Projeto e Analise de Algoritmos
Prova 3 – 29/11/2005

NOME: _____
MATRICULA: _____ [ ] Regular [ ] Especial
PROFESSORA: Diane Castonguay

JUSTIFIQUE TODAS AS SUAS RESPOSTAS
(exepção Questão 3)

Questão 1 [3 pts] Algoritmos em grafos

(a) Dar a representação de lista de adjacência do grafo G ***(ver verso).***

(b) Aplicar a busca em profundidade ao grafo G do item (a) identificando as arestas de árvore, de retorno, direitas e cruzadas. ***A ordem a seguir é a ordem numérica.***

Questão 2 [3 pts] Fluxo

(a) Dado o fluxo em redes e o fluxo seguintes, ***(ver verso)***, aplicar a idea do algoritmo de Edmonds-Karp, para achar o fluxo máximo. ***A ordem a seguir é a ordem numérica***.
Para cada iteração do loop, identificar a rede residual G_f, o caminho aumentante e o valor do fluxo intermediário. Na ultima iteração, identificar um corte (S, T) no grafo residual tal que não tem flechas de S para T illustrando a inexistência de caminho aumentante. Este corte deve corresponder a um corte mínimo no grafo G.

(b) IMPLEMENTAÇÃO DE EDMONDS-KARP (ver IMPLEMENTACAO-EDMONDS-KARP em anexo)
1. Escreva o algortimo RedeResidual que constui a rede residual a partir de G=(V, E, c) e f.
2. Escreva o algoritmo NOVOF que calcula o novo fluxo a partir do valor do fluxo e do valor da capacidade residual do caminho aumentante definido a partir de BFS*(G_f, s, t).

Questão 3 [2 pts] NP-completo. *Correção negativa. - 0,5 por resposta errada.*

VERDADEIRO OU FALSO – SEM JUSTIFICATIVA
(a) Sabendo que a cada dois iterações do loop enquanto a variavel r diminui para, pelo menos, metade e supondo que o algoritmo MOD é constante, o algortimo MDC é eficiente.

(b) Se P=NP então todos os problemas NP-difícil são polinomiais.

(c) Temos que P=NP, se e somente se, existe um problema NP-completo que pertence a P.

(d) O problema de ordenação de um vetor pertence a NP.

Questão 4 [2 pts] NP-completo
Sabendo que HAM-CYCLE é NP-completo, mostre que TSP é NP-completo.

HAM-CYCLE(G): Existe um ciclo hamiltoniano em um grafo **não orientado** G?
TSP(G, p, k) : Existe um ciclo hamiltoniano de peso menor ou igual a k em um grafo **completo não orientado** G?

ALGORITMOS

Questão 1

Busca em largura

```
BFS(G, s)
1. para cada vértice u ∈ V[G] – {s} faça
2.                          cor[u] ← BRANCO
3.                          d[u] ← ∞
4.                          π[u] ← NIL
5. fim-para
6.          cor[s] ← CINZA
7.          d[s] ← 0
8.          π[s] ← NIL
9. Q ← 0
10. ENQUEUE(Q, s)
11. enquanto Q ≠ 0 faça
12.                 u ← DEQUEUE(Q)
13.                 para cada vértice v ∈ Adj[u] faça
14.                                 se cor[v] = BRANCO
15.                                 então cor[v] ← CINZA
16.                                                 d[v] ← d[u] + 1
17.                                                 π[v] ← u
18.                                                 ENQUEUE(Q, v)
19.                         fim-se
20.                 fim-para
21.         cor[u] ← PRETO
22.         fim-enquanto
```

Busca em profundidade

```
DFS(G)
1. para cada vértice u ∈ V[G] faça
2.                          cor[u] ← BRANCO
3.                          π[u] ← NIL
4. fim-para
5. tempo ← 0
6. para cada vértice u ∈ V[G] faça
7.                          se cor[u] = BRANCO
8.                                  então DFS-VISTIT(u)
9.                          fim-se
10. fim-para


DFS-VISIT(u)
1. cor[u] ← CINZA
2. tempo ← tempo + 1
3. d[u] ← tempo
4. para cada vértice v ∈ Adj[u] faça
5.                          se cor[v] = BRANCO
6.                                  então       π[v] ← u
7.                                                  DFS-VISTIT(v)
8.                          fim-se
9. fim-para
10. cor[u] ← PRETO
11. tempo ← tempo + 1
12. f[u] ← tempo
```

Questão 2

Fluxo máximo

```
FORD-FULKERSON(G, s, t)
1. para cada aresta (u, v) ∈ E[G] faça
2.                          f[u, v] ← 0
3.                          f[v, u] ← 0
4. fim-para
5. enquanto existir um caminho p de s até t em Gf faça
6.                          cf(p) ←MIN { cf(u, v) : (u, v) está em p }
7.                          para cada aresta (u, v) em p faça
8.                                          f[u, v] ← f[u, v] + cf(p)
9.                                          f[v, u] ← - f[u, v]
10.                 fim-para
11. fim-enquanto
```

Questão 2

```
BFS*(G', s, t)
1. para cada vértice u ∈ V[G'] – {s} faça
2.                                  cor[u] ← BRANCO
3.                                  d[u] ← ∞
4.                                  π[u] ← NIL
5. fim-para
6.          cor[s] ← CINZA
7.          d[s] ← 0
8.          π[s] ← NIL
9. Q ← 0
10. ENQUEUE(Q, s)
11. enquanto Q ≠ 0 faça
12.                 u ← DEQUEUE(Q)
13.                 para cada vértice v ∈ Adj[u] tal que c'(u, v) ≠ 0 faça
14.                                 se  v ≠ t
15.                                             então se cor[v] = BRANCO
16.                                                         então cor[v] ← CINZA
17.                                                                     d[v] ← d[u] + 1
18.                                                                     π[v] ← u
19.                                                                     ENQUEUE(Q, v)
20.                                                         fim-se
21.                                             senão Q ← 0
22.                                                         cor[v] = PRETO
23.                                 fim-se
24.                 fim-para
25.         cor[u] ← PRETO
26.         fim-enquanto
```

```
IMPLEMENTACAO-EDMONDS-KARP(G, s, t)
1. para cada aresta (u, v) ∈ E[G] faça
2.                                  f[u, v] ← 0
3.                                  f[v, u] ← 0
4. fim-para
5 G' ← RedeResidual(G, f)
6. BFS*(G', s, t)
7. enquanto π[t] ≠ NIL faça
8.                  c_f(p) ←MIN(G', s, t) // c_f(p) é o min { c'(u, v) : (u, v) está em p }
9.                  f ← NOVOF(f, c_f(p))
10.                 G' ← RedeResidual(G, f)
11.                 BFS*(G', s, t)
12. fim-enquanto
```

```
MIN(G', s, t)
1. m ← 0
2. v ← t
3. enquanto π[v] ≠ NIL faça
4.          se m > c'(π[v], v)
5.                  então m ← c'(π[v], v)
6.          fim-se
7.          v ← π[v]
8. fim-enquanto
9. retorna m
```

Questão 3

```
/* a e b são inteiros positivos */
MDC(a,b)
1. se a > b
2.          então       n ← a
3.                      m ← b
4.          senão       n ← b
5.                      m ← a
6. fim-se
7. r ←n
8. enquanto r ≠ 0 faça
9.          r ← n MOD m
10.         n ← m
11.         m ← r
12. fim-enquanto
13. retorna m
```