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.

**Resposta**

***(use o verso se necessário)***

**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 para a sua resposta.

* 1. (1,0 ponto) Usando o 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 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

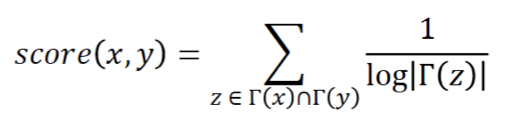
Passo: Passo: Passo: Passo: Passo:

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

**Resposta**

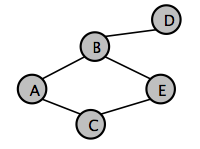
***(use o verso se necessário)***

**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**

***(use o verso se necessário)***

**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**

***(use o verso se necessário)***