Departamento de Ciência de Computadores FCUP

Algoritmos (CC4010)

2020/2021

Teste (13.01.2021) Cotação: 4.0, 1.5, 3.0 , 4.0, 6.0, 1.5 *duração: 2h (+30')*

N.º Nome

1. Recorde o problema *unit task scheduling*, de calendarização de n tarefas unitárias com penalização total mínima. Admita que d_i, com $1 \leq d_i \leq n$, é o prazo limite para execução da tarefa i e p_i o montante a pagar se a executar após esse prazo. As tarefas ficam concluídas no dia em que são executadas, só podendo executar uma tarefa em cada dia k, com $1 \leq k \leq n$.

a) **Enuncie** e **justifique sucintamente** o critério dado nas aulas para decidir *se é possível* calendarizar, sem penalização, um conjunto S de tarefas (dado S).

b) Apresente os passos principais de um **algoritmo** com complexidade temporal $O(n^2)$ que determine uma solução ótima para *unit task scheduling*. Em caso de empate, optará pela tarefa com **identificador menor**.

c) Ilustre a aplicação do algoritmo à instância seguinte.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
d_i	5	4	4	2	1	5	3	7	2	3	1	8	5	8
p_i	2	3	2	1	3	5	5	2	5	5	3	1	1	2

Apresente a **ordem** *pela qual as tarefas são calendarizadas* pelo algoritmo que apresentou, a **calendarização** que produziu (para as 14 tarefas) e a **penalização** total.

2. Suponha que temos um ficheiro com 100000 caracteres, em que ocorrem apenas os caracteres `a`, `b`, `c`, `d`, `e`, `f`, com as frequências seguintes (em milhares): `a` - 5, `b` - 9, `c` - 16, `d` - 13, `e` - 12, `f` - 45. Indique os códigos que resultam da aplicação do algoritmo de Huffmann para compressão. Apresente a árvore (de prefixos) que constrói (indicando os valores nos nós e nos ramos e o seu significado).

3. O Clay Mathematics Institute atribuirá um prémio de 1 milhão de dólares a quem resolver o problema **"P = NP?"**. Admita que alguém descobria um algoritmo polinomial que, dada uma qualquer instância de TSP, com n nós e peso $d(u, v) \in \mathbb{Z}^+$ para o ramo (u, v), determinava um ciclo de Hamilton C com $d(C) \leq n^2 d(C^\star)$, sendo $C^\star$ um ciclo de Hamilton com peso mínimo. Explique porque é que ganharia o prémio. Recordando que o algoritmo de Christofides garante um fator de aproximação $3/2$, não serviria?

4. Sejam $A_1, A_2, \ldots, A_n$ matrizes de inteiros, tendo A_k dimensão $d_k \times d_{k+1}$, i.e., d_k linhas e d_{k+1} colunas), para $1 \leq k \leq n$. Pretendemos calcular o produto $A_1 A_2 \cdots A_n$. Será usada a **definição usual de produto de matrizes**, mas queremos minimizar o número total de multiplicações (de inteiros) efetuadas.

Note que, por exemplo, para $A_1 : 2 \times 3$, $A_2 : 3 \times 5$ e $A_3 : 5 \times 2$, se usarmos $A_1(A_2A_3)$ efetuamos $30 + 12 = 42$ multiplicações e se usarmos $(A_1A_2)A_3$ efetuamos $30 + 20 = 50$. Recorde que se $C = A_1A_2$ então C tem dimensão $d_1 \times d_3$ e no cálculo de $C[i, j] = \sum_{p=1}^{d_2} A_1[i, p] \times A_2[p, j]$ efetuamos d_2 multiplicações (de inteiros), para $1 \leq i \leq d_1$, $1 \leq j \leq d_3$.

Para isso, exploramos o facto de o produto de matrizes ser associativo. Por exemplo, para $A_1A_2A_3A_4$, teríamos de considerar quatro possibilidades $A_1(A_2(A_3A_4))$, $A_1((A_2A_3)A_4)$, $(A_1A_2)(A_3A_4)$, $(A_1(A_2A_3))A_4$, $((A_1A_2)A_3)A_4$. Em geral, o número de casos é exponencial. Mas, podemos calcular uma solução ótima usando **programação dinâmica**.

N.º | Nome

Apresente **um algoritmo**, em pseudocódigo, que use **programação dinâmica** para resolver o problema. Deve calcular o **mínimo** $M[k, m]$ para $A_k A_{k+1} \dots A_m$, com $1 \leq k \leq m \leq n$, e o **índice** $P[k, m]$ que indica como partir. $P[k, m] = s$ corresponderia a $(A_k A_{k+1} \dots A_s)(A_{s+1} \dots A_m)$. Qual é a sua complexidade temporal e espacial (admitindo que os valores podem ser calculados em $O(1)$)? Justifique sucintamente.

5. No problema **Load balancing** existem n tarefas que terão de ser realizadas por máquinas do mesmo tipo. Cada tarefa é processada sem interrupções por uma máquina. Cada máquina só pode estar a realizar uma tarefa em cada instante. A tarefa j tem duração d_j, para $j = 1, \dots, n$. Existem m máquinas idênticas. As tarefas podem ser realizadas por qualquer ordem. Pretendemos atribuir tarefas às máquinas de modo a *minimizar a carga máxima* L das máquinas, definida por $L = \max_i C[i]$, sendo $C[i]$ a soma das durações das tarefas atribuídas à máquina i, para $i = 1, \dots, m$. Admita que os valores d_j são inteiros positivos.

a) Justifique que qualquer que seja a instância se tem $L^\star \geq \max_j d_j$ e $L^\star \geq \frac{1}{m} \sum_j d_j$, sendo $L^\star$ o ótimo.

b) No problema **Partition**, que é NP-completo, são dados n inteiros $a_1, \dots, a_n$ e pretendemos saber se existe um subconjunto J de $\{1, \dots, n\}$ tal que $\sum_{j \in J} a_j = \sum_{j \notin J} a_j$. Usando **Partition**, prove que o problema de decisão correspondente a *Load Balancing* é **NP-completo**, para $m = 2$.

c) Considere o algoritmo apresentado à direita, suportado por uma *heap de mínimo*, em que, na operação INCREASEKEY, a localização do nó correspondente à máquina k pode ser efetuada em $O(1)$.

1. Indique a complexidade temporal do algoritmo. Justifique sucintamente.

2. Sabendo que **Load-Balancing** é NP-hard e assumindo $\mathbf{P}\neq\mathbf{NP}$, comente a veracidade das afirmações, onde L é o **valor obtido** pelo algoritmo para a instância e $L^\star$ o **ótimo** correspondente.

   (i) Para alguma instância (d, n, m), tem-se $L \neq L^\star$.

   (ii) Qualquer que seja a instância, $L \geq (1+\varepsilon)L^\star$, para algum $\varepsilon > 0$.

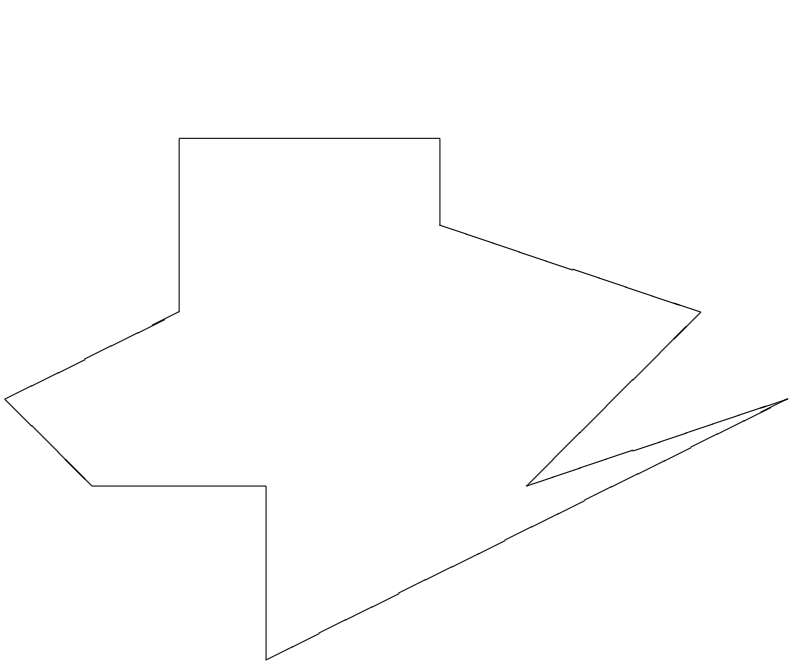
```
LOADBALANCING(d, n, m, S, C)
 1.  L = 0
 2.  for i = 1 to m
 3.      C[i] = 0
 4.  Q = MAKEHEAPMIN(C, m)
 5.  for j ← 1 to n
 6.      k = EXTRACTMIN(Q)
 7.      S[j] = k
 8.      C[k] = C[k] + d[j]
 9.      INCREASEKEY(Q, k, C[k])
10.      if C[k] > L then L = C[k]
11   return L
```

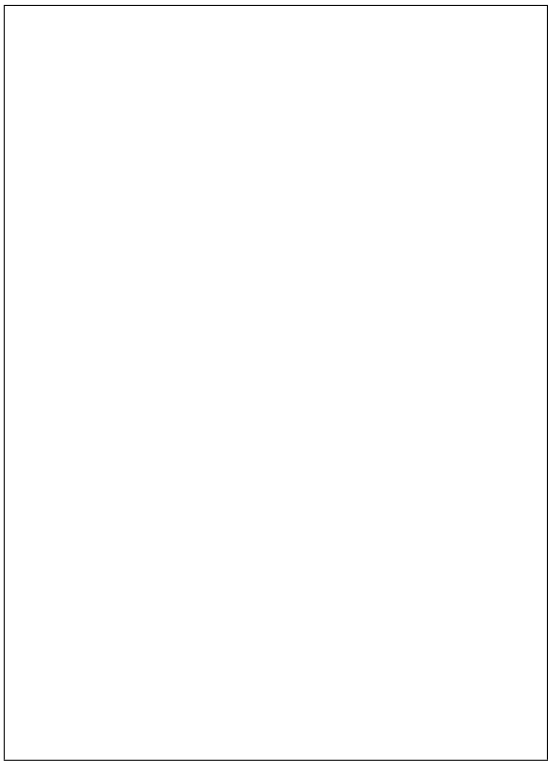
d) Seja L o valor retornado pelo algoritmo. Seja k' uma máquina que fica com carga L na solução (ou seja, $L = C[k']$). Seja j' a última tarefa atribuída à máquina k'. Tendo em conta a estratégia *greedy* que o algoritmo implementa, justifique que $L - d_{j'} \leq C[i]$, para todas as máquinas i, considerando as cargas no momento em que processou j' no algoritmo e as cargas finais. Usando esse facto e **5a)**, conclua que, na linha 11, se tem $L - d_{j'} \leq \frac{1}{m}\sum_i C[i] = \frac{1}{m}\sum_i d_i \leq L^\star$, e que o algoritmo apresentado produz uma aproximação de fator 2, pelo que **Load Balancing** pertence à classe APX.

N.º | | Nome | |

6. Usando a figura, explique a prova de Fisk para o teorema de Chvátal, de que bastam $\lfloor n/3 \rfloor$ para vigiar qualquer polígono simples com n vértices.

Sabendo que os vértices são $(3,0), (9,3), (6,2), (8,4), (5,5), (5,6), (2,6), (2,4), (0,3), (1,2), (3,2)$, indique qual seria o mínimo para esta instância.





Master theorem:

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$, where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$, for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Stirling's approximation:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1 + \Theta(1/n)) = \sqrt{2\pi n}\left(\frac{n}{e}\right)^{\alpha_n}, \quad \text{with } 1/(12n+1) < \alpha_n < 1/(12n)$$

Some useful results:

$$\log(\prod_{k=1}^{n} a_k) = \sum_{k=1}^{n} \log a_k \qquad \sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2}, \quad \text{for } |x| < 1$$

If $(u_k)_k$ is an arithmetic progression (i.e., $u_{k+1} = r + u_k$, for some constant $r \neq 0$), then $\sum_{k=1}^{n} u_k = \frac{(u_1+u_n)n}{2}$.

If $(u_k)_k$ is a geometric progression (i.e., $u_{k+1} = ru_k$, for some constant $r \neq 1$), then $\sum_{k=1}^{n} u_k = \frac{u_{n+1}-u_1}{r-1}$.

If $f \geq 0$ is continuous and a monotonically increasing function, then

$$\int_{m-1}^{n} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx$$