MAC 338 – Anáise de Algoritmos

Primeiro Semestre de 2008

Gabarito parcial da Terceira Prova – 3 de julho

1. [3,0 pontos]
   Projete uma estrutura de dados que suporte as seguintes duas operações sobre um conjunto S de inteiros:

   (a) **insere**(S, x): insere x no conjunto S;

   (b) **remova_maior_metade**(S): remove os maiores $\lceil |S|/2 \rceil$ elementos de S.

   Explique como implementar essa estrutura de dados de maneira que m operações consumam tempo $O(m)$.

   Mais precisamente, diga como S é armazenado, e escreva os dois algoritmos acima. Diga quanto tempo cada um deles consome no pior caso. Depois mostre que uma seqüência arbitrária de m chamadas a **insere** ou **remova_maior_metade** sobre um conjunto S inicialmente vazio tem custo total $O(m)$ no pior caso.

   Resolução: Vamos usar um inteiro n e um vetor para representar um conjunto S. O inteiro guarda $|S|$ e o vetor guarda os elementos de S nas posições de 1 a n. Vamos chamar o vetor de s.

   IINSERE (s, n, x)
   $n \leftarrow n + 1$
   $s[n] \leftarrow x$

   É claro que **IINSERE** (s, n, x) consome tempo $\Theta(1)$.

   REMOVA-MAIOR-METADE (s, n)
   $q \leftarrow$ **SELECT-BFPRT-IND**$(s, 1, n, \lfloor n/2 \rfloor)$
   $s[n] \leftarrow s[q]$
   PARTICIONE$(s, 1, n)$
   $n \leftarrow \lfloor n/2 \rfloor$

   O algoritmo **SELECT-BFPRT-IND**(A, p, r, k) é basicamente o **SELECT-BFPRT** visto em aula, porém que devolve o *índice* onde o k-ésimo mínimo do vetor $A[p \,..\, r]$. É fácil ver que tal variante consome o mesmo tempo que o **SELECT-BFPRT**, ou seja, consome no pior caso tempo $\Theta(t)$, onde $t = r - p + 1$. O algoritmo **PARTICIONE**(A, p, r), também visto em aula, consome no pior caso tempo $\Theta(t)$ (onde novamente $t = r - p + 1$). (Aqui ignoramos o valor devolvido pelo algoritmo, pois já sabemos que será $\lfloor n/2 \rfloor$.) Com isso é fácil concluir que **REMOVA-MAIOR-METADE** (s, n) consome tempo $\Theta(n)$.

   Podemos pensar então que o custo real de uma chamada a **IINSERE** (s, n, x) é 1 e o custo real de uma chamada a **REMOVA-MAIOR-METADE** (s, n, x) é n.

   Uma análise amortizada pode mostrar que as m operações sobre um S inicialmente vazio consomem, no total, tempo $O(m)$. Atribua 3 créditos para a operação **IINSERE** e 0 créditos para a **REMOVA-MAIOR-METADE**. O **IINSERE** paga por si mesmo e deixa os 2 créditos remanescentes sobre o elemento inserido. O **REMOVA-MAIOR-METADE** é pago pelos $2x\lceil n/2 \rceil$ créditos que estão sobre os $\lceil n/2 \rceil$ elementos removidos. Observe que, desta forma, sempre há $2n$ créditos na estrutura. Como o total de créditos atribuídos é $O(m)$, e estes são suficientes para pagar pelas operações realizadas, o custo total das m operações é $O(m)$.

2. [1,5 pontos]
Considere a implementação do union-find por árvores enraizadas. Segue a implementação recursiva do FINDSET com compressão de caminhos.

```
FINDSET (x)
se x ≠ pai[x]
   então pai[x] ← FINDSET (pai[x])
devolva pai[x]
```

Escreva uma versão não recursiva do FINDSET com compressão de caminhos. O consumo de tempo da sua função deve ser assintoticamente igual ao da acima.

Resolução:

```
FINDSET (x)
r ← x
enquanto r ≠ pai[r] faça
   r ← pai[r]
y ← x
enquanto y ≠ r faça
   z ← pai[y]
   pai[y] ← r
   y ← z
devolva r
```

Observe que o custo dessa versão é $\Theta(m)$, onde m é o comprimento do caminho de x até seu representante quando a busca foi acionada. Esse é também o custo da versão recursiva do algoritmo mostrada acima.

3. [3,0 pontos]
Mostre que, se todos os caracteres do padrão $P[1\,..\,m]$ são distintos, o algoritmo ingênuo que busca P em um texto $T[1\,..\,n]$ pode ser modificado para consumir tempo $O(n)$.

Mais precisamente, escreva uma versão modificada do algoritmo ingênuo de busca de padrão que, dado $T[1\,..\,n]$ e $P[1\,..\,m]$ onde todos os caracteres de P são distintos, consome tempo $O(n)$ para imprimir todas as posições do texto $T[1\,..\,n]$ em que $P[1\,..\,m]$ aparece. Mostre que seu algoritmo de fato consome tempo $O(n)$ no pior caso.

Resolução:

```
BUSCA (P, m, T, n)
i ← 1
j ← 1
enquanto i + (m − j + 1) ≤ n faça
   se T[i] ≠ P[j]
      então i ← i + 1
            j ← j + 1
            se j > m
               então imprima "Padrão ocorre na posição ", i − m + 1
                     j ← 1
      senão se j = 1
            então i ← i + 1
            senão j ← 1
```

A idéia do algoritmo ingênuo é, para cada posição i do texto, para i de 1 até $n-m+1$, verificar se P é igual a $T[i\,..\,i+m-1]$. Caso tenhamos feito comparações bem sucedidas entre P e $T[i\,..\,i+m-1]$ até a posição k de P e ou $k=m$ ou $P[k+1] \neq T[i+k]$, passamos a testar se P é igual a $T[i+1\,..\,i+m]$. Ou seja, testamos P contra a posição seguinte do texto. Com a hipótese de que os caracteres de P são distintos, podemos aprimorar o algoritmo, pois os caracteres de T para os quais a comparação foi bem sucedida não irão coincidir com nenhum outro caracter de P. Por isso, caso tenhamos feito comparações bem sucedidas entre P e $T[i\,..\,i+m-1]$ até a posição k de P e ou $k=m$ ou $P[k+1] \neq T[i+k]$, podemos passar a testar se P é igual a $T[i+k\,..\,i+m-1+k]$. Em outras palavras, se uma posição texto é comparada com uma posição do padrão e são iguais, ela não precisa ser comparada novamente. Se uma posição do texto é comparada com uma posição $P[j]$ do padrão e elas são distintas, então, se $j>1$, o padrão é deslocado, e passa-se a comparar a posição $P[1]$ com essa mesma posição do texto. Se $j=1$, essa posição do texto não é mais comparada e passa-se a comparar a posição $P[1]$ com a próxima do texto. Isso é o que está implementado na função acima.

O número de iterações que são feitas no **enquanto** é o número de vezes que as linhas 5, 11 e 12 são executadas. As linhas 5 e 11 são executadas não mais que n vezes. Na linha 12, o valor de j é decrementado (pois no início de toda iteração $1 \leq j \leq m$). No entanto, j é sempre pelo menos 1 e é incrementado (de 1) no máximo n vezes (na linha 6). Portanto a linha 12 é executada no máximo o número de vezes que a linha 6 é executada, ou seja, no máximo n vezes. Isso nos permite concluir que o **enquanto** executa no máximo $2n$ iterações e o consumo de tempo do algoritmo, no pior caso, é $O(n)$.

4. [1,0 + 1,5 pontos]
Seja $G=(V,E)$ um grafo. Um conjunto $S \subseteq V$ é *independente* se quaisquer dois vértices de S não são adjacentes. Ou seja, não há nenhuma aresta do grafo com as duas pontas em S. O problema IS consiste no seguinte: dado um grafo G e um inteiro $k \geq 0$, existe um conjunto independente em G com k vértices? Mostre que IS é NP-completo.

Resolução:

(1) IS $\in$ NP.
Tome como certificado um conjunto independente no grafo G com k vértices. Claro que um tal conjunto existe apenas para as instâncias G, k de resposta SIM. Ademais, dado um candidato a certificado, podemos verificar, em tempo polinomial no tamanho de G, se este de fato descreve um conjunto de k vértices distintos de G e se tais k vértices são dois a dois não adjacentes.

(2) CLIQUE $\leq_p$ IS.
Dada uma instância G, k do CLIQUE, tome como instância do IS o grafo complementar $\bar{G}$ e o próprio k. Claro que um conjunto S de vértices de G é um clique em G se e somente se S é um conjunto independente em $\bar{G}$. Assim G, k é uma instância de resposta SIM do CLIQUE se e somente se $\bar{G}$, k é uma instância de resposta SIM do IS. Ademais, a construção de $\bar{G}$ a partir de G pode ser feita facilmente em tempo polinomial no tamanho de G.