Aluno(a):_____Matrícula:_____

1ª)	2.0	
2ª)	2.0	
3ª)	3.0	
4ª)	3.0	
	10.0	

I. A prova é individual e sem consulta. Qualquer tentativa de "cola" resultará na anulação da prova de ambos os alunos envolvidos.
II. A interpretação faz parte da questão. Em caso de dúvida escreva a dúvida e a sua interpretação na resposta. NÃO HÁ PERGUNTAS AO PROFESSOR DURANTE A PROVA.
III. O tempo de prova é 1:45 h.
IV. As respostas devem seguir as questões. Caso precise de rascunho use o verso da folha.
V. A prova pode ser feita a lápis.

Questão 1 (2.0 pontos). Considere uma rotina para inserção de chaves, definidas como números inteiros, em uma tabela de dispersão com endereçamento aberto e função de dispersão h definida como

$$h(K, i) = ((K \% N) + i) \% N$$

onde

K = chave a ser inserida
N = tamanho da tabela de dispersão
i = 0, 1, 2, ... = número da tentativa de inserção da chave na tabela

(a) (1.0 ponto) Assuma que a tabela está originalmente vazia e que N=1007. Mostre a computação da função de dispersão e indique em que posição da tabela cada chave será inserida, nesta ordem: 17, 1024, 18, 1031, 2031.

(b) (1.0 ponto) Escreve uma rotina em C para inserção de chaves, definidas como números inteiros, em uma tabela de dispersão com endereçamento aberto com a função de dispersão definida acima. Assuma que as posições são preenchidas, inicialmente, com o valor -1 e que a tabela seja declarada como:

```
#define N 1007;     /* tamanho da tabela de dispersão       */
#define VAZIO (-1); /* indica que a posição está desocupada */
int tabela[N];      /* tabela de dispersão                  */
```

A função deve ter o seguinte protótipo:

```
int insere(int K)
```

e deve retornar:

`-1` se a chave não puder ser inserida
`0` se a chave já existe
`1` se a chave for inserida corretamente

Resposta:

(a) h(17,0) = ((17%1007)+0)%1007 = 17 --- inserida na posição 17

h(1024,0) = ((1024%1007)+0)%1007 = ((17)+0)%1007 = 17 --- colisão
h(1024,1) = ((1024%1007)+1)%1007 = ((17)+1)%1007 = 18 --- inserida na posição 18

h(18,0) = ((18%1007)+0)%1007 = ((18)+0)%1007 = 18 --- colisão
h(18,1) = ((18%1007)+1)%1007 = ((18)+1)%1007 = 19 --- inserida na posição 19

h(1031,0) = ((1031%1007)+0)%1007 = ((24)+0)%1007 = 24 --- inserida na posição 24

h(2031,0) = ((2031%1007)+0)%1007 = ((17)+0)%1007 = 17 --- colisão
h(2031,1) = ((2031%1007)+1)%1007 = ((17)+1)%1007 = 18 --- colisão
h(2031,2) = ((2031%1007)+2)%1007 = ((17)+2)%1007 = 19 --- colisão
h(2031,3) = ((2031%1007)+3)%1007 = ((17)+3)%1007 = 20 --- inserida na posição 20

(b)

```
int insere(int K)
{
        int p, i;
        for (i = 0; i < N; i++)
        {
              p = ((K % N) + i) % N;
              if (tabela[p] == VAZIO)       /* Insere a chave    */
              {
                 tabela[p] = K;
                 return 1;
              }
              if (tabela[p] == K)           /* Chave já existe   */
                 return 0;

        }
        return (-1);
}
```

1) **Questão 2 (2.0 pontos).** Considere a seguinte sequencia de inteiros: 90, 60, 30, 15.

   a) (0.5 ponto) Mostre, passo a passo, como o vetor armazenando um *heap* mínimo é construído pela inserção sucessiva destes 4 elementos, na ordem dada. Comente brevemente cada passo do algoritmo de inserção.

   b) (0.5 ponto) Mostre, passo a passo, como fica o vetor após a remoção do menor elemento e do segundo menor elemento do heap construído no Item (a). Comente brevemente cada passo do algoritmo de remoção.

   c) (1.0 ponto) Descreva como você implementaria a operação

   ```
   remove(n, Heap* h)
   ```

   que remove e devolve os `n` menores elementos de um heap mínimo `h`, *sem usar um vetor auxiliar de* `n` *posições*. Não é necessário implementar a operação em C, mas apenas descrever a solução em alto nível.

Resp:

a)

90				
90	60			Acrescente 60 ao final do heap
60	90			Compare com o pai e troque
60	90	30		Acrescente 30 ao final do heap
30	90	60		Compare com o pai e troque
30	90	60	15	Acrescente 15 ao final do heap
30	15	60	90	Compare com o pai e troque
15	30	60	90	Compare com o pai e troque

b)

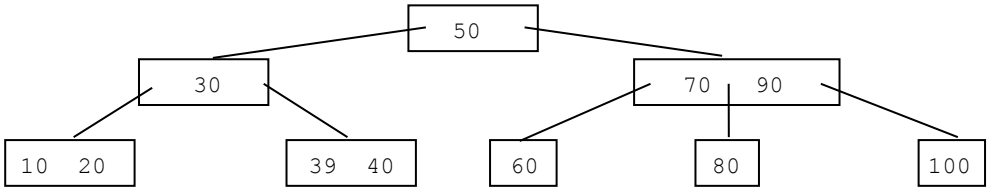
15	30	60	90	Devolva a raiz
90	30	60		Coloque o último na raiz
30	90	60		Troque com o menor filho
30	90	60		Devolva a raiz
60	90			Coloque o último na raiz
60	90			Pare: em ordem

c) Use a mesma ideia do heapsort:

1) Troque o menor elemento, que está na raiz, com o último elemento no vetor que representa o heap.
2) Reorganize o heap, como no algoritmo de remoção.
3) Repita os passos (1) e (2) até retirar os n menores elementos do heap.
4) Os n menores elementos estarão nas n últimas posições do heap.

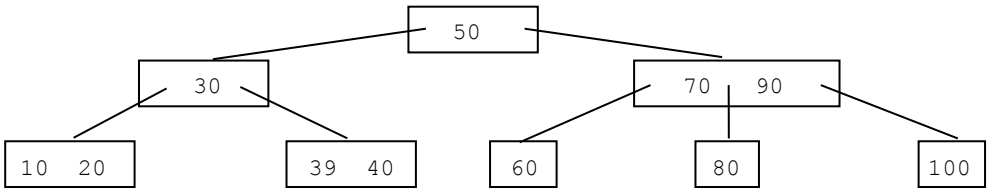
Questão 3 (3.0 pontos).

a) **(0.5 ponto)** Insira a chave 31 na árvore B de ordem 4 abaixo:



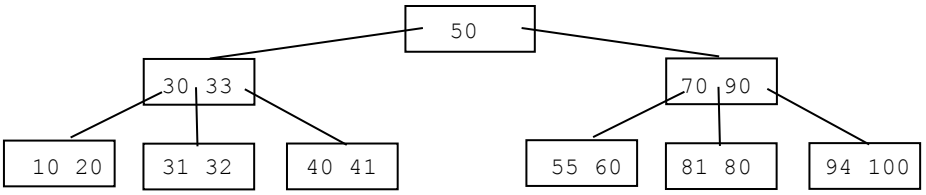
Descreva, passo a passo, todas as modificações sofridas, redesenhando apenas a parte da árvore modificada a cada passo. Justifique porque não há mais modificações a fazer.

b) **(0.5 ponto)** Insira a chave 31 na árvore 2-3 abaixo:



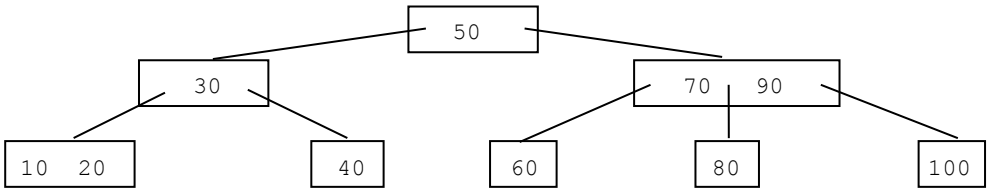
Descreva, passo a passo, todas as modificações sofridas, redesenhando apenas a parte da árvore modificada a cada passo. Justifique porque não há mais modificações a fazer.

c) **(1.0 ponto)** Remova a chave 70 da árvore B de ordem 4 abaixo:.



Descreva, passo a passo, todas as modificações sofridas, redesenhando apenas a parte da árvore modificada a cada passo. Justifique porque não há mais modificações a fazer.

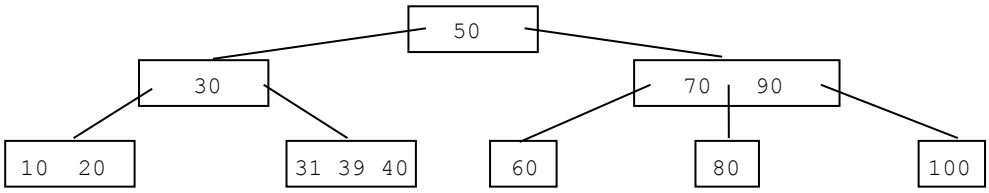
d) **(1.0 ponto)** Remova a chave 70 da árvore B de ordem 3 abaixo:.



Descreva, passo a passo, todas as modificações sofridas, redesenhando apenas a parte da árvore modificada a cada passo. Justifique porque não há mais modificações a fazer.

Resp.:

a) Insira a chave 31 na árvore B de ordem 4 abaixo:



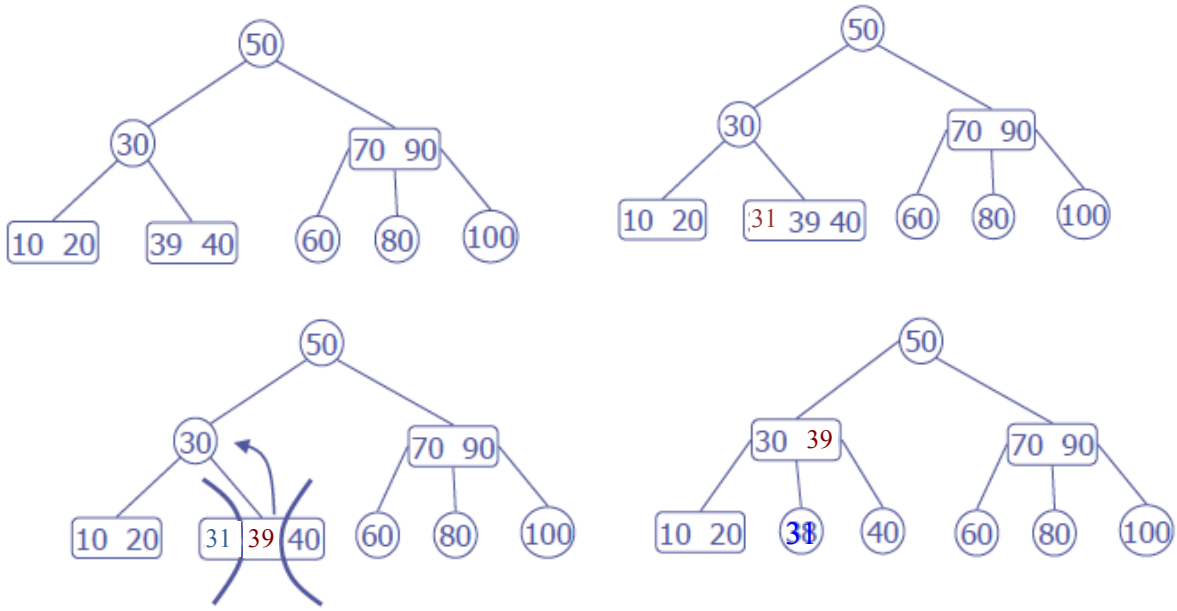
A inserção pode ser feita sem modificações nos nós pois uma árvore B de ordem 4 admite no máximo

$$k_{max} = (4\text{-}1)=3$$

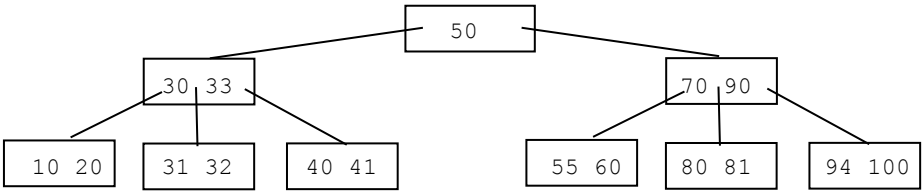
chaves por nó.

b) Insira a chave 31 na árvore 2-3 abaixo:

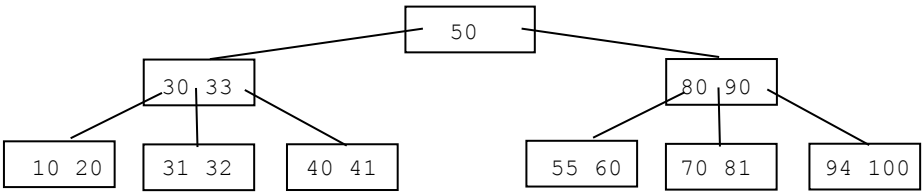
(Ver notas de aula sobre árvores 2-3, página 11, trocando 38 por 31)



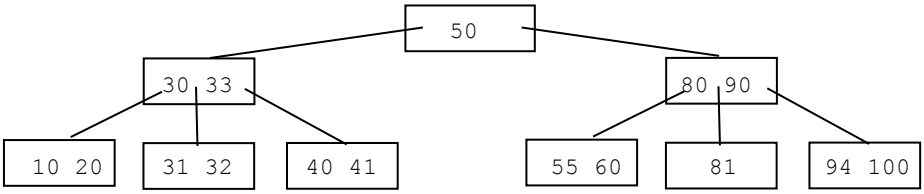
c) Remova a chave 70 da árvore B de ordem 4 abaixo:.



Troque 70 a sua sucessora 80.



Remova 70.



Não é necessário fazer qualquer outra alteração pois uma árvore B de ordem 4 admite no mínimo

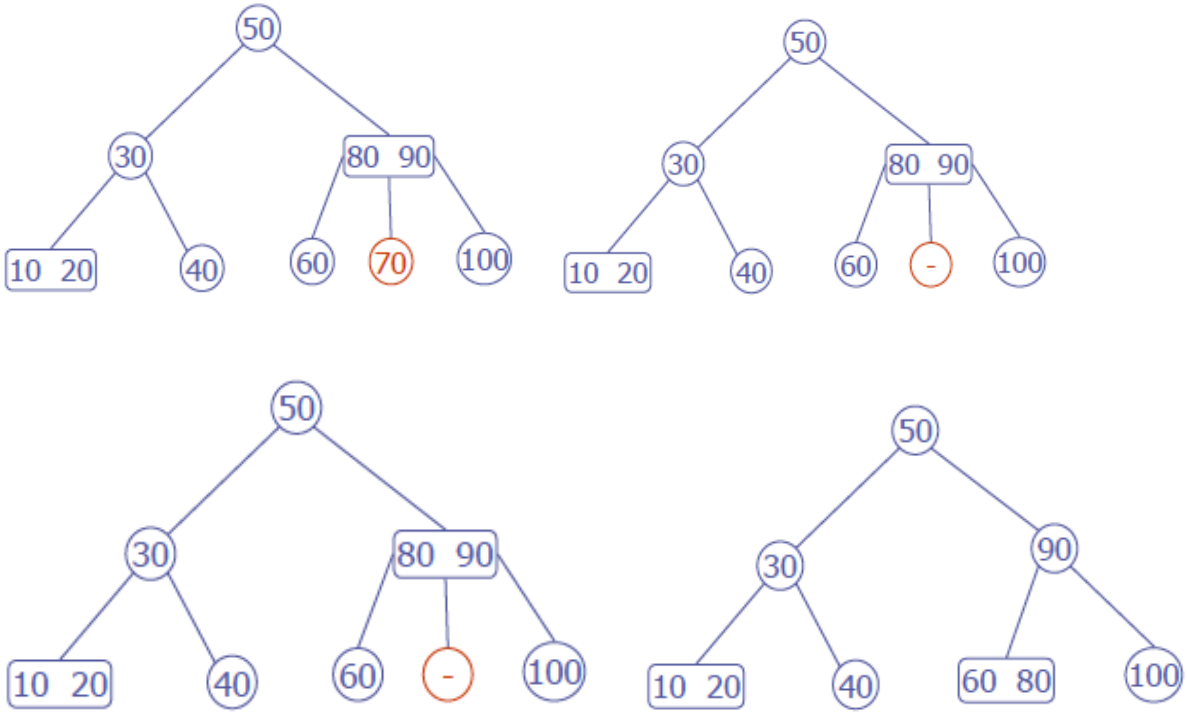
$$k_{min} = (4/2\text{-}1)=1$$

chaves por nó.

d) Remova a chave 70 da árvore B de ordem 3 abaixo:.

(Ver notas de aula sobre árvores 2-3, página 30)

Troque 70 com 80 inicialmente. Em seguida, prossiga da seguinte forma:

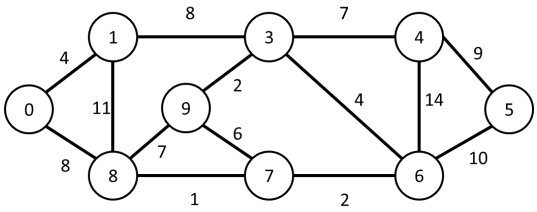


Questão 4 (3.0 pontos).

(a) **(2.0 pontos)** Mostre os passos do algoritmo de Kruskal para calcular uma árvore geradora mínima do grafo mostrado na figura 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.

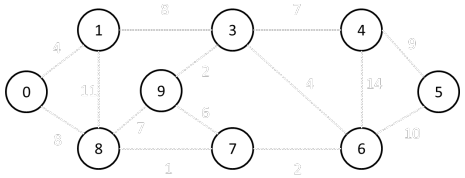
(b) **(1.0 ponto)** Explique qual a vantagem, em termos de custo do processamento, de adotar uma partição dinâmica dos nós do grafo, representada por uma floresta, com a implementação de UNION por altura e FIND com compressão de caminhos?



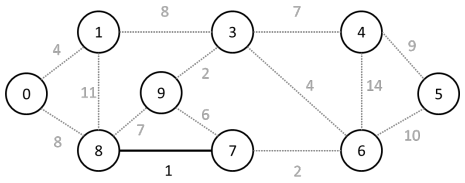
Resposta

(a)

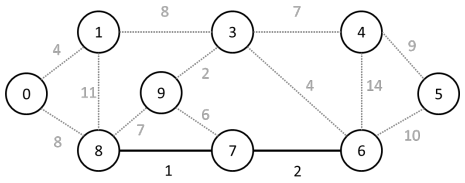
a) Crie uma partição dinâmica onde cada vértice é um conjunto isolado.
P_d={{0},{1},...{7},{8},{9}}.

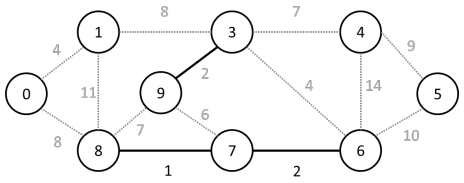


b) Adicione uma aresta de menor peso dentre as disponíveis. Ao inserir uma aresta faça uma união dos dois conjuntos correspondentes aos vértices dela. P_d={{0},{1},...**{7,8}**,{9}}.

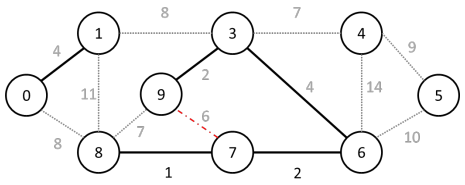


c) Continue adicionando uma aresta de menor peso dentre as disponíveis desde que sua inserção não forme ciclos na árvore que está sendo gerada. Ou seja, desde que seus vértices não estejam numa mesma partição. P_d={{0},{1},...**{6,7,8},{3,9}**}.

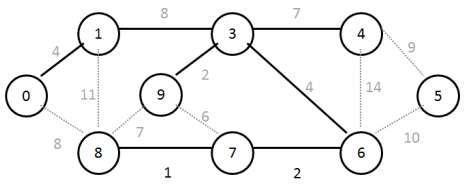




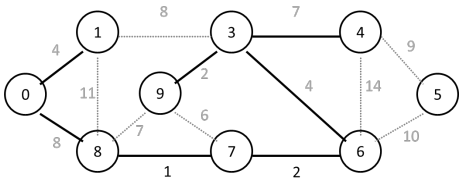
d) Note que a aresta (7,9) apesar de ser a de menor peso dentre as disponíveis não pode ser inserida, pois forma um ciclo, detectado pelo fato de 3 e 9 estarem na mesma partição.
P_d={**{0,1},...{6,7,8,3,9}**}.)



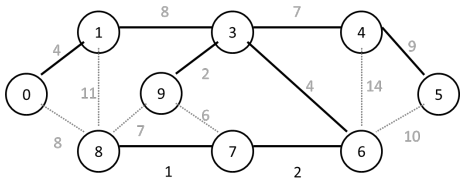
e) O algoritmo segue e encontra uma situação onde duas arestas de menor peso dentre as disponíveis tem o mesmo peso e são excludentes e daí bifurca para duas soluções equivalentes.



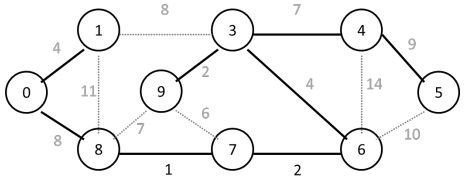
e



f) 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).