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

|  |  |  |
| --- | --- | --- |
| 1a) | 2,0 |  |
| 2a) | 3,5 |  |
| 3a) | 4,5 |  |
|  | 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 (2,0 pontos)** Considere o grafo não dirigido ponderado da Figura 1.

(a) **(1,5 pontos)** Usando o pseudo-código do Algoritmo de Dijkstra apresentado na Figura 2, mostre esquematicamente todos os valores da variável u e dos vetores dist e prev durante a computação do caminho mínimo de C a A para o grafo da Figura 1. Use a Tabela 1 como base para a sua resposta.

(b) **(0,5 ponto)** Mostre como reconstruir o caminho mínimo de C a A para o grafo da Figura 1 a partir das estruturas computadas no item (a).



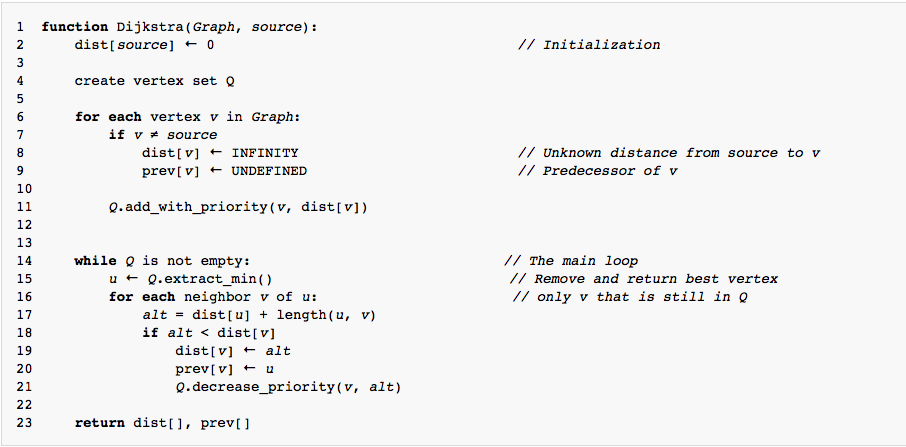
1

2

3

9

**Figura 1.** Grafo não dirigido.



**Figura 2.** Pseudo-código do Algoritmo de Dijkstra.

***Resposta***

a)

**Tabela 1. Estruturas de dados para o Algoritmo de Dijkstra.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 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) Usando ao vetor prev, temos: prev(A) = G caminho = (G,A)

prev(G) = F caminho = (F,G,A)

prev(F) = C caminho = (C,F,G,A)

**Questão 2 (3,5 pontos)**. Seja *G=(C,E)* um grafo não dirigido. Um vértice *x é vizinho* de um vértice *y* em *G* sse há uma aresta entre *x* e *y* em *G*. Seja Γ(*x*) o conjunto de vizinho de um vértice *x* em *G* e denote por |*s*| a cardinalidade de um conjunto *s*. Dados dois vértices *x* e *y* em *G*, o *Coeficiente de Jaccard* entre *x* e *y* é definido como:

(a) **(1,0 ponto**) Considere o grafo da Figura 3. Compute os conjuntos de vizinhos dos vértices   
*a*, *d*, *f*, *g*. Em seguida, compute *jc(d,g)* e *jc(a,f).*

*g*

*f*

*a*

*c*

*b*

*e*

*d*

**Figura 3.** Grafo não-dirigido.

(b) **(2,5 pontos)**. Implemente uma função em C que receba como entrada um grafo não dirigido *G* e um vértice *x* do grafo *G* e devolva o vizinho *y* de *x* com o maior *Coeficiente de Jaccard* com relação a *x*. 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:

int MaxJaccard(Graph\* G, int x);

onde x é um inteiro indicando a posição do vértice no vetor vv. A função deve devolver um inteiro indicando a posição no vetor vv do vizinho de x com maior *Coeficiente de Jaccard*, ou -1, se x não tiver vizinhos.

***Resposta***

a) Γ(*a*) = *{b, c} ={f}*

Γ(*d*) = *{b, f} ={b, c, f}*

Γ(*f*) = *{d, b, e, g} ={b}*

Γ(*g*) = *{c, f} ={b, c, d, e, g}*

*jc(d,g) = 1/3 jc(a,f) = 1/5*

b)

int MaxJaccard(Graph\* G, int x)

{

int y\_max = -1, y; /\* x não tem vizinhos: retorna -1 \*/

int jc\_max = 0, jcy; /\* jc\_max e jc entre x e y \*/

listNode\* u, v, w;

int gammax = 0, gammay, gammaxy;

for (u=G->vv[x]; u!=NULL; u=u->link) /\* calcula gammax \*/

gammax = gammax + 1;

gammay = 0;

for (u=G->vv[x]; u!=NULL; u=u->link) /\* computa gammay e gammaxy \*/

{ /\* para cada vizinho y de x \*/

y = u->vertex;

gammay = 0;

gammaxy = 0;

for (v=G->vv[y]; v!=NULL; v=v->link) {

gammay = gammay +1; /\* computa gammay \*/

for (w=G->vv[x]; w!=NULL; w=w->link)

if (v->vertex == w->vertex) gammaxy = gammaxy + 1; /\* computa gammaxy \*/

}

jcy = gammaxy / (gammax + gammay – gammaxy); /\* |A∪B|=|A|+|B|-|A∩B| \*/

if (jcy > jc\_max) { jc\_max = jcy; y\_max = y; };

}

return y\_max;

}

**Questão 3 (4,5 pontos)**. O *diâmetro* de um grafo é o maior caminho mínimo dentre todos os possíveis caminhos mínimos entre pares de vértices pertencentes ao grafo.

(a) **(1,5 ponto)** Implemente uma função em C que receba como entrada um grafo não dirigido ponderado *G*, com pelo menos 1 vértice, e retorne o diâmetro do grafo. Assuma que o peso de uma aresta é sempre positivo. A função deve ter o seguinte protótipo:

float diametro(Graph\* G);

Considere que o grafo está representado pela matriz de pesos:

typedef struct graph Graph;

struct graph {

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

float\*\* peso; /\* matriz de pesos do grafo \*/

};

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

void FloydWarshall(Graph\* G, float\*\* mat);

onde G é o grafo recebido como entrada, representado pela matriz de pesos, e mat é a matriz de saída do algoritmo de Floyd-Warshall. Não é necessário incluir o código em C do algoritmo de Floyd-Warshall na solução da questão, mas a função diametro deve alocar a matriz mat e passá-la como parâmetro para a função FloydWarshall.

(b) **(1,0 ponto)** Argumente porque a sua implementação da função diametro está correta.

(c) **(1,0 ponto)** Qual a complexidade de tempo, no pior caso, da função diametro, incluindo a execução da função FloydWarshall. Explique a sua resposta.

(d) **(1,0 ponto)** Esta é a forma mais eficiente de computar o diâmetro de um grafo? Explique sua resposta.

***Resposta***

a)

float diametro(Graph\* G) {

int i, k, n;

float diametro = 0;

float\*\* mat;

mat = (float\*\*) malloc(G->nv \* sizeof(int\*));

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

mat[i] = (float\*) malloc(G->nv \* sizeof(int));

FloydWarshall(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] > diametro) diametro = mat[i][j];

return diametro;

}  
b) O algoritmo de Floyd-Warshall retorna em mat o custo do caminho mínimo entre cada par de vértices do grafo. Portanto, a função diametro está correta pois o duplo loop computa o maior destes caminhos mínimos, varrendo a matriz mat.

c) O algoritmo de Floyd-Warshall tem complexidade de tempo, no pior caso, O(|*V*|3), onde *V* é o conjunto de vértices do grafo. Logo, diametro terá complexidade O(|*V*|3).

d) Será a forma mais eficiente, se o grafo for denso; caso contrário o Algoritmo de Dijkstra será mais eficiente.