Aluno(a):____Matrícula:____________

1ª)	3.0	
2ª)	2.0	
3ª)	3.0	
4ª)	2.0	
	10.0	

I. A prova é individual e sem consulta. Qualquer tentativa de “cola” resultará na anulação da prova de ambos os alunos envolvidos.
II. A interpretação faz parte da questão. Em caso de dúvida escreva a dúvida e a sua interpretação na resposta.
III. O tempo de prova é 1:45 h.
IV. As respostas devem seguir as questões. Caso precise de rascunho use o verso da folha.
V. A prova pode ser feita a lápis.

Questão 1 (3.0 pontos).

(a) **(1.0 ponto)** Implemente uma função em C que receba como entrada um grafo não dirigido *G*, com pelo menos 1 nó, e retorne 1, se há um caminho entre quaisquer pares de nós de *G*, e 0, em caso contrário. A função deve ter o seguinte protótipo:

```
int teste(Graph* G);
```

Considere que o grafo está representado pela sua matriz de adjacências:

```
typedef struct graph Graph;

struct graph {
      int nv;          /* número de nós no grafo */
      int** adj;       /* matriz de adjacências do grafo */
  };
```

A função deverá basear-se apenas no algoritmo de Warshall, usado como subrotina e implementado como uma função em C com o seguinte protótipo:

```
void warshall(Graph* g, int** mat);
```

onde `g` é o grafo recebido como entrada e `mat` é a matriz de saída do algoritmo de Warshall. Não é necessário incluir a definição em C do algoritmo de Warshall na solução da questão, mas a função `teste` deve alocar a matriz `mat` e passá-la como parâmetro para a função `warshall`.

Argumente brevemente porque a sua implementação da função `teste` está correta.

(b) **(1.0 ponto)** Qual a complexidade de tempo da função `teste`, incluindo a execução da função `warshall`. Explique a sua resposta.

(c) **(1.0 ponto)** Esta é a forma mais eficiente, em termos de complexidade de tempo, para implementar uma função em C que receba como entrada um grafo não dirigido *G*, com pelo menos 1 nó, e retorne 1, se há um caminho entre quaisquer pares de nós de *G*, e 0, em caso contrário? Explique a sua resposta.

Resposta:

(a)

```
int teste(Graph* g) {
  int i, k, n;
  int** mat;
  mat = (int**) malloc(g->nv * sizeof(int*));
  for (i=0; i< g->nv; i++)
     mat[i] = (int*) malloc(g->nv * sizeof(int));
  warshall(g, mat);
  for (i=0; i < g->nv; i=i+1)
     for (j=0; j < g->nv; j=j+1)
        if (i != j && mat[i][j] == 0) return 0;
  return 1;
}
```

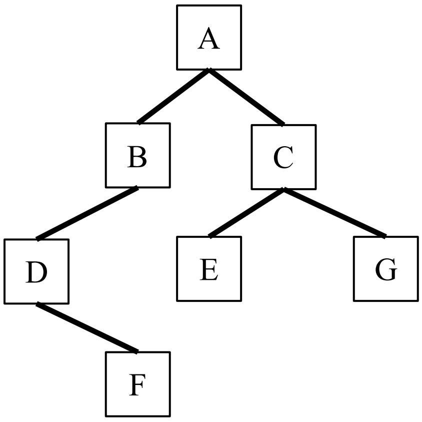
O algoritmo de Warshall retorna em `mat` o fecho transitivo da matriz de adjacências do grafo. Portanto, há um caminho entre um par de nós `i` e `j` sse `mat[i][j]=1`.

(b) O algoritmo de Warshall tem complexidade de tempo $O(|V|^3)$, onde V é o conjunto de nós do grafo. Logo, `teste` terá complexidade $O(|V|^3)$.

(c) Não é a forma mais eficiente. Um caminhamento em profundidade (ou amplitude) marcará todos os nós do grafo, se ele for conexo, ou seja, se há um caminho entre qualquer par de nós. A complexidade do teste será então $O(|V|+|E|)$, para um grafo $G=(V,E)$. Como $|E| \leq |V|^2$, esta implementação será mais eficiente do que a do item (a).

Questão 2 (2.0 pontos). O diâmetro de um grafo é definido como o maior caminho mínimo, dentre todos os possíveis caminhos mínimos entre pares de vértices pertencentes ao grafo. Note que este conceito também se aplica a árvores, entendidas como grafos.

(a) **(0.5 pontos).** Calcule o diâmetro da árvore abaixo. Explique sua resposta.



(b) **(1.5 pontos).** Estabeleça uma relação entre o diâmetro de uma árvore e a sua altura. Lembre-se que a altura é o comprimento do maior ramo de uma árvore. Explique sua resposta.

Resposta:

(a) O diâmetro da árvore é igual ao comprimento do maior ramo (ou seja, a sua altura), 3, somado ao comprimento do segundo maior ramo, 2. Logo, o diâmetro será 5.

(b) Dados dois nós M e N de uma árvore, só há um caminho entre M e N. Além disto, o comprimento do caminho entre M e N em uma árvore com raiz R é:

- a soma do número de arestas entre M e N, se M e N estão no mesmo ramo, ou
- a soma do número de arestas entre M a R adicionada à soma do número de arestas entre N a R

Logo, o diâmetro d da árvore é:

- Se a árvore possui um único ramo, trivialmente, d é igual ao comprimento do único ramo, que é a sua altura h. Ou seja, $d=h$.
- Se a árvore possui mais de um ramo, d é igual ao maior caminho entre duas folhas, isto é, d é igual ao comprimento do maior ramo, que é a sua altura h, somado ao comprimento do segundo maior ramo, que é sempre menor ou igual à sua altura. Logo, $d \leq 2h$.

Portanto, temos que $h \leq d \leq 2h$.

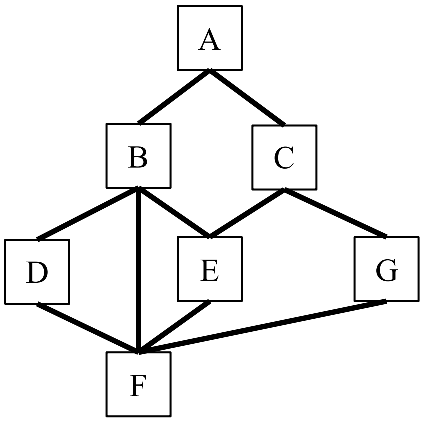
Questão 3 (3.0 pontos). Seja *G=(C,E)* um grafo não dirigido. Um nó *x é vizinho* de um nó *y* em *G* sse há uma aresta entre *x* e *y* em *G*. Seja $\Gamma(x)$ o conjunto de vizinho de um nó *x* em *G* e denote por |C| a cardinalidade de um conjunto *C*. Dados dois nós *x* e *y* em *G*, considere as seguintes métricas entre *x* e *y*:

Preferential Attachment: $pa(x,y) = |\Gamma(x)| \, . \, |\Gamma(y)|$

Common Neighbors: $cn(x,y) = |\Gamma(x) \cap \Gamma(y)|$

Jaccard: $jc(x,y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$

(a) **(1.0 ponto)** Compute *pa(A,E)*, *cn(A,E)*, *jc(A,E), pa(A,F)*, *cn(A,F)* e *jc(A,F)* para o grafo abaixo. Mostre todos os passos intermediários da computação.



(b) **(2.0 pontos)** Argumente qual das três métricas é a mais apropriada para computar um *score* que, dado um nó x, indique qual o nó y mais semelhante a *x* em termos de vizinhança de *x* e *y* no grafo.

Resposta:

(a) $\Gamma(A)$ = *{B, C}* $\qquad$ $\Gamma(A) \cap \Gamma(E)$=*{B, C}*

$\Gamma(E)$ = *{B, C, F}* $\qquad$ $\Gamma(A) \cup \Gamma(E)$=*{B, C, F}*

$\Gamma(F)$ = *{D, B, E, G}* $\qquad$ $\Gamma(A) \cap \Gamma(F)$=*{B}*

$\Gamma(A) \cup \Gamma(F)$=*{B, C, D, E, G}*

pa(A,E) = 2 × 3 = 6 $\qquad$ *pa(A,F)* = 2 × 4 = 8

cn(A,E) = 2 $\qquad$ *cn(A,F)* = 1

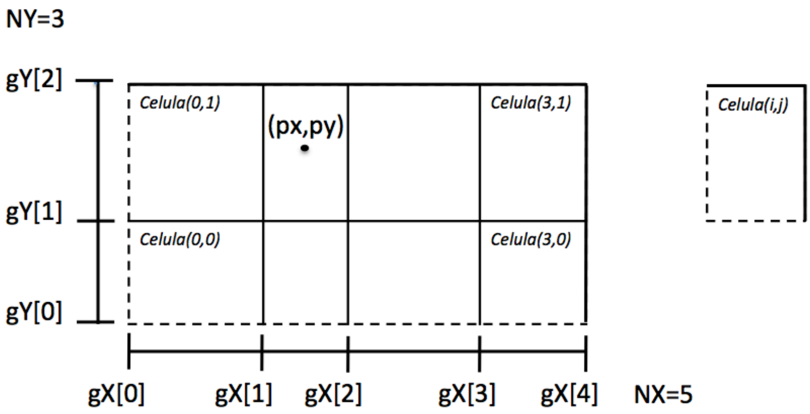
jc(A,E) = 2/3 $\qquad$ *jc(A,F)* = 1/5

(b) *Preferential attachment* leva em consideração apenas o número de vizinhos de cada nó: quanto mais vizinhos, maior será o *score*, independentemente dos vizinhos em comum. Por esta razão, não é uma boa métrica para se computar um *score* que leve em consideração a vizinhança no grafo.

Common neighbors leva em consideração apenas o número de vizinhos EM COMUM dos nós, mas não normaliza o *score*: os nós podem ter muitos vizinhos, mas poucos em comum. Por esta razão, também não é uma boa métrica para se computar um *score* que leve em consideração a vizinhança no grafo.

Já *Jaccard* leva em consideração os vizinhos EM COMUM dos nós e NORMALIZA o score pelo número total de vizinhos dos dois nós. Por esta razão, é uma boa métrica para se computar um *score* que leve em consideração a vizinhança no grafo.

Questão 4 (2.0 pontos). Considere uma grade não regular bidimensional tal que (ver figura abaixo): as bordas lateral esquerda e inferior não pertencem à grade; para cada célula, as bordas lateral esquerda e inferior não pertencem à célula.



Cada célula da grade armazena uma lista encadeada de pontos. Considere os seguintes tipos que representam a grade:

```
typedef struct lista Lista;
struct lista {
  float x, y;    /* ponto na lista */
  Lista* prox;   /* ponteiro para próximo elemento da lista */
};

typedef struct grade Grade;
#define NX 200          /* número de elementos do vetor gX */
#define NY 135          /* número de elementos do vetor gY */
struct grade {
  float gX[NX];         /* vetor que define as divisões no eixo X da grade */
  float gY[NY];         /* vetor que define as divisões no eixo Y da grade */
  Lista* prim[NX][NY];/* lista por célula (inicializada com NULL)          */
};
```

Implemente uma função que receba como entrada uma grade *g*, com a definição acima, e um ponto *(px, py)*, como na figura, e retorne

0	se o ponto não foi encontrado na grade
1	se o ponto foi encontrado na grade
-1	se o ponto está fora da grade

A função deve seguir o seguinte protótipo:

```
int ocorre(Grade* g, float px, float py);
```

Resposta:

```
int ocorre(Grade* g, float x, float y)
{ int i, j;
  Lista* w;
  if (x <= gX[0] || x > gX[NX-1] || y <= gY[0] || y > gX[NY-1]) return -1;
  for(i = 1; i < NX && x > gX[i]; i++)     /* calcula posição x da célula  */
  for(j = 1; j < NY && y > gY[j]; j++)     /* calcula posição y da célula  */
  for(w = g->prim[i-1][j-1]; w != NULL; w = w->prox) /* procura ponto na  */
    if (w->x == x && w->y == y) return 1;            /* lista da célula   */
  return 0
}
```