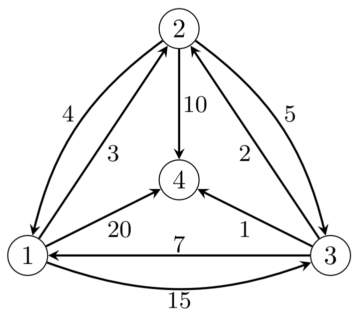
Universidade Estadual de Maringá
Departamento de Informática – Bacharelado em Informática
5189 – Algoritmos em grafos / Prof. Daniel Kikuti

Aluno(a): ______________________________

Primeira avaliação (Valor: 10,0)

1. [Valor: 4,0] No grafo $G = (V, E)$ a seguir, os vértices representam estados em um jogo e, cada aresta (u, v) indica que é possível sair do estado u e alcançar diretamente o estado v com um custo $w(u, v)$.
   (a) [Valor: 1,5] Represente este grafo através de matriz e lista de adjacência.
   (b) [Valor: 1,0] Explique se há diferença entre as duas representações, em termos de complexidade computacional, na consulta: "É possível alcançar diretamente o estado v a partir do estado u?"
   (c) [Valor: 1,5] Dados dois vértices $s, t \in V$, onde s é o estado inicial do jogo e t o estado final, escreva um algoritmo de tempo linear (ou seja, de complexidade $O(V + E)$), que informe se o jogo tem fim.



2. [Valor: 3,0] Considere o problema de encontrar o caminho de custo mínimo em um grafo orientado $G = (V, E)$.
   (a) [Valor: 1,0] Mostre que o algoritmo de Dijkstra falha se G possui arestas com pesos negativos (explique detalhadamente).
   (b) [Valor: 1,0] Considere agora a seguinte ideia. Se G possui arestas com pesos negativos, então some uma constante c a todas as arestas do grafo, tal que todas fiquem com pesos positivos. Execute o algoritmo de Dijkstra e obtenha os caminhos mínimos neste grafo alterado. Para obter os caminhos mínimos no grafo original, para cada vértice $v \in V$, subtraia do valor da distância calculada o valor $k \times c$, onde k é o número de arestas no caminho do vértice origem s até v (considerando a árvore de predecessores). Justifique se esta ideia funciona corretamente ou não.
   (c) [Valor: 1,0] Se G contém um ciclo com custo negativo no percurso do vértice origem s a um vértice qualquer v, então não existe um caminho de custo mínimo de s a todos os vértices que fazem parte deste ciclo negativo e demais vértices alcançáveis dos vértices pertencentes a este ciclo. Esta afirmação está correta? Caso haja um ciclo de custo negativo em G, é possível existir caminhos mínimos e determiná-los? Se sim, indique como. Caso contrário justifique.

3. [Valor: 3,0] Uma propriedade interessande do algoritmo de busca em profundidade é que podemos utilizá-lo para classificar as arestas do grafo de entrada $G = (V, E)$. *Arestas de árvores*: são arestas da floresta de busca em profundidade G_π. A aresta (u, v) é uma aresta de árvore se v foi descoberto pela primeira vez ao explorar a aresta (u, v). *Arestas de retorno*: são arestas (u, v) conectando um vértice u a um antecessor v em uma árvore de busca em profundidade. *Arestas de avanço*: são arestas (u, v) que não pertencem à árvore de busca em profundidade, mas conectam um vértice u a um descendente v que pertence à árvore de busca em profundidade. *Arestas de cruzamento*: são todas as outras arestas, as quais podem conectar vértices na mesma árvore de busca em profundidade ou em duas árvores de busca em profundidade diferentes. Assim, pede-se:
   (a) [Valor: 1,5] Desenvolva um algoritmo que dado um grafo $G = (V, E)$ classifique cada aresta de acordo com seu tipo. [Não precisa armazenar esta informação, pode imprimir na tela "(u, v) é X", onde X deve ser substituído por uma das quatro classificações].
   (b) [Valor: 1,5] Considere a afirmação: "Um grafo orientado $G = (V, E)$ possui uma ou mais ordenações topológicas se não possui arestas de retorno." A afirmação está correta? Justifique.

Boa Prova!!!

Material de apoio

Busca em largura

```
BFS(G,s)
for each vertex u ∈ G.V - {s}
  u.color = WHITE
  u.d = ∞
  u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q,s)
while Q ≠ ∅
  u = DEQUEUE(Q)
  for each v ∈ G.Adj[u]
    if v.color == WHITE
      v.color = GRAY
      v.d = u.d + 1
      v.π = u
      ENQUEUE(Q,v)
  u.color = BLACK
```

Busca em profundidade

```
DFS(G)
for each vertex u ∈ G.V
  u.color = WHITE
  u.π = NIL
time = 0
for each vertex u ∈ G.V
  if u.color == WHITE
    DFS-VISIT(G,u)
```

```
DFS-VISIT(G,u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
  if v.color == WHITE
    v.π = u
    DFS-VISIT(G,v)
u.color = BLACK
time = time + 1
u.f = time
```

Ordenação topológica

```
Topological-Sort(G)
1 Chamar DFS(G) para calcular o tempo de
  término v.f para cada vértice v;
2 à medida que cada vértice é terminado,
  insera-o à frente de uma lista ligada;
3 devolva a lista ligada de vértices.
```

Caminhos mínimos

```
initialize-single-source(G, s)
for each vertex v ∈ G.V
 v.D = ∞
 v.π = NIL
 s.d = 0
```

```
relax(u,v,w)
if v.d > u.d + w(u,v)
  v.d = u.d + w(u,v)
  v.π = u
```

```
Bellman-Ford(G, w, s)
initialize-single-source(G, s)
for i = 1 to |G.V| - 1
   for each edge (u, v) in G.E
      relax(u, v, w)
for each edge (u, v) in G.E
   if v.d > u.d + w(u, v)
      return false
return true
```

```
Dijkstra(G,w,s)
initialize-single-source(G,s)
S = ∅
Q = G.V
while Q ≠ ∅
  u = Extract-Min(Q)
  S = S ∪ {u}
  for each vertex v ∈ G.Adj[u]
    relax(u,v,w)
```

```
Floyd-Warshall(W)
n = W.linhas
D^(0) = W
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```