Aluno(a):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Matrícula:\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| 1a) | 3.0 |  |
| 2a) | 2.0 |  |
| 3a) | 2.5 |  |
| 4a) | 2.5 |  |
|  | 10.0 |  |

**LEIA COM CUIDADO**

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

**Questão 1 (3.0 pontos)** Implemente uma função em C que receba como entrada um grafo não dirigido *G*, com pelo menos 1 nó, e retorne 1, se *G* for um grafo conexo, e 0, em caso contrário. A função deve ter o seguinte protótipo:

int conexo(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 se basear em algum dos algoritmos de grafos apresentados em sala, ***de menor custo para a tarefa***. Não é necessário incluir o algoritmo na solução, mas indique o protótipo do algoritmo adotado e qualquer outra estrutura de dados necessária para a sua implementação.

A sua implementação não deverá modificar a estrutura de dados passada como entrada.

Argumente porque a sua implementação da função conexo está ***correta*** e é de ***menor custo*** para a tarefa.

***Resposta***

* Usando DFS:

int conexo(Graph\* g) {

int i, k, n;

int vis[g->nv];

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

vis[i] = 0;

DFS(0);

for (i=0; i < g->nv; i=i+1)

if (vis[i] == 0) return 0;

return 1;

}

Usa o algoritmo de DFS como subrotina, com o seguinte protótipo:

void DFS(int n);

DFS(0)retorna em vis os nós visitados a partir do nó 0. Portanto, o grafo não dirigido será conexo, ou seja, haverá um caminho entre um par de nós, sse vis[i]=1, para todo nó i. O custo será *O(|V|2)* e, portanto, ótimo pois: é necessário percorrer todas as arestas para decidir se um grafo não dirigido é conexo; há no máximo (*|V|2*)/2 arestas e o grafo está representado pela matriz de adjacências, de dimensão *|V|* × *|V|*.

* Usando Dijkstra:

int conexo(Graph\* g) {

int i, k, n;

int dist[g->nv];

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

vis[i] = -1;

Dijkstra(0);

for (i=0; i < g->nv; i=i+1)

if (dist[i] == -1) return 0;

return 1;

}

Usa o algoritmo de Dijkstra como subrotina, com o seguinte protótipo:

void Dijkstra(int n);

Dijkstra(0)retorna em dist a distância dos nós a partir do nó 0. Portanto, o grafo não dirigido será conexo, ou seja, haverá um caminho entre um par de nós, sse dist[i]!=-1, para todo nó i. O custo do algoritmo de Dijkstra é *O(|V|2)* e, portanto, ótimo pelas razões já apontadas para DFS.

* Usando Warshall:

int conexo(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;

}

Usa o algoritmo de Warshall como subrotina, 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. A função conexo aloca a matriz mat e a passa como entrada para a função warshall.O algoritmo de Warshall retorna em mat o fecho transitivo da matriz de adjacências do grafo. Portanto, o grafo é conexo, ou seja, há um caminho entre um par de nós ie j, sse mat[i][j]=1. Porém, o custo de Warshall é *O(|V|3)* e, portanto, não é a melhor escolha.

**Questão 2 (2.0 pontos)** Escreva uma rotina em C para codificar um ponto, de um espaço tridimensional, usando a curva de Morton. A rotina deve ter o seguinte protótipo:

long encodeMorton(Ponto\* p)

A rotina:

* recebe como entrada um ponteiro para um ponto p, conforme definido abaixo
* devolve como saída um inteiro longo m, onde
  + m possui 32 bits
  + m codifica o ponto p, considerando as coordenadas na ordem x, y e z

Assuma, por simplicidade, que um ponto possui coordenadas inteiras, definidas pelas 8 bits mais à direita de números inteiros. O tipo Ponto é definido como

typedef struct ponto Ponto;

struct ponto {

int x; /\* usa apenas os 8 bits mais à direita do int \*/

int y; /\* usa apenas os 8 bits mais à direita do int \*/

int z; /\* usa apenas os 8 bits mais à direita do int \*/

}

***Resposta***

*(Simples, deselegante)*

long encodeMorton(Ponto\* p)

{

int i, j, x, y, z;

long M = 0;

if (p == NULL) {printf("Ponto indefinido!\n"); exit(1);}

x = p->x;

y = p->y;

z = p->z;

for (i = 0; i < 8; i++) /\* extrai 8 bits de x, y e z \*/

{ /\* (os bits serão intercalados em M) \*/

j = x % 2; /\* copia o último bit do valor corrente de x \*/

x = x >> 1; /\* retira o último bit do valor corrente de x \*/

M = M << 1; /\* ajusta o valor de M para receber o bit \*/

M = M + j; /\* acrescenta o bit como último bit de M \*/

j = y % 2; /\* copia o último bit do valor corrente de y \*/

y = y >> 1; /\* retira o último bit do valor corrente de y \*/

M = M << 1; /\* ajusta o valor de M para receber o bit \*/

M = M + j; /\* acrescenta o bit como último bit de M \*/

j = z % 2; /\* copia o último bit do valor corrente de z \*/

z = z >> 1; /\* retira o último bit do valor corrente de z \*/

M = M << 1; /\* ajusta o valor de M para receber o bit \*/

M = M + j; /\* acrescenta o bit como último bit de M \*/

}

return p;

}

*(Simples, elegante)*

long encodeMorton(Ponto\* p)

{

int i, j =0, x, y, z;

long M = 0;

if (p == NULL) {printf("Ponto indefinido!\n"); exit(1);}

x = p->x;

y = p->y;

z = p->z;

for (i = 0; i < 8; i++) /\* extrai 8 bits de x, y e z \*/

{ /\* (os bits serão intercalados em M) \*/

M = (((x >> i) & 1) << j) | M; /\* extrai o i-ésimo bit de x e \*/

/\* coloca na j-ésima posição de M \*/

j = j + 1; /\* avança j \*/

M = (((y >> i) & 1) << j) | M; /\* extrai o i-ésimo bit de y e \*/

/\* coloca na j-ésima posição de M \*/

j = j + 1; /\* avança j \*/ M = (((z >> i) & 1) << j) | M; /\* extrai o i-ésimo bit de z e \*/

/\* coloca na j-ésima posição de M \*/

j = j + 1; /\* avança j \*/

}

return p;

}

*(Simples, elegante)*

long encodeMorton(Ponto\* p)

{

int i, mask=1; /\* máscara inicializada com bit mais à dir. \*/

long M = 0;

if (p == NULL) {printf("Ponto indefinido!\n"); exit(1);}

for (i = 0; i < 8; i++) /\* extrai 8 bits de x, y e z \*/

{ /\* (os bits serão intercalados em M) \*/

M = ((p->x & mask)<< 3\*i ) | M; /\* extrai o i-ésimo bit de x e move para M \*/

M = ((p->y & mask)<<(3\*i+1) | M; /\* extrai o i-ésimo bit de y e move para M \*/

M = ((p->z & mask)<<(3\*i+2) | M; /\* extrai o i-ésimo bit de z e move para M \*/

mask = mask<<1; /\* ajusta a máscara \*/

}

return p;

}

**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:

* + - 1. Se Γ(x) ∩ Γ(y)=∅, então *score(x, y)* = 0
      2. Em caso contrário, *score(x, y)* é dado pela equação abaixo:

onde Γ(*w*) denota o conjunto de nós ligados por arestas a um nó *w*, |*s*| denota a cardinalidade de um conjunto *s* e log(x) denota o logaritmo natural de x, ou seja, o logaritmo na base *e*.

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.

Nota: use a função logf para computar o valor do logaritmo natural do argumento x. Esta função recebe e devolve um float.

***Resposta***

float Adamic-Adar(Graph\* G, int x, int y)

{

int #z = 0; /\* cardinalidade de gamma(z) \*/

float sxy = 0; /\* score de x e y \*/

listNode\* u=G->vv[x], v=G->vv[y], w;

while(u) /\* computa gamma(x) inter gamma(y) \*/

{ while(v)

{ if (u->vertex == v->vertex)

{w = G->vv[u->vertex];

#z = 0;

while(w) {w = w->link; #z = #z + 1}; /\* computa |gamma(z)| \*/

if (#z >0) sxy = sxy + 1.0/logf(#z); /\* atualiza sxy, se #z>0 \*/

}

v = v->link;

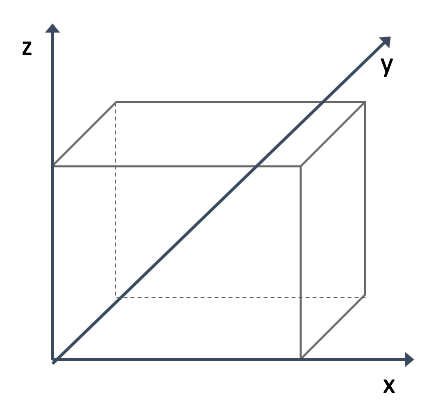
}

u = u->link;

}

return sxy;

}

**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)* 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 x, float y, float z);

***Resposta***

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

{ int i, j, k;

Lista\* 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 \*/

for(w = g->prim[i-1][j-1][k-1]; w != NULL; w = w->prox) /\* procura ponto \*/

if (w->x == x && w->y == y && w->z == z) return 1; /\* na célula \*/

return 0

}