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

|  |  |  |
| --- | --- | --- |
| 1a) | 3.0 |  |
| 2a) | 3.0 |  |
| 3a) | 2.0 |  |
| 4a) | 2.0 |  |
|  | 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)** Considere o grafo ponderado não dirigido da Figura 1.

* 1. **(2.0 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 G para o grafo da Figura 1. Use a Figura 3 como base para a sua resposta.
  2. **(1.0 ponto)** Com base no resultado do item (a), mostre passo a passo como recuperar o caminho mínimo de C a A.



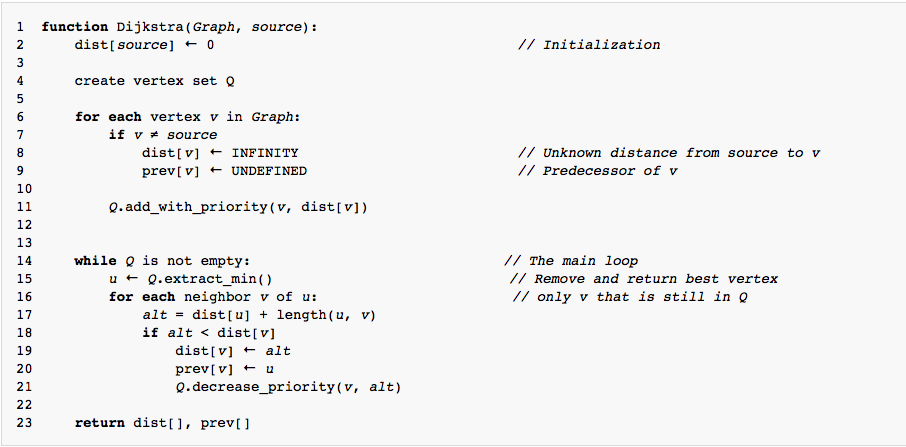
1

2

3

9

**Figura 1.** Grafo não dirigido.



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

**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:

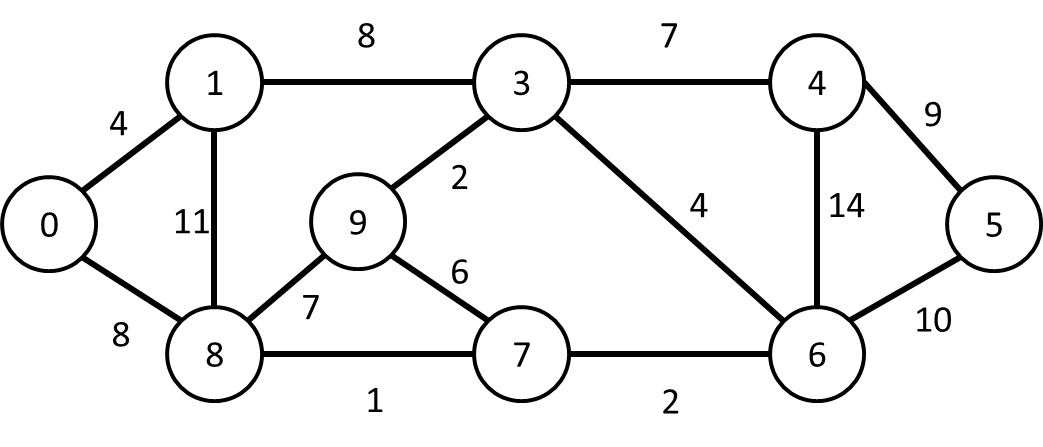
**b)**

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 2 (3.0 pontos)**

(a) **(2.0 pontos)** Mostre os passos do algoritmo de Kruskal para calcular uma árvore geradora mínima do grafo mostrado na Figura 4. Considere que o algoritmo adota uma partição dinâmica dos nós do grafo para otimizar o processamento. Organize a sua resposta como apresentado abaixo.

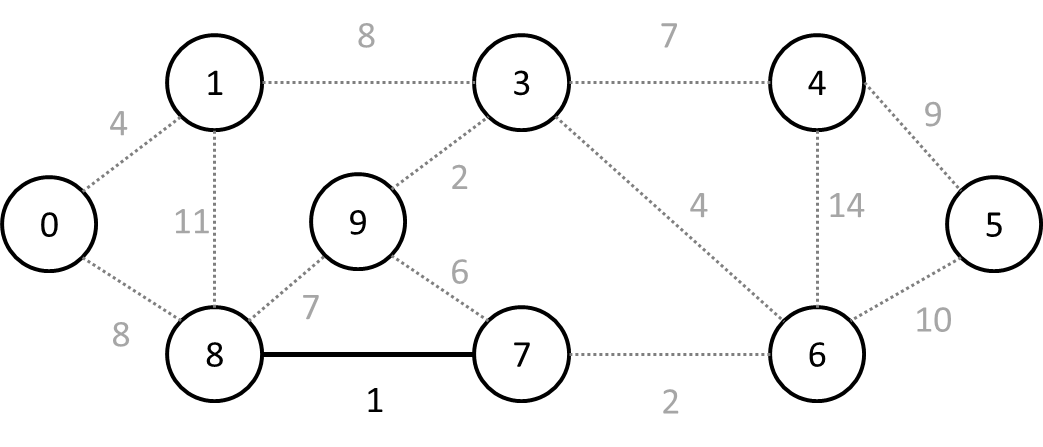
(b) **(1.0 ponto)** Explique qual a vantagem, em termos de custo do processamento, de adotar uma partição dinâmica dos nós do grafo, representada por uma floresta, com a implementação de UNION por altura e FIND com compressão de caminhos?



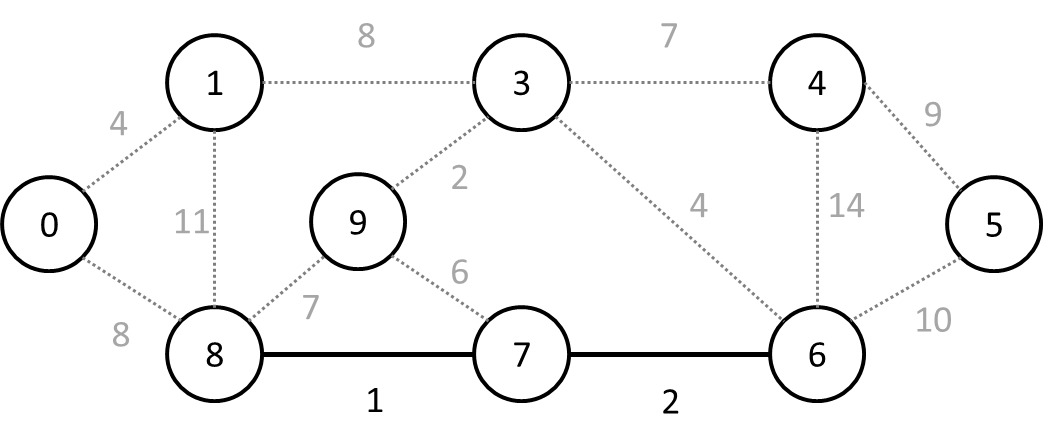
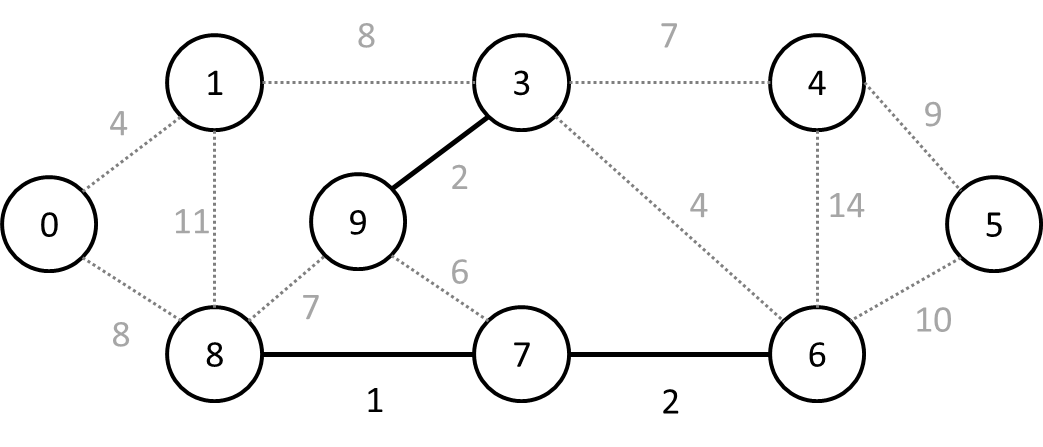
**Figura 4.** Grafo não dirigido.

**Resposta**

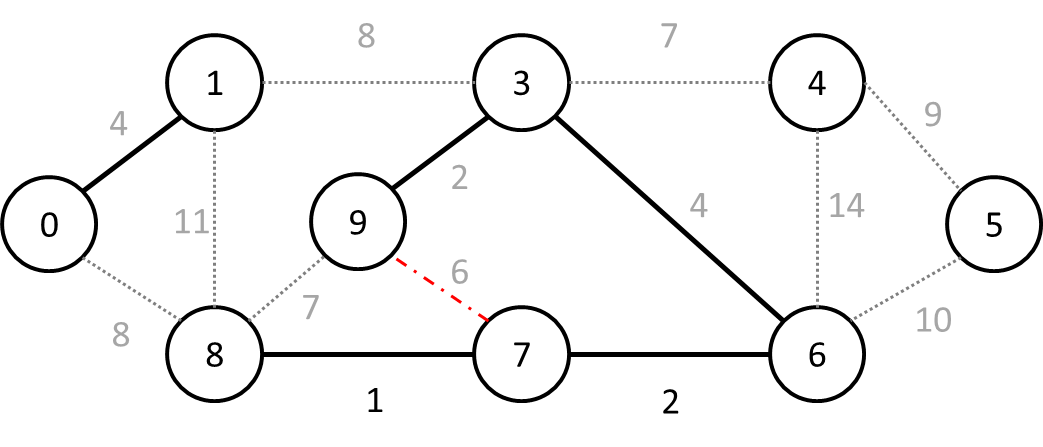
* 1. Adicione uma aresta de menor peso dentre as disponíveis. Ao inserir uma aresta faça uma união dos dois conjuntos correspondentes aos vértices dela. Pd={{0},{1},...**{7,8}**,{9}}.



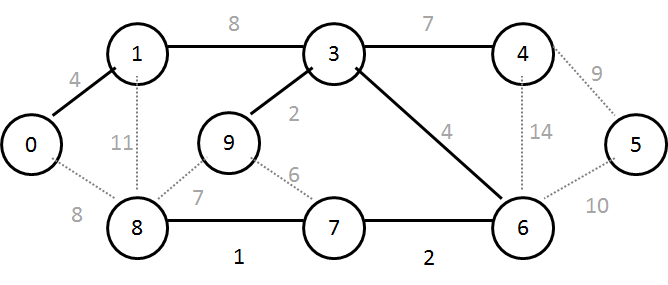
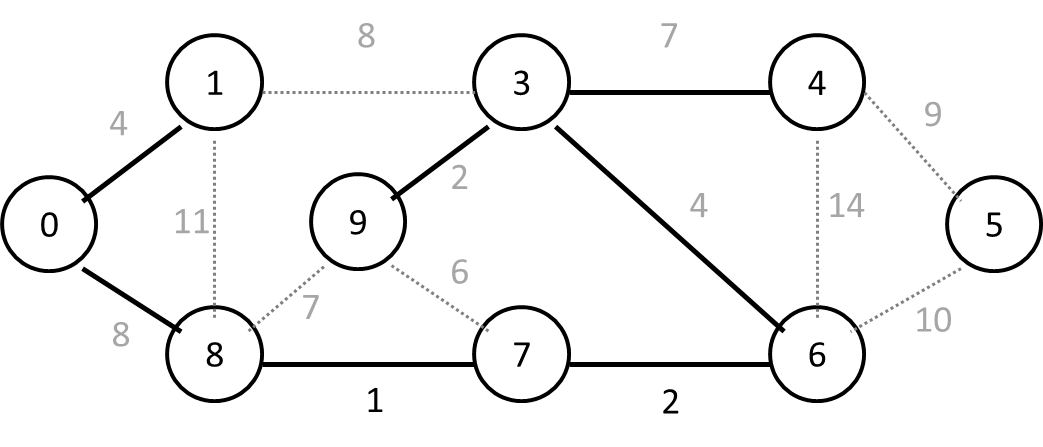
* 1. Continue adicionando uma aresta de menor peso dentre as disponíveis desde que sua inserção não forme ciclos na árvore que está sendo gerada. Ou seja, desde que seus vértices não estejam numa mesma partição. Pd={{0},{1},...**{6,7,8}**,**{3,9}**}.

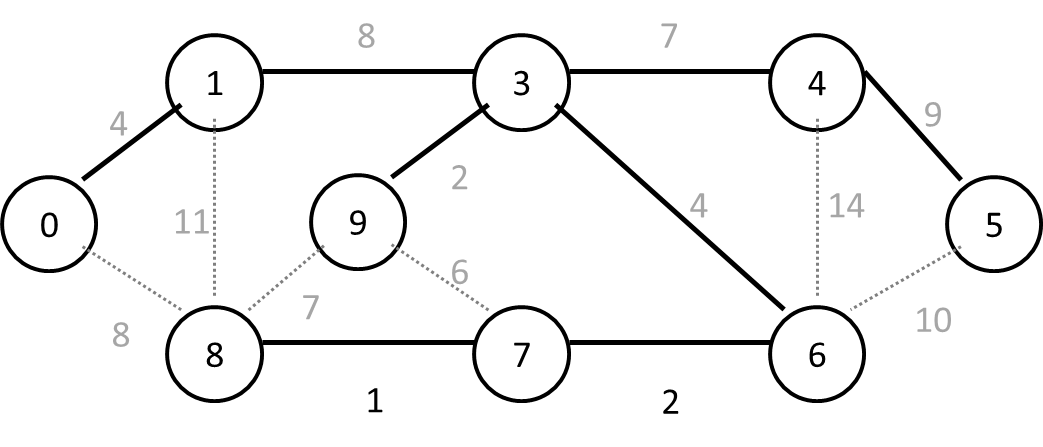
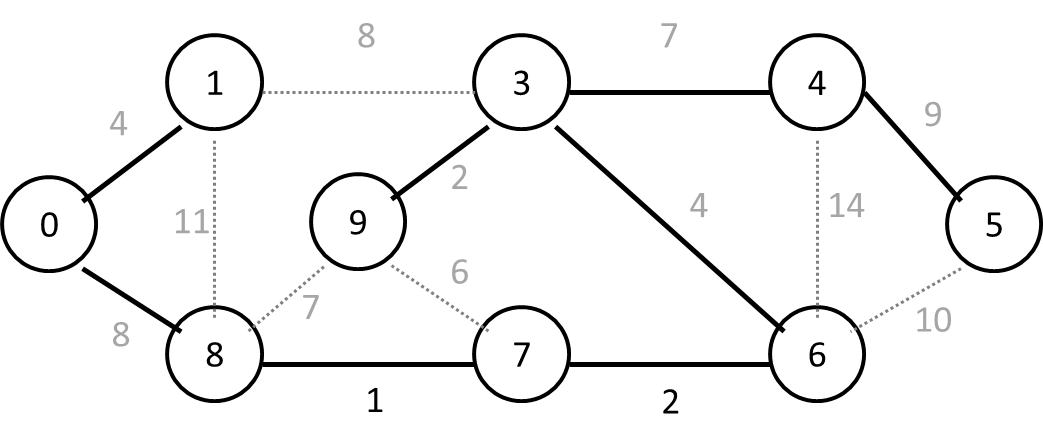
* 1. Note que a aresta (7,9) apesar de ser a de menor peso dentre as disponíveis não pode ser inserida, pois forma um ciclo, detectado pelo fato de 3 e 9 estarem na mesma partição. Pd={**{0,1}**,...**{6,7,8**,**3,9}**}.)



* 1. O algoritmo segue e encontra uma situação onde duas arestas de menor peso dentre as disponíveis tem o mesmo peso e são excludentes e daí bifurca para duas soluções equivalentes.

 e 

* 1. Dai resultam as duas soluções de custo total 37 mostradas abaixo.

 e 

(b) Ao adotar uma partição dinâmica dos nós do grafo, representada por uma floresta, com a implementação de UNION por altura e FIND com compressão de caminhos é vantajoso pois:

* Cada partição corresponde aos nós de uma árvore da floresta. Logo, determinar se uma aresta {*n, m*} pode ser adicionada, ou seja, se *n* e *m* pertencem à mesma árvore, reduz-se a determinar se *n* e *m* pertencem à mesma partição.
* Para uma sequencia de n Unions e Finds, a implementação partições utilizando UNION por altura e FIND com compressão de caminhos possui complexidade O(n log\*n), no pior caso, e tempo médio por operação de O(log\*n).

**Questão 3 (2.0 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 Γ(*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:*

*Common Neighbors*:

*Jaccard:*

(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 da Figura 5. Mostre todos os passos intermediários da computação.

G

F

A

C

B

E

D

**Figura 5.** Grafo não dirigido.

(b) **(1.0 ponto)** Argumente qual das três métricas é a MENOS 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)* Γ(*A*) = *{B, C} ={B, C}*

Γ(*E*) = *{B, C, F} ={B, C, F}*

Γ(*F*) = *{D, B, E, G} ={B}*

*={B, C, D, E, G}*

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

*cn(A,E) = 2 cn(A,F) = 1*

*jc(A,E) = 2/3 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, é a menos apropriada para se computar um *score* que leve em consideração a vizinhança no grafo.

**Questão 4 (2.0 pontos)**. Implemente uma função em C que receba como entrada um grafo não dirigido *G* e um nó *x* do grafo *G* e devolva o vizinho *y* de *x* com o maior *Common Neighbors* *score* com relação a *x*, ou seja, com o maior *cn(x,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:

int Max-Common-Neighbors(Graph\* G, int x);

onde x é um inteiro indicando a posição do nó no vetor vv. Da mesma forma, a função devolve um inteiro indicando a posição no vetor vv do vizinho de x com maior *Common Neighbors* *score*, ou -1, se x não tiver vizinhos.

***Resposta***

int Max-Common-Neighbors(Graph\* G, int x)

{

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

int cn\_max = 0, cny; /\* CN max e CN score entre x e y \*/

listNode\* u, v, w;

for (u=G->vv[x]; u!=NULL; u=u->link) /\* calcula o CN para cada viz. y de x \*/

{

y = u->vertex;

cny = 0;

for (v=G->vv[x]; v!=NULL; v=v->link) /\* computa gamma(x) inter gamma(y) \*/

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

if (v->vertex == w->vertex) cny = cny + 1;

if (cny > cn\_max) { cn\_max = cny; y\_max = y; };

}

return y\_max;

}