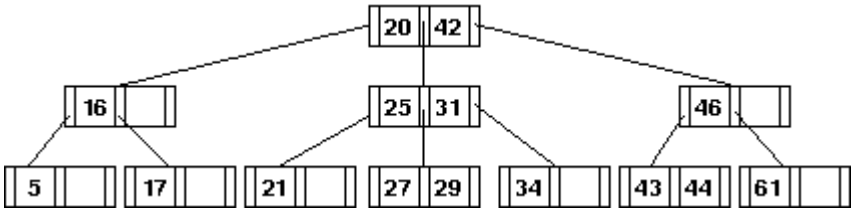
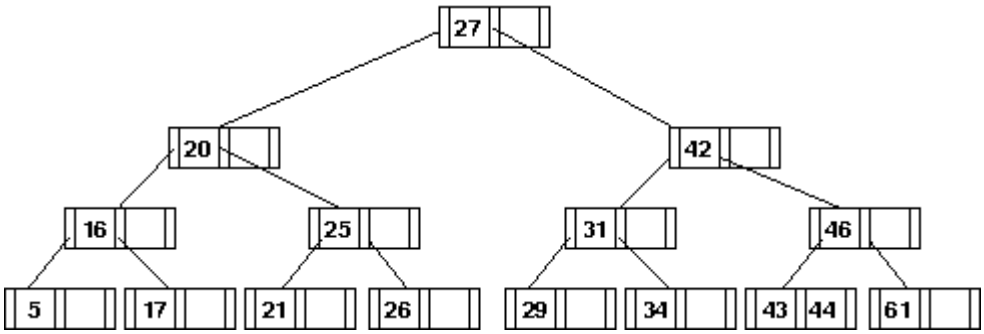
Questão 1

Indique no espaço abaixo uma chave cuja inserção provocará o aumento da altura total da árvore. Desenhe a árvore resultante após a inserção.



Chave a ser inserida: 26 (28 ou 30 também são válidas)



Questão 2

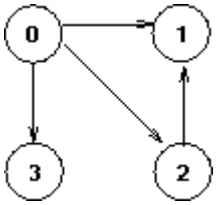
Considere a seguinte declaração para os nós de uma árvore 2-3, com as restrições apresentadas em aula. Escreva uma função que retorna 1 caso a inserção de uma dada chave na árvore implique no aumento da altura da árvore e 0 caso contrário. Lembre-se que chaves repetidas não são inseridas (e, portanto, não aumentam a altura da árvore).

```
typedef struct no23{
  int nch;                       /* Número de chaves (1 ou 2) no nó         */
  struct no23 *esq, *cen, *dir;  /* Apontadores esquerdo, central e direito */
  int chesq, chdir;              /* Chaves esquerda e direita               */
} No23;

  /* Retorna 1 se a altura de arv aumenta após a inserção de chave */
  /* Retorna 0 caso contrário. Chaves repetidas não são inseridas. */

int cresce(No23* arv, int chave) {
  if (arv == NULL)          /* Árvore vazia, qualquer inserção aumentará a
altura */
    return 1;
  if (arv->chesq == chave ||
      arv->nch == 2 && arv->chdir == chave)
        return 0;           /* Chaves repetidas não são inseridas. */
  if (arv->esq == NULL)     /* Inserção em uma folha */
    return arv->nch == 2;       /* a altura aumenta se não há espaço */
  int c;
  if (chave < arv->chesq)   /* Chamada recursiva na sub-árvore apropriada */
   c = cresce(arv->esq, chave);
  else
    if (arv->nch == 1 || chave < arv->chdir)
      c = cresce(arv->cen, chave);
    else
      c = cresce(arv->dir, chave);
  return c && arv->ch == 2; /* A altura aumenta se houve aumento durante
```

```
                                a chamada recursiva e não há espaço no nó
corrente */
}
```

Questão 3

Considere um grafo dirigido representado utilizando-se uma matriz de adjacências, como no exemplo abaixo. (a) Implemente a função `dist_max_duas_arestas()` cujo objetivo é retornar 1 caso exista um caminho do vértice `u` ao vértice `v` composto por uma ou duas arestas. A função recebe como parâmetro a matriz `M` sendo que `M[i][j] == 1` se existe uma aresta que liga `i` a `j` ou 0 em caso contrário. (b) Implemente também a função `dist_max_tres_arestas` que {\bf utiliza a função {\tt dist_max_duas_arestas e responde se existe um caminho de no máximo três arestas entre os vértices `u` e `v`.




```
0  1  1  1
0  0  0  0
0  1  0  0
0  0  0  0
```

```
int dist_max_duas_arestas(int u, int v, int n, int M[][]) {
  if (M[u][v])
    return 1;
  int i;
  for (i = 0; i < n; i++)
    if (M[u][i] && M[i][v])
      return 1;
  return 0;
}

int dist_max_tres_arestas(int u, int v, int n, int M[][]) {
  if (dist_max_duas_arestas(u,v,n,M))
    return 1;
  int i;
  for (i = 0; i < n; i++)
    if (M[u][i] && dist_max_duas_arestas(u,v,n,M))
      return 1;
  return 0;
}
```

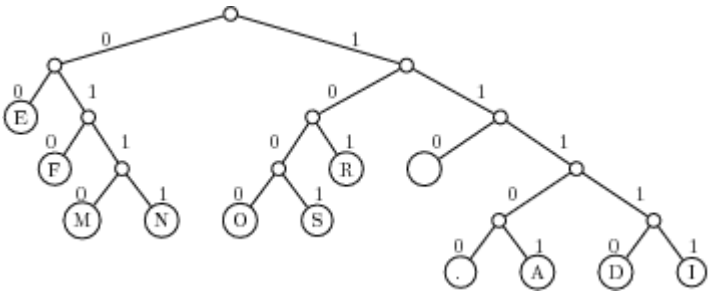
Questão 4

Versão A

O algoritmo de Huffman sobre aplicado sobre a frase "FREE AS IN FREEDOM." gerou a seguinte codificação. Desenhe a árvore de Huffman correspondente.

```
' ': 110
'.': 11100
'A': 11101
```

```
'D': 11110
'E': 00
'F': 010
'I': 11111
'M': 0110
'N': 0111
'O': 1000
'R': 101
'S': 1001
```



Versão B

O algoritmo de Huffman sobre aplicado sobre a frase "FREE AS IN FREEDOM" gerou a seguinte codificação. Desenhe a árvore de Huffman correspondente.

```
' ': 110
'A': 11110
'D': 11111
'E': 00
'F': 010
'I': 0110
'M': 0111
'N': 1000
'O': 1001
'R': 101
'S': 1110
```