Projeto e Análise de Algoritmos (2022-1)

Seleção de exercícios para a segunda avaliação

Prof. Flávio L. C. de Moura

September 17, 2022

1. Mostre como podemos ordenar n inteiros contidos no intervalo de 0 a $n^2 - 1$ em tempo linear, ou seja, em $O(n)$.

   Solução:

   Inicialmente iteramos pela lista dos números, convertendo cada um deles para a base n. Depois aplicamos o algoritmo radix-sort sobre a lista, utilizando o counting-sort como o algoritmo de ordenação dos dígitos da lista de números. Dada a nova base, cada número possuirá 2 dígitos, uma vez que $\log_n(n^2) = 2$, onde cada dígito estará em um intervalo entre 0 e $n - 1$. Sabendo disso, vemos que radix-sort ordenará n números de 2 dígitos, usando o counting-sort em um intervalo de 0 à $n - 1$, resultando em uma complexidade de $O(2(n + n)) = O(n)$.

2. Mostre como podemos ordenar n inteiros contidos no intervalo de 0 a $n^3 - 1$ em tempo linear, ou seja, em tempo $O(n)$.

   Solução:

   Inicialmente iteramos pela lista dos números, convertendo cada um deles para a base n. Depois aplicamos o algoritmo radix-sort sobre a lista, utilizando o counting-sort como o algoritmo de ordenação dos dígitos da lista de números. Dada a nova base, cada número possuirá 3 dígitos, uma vez que $\log_n(n^3) = 3$, onde cada dígito estará em um intervalo entre 0 e $n - 1$. Sabendo disso, vemos que radix-sort ordenará n números de 3 dígitos, usando o counting-sort em um intervalo de 0 à $n - 1$, resultando em uma complexidade de $O(3(n + n)) = O(n)$.

3. Prove que o algoritmo counting-sort é estável.

```
1  let C[0..h − l] be a new array;
2  for i = 0 to h − l do
3  |  C[i] ← 0;
4  end
5  for i = 0 to n − 1 do
6  |  C[A[i] − l] ← C[A[i] − l] + 1;
7  end
8  for j = 1 to h − l do
9  |  C[j] ← C[j] + C[j − 1];
10 end
11 for i = n − 1 downto 0 do
12 |  j ← A[i] − l;
13 |  B[C[j] − 1] ← A[i];
14 |  C[j] ← C[j] − 1;
15 end
16 return B;
```

Algorithm 1: counting-sort($A[0..n-1], l, h$)

   Solução:

   Suponha que $A[k] = A[i]$ com $0 \leq k < i \leq n - 1$. Considere o laço das linhas 11-15 onde o vetor B é construído. Como $i > k$, o elemento $A[i]$ será examinado antes do elemento $A[k]$, e $A[i]$ será

alocado na posição $m = C[j] - 1$ (linha 13), e como $C[j]$ é decrementado na linha 14, e não é mais incrementado, podemos concluir que. ao término da execução do laço, $A[k]$ será alocado em uma posição $C[j] - 1 < m$. Ou seja, $A[k]$ será alocado antes de $A[i]$ garantindo assim a estabilidade de *counting sort*.

4. Prove que o algoritmo *insertion sort* é estável.

```
1 for j = 1 to n − 1 do
2 |   key ← A[j];
3 |   i ← j − 1;
4 |   while i ≥ 0 and A[i] > key do
5 |   |   A[i + 1] ← A[i];
6 |   |   i ← i − 1;
7 |   end
8 |   A[i + 1] ← key;
9 end
```

Algorithm 2: InsertionSort($A[0..n-1]$)

Solução:

5. Prove que o algoritmo *merge sort* é estável.

```
1 if p < r then
2 |   q = ⌊(p+r)/2⌋;
3 |   mergesort(A, p, q);
4 |   mergesort(A, q + 1, r);
5 |   merge(A, p, q, r);
6 end
```

Algorithm 3: mergesort(A, p, r)

onde

```
1  n1 = q − p + 1 ;                          // Qtd. de elementos em A[p..q]
2  n2 = r − q ;                              // Qtd. de elementos em A[q + 1..r]
3  let L[1..n1 + 1] and R[1..n2 + 1] be new arrays;
4  for i = 1 to n1 do
5  |   L[i] = A[p + i − 1];
6  end
7  for j = 1 to n2 do
8  |   R[j] = A[q + j];
9  end
10 L[n1 + 1] = ∞;
11 R[n2 + 1] = ∞;
12 i = 1;
13 j = 1;
14 for k = p to r do
15 |   if L[i] ≤ R[j] then
16 |   |   A[k] = L[i];
17 |   |   i = i + 1;
18 |   end
19 |   else
20 |   |   A[k] = R[j];
21 |   |   j = j + 1;
22 |   end
23 end
```

Algorithm 4: merge(A, p, q, r)

Solução:

6. Considere uma enumeração qualquer $1, 2, \ldots, |V|$ dos vértices de G. A matriz de adjacências $G.A$ de dimensão $|V| \times |V|$ é dada por:

$$G.A[i][j] = \begin{cases} 1, & \text{se } (i,j) \in G.E \\ 0, & \text{caso contrário.} \end{cases} \tag{1}$$

O pseudocódigo a seguir apresenta o algoritmo BFS onde o grafo G é representado por sua matriz de adjacências:

```
1  for i = 1 to |V| do
2  |  i.color ← WHITE;
3  end
4  s.color ← GRAY;
5  Q ← ∅;
6  enqueue(Q, s);
7  while Q ≠ ∅ do
8  |  u ← dequeue(Q);
9  |  for i = 1 to |V| do
10 |  |  if G.A[u][i] = 1 and i.color = WHITE then
11 |  |  |  i.color ← GRAY;
12 |  |  |  enqueue(Q, i);
13 |  |  end
14 |  end
15 end
```

Algorithm 5: BFS(G, s)

Qual é a complexidade de tempo de BFS neste caso?

Solução:

O laço das linhas 1-3 é executado $|V|$ vezes, enquanto que as operações das linhas 4, 5 e 6 são executadas em tempo constante. O laço das linhas 7-15 é executado enquanto a fila Q possuir algum elemento. A fila é iniciada com o vértice s que é removido da fila na linha 8, quando o laço FOR (linhas 9-14) percorre toda a linha da matriz de adjacências $G.A$ correspondente ao vértice s, ou seja, é executado $|V|$ vezes. Cada vértice desta linha, ou seja, cada vértice adjacente a s é marcado como visitado (linha 11) e inserido na fila Q (linha 12). Em seguida o laço FOR é executado novamente para o vértice que foi removido da fila. Note que cada vértice é inserido na fila uma única vez depois que ele é marcado como visitado (GRAY), e portanto o laço FOR é executado uma vez para cada um dos $|V|$ vértices visitados. Como cada execução percorre toda a linha da matriz, o custo total do laço WHILE é quadrático. Assim, podemos concluir que a complexidade de BFS neste caso é $O(V^2)$.

7. Escreva o pseudocódigo para o algoritmo DFS onde o grafo G dado como argumento é representado por sua matriz de adjacências, e em seguida faça a análise assintótica do seu pseudocódigo.

Solução:

Análogo ao exercício anterior.

8. Mostre que 2-SAT $\in$ P.

Solução:

9. Assumindo que SAT é um problema NP-completo, mostre que 3-SAT é NP-completo. Isto é, mostre que 3-SAT $\in$ NPC.

Solução: Ver aula 23.

10. Mostre que CLIQUE $\in$ NPC.

Solução: Ver aula 24.

11. Mostre que VERTEX-COVER $\in$ NPC.

Solução: Ver aula 25.

12. Mostre que HAMPATH $\in$ NPC.
    Solução: