Aluno(a):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ matrícula:\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| 1a) | 2,5 |  |
| 2a) | 2,5 |  |
| 3a) | 2,5 |  |
| 4a) | 2,5 |  |
|  |  |  |

* A prova é individual e sem consulta.
  + **Qualquer tentativa de “cola” resultará na anulação da prova do aluno ou dos alunos envolvidos**.
  + Os aparelhos celulares deverão permanecer desligados durante toda a prova. **Aparelhos celulares ligados serão tratados como tentativa de “cola”**.
* A interpretação faz parte da questão.
  + **Não há perguntas durante a prova.**
  + Em caso de dúvida escreva a dúvida e a sua interpretação na resposta.
* O tempo de prova é 1:45 h.
* **Após o início da prova, não será possível sair e depois voltar à sala.**
* As respostas devem seguir as questões. Caso precise de rascunho use o verso da folha.
* A prova pode ser feita a lápis.

**Questão 1** (2,5 pontos).

0

1

2

3

4

5

6

7

**Figura 2.** Exemplo de um grafo.

Considere a representação de grafos por listas de adjacência em C abaixo:

typedef struct graph Graph;

typedef struct listNode ListNode;

struct listNode {

int vertex;

ListNode\* link;

};

struct graph {

int nv; /\* número de vértices do grafo \*/

int\* vis; /\* vis[i]=1 sse o vertice i foi visitado \*/

ListNode\*\* vv; /\* vv[i] aponta p/ lista de vértices adjacentes a i\*/

};

* 1. (1,5 ponto) Implemente uma função em C que retorne o número de componentes conexas de um grafo não dirigido, utilizando como subrotina a rotina de busca em profundidade em grafos, modificada se for necessário. A função deve ter o seguinte protótipo:

int nconnected(Graph\* g);

onde g é um grafo representado como acima. A implementação em C da rotina de busca em profundidade deve ser incluída na resposta.

* 1. (1,0 ponto) Usando o grafo da Figura 2 como exemplo, argumente porque o seu algoritmo está correto.

a) (g->vis[n] = j indica que o nó n pertence à j-ésima componente conexa do grafo)

void dfs(Graph\* g, int N, int j){

ListNode\* w;

g->vis[N] = j;

for (w = g->vv[N]; w != NULL; w = w->link)

if (g->vis[w->vertex] == 0) dfs(g, w->vertex, j);

}

int nconectado(Graph\* g){

int i, cont=1;

for (i=1; i < g->nv; i++) g->vis[i]=0; /\* inicializa vis \*/

for (i=0, i < g->nv; i++)

if (g->vis[i] == 0){

dfs(g,i,cont);

cont++;}

return cont;

}  
b) O procedimento implementado em (a) opera da seguinte forma:

1. Marque todos os nós como não visitados.
2. Se há algum nó não foi visitado, escolha um nó não visitado e inicie novamente o procedimento em (2) para determinar a segunda componente conexa, e assim por diante.
3. Pare quando todos os nós tenham sido visitados. O número de componentes conexas será o número de vezes que o procedimento em (2) foi executado.

Este procedimento está correto pois o passo (2) marca todos os nós de cada componente conexa com um inteiro distinto, cont. No passo (3), o valor final da variável cont é retornado. Este valor será o número de componentes conexas.

Por exemplo, escolhendo-se o nó 0 como nó inicial, o passo (2) marcará com “1” os nós 1, 2 e 4. Em seguida, escolhendo-se o nó 3 como nó inicial, o passo (2) marcará com “2” apenas o nó 3. Por fim, escolhendo-se o nó 5 como nó inicial, o passo (2) marcará com “3” os nós 5, 6 e 7. O passo (3) retornará então “3” como o número de componentes conexas do grafo.

**Questão 2** (2,5 pontos) Seja o grafo ponderado não dirigido da Figura 1.

* 1. (1,5 ponto) Considere a versão do algoritmo de Dijkstra apresentada em sala que possibilita a recuperação dos caminhos mínimos, onde a cada passo n:
* A variável u contém o nó selecionado no passo n
* O vetor dist é tal que dist(v) é a distância do nó v ao nó inicial computada até o passo n
* O vetor prev é tal que prev(v) é o predecessor de v computado até o passo n

Mostre esquematicamente todos os valores da variável u e dos vetores dist e prev durante a computação do caminho mínimo de C a G para o grafo da Figura 1. Use a Figura 2 como base para a sua resposta.

* 1. (1,0 ponto) Com base no resultado do item (a), mostre passo a passo como recuperar o caminho mínimo de C a A.



1

2

3

9

**Figura 1.** Grafo não dirigido.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |
| A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| F |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Inicialização Passo: 1 Passo: Passo: Passo:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |
| A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| F |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Passo: Passo: Passo: Passo: Passo:

**Figura 2**. Apresentação esquemáticaos estados dos vetores dist e prev e da variável u.

**Resposta**

**a)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |
| A |  |  | ∞ |  |  |  | ∞ |  |  |  | ∞ |  |  |  | ∞ |  |  |  | ∞ |  |
| B |  |  | ∞ |  |  |  | 5 | C |  |  | 5 | C |  |  | 5 | C |  |  | 4 | F |
| C |  |  | 0 |  | X | C | 0 |  | X |  | 0 |  | X |  | 0 |  | X |  | 0 |  |
| D |  |  | ∞ |  |  |  | ∞ |  |  |  | 2 | E | X | D | 2 | E | X |  | 2 | E |
| E |  |  | ∞ |  |  |  | 1 | C | X | E | 1 | C | X |  | 1 | C | X |  | 1 | C |
| F |  |  | ∞ |  |  |  | 3 | C |  |  | 3 | C |  |  | 3 | C | X | F | 3 | C |
| G |  |  | ∞ |  |  |  | ∞ |  |  |  | ∞ |  |  |  | ∞ |  |  |  | 6 | F |

Inicialização Passo: 1 Passo: 2 Passo: 3 Passo: 4

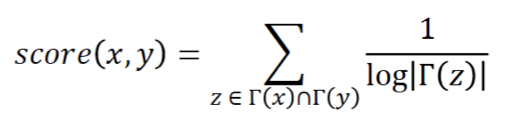
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | u | dist | prev |  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |  | u | dist | prev |
| A |  |  | 9 | B |  |  | 8 | G | X | A | 8 | G |  |  |  |  |  |  |  |  |
| B | X | B | 4 | F | X |  | 4 | F | X |  | 4 | F |  |  |  |  |  |  |  |  |
| C | X |  | 0 |  | X |  | 0 |  | X |  | 0 |  |  |  |  |  |  |  |  |  |
| D | X |  | 2 | E | X |  | 2 | E | X |  | 2 | E |  |  |  |  |  |  |  |  |
| E | X |  | 1 | C | X |  | 1 | C | X |  | 1 | C |  |  |  |  |  |  |  |  |
| F | X |  | 3 | C | X |  | 3 | C | X |  | 3 | C |  |  |  |  |  |  |  |  |
| G |  |  | 6 | F | X | G | 6 | F | X |  | 6 | F |  |  |  |  |  |  |  |  |

Passo: 5 Passo: 6 Passo: 7 Passo: Passo:

**b)**

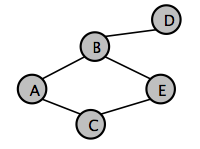
O caminho mínimo de C a A de custo 6 é dado pelo vetor prev, em oredm reversa : A¬G¬F¬C. Revertendo a ordem, temos o caminho: (C,F,G,A).

**Questão 3** (2,5 pontos). Dado um grafo não dirigido, o *coeficiente de Adamic-Adar* de dois nós *x* e *y*, *score(x,y)*, é definido como (log na base 10):



onde Γ(*n*) denota o conjunto de nós ligados por arestas a um nó *n* e |*s*| denota a cardinalidade de um conjunto *s*.

a) (1,0 ponto) Compute *score(A,D)* e *score(A,E)* no grafo abaixo.



b) (1,5 ponto) Implemente uma função em C que receba como entrada um grafo não dirigido *G* e dois nós, *x* e *y*, do grafo *G* e retorne o coeficiente de Adamic-Adar de *x* e *y*. Considere que o grafo está representado por listas de adjacências:

typedef struct graph Graph;

typedef struct listNode ListNode;

struct listNode {

int vertex;

ListNode\* link;

};

struct graph {

int nv; /\* número de vértices no grafo \*/

ListNode\*\* vv; /\* vv[i] aponta p/ lista de vértices adjacentes a i \*/

};

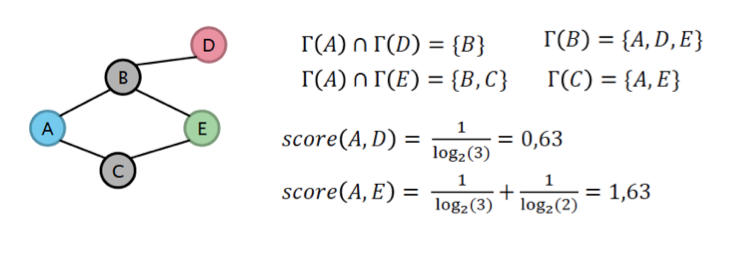
A função deve ter o seguinte protótipo:

float adamic-adar(Graph\* G, int x, int y);

onde x e y são dois inteiros indicando a posição dos nós no vetor vv.

Resposta:

(a)



(b)

float adamic-adar(Graph\* G, int x, int y)

{

float aa=0; /\* adamic-adar \*/

int #z ; /\* cardinalidade de gamma(z) \*/

listNode\* u=vv[x], v=vv[y], t;

if (!u && !v) return 0; /\* gamma(x)=gamma(y)=0 implica adamic-adar = 0 \*/

while(u) /\* computa a interseção de gamma(x) e gamma(y) \*/

{ while(v)

{if (u->vertex == v->vertex) /\* Se z em gamma(x) inter gamma(y), \*/

{#z = 0; /\* computa |gamma(z)| . \*/

t = vv[z]; /\* Note que x e y pertencem a gamma(z).\*/

while(t){#z=+1; t=t->link}; /\* Logo |gamma(z)| >= 2 . \*/

aa = aa + 1.0/log10(#z) /\* Computa adamic-adar \*/

}

v = v->link

}

u = u->link

}

return aa;

}

**Questão 4** (2,5 pontos). Considere uma grade não regular tridimensional tal que as fronteiras lateral esquerda, inferior e frontal não pertencem à grade; para cada célula, as fronteiras lateral esquerda, inferior e frontal 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, z; /\* 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 \*/

#define NZ 400 /\* número de elementos do vetor gZ \*/

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 \*/

float gZ[NZ]; /\* vetor que define as divisões no eixo Z da grade \*/

Lista\* prim[NX][NY][NZ]; /\* 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, pz)*, remove o ponto da grade, se existir, e retorne

0 se o ponto não existia na grade

1 se o ponto existia e foi removido da grade

-1 se o ponto está fora da grade

A função deve seguir o seguinte protótipo:

int remove(Grade\* g, float x, float y, float z);

***Resposta***

int remove(Grade\* g, float x, float y, float z)

{ int i, j, k;

Lista\* ant, w;

if (x <= g->gX[0] || x > g->gX[NX-1] ||

y <= g->gY[0] || y > g->gY[NY-1] ||

z <= g->gZ[0] || z > g->gZ[NZ-1]) return -1;

for(i = 1; i < NX && x > g->gX[i]; i++) /\* calcula posição x da célula \*/

for(j = 1; j < NY && y > g->gY[j]; j++) /\* calcula posição y da célula \*/

for(k = 1; k < NZ && z > g->gZ[k]; k++) /\* calcula posição z da célula \*/

ant = NULL;

w = g->prim[i-1][j-1][k-1];

while(w != NULL && (w->x != x || w->y != y || w->z != z) /\* procura pt \*/

{ant = w; w = w->prox:} /\* na célula \*/

if (w == NULL) /\* não achou ponto \*/

return 0;

if (ant == NULL) /\* retira ponto do início da lista \*/

g->prim[i-1][j-1][k-1] = w->prox;

else p->prox = w->prox; /\* retira ponto do meio da lista \*/

free(w);

return 1;

}