Teste 9 Novembro 2019

Algoritmos e Complexidade

Universidade do Minho

Questão 1 [5 valores]
Considere a função `maxarray` em baixo. Sejam

- $\phi \equiv 0 \leq a \leq b$
- $\psi \equiv a \leq r \leq b \quad \wedge \quad \forall\ k.\ a \leq k \leq b \rightarrow v[k] \leq v[r]$

Apresente um **invariante e um variante para o ciclo,** que permitam provar a correcção total da função face à pré-condição ϕ e à pós-condição ψ, em que r denota o valor devolvido pela função.

```
int maxarray(int v[], int a, int b) {
  int i = a+1, r = a;
  while (i <= b) {
    if (v[i] > v[r]) { r = i; }
    i = i+1; }
  return r;
}
```

Resolução:
Invariante: $i \leq b+1 \quad \wedge \quad a \leq r < i \quad \wedge \quad \forall j.a \leq j < i \rightarrow v[j] \leq v[r]$ (poderia também ser $a \leq r \leq b$)
Variante: $b - i + 1$ (por exemplo)

Questão 2 [3 valores]
Considere o cálculo de máximos de um array de comprimento N por uma *sliding window* (janela deslizante) de comprimento k. A ideia é calcular os máximos de todas as subsequências (contíguas) de comprimento k, guardando-os num array de resultados.
Por exemplo com N=5, k=3, o resultado para o array [50, 10, 30, 20, 0] seria [50, 30, 30], correspondente aos máximos de [50, 10, 30], [10, 30, 20], e [30, 20, 0].

A função seguinte resolve recursivamente este problema, utilizando como auxiliar a função `maxarray` da Questão 1.

```
void MaxWindowRec(int v[], int r[], int N, int k) {
  if (N >= k) {
    r[0] = v[maxarray(v, 0, k-1)];
    MaxWindowRec(v+1, r+1, N-1, k);
  }
}
```

Escreva uma especificação (**pré- e pós-condição**) para esta função.

Resolução:
Pré-condição: $0 < k \leq N$
Pós-condição:
$\forall i.\ 0 \leq i \leq N-k \ \rightarrow (r[i] \in v[i..i+k-1] \ \wedge \ \forall j. i \leq j < i+k \ \rightarrow \ v[j] \leq r[i])$
Ou alternativamente:
$\forall i.\ 0 \leq i \leq N-k \ \rightarrow r[i] = v[maxarray(v, i, i+k-1)]$

Questão 3 [5 valores]
(i) Escreva e resolva uma **recorrência** para o número de comparações $T(N, k)$ efectuadas entre elementos do array, em função de N e de k.

Resolução:
$T(N, k) = 0$ para $N < k$
$T(N, k) = k - 1 + T(N-1, k)$ para $N \geq k$

Logo $T(N, k) = \sum_{i=k}^{N}(k-1) = (N-k+1)(k-1)$

(ii) Tendo agora em conta os diferentes valores que k pode tomar, e assumindo que todos esses valores ocorrem com igual probabilidade, apresente um somatório que permita calcular o **caso médio** do tempo de execução $T(N)$ de `MaxWindowRec`.

Resolução:
$T^{avg}(N) = \sum_{k=1}^{N} \frac{1}{N} T(N, k) = \sum_{k=1}^{N} \frac{1}{N}(N-k+1)(k-1)$

Questão 4 [5 valores]
Considere um algoritmo de inserção ordenada numa lista (crescente), com uma particularidade: são apagados os nós iniciais da lista contendo valores inferiores ao que está a ser inserido. Por exemplo, a inserção de 30 na lista [10, 20, 40, 50] resulta na lista [30, 40, 50]. A função seguinte implementa este algoritmo em C.

```
node *insert_rem (node *p, int x) {
  node *new = malloc(sizeof(node)); new->value = x;
  while (p && x > p->value)
    { aux = p; p = p->next; free (aux); }
  new->next = p;
  return new;
}
```

(i) Analise o tempo de execução assimptótico de `insert_rem`, identificando o **pior e o melhor caso**.

Resolução:
Tendo em conta a comparação `x > p->value`, o melhor caso corresponde às situações em que x é inferior ou igual ao elemento que se encontra no início da lista (e logo a todos os elementos da lista), tempo $\Theta(1)$. O pior caso corresponde às situações em que x é superior aos elementos que se encontram nos N-1 primeiros nós da lista, sendo inserido na última ou penúltima posição, em tempo $\Theta(N)$. Logo $T(N) = \Omega(1), \mathrm{O}(N)$.

(ii) Em termos amortizados a operação de inserção da questão anterior executa em tempo constante. Efectue a sua **análise agregada** considerando a sequência de inserções 20, 70, 60, 30, 40, 50, 10, 80 (partindo de uma lista vazia). Considere que o custo real de cada inserção/remoção efectuada à cabeça da lista é 1, por isso a inserção de 30 na lista [10, 20, 40, 50] tem custo 3. Apresente ainda uma **função de potencial** apropriada sobre as listas, e calcule a partir dela o custo amortizado constante desta operação `insert_rem`.

Resolução:

O custo real de cada inserção da sequência dada é 1, 2, 1, 1, 2, 2, 1, 5, resultando no custo total $\sum_i = 15$, e no custo agregado $c_i = 15/8 < 2$.
Uma função de potencial adequada é simplesmente

Φ = comprimento da lista.

Para o cálculo do custo amortizado, consideremos o caso geral de uma inserção em que são apagados k elementos. Então o custo real desta operação é $k + 1$ (k remoções e uma inserção), e a diferença de potencial é negativa: o comprimento da lista diminui $k - 1$ unidades. Temos então $c_i = t_i + \Phi_i - \Phi_{i-1} = k + 1 - (k - 1) = 2$.

Questão 5 [2 valores *]
Pretende-se reimplementar o cálculo de máximos em janela deslizante (Questões 2 e 3), agora em tempo $\mathrm{O}(N)$, não dependendo de k, mas podendo para isso utilizar-se uma estrutura de dados auxiliar ocupando espaço em $\mathrm{O}(k)$. Descreva a sua solução de forma clara, justificando o tempo de execução no pior caso (pode utilizar pseudo-código se assim entender).

Resolução:
Uma solução possível passa por identificar exactamente os elementos da janela actual, com comprimento k, que é indispensável guardar para utilização futura quando a janela avançar.
Considere-se o array [20, 30, 10, 5, 8, 16], com N = 6, e k = 3. Quando a janela for [20, 30, **[10, 5, 8]**, 16], a estrutura de dados auxiliar poderá conter apenas os elementos 10 e 8, porque 5 não poderá nunca ser o máximo quando a janela deslizar (saindo 10), uma vez que o elemento 8, que se encontra à sua direita, impede que 5 venha a ser máximo. Elimina-se assim a necessidade de se guardar este elemento. Por outro lado, se se mantiver esta estrutura ordenada, o máximo da janela actual poderá sempre ser obtido em tempo constante.

Quando a janela avança uma posição, depois de consultado o máximo, se este for o elemento que vai saír da janela ele poderá ser removido da estrutura (em tempo constante). Por outro lado, a inserção do novo elemento que entra para a janela pode ser feita por uma função de inserção ordenada como a da Questão 4, que apaga os elementos inferiores ao agora inserido, e que executa em tempo amortizado constante.

Simulação da execução para o exemplo acima:

array e janela	estrutura auxiliar	array resultado

deslizante		
[**[20, 30, 10]**, 5, 8, 16]	[**30**, 10]	[30, —, —, —]
[20, **[30, 10, 5]**, 8, 16]	[**30**, 10, 5] (máximo 30 não sai; inserção de 5)	[30, 30, —, —]
[20, 30, **[10, 5, 8]**, 16]	[**10**, 8] (máximo 30 sai; inserção de 8)	[30, 30, 10, —]
[20, 30, 10, **[5, 8, 16]**]	[**16**] (máximo 10 sai; inserção de 16)	[30, 30, 10, 16]

Note-se que no primeiro passo o elemento 20 que sai da janela, não estando na extremidade esquerda da estrutura auxiliar, não pode estar dentro desta, pois terá seguramente sido removido pela inserção de 30.