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

|  |  |  |
| --- | --- | --- |
| 1a) | 2,0 |  |
| 2a) | 2,0 |  |
| 3a) | 2,0 |  |
| 4a) | 2,0 |  |
| 5a) | 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 (2,0 pontos)** Uma tabela de dispersão (*hash table*) de tamanho 11 é implementada com *encadeamento externo* através da seguinte função de dispersão:   
Nela são inseridas 8 dados que possuem as seguintes chaves de busca (nesta ordem):

7,10,15,14,17,3,21,25

1. (0,5 ponto) Desenhe a estrutura de dados após a inserção destas chaves.
2. (1,0 ponto) Explique, com base no exemplo anterior, quais são os casos que devem ser considerados para implementar a operação de remoção de uma chave x:

remocao(x)

Entrada: um valor x de chave

Saída: NULL, se x não é encontrada

p, ponteiro para o elemento que contém x

1. (0,5 ponto) Qual a complexidade temporal, no pior caso, da operação de remoção? Explique sua resposta.

**Resposta**

a)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *x* | *h(x)* |  |  |  |  |
| 7 | **7** |
| 10 | **10** |
| 15 | **4** |
| 14 | **3** |
| 17 | **6** |
| 3 | 3 |
| 21 | 10 |
| 25 | 3 |

|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |  | 14 |  | 3 |  | 25 |
| 4 |  |  | 15 |
| 5 |  |  |  |
| 6 |  |  | 17 |
| 7 |  |  | 7 |
| 8 |  |
| 9 |  |
| 10 |  |  | 10 |  | 21 |

1. Há 4 casos a considerar:

Caso 1: x não está na estrutura. Devolva NULL.

Caso 2: x está no único elemento de uma lista. Remova x e coloque NULL na entrada da tabela.

Caso 3: x está no primeiro elemento p de uma lista. Remova x e faça a tabela apontar para o elemento seguinte a p na lista.

Caso 4: Nenhum dos casos acima. Remova x da lista a que pertencia, como usual.

c) No pior caso, a função de remoção é *O*(*n*), onde *n* é o número de chaves na tabela, já que a tabela pode degenerar em uma única lista encadeada contendo todas as chaves.

**Questão 2 (2,0 pontos)** Considere a seguinte sequencia de inteiros: 90, 60, 30, 15, 45.

* 1. (0,5 ponto) Mostre, passo a passo, como o vetor armazenando um *heap* mínimo é construído pela inserção sucessiva destes 5 elementos, na ordem dada. Comente brevemente cada passo do algoritmo de inserção.
  2. (1,0 ponto) Mostre, passo a passo, como fica o vetor após a remoção apenas do segundo menor elemento do *heap* construído no Item (a).
  3. (0,5 ponto) Sugira um algum algoritmo mais eficiente para construção de um *heap* mínimo com estes mesmos inteiros e comente se ao final o vetor ficará exatamente igual ou não ao do item (a)?

***Resposta***

a)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **Explicação** |
| 90 |  |  |  |  | Acrescente 90 depois do final do heap (que está vazio). | |
| 90 | 60 |  |  |  | Acrescente 60 depois do final do heap. | |
| 60 | 90 |  |  |  | Compare com o pai ((1-1)/2=0) e troque. | |
| 60 | 90 | 30 |  |  | Acrescente 30 depois do final do heap. | |
| 30 | 90 | 60 |  |  | Compare com o pai ((2-1)/2=0) e troque. | |
| 30 | 90 | 60 | 15 |  | Acrescente 15 depois do final do heap. | |
| 30 | 15 | 60 | 90 |  | Compare com o pai ((3-1)/2=1) e troque. | |
| 15 | 30 | 60 | 90 |  | Compare com o pai ((1-1)/2=0) e troque. | |
| 15 | 30 | 60 | 90 | 45 | Acrescente 45 depois do final do heap. | |
| 15 | 30 | 60 | 90 | 45 | Compare com o pai ((4-1)/2=1) e não troque. Pare. | |

b)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |  | **Explicação** |
| 15 | 30 | 60 | 90 | 45 |  | Remova a raiz, 15, trocando-a com o último, 45. |
| 45 | 30 | 60 | 90 | 15 |  | O último passa a estar na posição 3; o elemento retirado na posição 4. O heap precisa ser rearrumado. Troque 45 com o menor filho, 30. |
| 30 | 45 | 60 | 90 | 15 |  | O segundo menor elemento é a nova raiz, 30. Remova 30. Para remover 30, troque-a pelo último, 90. |
| 90 | 45 | 60 |  | 15 |  | O heap precisa ser rearrumado. Troque 90 com o menor filho, 45. |
| 45 | 90 | 60 |  | 15 |  | O último elemento salvo, 15, continua na posição 4. Reinsira 15 no heap, colocando-o depois do final do heap. |
| 45 | 90 | 60 | 15 |  |  | O heap precisa ser rearrumado; troque 15 com o seu pai, 90. |
| 45 | 15 | 60 | 90 |  |  | O heap precisa ser rearrumado; troque 15 com o seu pai, 45. |
| 15 | 45 | 60 | 90 |  |  | O heap não precisa mais ser rearrumado. Pare. |

c) É possível alocar os n/2+1 primeiros elementos a serem inseridos no final do vetor e depois ir inserido na posição anterior as preenchidas o próximo elemento, verificando a ordenação com seus filhos. Os heaps não precisam ser iguais. Por definição, basta que a prioridade do pai seja menor do que a dos filhos, mas a prioridade do filho à esquerda pode ser maior ou menor do que a do filho à direita.

**Questão 3 (2,0 pontos)** Implemente de forma não recursiva, visitando o menor número possível de nós, uma função que calcule a *largura* de uma árvore de busca binária, definida como a diferença entre o maior e o menor valor dentre os valores de chave na árvore. Por simplicidade, assuma que os valores de chave são inteiros. A função deve ter o seguinte protótipo:

int abb\_largura (Abb\* r)**;**

A função recebe como entrada um apontador para a raiz da árvore e retorna a largura da árvore.

Adote a seguinte estrutura para os nós:

typedef struct \_abb Abb;

struct \_abb {

int chave;

Abb\* esq;

Abb\* dir;

};

***Resposta***

int abb\_largura (Abb\* r){

Abb\* min, max;

if (r==NULL) return 0;

min = r;

while(min->esq != NULL)

min = min->esq;

max = r;

while(max->dir != NULL)

max = max->dir;

return (max->chave – min->chave);

}

**Questão 4 (2,0 pontos)** Considere uma família de árvores definida de forma semelhante a árvores B, exceto que:

* Os nós possuem tamanho variável entre 128 e 256 bytes
* As chaves possuem tamanho variável, entre 22 bytes e 44 bytes
* Os ponteiros ocupam 4 bytes
* Em cada nó, há um campo a mais, de 4 bytes, indicando o número de chaves que o nó efetivamente armazena

1. (1,0 ponto) Qual é o maior número de chaves que uma árvore de altura 2 armazena (uma árvore que só tem a raiz possui altura 0, por convenção)? Explique cuidadosamente sua resposta.
2. (0,5 ponto) Qual é o número mínimo de chaves, estando todos os nós completamente preenchidos, que uma árvore de altura 2 armazena? Explique cuidadosamente sua resposta.
3. (0,5 ponto) A escolha dos tamanhos mínimo e máximo dos nós é adequada para o tamanho das chaves? Explique sua resposta.

***Resposta***

a) O maior número de chaves será atingido quando todas as chaves possuírem o menor tamanho e os nós, o maior tamanho. Seja *m* o número de chaves que podem ser acomodadas em um nó. Cada chave é acompanhada por um ponteiro, exceto a primeira, que é acompanhada por 2 ponteiros. Há ainda 4 bytes adicionais para indicar o número de chaves. Para o caso de todas as chaves terem o menor tamanho, 16 bytes, e os e os nós, o maior tamanho, 256 bytes, temos:

256 = (4 bytes para número de chaves) +

(4 bytes para o primeiro ponteiro) +

(4 bytes para cada ponteiro + 22 bytes de cada chave) \* *m*

*m* = (256-8)/26 = 248/26 = 9 chaves

Logo, cada nó terá 10 filhos. Portanto, como a árvore possui altura 2, o número máximo de chaves será:

*CM* = 9 + 9\*10 + 9\*10\*10 = 9 + 90 + 900 = 999

b) O menor número de chaves, quando os nós estão completos, será atingido quando todas as chaves possuírem o maior tamanho e os nós, o menor tamanho. Cada nó completo terá *p* chaves de tamanho máximo, onde *p* é dado por:

128 = 4 + 4 + (4 + 44) \* *p*

*p* = (128-8)/48 = 120/48 = 2

Logo, cada nó terá 3 filhos. Portanto, como a árvore possui altura 2, o número de chaves de tamanho máximo em nós completos será:

*Cm* = 2 + 2\*3 + 2\*3\*3 = 2 + 6 + 18 = 26

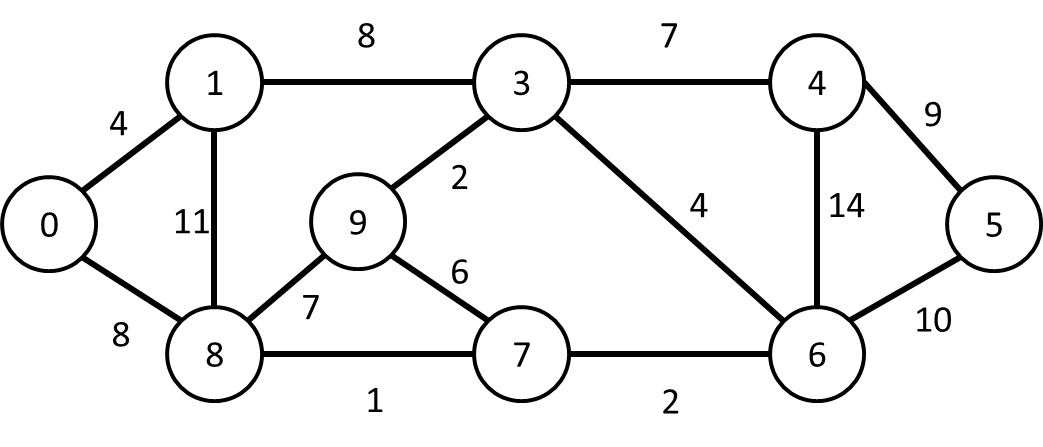
c) Os nós são demasiadamente pequenos. Mesmo no melhor caso - item (a), o número de chaves por nó é pequeno. No pior caso - item (b), a árvore degenera em uma árvore binária.

**Questão 5** **(2,0 pontos)**

(a) (1,0 ponto) Mostre os passos do algoritmo de Kruskal para calcular uma árvore geradora mínima do grafo mostrado na figura abaixo. Considere que o algoritmo adota uma partição dinâmica dos nós do grafo, representada por uma floresta, com a implementação de UNION por altura e FIND com compressão de caminhos.

**Respostas que não utilizarem uma partição dinâmica como pedido não serão consideradas.**

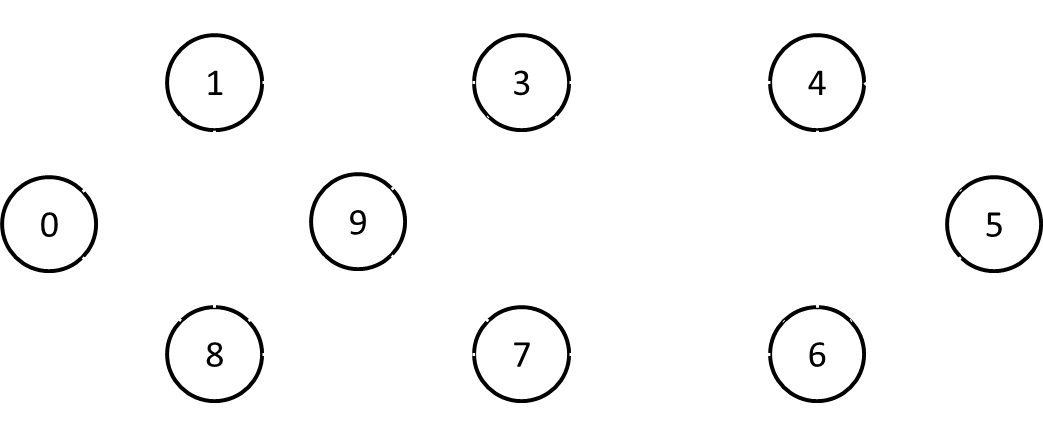
(b) (1,0 ponto) Explique qual a vantagem, em termos de custo do processamento, de adotar uma partição dinâmica dos nós do grafo, representada por uma floresta, com a implementação de UNION por altura e FIND com compressão de caminhos?



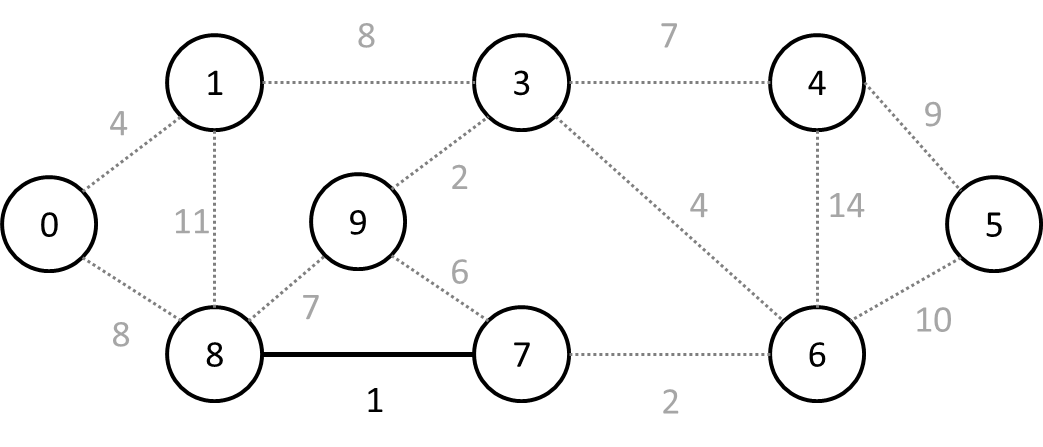
Resposta

(a)

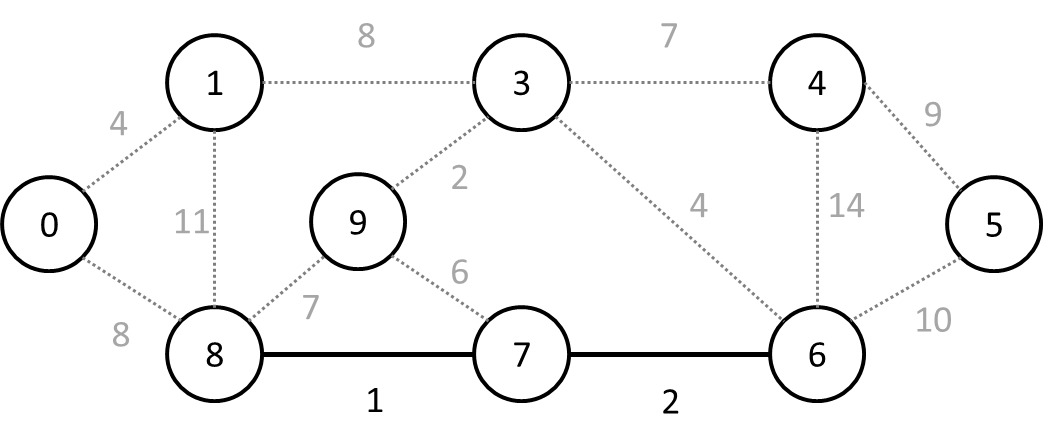
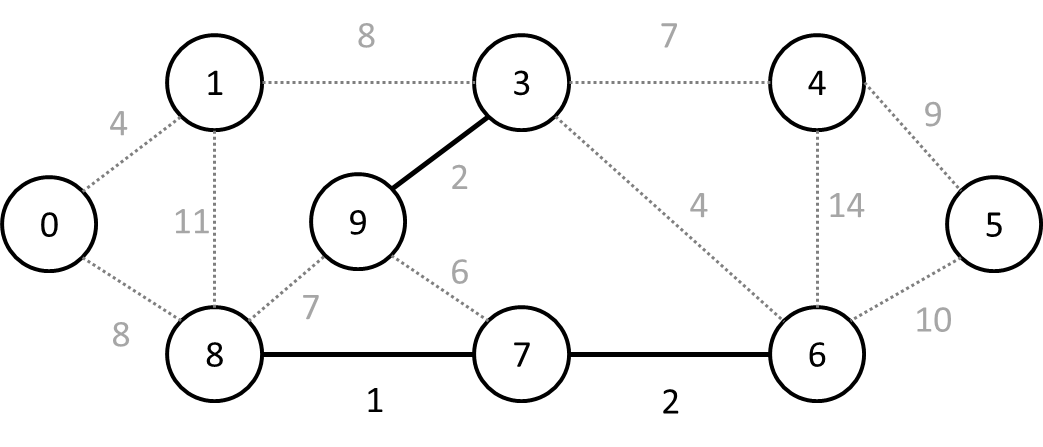
* 1. Crie uma partição dinâmica onde cada vértice é um conjunto isolado. Pd={{0},{1},...{7},{8},{9}}.



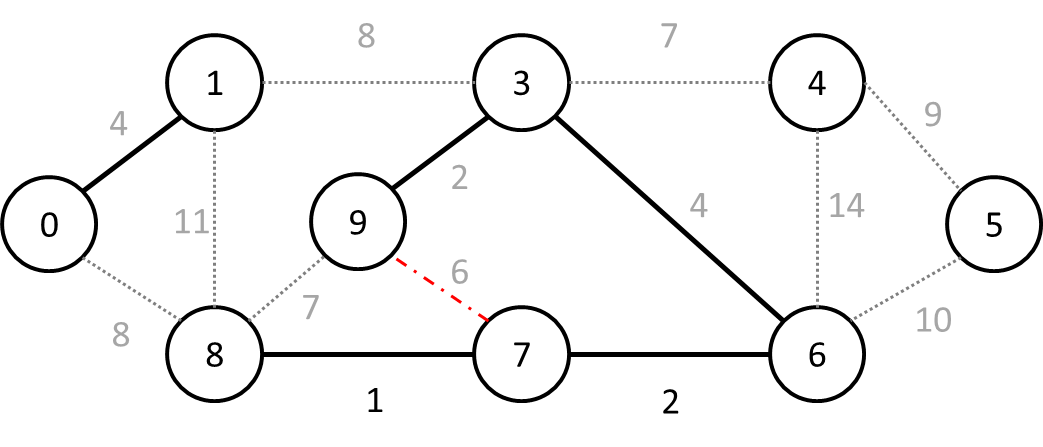
* 1. Adicione uma aresta de menor peso dentre as disponíveis. Ao inserir uma aresta faça uma união dos dois conjuntos correspondentes aos vértices dela. Pd={{0},{1},...**{7,8}**,{9}}.



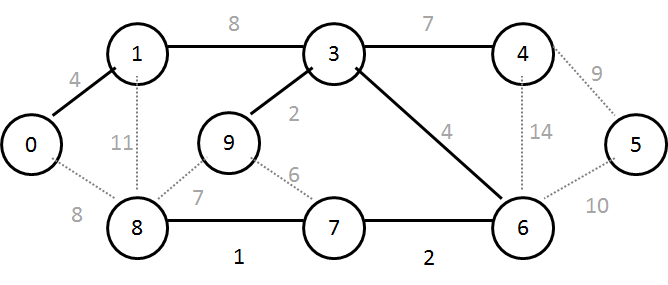
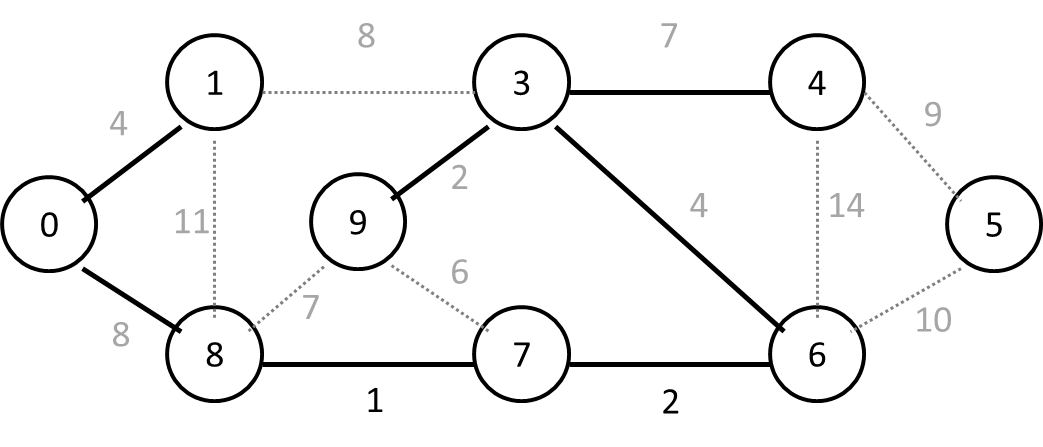
* 1. Continue adicionando uma aresta de menor peso dentre as disponíveis desde que sua inserção não forme ciclos na árvore que está sendo gerada. Ou seja, desde que seus vértices não estejam numa mesma partição. Pd={{0},{1},...**{6,7,8}**,**{3,9}**}.

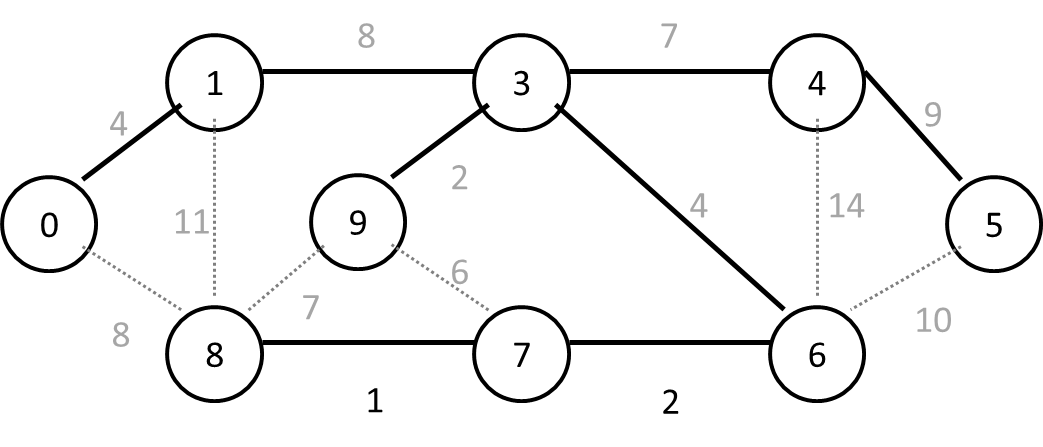
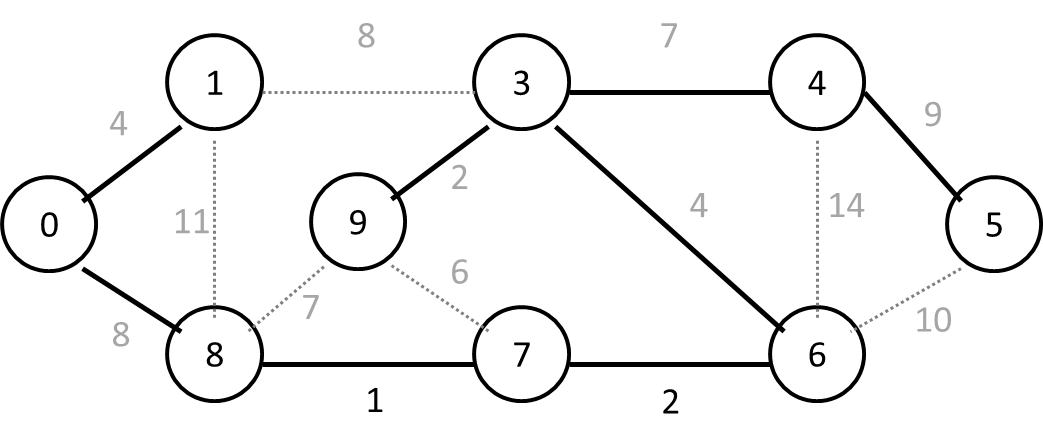
* 1. Note que a aresta (7,9) apesar de ser a de menor peso dentre as disponíveis não pode ser inserida, pois forma um ciclo, detectado pelo fato de 3 e 9 estarem na mesma partição. Pd={**{0,1}**,...**{6,7,8**,**3,9}**}.)



* 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).