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.



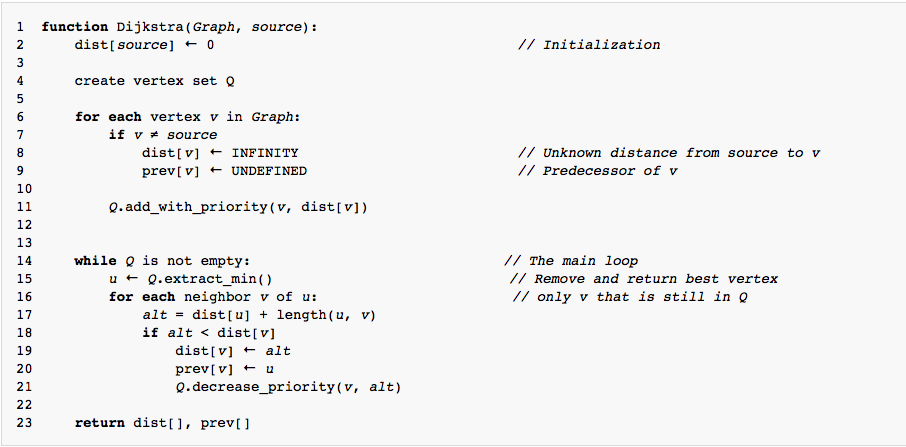
1

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 |  |  |  |  |  |  |  |  |  |  |  |  |
| C |  | 0 |  | X | C | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  | ∞ |  |  |  | ∞ |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E |  | ∞ |  |  |  | 1 | C |  |  |  |  |  |  |  |  |  |  |  |  |
| F |  | ∞ |  |  |  | 3 | C |  |  |  |  |  |  |  |  |  |  |  |  |
| 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 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

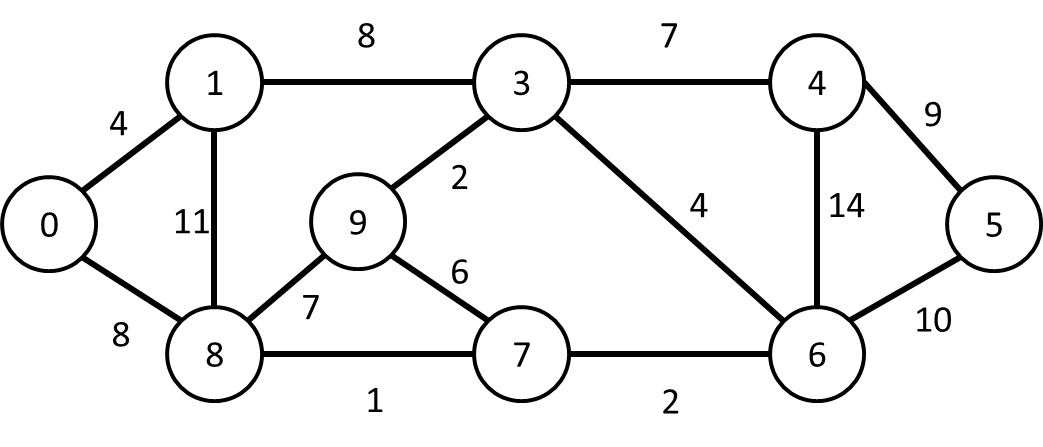
Passo: Passo: Passo: Passo: Passo:

**Figura 3**. Apresentação esquemáticaos estados dos vetores dist e prev e da variável u.

**(b)**

**Questão 2 (3.0 pontos)**

1. **(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.



**Figura 4.** Grafo não dirigido.

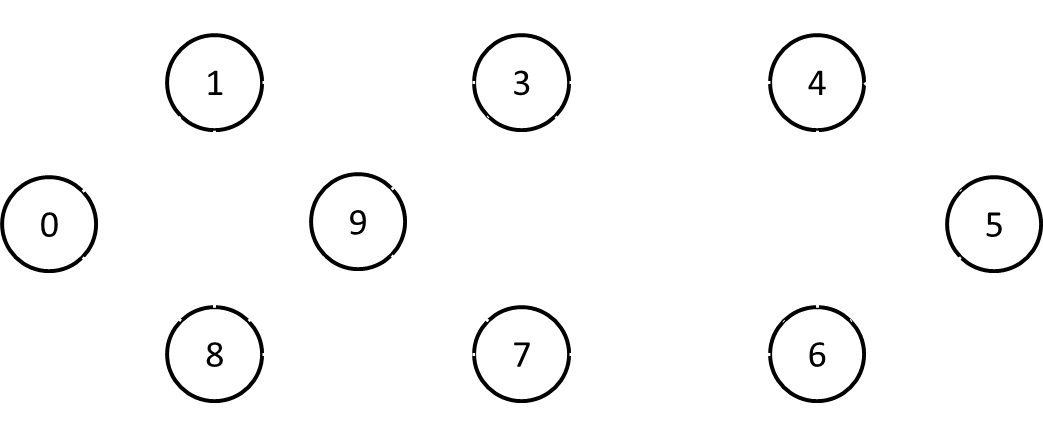
(b) **(1.0 ponto)** Explique qual a vantagem, em termos de custo do processamento, de adotar uma partição dinâmica dos nós do grafo, representada por uma floresta, com a implementação de UNION por altura e FIND com compressão de caminhos?

**Resposta**

(a) Passo inicial:

Partição: Pd={{0},{1},...,{7},{8},{9}}.

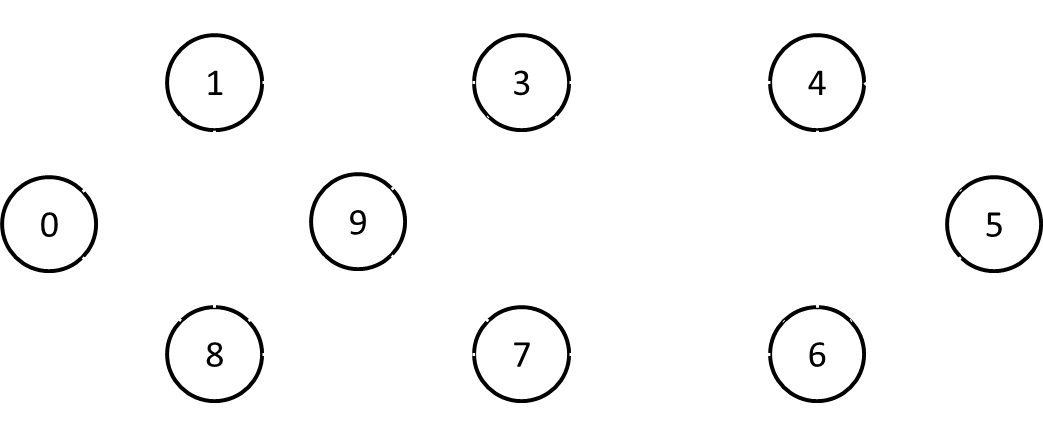
Árvore:



Passo 1:

Partição: Pd=

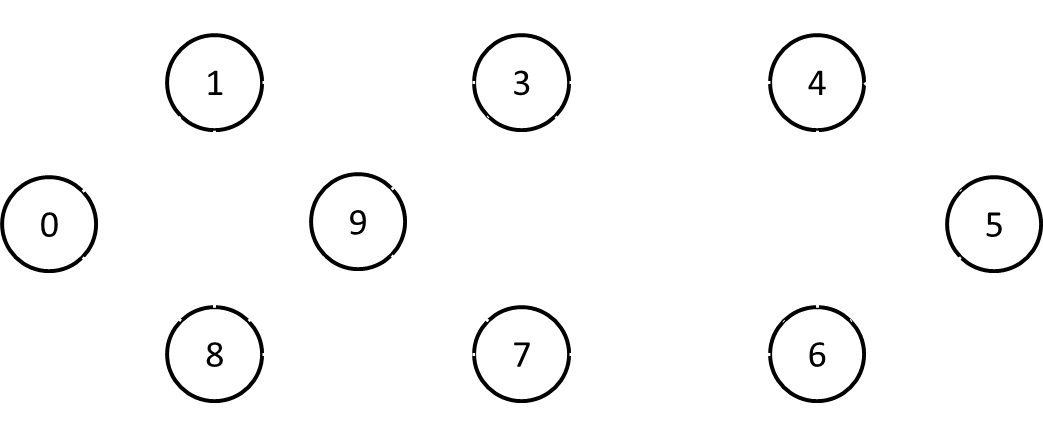
Árvore:



Passo 2:

Partição: Pd=

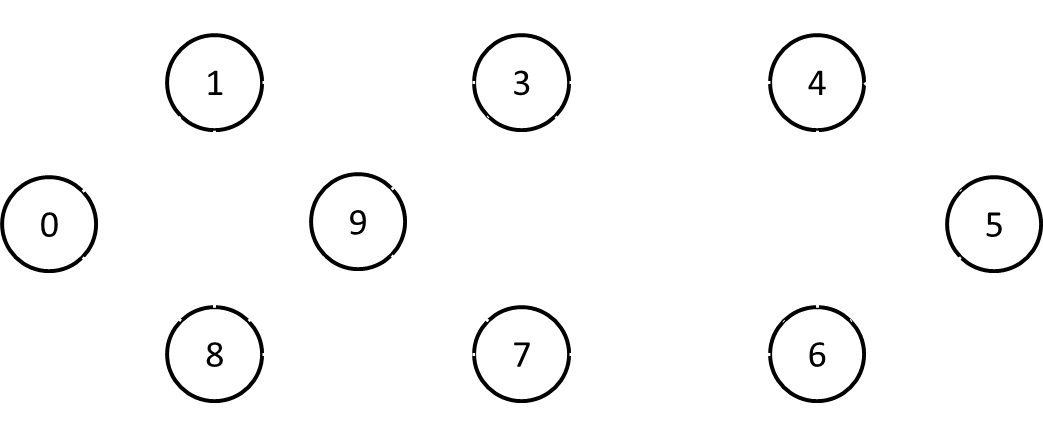
Árvore:



Passo 3:

Partição: Pd=

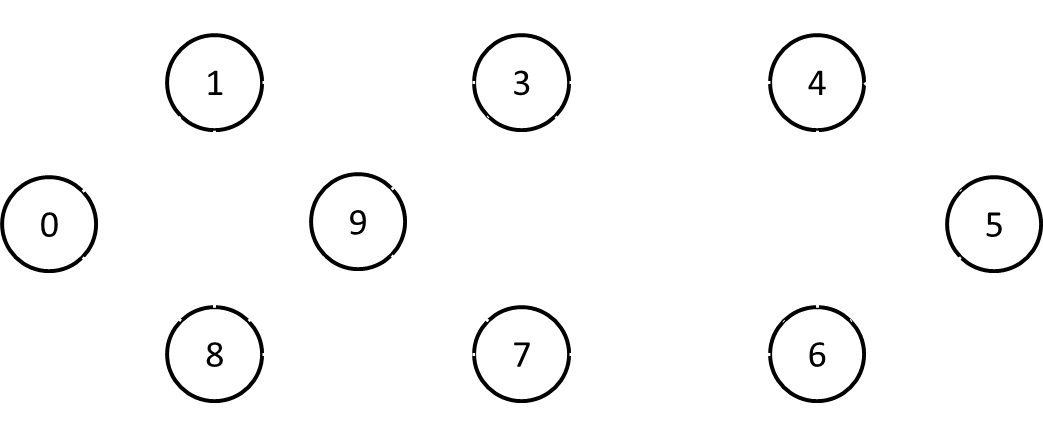
Árvore:



Passo 4:

Partição: Pd=

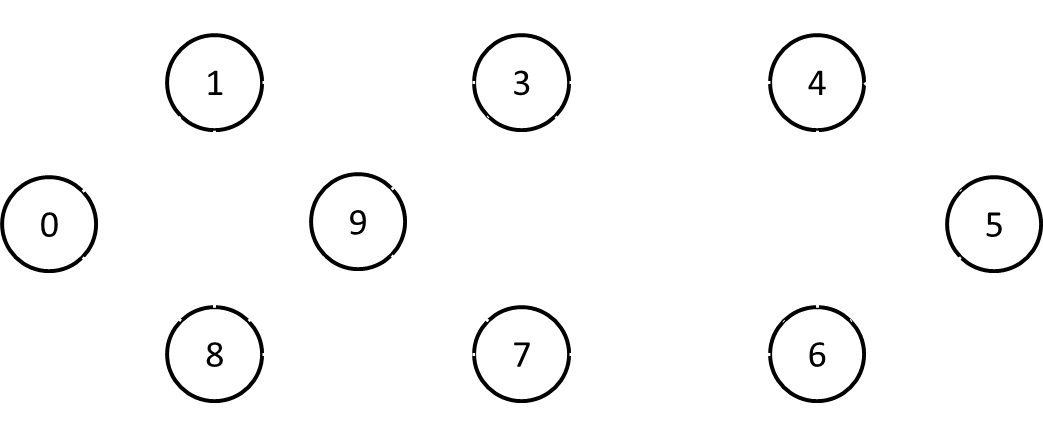
Árvore:



Passo 5:

Partição: Pd=

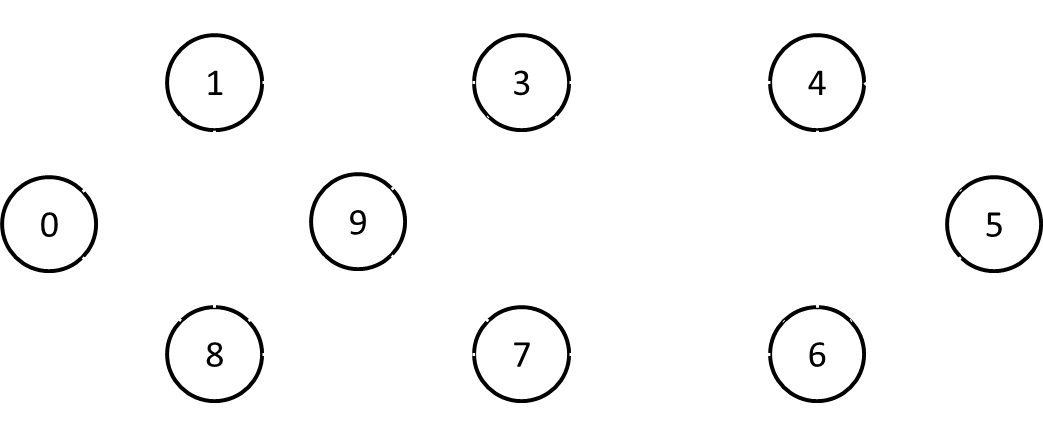
Árvore:



Passo 6:

Partição: Pd=

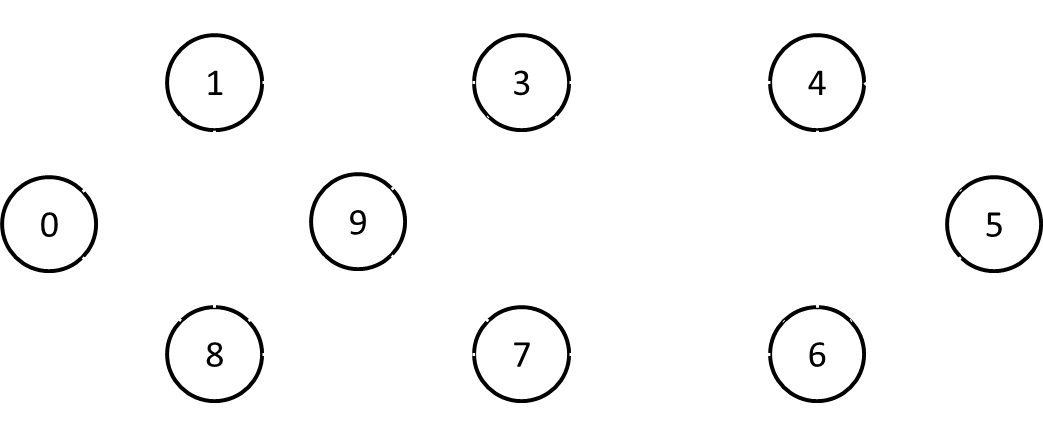
Árvore:



Passo 7:

Partição: Pd=

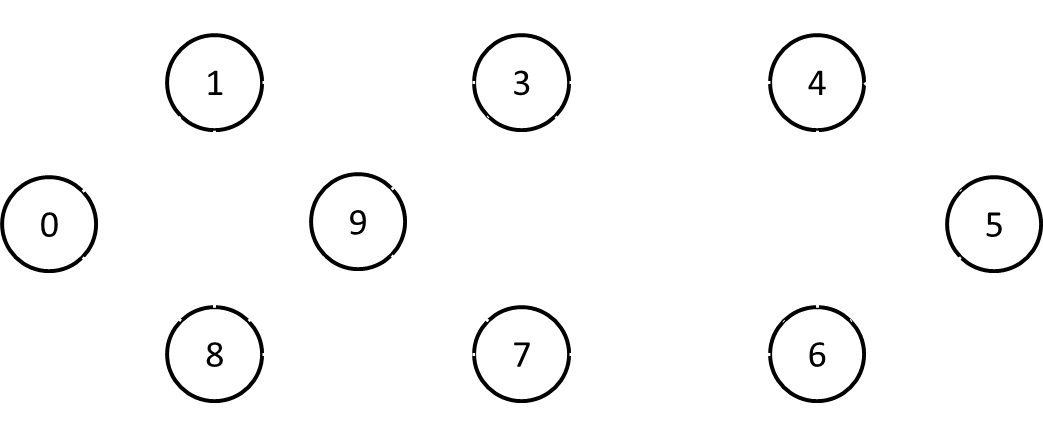
Árvore:



Passo 8:

Partição: Pd=

Árvore:



**(b)**

**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:**

**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:**