Prova 2 – Análise de Algoritmos (2016.1)
Professor: Marco Molinaro

Nome ____

- **Escreva as respostas em pseudo-códigos ou sentenças concisas.**
- **Caso utilize em suas soluções algoritmos vistos em aula (e.g. MergeSort, k-Selection, etc.) basta citá-los, não precisa reimplementá-los.**
- **Tente usar algoritmos conhecidos para resolver os problemas, evite reinventar a roda.**
- **Informação prometida: Lembre que o algoritmo de caminho mais curto de Dijkstra (usando heap) tem complexidade $O(m \log n)$.**
- **A prova tem 2 horas de duração**
- **Boa sorte!**

1. (2 pts) Responda as seguintes perguntas referentes a BFS (busca em largura) e DFS (busca em profundidade).

   (a) (1 pt) Construa um grafo não direcionado e sem pesos nas aresta onde a árvore DFS não dá os caminhos mais curtos da raiz a todos os outros nós. (Ou seja, existe um nó u no grafo onde o caminho na árvore DFS da sua raiz r até u não é um caminho mais curto no grafo entre r e u.) Note que aqui "caminho mais curto" significa o caminho com menos arestas.

   Resposta: Ciclo com três nos, ou seja $a - b - c - a$. A árvore DFS começando em a dá caminho de tamanho 2 pra c, mas caminho mais curto no grafo é 1.

   (b) (1 pt) Seja T a arvore gerada por uma BFS em um grafo **direcionado** G. Seja (u, v) uma aresta em G. Podemos afirmar que u e v estão no mesmo nível ou em níveis vizinhos em T? Justifique.

   Resposta: Não, como contra-exemplo tempos o ciclo direcionado com três nos, ou seja $a- > b- > c- > a$. A árvore BFS começando em a coloca nó a no nível 0 e nó c no nível 2, apesar da aresta (c, a) existir no grafo.

2. (2 pts) Seja G um grafo direcionado com pesos positivos na arestas e seja e uma aresta em E.

   (a) (1 pt) Explique como seria um algoritmo polinomial para determinar se G tem ou não uma ciclo que inclui a aresta e. Analise a complexidade do algoritmo. [Dica: Não tente obter o algoritmo mais rápido possivel, e sim o mais simples.]

   Resposta: Suponha que e seja a aresta direcionada $u- > v$. Um ciclo contendo $u- > v$ é da forma $u- > v \rightsquigarrow u$, onde $v \rightsquigarrow u$ é um caminho no grafo. Portanto, rode uma DFS/BFS a partir de v; se u for achado nessa busca, existe um ciclo incluindo $u- > v$, caso contrário não existe.

   A complexidade é $O(m + n)$.

   (b) (1 pt) Explique como seria um algoritmo polinomial para encontrar o ciclo de G com menor peso dentre aqueles ciclos que contem a aresta e. Analise a complexidade do algoritmo. Note que o peso de um ciclo é definido como a soma dos pesos de suas arestas.

   Resposta: Usando a notação da resposta acima, o menor ciclo contendo $u- > v$ é aquele que usa o menor caminho $u \rightsquigarrow v$. Portanto, rode Dijkstra começando por v e pegue o caminho mais curto até u; o menor ciclo é dado por esse caminho mais a aresta $u- > v$.

3. (2 pts) Dado um grafo direcionado, você precisa decidir se existe algum vértice v que alcança todos os outros vértices do grafo (ou seja, se existe caminho de v a todos os nós).

   (a) (0.5 pt) Descreva em palavras um algoritmo para resolver esse problema em tempo $O(n(m+n))$.

   Resposta: Rode uma BFS/DFS a partir de cada vértice e verifique se alguma dessas buscas alcança todos os vértices.

   (b) (1.5 pts) Agora o grafo é **acíclico**, oe seja, um **DAG**. Descreva em palavras um algoritmo para resolver esse problema em tempo $\mathbf{O}(\mathbf{m}+\mathbf{n})$.

   Resposta: Como o grafo é um DAG, **tem** ordenação topológica. Faça ordenação topológica. O único vértice que **potencialmente** pode alcançar todos os outros é o primeiro nessa ordem (os outros não alcançam esse primeiro vértice). Portanto, rode uma só BFS/DFS a partir desse primeiro vértice u_1; caso essa busca alcance todos os outros vértices, u_1 é o vértice desejado, caso contrário tal vértice não existe.

4. (2 pts) Considere o seguinte problema de escalonamento de processos para execução pelo sistema operacional. Existem n processos. O i-ésimo processo tem tempo de duração $d_i > 0$ e um valor de prioridade, representado por um número $w_i > 0$. O sistema operacional precisa escolher em que ordem executar os processos para minimizar o "custo"

$$\sum_{i=1}^{n} w_i \cdot \text{tempo de término do processo } i.$$

A execução dos processos inicia no tempo 0, e o tempo de termino de um processo é o seu tempo de início mais sua duração.

Por exemplo, suponha que temos 3 processos com $d_1 = d_2 = 1$, $d_3 = 2$ e $w_1 = w_2 = 10$ e $w_3 = 20$. Se o sistema realizar os processos na ordem *processo 1, processo 2, processo 3* (crescente por duração), ele obtém custo

$$w_1 d_1 + w_2(d_1 + d_2) + w_3(d_1 + d_2 + d_3) = 10 + 20 + 80 = 110,$$

mas se executar na ordem *processo 3, processo 1, processo 2* (decrescente por prioridade) ele obtém custo

$$w_3 d_3 + w_1(d_3 + d_1) + w_2(d_3 + d_1 + d_2) = 40 + 30 + 40 = 110.$$

Por acaso, nesse exemplo ambas as ordens obtém o mesmo custo, porém isso nem sempre acontece.

(a) (1 pt) Considere o algoritmo guloso que executa os processos por ordem **crescente de duração**. Esse algoritmo sempre retorna a ordem de menor custo? Se SIM argumente, se NÃO dê um contra-exemplo.

Resposta: Não. Considere $d_1 = 1, d_2 = 2$, mas $w_1 = 1$, $w_2 = 100$. A ordem gulosa 1,2 tem custo $1 + 300 = 301$, mas a ordem 2,1 tem custo menor $200 + 3 = 203$.

(b) (1 pt) Considere o algoritmo guloso que executa os processos por ordem **decrescente de prioridade** w_i. Esse algoritmo sempre retorna a ordem de menor custo? Se SIM argumente, se NÃO dê um contra-exemplo.

Resposta: Não. Considere $d_1 = 100, d_2 = 1$, mas $w_1 = 1$, $w_2 = 2$. A ordem gulosa 1,2 tem custo $100 + 202 = 302$, mas a ordem 2,1 tem custo menor $2 + 101 = 103$.

5. (2 pts) Seja $A[1..n]$ um vetor **ordenado** de n inteiros distintos (positivos e negativos). Você tem que desenhar um algoritmo que responda SIM se existe um índice j tal que $A[j] = j$, e responda NAO caso contrário.

   (a) (0.5 pts) Descreva um algoritmo em tempo polinomial para resolver esse problema e analise sua complexidade.

   Resposta: Teste para todo j se $A[j] = j$. Esse algoritmo é linear.

   (b) (1.5 pts) Descreva um algoritmo divisão-e-conquista com complexidade $\mathbf{O}(\log \mathbf{n})$ para resolver esse problema. Analise a complexidade do seu algoritmo.

   Resposta: Teste a posição do meio. Se $A[n/2] = n/2$, retorne SIM. Se $A[n/2] > n/2$ significa que todos os elementos "a direita" tem valor maior que seus indices, ou seja, para $j > n/2$ temos $A[j] > j$; portanto, descarte a segunda metade de A, e continue da mesma forma na lista restante. Se $A[n/2] < n/2$ significa que todos os elementos "a esquerda" tem valor menor que seus indices, ou seja, para $j < n/2$ temos $A[j] < j$; portanto, descarte a primeira metade de A, e continue da mesma forma na lista restante.

   Em pseudo código:

   Algo(A, i, j)
   1: If $A[(i + j)/2] = (i + j)/2$, **return** SIM
   2: If $A[(i + j)/2] > (i + j)/2$, **return** Algo(A, i, (i+j)/2 - 1)
   3: If $A[(i + j)/2] < (i + j)/2$, **return** Algo(A, (i+j)/2 + 1, j)
   End algorithm

   Como o algoritmo divide o tamanho da lista em 2 em cada iteração, tem um total de $\log n$ iterações. Cada iteração leva tempo constante, portanto o algoritmo é $O(\log n)$.