caminho mais curto entre os pares de vértices de grau ímpar: d-e (custo 6)

Passo 3) Grafo G' com vértices de grau ímpar ligados por aresta de peso igual a distância mínima

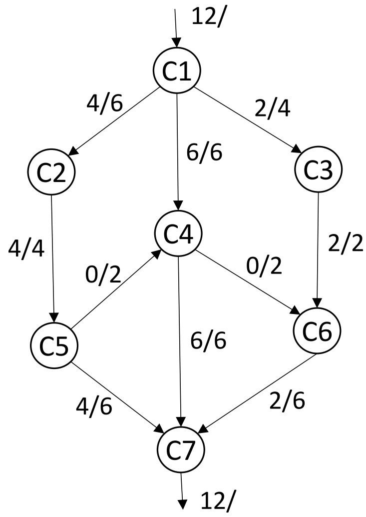
Passo 4) Emparelhamento perfeito de peso mínimo em G': d-e

Passo 5) Grafo G* duplicando aresta d-e no grafo original

Passo 6) Achar um circuito de Euler em G*, o que origina a solução acima

4.a) Trata-se de um problema de fluxo máximo em redes de transporte. Uma vez que as capacidades das arestas que saem do nó de partida são 6.000, 6.000 e 4.000, o fluxo máximo será sempre ≤ 16.000 veículos por hora, logo não é possível satisfazer o objetivo de 18.000 veículos por hora.

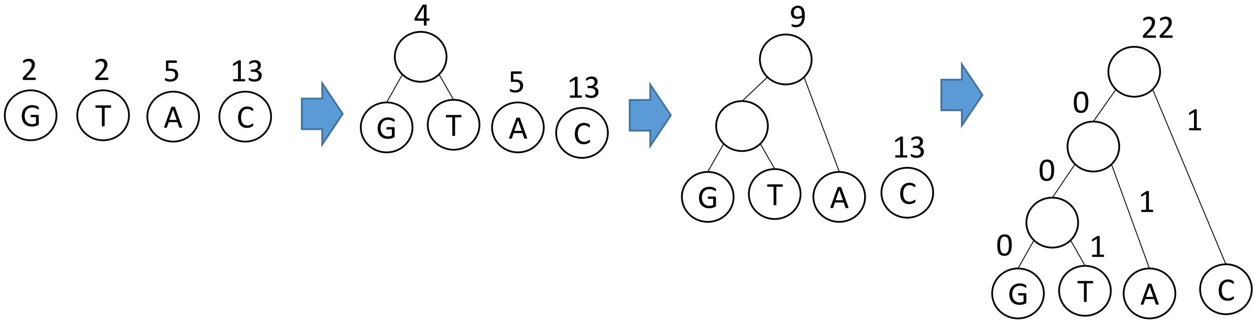
O grafo ao lado mostra uma possível solução de fluxo máximo, com "fluxo /capacidade" nas arestas em milhares de veículos por hora. O fluxo total máximo é de 12.000 veículos por hora.



4.b) Será no troço de auto-estrada com maior fluxo. Uma possibilidade é o troço C1-C4, pois em qualquer solução do problema de fluxo máximo tem fluxo de 6.000 veículos por hora. Já o troço C4-C7 tem fluxo de 6.000 veículos por hora na solução ao lado, mas teria fluxo de 4.000 veículos por hora noutra solução (2.000 veículos por hora desviados para C4-C6).

5.a) Como o nº de símbolos diferentes é 4, basta <u>2 bits</u> para codificar cada símbolo. Como o gene XPTO tem 22 símbolos, o tamanho seria de <u>44 bits</u>.

5.b) Aplicando o algoritmo de Huffman, obtém-se a seguinte codificação:



Ou seja, G = 000, T = 001, A = 01, C =1. Assim, o custo total é 2*3 + 2*3 + 5*2+13*1 = <u>35 bits</u>.

6.a) O problema de decisão consiste em determinar se é possível marcar os exames usando um número de *slots* ≤ k (nº natural), evitando que estudantes inscritos em vários cursos tenham exames sobrepostos. Assim, o problema pode ser reescrito como: É possível marcar os exames utilizando k ou menos slots, de forma a não haver exames sobrepostos?

6.b) O problema é NP-Completo (logo não resolúvel em tempo polinomial), pois:

- É NP, pois uma marcação candidata pode obviamente ser verificada em tempo polinomial. Basta (i) verificar se o nº de slots é efetivamente ≤ k e (ii) ercorrer a lista de estudantes e verificar se algum estudante tem 2 exames marcados no mesmo slot.
- É NP-difícil, pois o problema da Coloração de Grafos é redutível em tempo polinomial ao problema da Marcação de Exames (vide figura):
    - Dado um grafo G=(V,E), cada vértice é convertido num curso e cada aresta é convertida num estudante que está inscrito nos 2 cursos correspondentes aos vértices ligados pela aresta;
    - Os *slots* da solução do problema da marcação de exames correspondem a cores no problema da coloração de grafos;
    - Assim, 2 vértices ligados por uma aresta em *G* originam 2 cursos com um estudante em comum, logo terão *slots* de exame distintos, a que corresponderão 2 cores diferentes nos vértices de *G*. Assim, é possível colorir os vértices do grafo com *k* ou menos cores, se e só se for possível marcar os exames em *k* ou menos *slots*.

Exemplo de redução (não solicitado no enunciado):