Aluno(a):_____Matrícula:_____

| | | |
|---|---|---|
| 1ª) | 1.0 | |
| 2ª) | 1.0 | |
| 3ª) | 3.0 | |
| 4ª) | 2.0 | |
| 5ª) | 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 (1.0 ponto).** Qual a ordem da complexidade em tempo dos algoritmos A e B abaixo? Explique sua resposta.

**Respostas sem explicações não serão consideradas.**

a) (0.5 ponto)

```
for ( i=1; i < n; i *= 2 ) {
   for ( j = n; j > 0; j /= 2 ) {
         for ( k = j; k < n; k += 2 ) {
               sum += (-j * k) << i/2;
         }
   }
}
```

b) (0.5 ponto)

```
int B(int n) {
    int i,j,sum = 0;
    for (i=1;i<n;i*=2) {
          for (j=1;j<i;j*=2){
                sum+=i+2*j;
          }
    }
    return sum;
}
```

Resposta:

a) O tempo de execução do laço mais externo é proporcional a $log_2(i)$, visto que $i$ dobra a cada passo. O tempo de execução do laço do meio é semelhante, exceto que $j$ é dividido pela metade, começando em $n$. O tempo de execução do último laço é proporcional a $n/2$, pois soma-se 2 a $k$ a cada passo. Como os laços estão aninhados, devem-se multiplicar todos eles. Logo, a complexidade em tempo desse algoritmo é $O(nlog_2(n))$.

b) O tempo de execução do laço mais externo é proporcional a $log_2(i)$, visto que $i$ dobra a cada passo. O tempo de execução do laço mais interno é também proporcional a $log_2(j)$, visto que $j$ dobra a cada passo. o mais interno também $log_2(n)$ vezes pois $j$ dobra a cada passo. Logo, a complexidade em tempo desse algoritmo é $O((log_2(n))^2)$.

**Questão 2 (1.0 ponto).** Considere uma família de árvores definida de forma semelhante a árvores B, exceto que:

- Os nós possuem tamanho fixo em 72 bytes
- As chaves possuem tamanho variável, entre 4 bytes e 12 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

a) (0.5 ponto) Qual será o maior número de chaves, todas de menor tamanho, que é possível acomodar em uma árvore de altura 2? Explique cuidadosamente sua resposta.

b) (0.5 ponto) Qual será o número mínimo de chaves, todas de maior tamanho, que uma árvore de altura 2 armazena? Explique cuidadosamente sua resposta.

**Respostas sem explicações não serão consideradas.**

Resposta:

a) Seja m o número de chaves que podem ser acomodadas em um nó, assumindo que todas as chaves tem o menor tamanho. 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. Logo, m será:

$72 = 4+4+(4+4)*m$
ou seja,
$m = (72-8)/8 = 64/8 = 8$ chaves

Logo, cada nó terá 9 filhos no máximo.

Portanto, como a árvore possui altura 2, o número máximo de chaves de menor tamanho será:

$C_M = 8 + 9*8 + 9*9*8 = 728$

b) O número mínimo de chaves de maior tamanho é obtido da seguinte forma.

Cada nó terá no máximo p chaves, onde p é dado por:

$72 = 4+4+(12+4)*p$

$p = (72-8)/16 = 64/16 = 4$

O número mínimo de chaves de maior tamanho será então:

- A raiz possui no mínimo 1 chave e 2 filhos
- Cada nó no primeiro nível terá no mínimo p/2=2 chaves de maior tamanho e p/2+1=3 filhos
- Cada nó no terceiro e último nível terá no mínimo p/2=2 chaves de maior tamanho

Logo, o número mínimo de chaves de maior tamanho em uma árvore de altura 2 será:
$C_m = 1 + 2*2 + 2*3*2 = 17$

**Questão 3 (3.0 pontos).** Escreva uma rotina em C para recuperar um ponto representado pela curva de Morton. A rotina deve ter o seguinte protótipo:

```
Ponto* decodeMorton(int m)
```

A rotina:

- recebe como entrada um inteiro `m`, onde
  - `m` possui 16 bits
  - `m` codifica as coordenadas de um ponto `p`, começando com a coordenada `x`
- aloca e preenche as coordenadas do ponto `p` codificado em `m`
- devolve um ponteiro para o ponto `p`

Assuma, por simplicidade, que um ponto possui coordenadas inteiras, definidas pelas 8 bits mais à direita de números inteiros. O tipo Ponto é definido como

```
typedef struct ponto Ponto;
struct ponto {
 int x;      /* usa apenas os 8 bits mais à direita do int */
 int y;      /* usa apenas os 8 bits mais à direita do int */
}
```

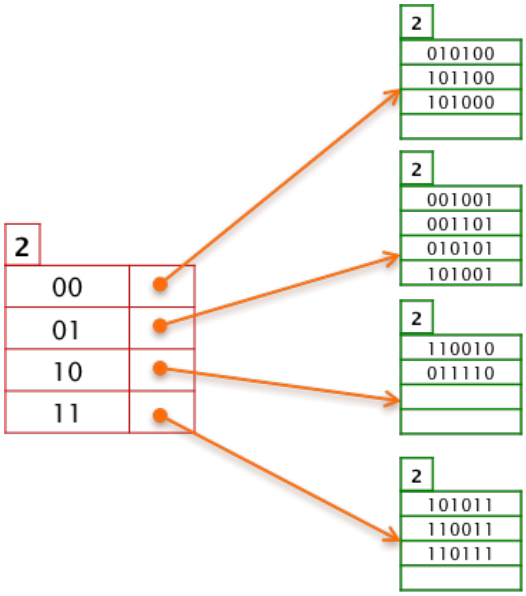
Resposta:

```
Ponto* decodeMorton(int m)
{
  int i, j, p, M;
  Ponto* p = (Ponto*)malloc(sizeof(Ponto));
  if (p == NULL) {
       printf("Memória insuficiente!\n");
       exit(1);
  }
  p->x = 0;
  p->y = 0;
  M = m;
  k = 0;                    /* posição onde o novo bit deve ser inserido   */
                            /* (da direita para a esquerda)                */
  for (i = 0; i < 8; i++)   /* extrai 8 bits para p->x e 8 bits para p->y  */
  {                         /* (os bits estão intercalados em M)           */
     j = M % 2;             /* retira o último bit do valor corrente de M  */
     M = M >> 1;
     j = j << k;            /* "shift" do bit para a posição correta       */
     p->x = p->x + j;       /* acres. bit retirado na pos. correta de p->x */
     j = M % 2;             /* retira o último bit do valor corrente de M  */
     M = M >> 1;
     j = j << k;            /* "shift" do bit para a posição correta       */
     p->y = p->y + j;       /* acrescenta o bit retirado à direita de p->y */
     k = k + 1;             /* incrementa a posição onde o próximo bit     */
                            /* deve ser inserido                           */
  }
  return p;
}
```

**Questão 4 (2.0 pontos).**

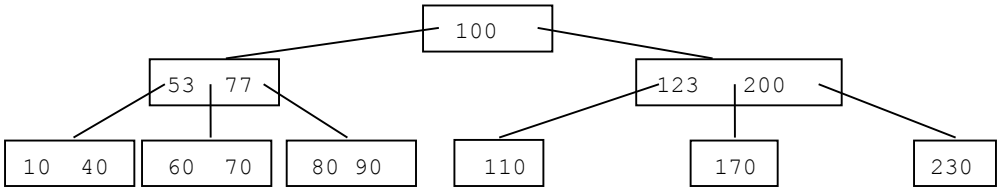
a) **(1.0 ponto)** Considere a estrutura de hash expansível abaixo. Realize, de duas maneiras diferentes, a remoção da chave *K* tal que a função de hash mapeia *K* no string de bits '101000'. Indique quais transformações são realizadas. Explique em que cenários cada alternativa seria a melhor.

**Respostas que não indicarem as transformações não serão consideradas.**



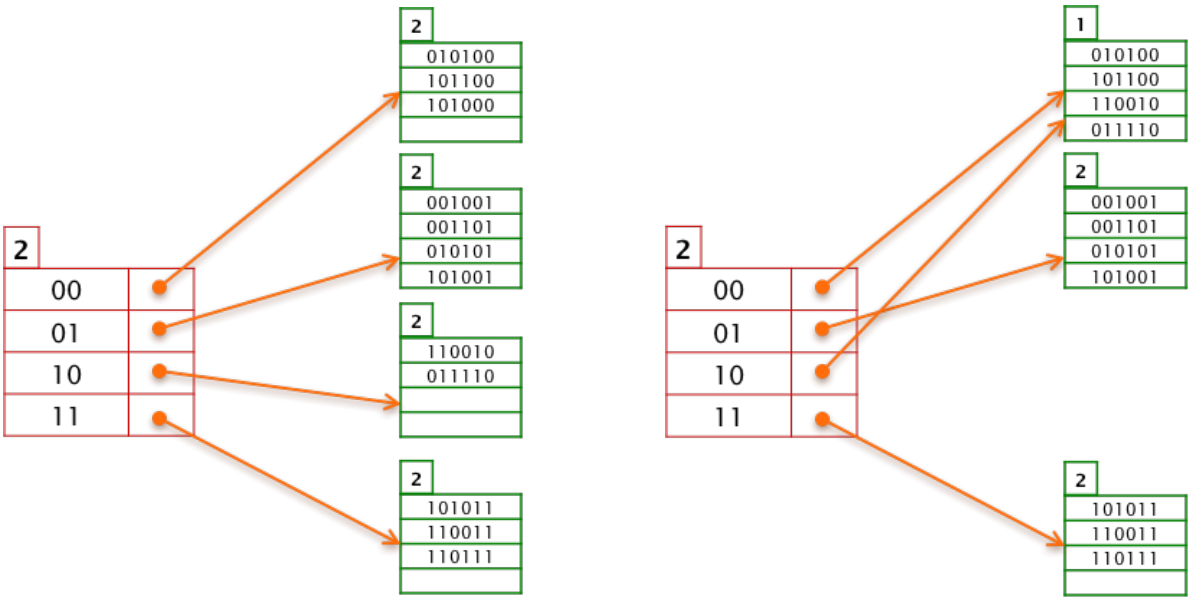

b) **(1.0 ponto)** Considere a árvore 2-3 abaixo. Realize a remoção da chave 100 de duas maneiras diferentes. Indique quais transformações são realizadas.

**Respostas que não indicarem as transformações não serão consideradas.**



**Resposta:**

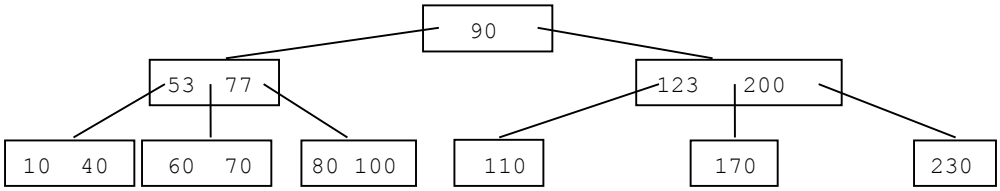
(a) A chave é removida do primeiro bucket e os os buckets 1 e 3 são combinados. Alternativamente, poderíamos simplesmente remover a chave do primeiro bucket, mantendo os buckets 1 e 3 parcialmente vazios.



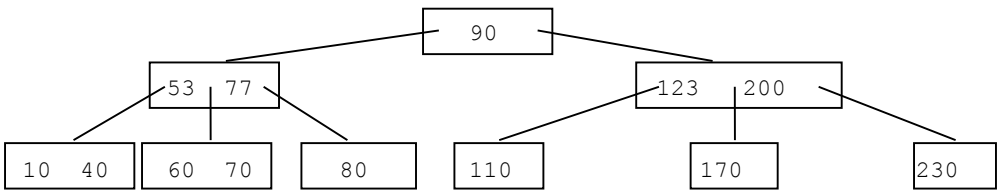

A primeira alternativa seria preferível se o número esperado de remoções for maior do que o de inserções. A segunda alternativa seria melhor se o número esperado de inserções e remoções for equilibrado.

b) Remoção da chave 100 – primeira forma.

Troque a chave 100 com a sua antecessora, a chave 90.

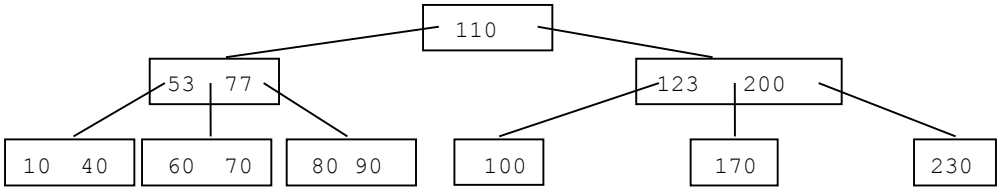



Remova a chave 100 da folha. A remoção da chave 100 termina pois a árvore resultante é uma árvore 2-3 correta.

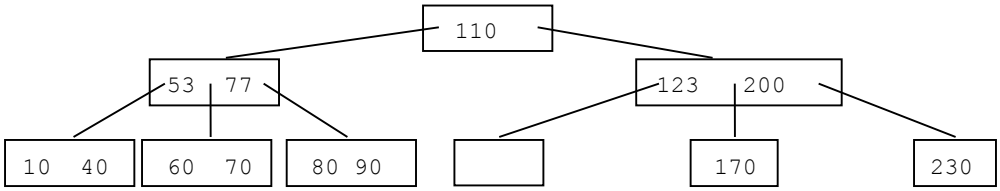


b) Remoção da chave 100 – segunda forma.

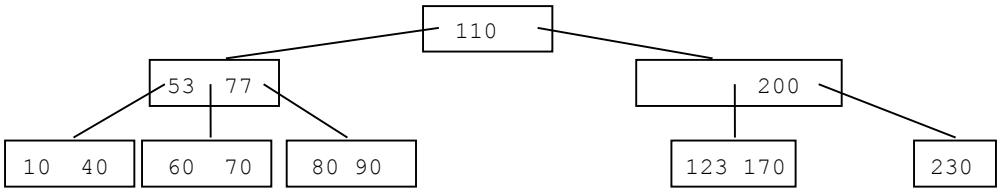
Troque a chave 100 com a sua sucessora, a chave 110.




Remova a chave 100 da folha.

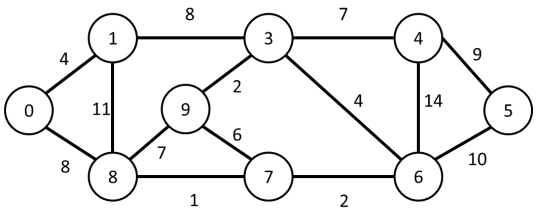



Redistribua e combine as folhas mais à direita (Caso 6 das notas de aula).



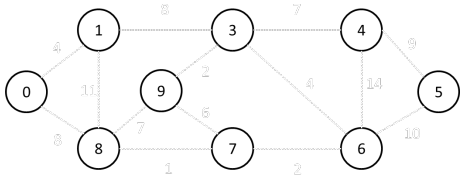
**Questão 5 (3.0 pontos).** Mostre os passos do algoritmo de Kruskal para calcular uma árvore geradora mínima do grafo mostrado na figura abaixo. Considere que o algoritmo adota uma partição dinâmica dos nós do grafo, representada por uma floresta, com a implementação de UNION por altura e FIND com compressão de caminhos.

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

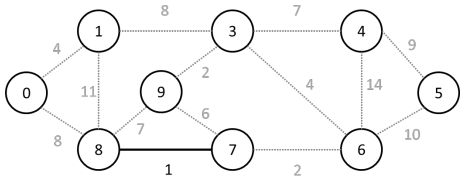



Resposta

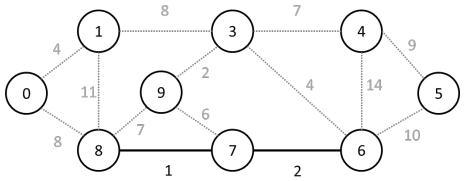
c) Crie uma partição dinâmica onde cada vértice é um conjunto isolado.
$P_d$={{0},{1},...{7},{8},{9}}.

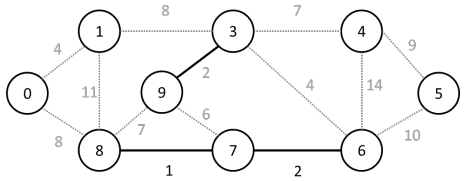


d) Adicione uma aresta de menor peso dentre as disponíveis. Ao inserir uma aresta faça uma união dos dois conjuntos correspondentes aos vértices dela. $P_d$={{0},{1},...**{7,8}**,{9}}.

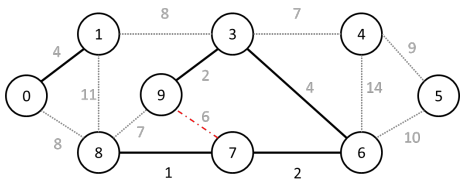


e) Continue adicionando uma aresta de menor peso dentre as disponíveis desde que sua inserção não forme ciclos na árvore que está sendo gerada. Ou seja, desde que seus vértices não estejam numa mesma partição. $P_d$={{0},{1},...**{6,7,8},{3,9}**}.

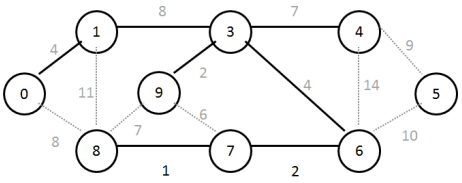




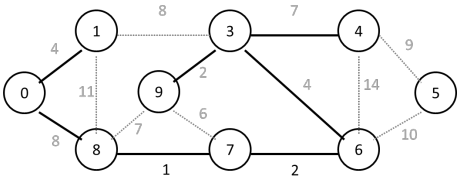
f) Note que a aresta (7,9) apesar de ser a de menor peso dentre as disponíveis não pode ser inserida, pois forma um ciclo, detectado pelo fato de 3 e 9 estarem na mesma partição.
$P_d$={**{0,1},...{6,7,8,3,9}**}.)



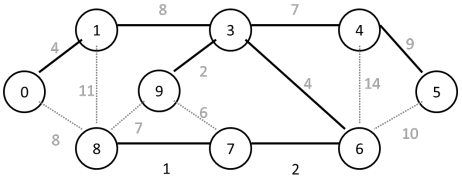
g) O algoritmo segue e encontra uma situação onde duas arestas de menor peso dentre as disponíveis tem o mesmo peso e são excludentes e daí bifurca para duas soluções equivalentes.




e




h) Dai resultam as duas soluções de custo total 37 mostradas abaixo.




e