Departamento de Ciência de Computadores FCUP

Desenho e Análise de Algoritmos (CC211) 2012/13

1º Teste Escrito (06.11.2012) *duração: 1h30*

Cotação: (0.5, 1, 1+2+0.5, 1+1+3), 2, 1×3, (1.5, 3.5)

UMA RESOLUÇÃO
(terá de ser adaptada para as outras versões do enunciado)

1. Considere o algoritmo seguinte, assumindo que as variáveis a, b e x guardam inteiros e que cada um dos valores lidos é um inteiro (positivo, negativo ou nulo).

```
a ← 0;
b ← 0;
ler(x);
Enquanto (x ≠ 0) fazer
    se (x > b) então
        se (x > a) então
            b ← a;
            a ← x;
        senão
            b ← x;
    ler(x);
escrever(a);
escrever(b);
```

Resposta 1d) continuação:

	(pior caso)	(melhor caso)
$a \leftarrow 0$;	c_1	c_1
$b \leftarrow 0$;	c_1	c_1
ler(x);	c_2	c_2
Enquanto ($x \neq 0$) fazer	$c_3 \times n$	$c_3 \times n$
se ($x > b$) então	$c_4 \times (n-1)$	$c_4(n-1)$
se ($x > a$) então	$c_4 \times (n-1)$	
$b \leftarrow a$;	$c_5 \times (n-1)$	
$a \leftarrow x$;	$c_5 \times (n-1)$	
senão		
$b \leftarrow x$;		
ler(x);	$c_2 \times (n-1)$	$c_2 \times (n-1)$
escrever(a);	c_6	c_6
escrever(b);	c_6	c_6

a) Indique qual seria o *output* do algoritmo se a sequência que o utilizador pretendia dar fosse a seguinte, supondo que os valores são indicados um a um se solicitados.

```
-3 -7 5 2 3 8 -10 -3 -6 8 -12 0 4 10 0 -3 20 -10 -4 23 13 0 5 8
```

Resposta:

```
8 8
```

b) Escreva o enunciado de um problema que possa ser resolvido pelo algoritmo dado.

Resposta:

Escrever um algoritmo para ler uma sequência de inteiros terminada por zero e imprimir os dois maiores inteiros positivos que contém. Deve apresentá-los por ordem não crescente (podendo estes ser iguais). Caso não existam números positivos deve imprimir dois zeros e se só existir um número positivo imprime-o e depois imprime zero.

c) Mostre que o algoritmo resolve corretamente o problema enunciado. Para isso, designe por n o número de valores lidos e por $d_1, d_2, \ldots, d_n$ a sequência desses valores lidos, e:

1. caracterize o estado das variáveis quando a condição de ciclo é testada pela k-ésima vez, para $k \geq 1$;

   Resposta:

   Quando a condição de ciclo é testada pela k-ésima vez (para $1 \leq k \leq n$), o estado das variáveis x, a e b é o seguinte:

   - x guarda d_k,
   - a guarda o maior inteiro existente na sub-sequência $0, d_1, \ldots, d_{k-1}$ (ou seja, 0 se k for 1 ou não existirem inteiros positivos nessa subsequência; senão guarda o máximo de $d_1, \ldots, d_{k-1}$);
   - b guarda o segundo maior inteiro existente na sub-sequência $0, d_1, \ldots, d_{k-1}$ excluindo o valor de a se for positivo (os valores de a e de b são iguais se o máximo ocorrer repetido).

2. mostre essa propriedade do algoritmo (por indução matemática);

   Resposta:

   Para podermos concluir por indução matemática que a propriedade se verifica para todo $k \geq 1$, vamos mostrar que se verifica para $k = 1$ e que é hereditária.

   (Caso de base) A condição de ciclo é testada pela primeira vez depois de se ter executado a sequência de instruções $a \leftarrow 0; b \leftarrow 0; ler(x)$. Assim, a propriedade verifica-se se k for 1, já que a sequência $d_1, \ldots, d_{k-1}$ é vazia, o valor de x é d_1, e os valores de a e b são zero.

   (hereditariedade) Como hipótese de indução, supomos que a propriedade se verifica para um certo k fixo, tal que $1 \leq k < n$, e vamos mostrar que então se verifica para $k + 1$. Como $k < n$, tem-se $d_k \neq 0$, e o algoritmo efetua mais uma iteração do ciclo "enquanto". Nessa iteração o valor de d_k é testado, e os valores de a e b atualizados do modo seguinte:

   - *Se $d_k > b$ e $d_k > a$, então d_k substituirá a e o valor anterior de a ficará em b.*

     Como, por hipótese de indução, $b \leq a$ e a guardava o máximo de $0, d_1, \ldots, d_{k-1}$, então, com esta atualização, a fica com o máximo de $0, d_1, \ldots, d_{k-1}, d_k$. Por outro lado, b ficará com o máximo de $0, d_1, \ldots, d_{k-1}$, ou seja, o segundo maior elemento da sequência $0, d_1, \ldots, d_{k-1}, d_k$.

   - *Se $d_k > b$ e $d_k \leq a$, então d_k substitui b.*

     Com esta atualização, a fica com o máximo de $0, d_1, \ldots, d_{k-1}, d_k$ (que é o máximo de $0, d_1, \ldots, d_{k-1}$) e sendo $b < d_k \leq a$ antes da atualização, a atribuição de d_k a b fará com que b fique com o segundo maior elemento da sequência $0, d_1, \ldots, d_{k-1}, d_k$.

   - *Se $d_k \leq b$, os valores de a e b mantêm-se.*

     Por hipótese de indução, os valores de a e de b são o maior e o segundo maior elemento de $0, d_1, \ldots, d_{k-1}$ (podendo ser iguais se o maior ocorrer repetido). Como $d_k \leq b \leq a$, esses valores de a e de b são ainda o maior e o segundo maior elemento da sequência $0, d_1, \ldots, d_{k-1}, d_k$ como pretendemos, pelo que deverão ser mantidos.

   Em suma, a atualização dos valores de a e de b na k-ésima iteração mantém a propriedade enunciada sobre a e b. Essa atualização é seguida da leitura de um novo valor para x, a qual colocará em x o valor d_{k+1} (pois, segundo a hipótese, o anteriormente lido era d_k). Assim, quando a condição de ciclo é testada pela $(k + 1)$-ésima vez, o estado das variáveis x, a e b verifica a condição enunciada, ou seja, o invariante de ciclo que enunciámos. (c.q.d.)

3. indique o estado das variáveis quando o ciclo termina, e conclua.

Resposta:

O ciclo termina quando $x = 0$, logo $d_n = 0$ (pois estamos a supor que são lidos n inteiros exatamente). Considerando o invariante de ciclo que mostrámos acima, podemos afirmar que, quando a condição de ciclo é testada pela n-ésima vez:

- x guarda d_n,
- a guarda o maior inteiro existente na sub-sequência $0, d_1, \ldots, d_{n-1}, d_n$, e
- b guarda o segundo maior inteiro existente na sub-sequência $0, d_1, \ldots, d_{n-1}, d_n$ (que pode ser igual ao primeiro se o máximo ocorrer repetido).

Consequentemente, como d_n é zero, os valores de a e de b no fim do ciclo correspondem aos que definimos no enunciado do problema em 1b). Segundo o algoritmo, o valor de a será escrito antes do valor de b, concluindo-se que o resultado satisfaz as condições indicadas em 1b) quanto à ordem de apresentação dos valores.

d) Caracterize duas instâncias que determinem a complexidade temporal do algoritmo no pior caso e no melhor caso, se forem lidos n valores. Mostre que a complexidade temporal do algoritmo dado é $\Theta(n)$, sendo n o número de valores lidos.

Resposta:

O pior caso ocorre quando todos os inteiros da sequência são distintos e positivos (com excepção de d_n que é zero) e a sequência está ordenada por ordem estritamente crescente.

O melhor caso ocorre quando nenhum dos inteiros da sequência é positivo.

Efetuando a análise da complexidade temporal do algoritmo nos dois casos, como consta do quadro que está representado acima, concluímos que qualquer que seja $n \geq 1$, o tempo máximo que demora em instâncias com n valores, designado por $T(n)$, satisfaz

$$\underbrace{(2c_1 + 2c_6 - c_4)}_{\beta_1} + \underbrace{(c_2 + c_3 + c_4)}_{\alpha_1} n \leq T(n) \leq \underbrace{(2c_1 + 2c_6 - 2c_4 - 2c_5)}_{\beta_2} + \underbrace{(c_2 + c_3 + 2c_4 + 2c_5)}_{\alpha_2} n$$

em que c_i, com $1 \leq i \leq 6$, são constantes. Por definição, $T(n) \in \Theta(n)$ se e só se existirem constantes $C, C' > 0$ e n_0 tais que $C'n \leq T(n) \leq Cn$ para todo $n \geq n_0$. Podemos mostrar que essas constantes existem pois vimos que $T(n)$ estava enquadrado por duas funções afins de variável n

$$\beta_1 + \alpha_1 n \leq T(n) \leq \beta_2 + \alpha_2 n$$

sendo $0 < \alpha_1 < \alpha_2$. Se tomarmos $C' = \frac{1}{2}\alpha_1$ e $C = (\alpha_2 + 1)$ e, a seguir, definirmos n_0 como $\max(1, \lceil \frac{-2\beta_1}{\alpha_1} \rceil, \lceil |\beta_2| \rceil)$, teremos $C'n \leq T(n) \leq Cn$, para todo $n \geq n_0$, porque

$$C'n = \frac{1}{2}\alpha_1 n \ \leq \ \beta_1 + \alpha_1 n \leq T(n) \leq \beta_2 + \alpha_2 n \ \leq \ (\alpha_2 + 1)n = Cn.$$

2. "Os algoritmos de Prim e de Kruskal seguem estratégias *greedy* (ávidas, gulosas, gananciosas)". Que significado tem essa afirmação? Em que passo se utiliza essa estratégia em cada um deles?

Resposta:

Numa estratégia "greedy", a opção tomada em cada passo da resolução é apenas localmente óptima e não tem em conta informação global sobre a instância a resolver, o que pode não conduzir a uma solução que seja globalmente óptima para essa instância.

Para o cálculo de uma árvore de cobertura com peso mínimo para um grafo não dirigido pesado, o algoritmo de Prim segue uma estratégia greedy na escolha do próximo vértice a incluir na sub-árvore de suporte construída até um dado passo. Opta por ligar a essa estrutura aquele nó que apresentar o custo menor. Esse custo é dado pelo peso do ramo de peso mínimo incidente nesse nó e num nó que já esteja na sub-árvore. Não analisa assim ligações entre nós futuros que, como esse, ainda não fazem parte dessa estrutura, e toma uma decisão baseada em informação parcial.

Para o mesmo problema, o algoritmo de Kruskal considera os ramos por ordem crescente de peso e segue uma estratégia greedy na escolha do próximo ramo que fará parte da árvore de cobertura. Esse ramo será sempre qualquer um dos que tenha peso mínimo entre os ainda não analisados e será incluído na estrutura desde que não forme um ciclo com ramos anteriormente incluídos. Assim, não pondera as implicações que a escolha de um ramo possa ter ao impossibilitar a escolha futura de outros ramos.

A análise das propriedades das árvores de cobertura de peso óptimo permite mostrar que a aplicação destas estratégias greedy conduz a uma árvore de cobertura de peso global mínimo (e, portanto, concluir que os algoritmos de Prim e de Kruskal determinam soluções corretas).

3. Dê exemplo de três problemas **simples** que envolvam um vetor de n inteiros, não ordenado, e que possam ser resolvidos em tempo $O(1)$, $\Omega(n)$ e $\Theta(\log n)$, respectivamente. Apresente em pseudocódigo os algoritmos que os resolvem e têm essas complexidades.

Resposta: Seja $v[\cdot]$ o vetor.

- $O(1)$: Escrever o valor do primeiro elemento do vetor

  escrever($v[0]$);

- $\Omega(n)$: Calcular na variável *soma* a soma dos elementos do vetor

  $soma \leftarrow 0$;
  Para $i \leftarrow 0$ até $n-1$ fazer
  $\quad soma \leftarrow soma + v[i]$;

- $\Theta(\log n)$: Para cada i tal que $i \geq 0$ e $2^i \leq n$, escrever o elemento que está na posição $2^i - 1$ do vetor.

  $pot \leftarrow 1$;
  Enquanto ($pot \leq n$) fazer
  $\quad$ escrever($v[pot - 1]$);
  $\quad pot \leftarrow pot * 2$;

4. Considere um sistema de rega de irrigação de um terreno. O sistema é modelado por um grafo não dirigido em que os vértices identificam locais relevantes, os ramos representam tubos que ligam pares de locais, e as dimensões dos tubos são iguais. Um dos locais está ligado à rede de distribuição de água e tem uma torneira que se abrirá para fazer chegar a água aos restantes locais. Suponha que a informação sobre o sistema **já está guardada** convenientemente em memória. Pretende-se um algoritmo que imprima os locais por ordem não decrescente de chegada de água.

a) Contextualize o problema no âmbito da matéria dada e relacione-o com algum dos problemas propostos nas aulas práticas. Explique por que razão não se trata da determinação de uma ordenação topológica dos nós do grafo.

Resposta:

Não se trata da determinação de uma ordenação topológica dos nós do grafo porque o grafo é não dirigido e, consequentemente, não define qualquer relação de precedência entre os seus nós.

Para resolver o problema pode-se explorar uma propriedade do método de pesquisa em largura (BFS), nomeadamente o facto de os nós serem colocados na fila por ordem não decrescente de distância ao nó de partida. Aqui, a distância é dada pelo número de ramos do caminho que tem menos ramos e que liga o nó ao nó de partida. Esta propriedade foi explorada no problema "Spreading News" tratado nas aulas práticas.

b) Apresente um algoritmo que resolva o problema em tempo $O(L+T)$ sendo L o número de locais e T o número de tubos (isto é, T é o número de pares de locais ligados diretamente por um tubo). Que estruturas de dados utiliza? De que modo essas estruturas são importantes para que o algoritmo tenha a complexidade pretendida?

Resposta:

```
BFS_Visit_Adaptado(s, G)
    Para cada v ∈ V fazer
        cor[v] ← branco;
        dist[v] ← |V|;      /* para definir ∞ pois os caminhos mínimos têm no máximo |V| − 1 ramos */
    cor[s] ← preto;      /* s é o nó onde se localiza a torneira */
    dist[s] ← 0;
    dnivel ← 0;
    Q ← MK_EMPTY_QUEUE();
    ENQUEUE(s, Q);
    Enquanto(QUEUE_IS_EMPTY(Q) ≠ true ) fazer
        v ← DEQUEUE(Q);
        Se (dnivel ≠ dist[v]) então
            dnivel ← dnivel + 1;          /* pelas propriedades de BFS, dist[v] terá de ser dnivel + 1 */
            escrever(mudança de linha);      /* separar nós por distâncias */
        escrever(espaço);
        escrever(v);
        Para cada w ∈ Adjs[v] fazer
            Se cor[w] = branco então
                dist[w] ← dist[w] + 1;
                ENQUEUE(w, Q);
                cor[w] ← preto;
```

Supõe-se também que a estrutura que representa o grafo é semelhante à descrita nas aulas: o conjunto de vértices é representado por um vetor (array) em que cada elemento aponta a lista de adjacentes do vértice respectivo (no algoritmo, $Adjs[v]$ representa a lista dos adjacentes de v). Esta representação é importante para que a complexidade do algoritmo seja $O(|V|+|E|)$ (ou seja, $O(L+T)$), o que não se conseguiria se se utilizasse uma matriz de adajacências. Supõe-se também que a estrutura tem um campo que guarda o número de nós do grafo (como a definida nas aulas). Tal permitirá aceder a $|V|$ em tempo constante. As operações com a fila Q são realizadas em tempo constante, permitindo garantir $O(|V|)$ na exploração dos nós (cada nó só entra na fila uma vez e só sai uma vez). A utilização do vetor $cor[\cdot]$ permite evitar que BFS entre em ciclo e o vetor $dist[\cdot]$ é importante para poder separar os nós por níveis de distância no *output*. (FIM)