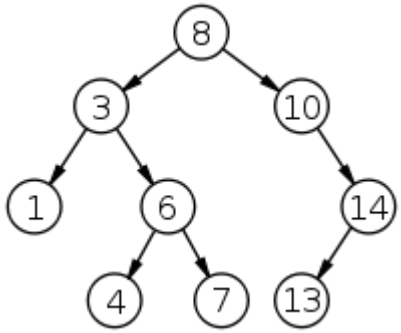
Prova Final de Estrutura de Dados II

(1.0) Percorra em pré-ordem, em-ordem e pós-ordem a seguinte árvore binária de busca

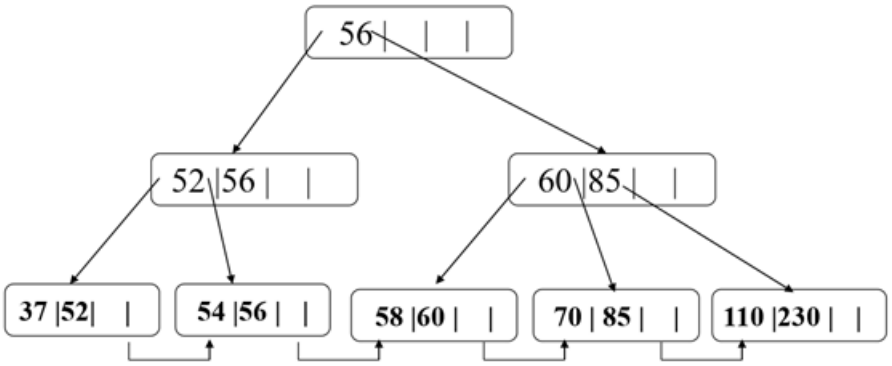


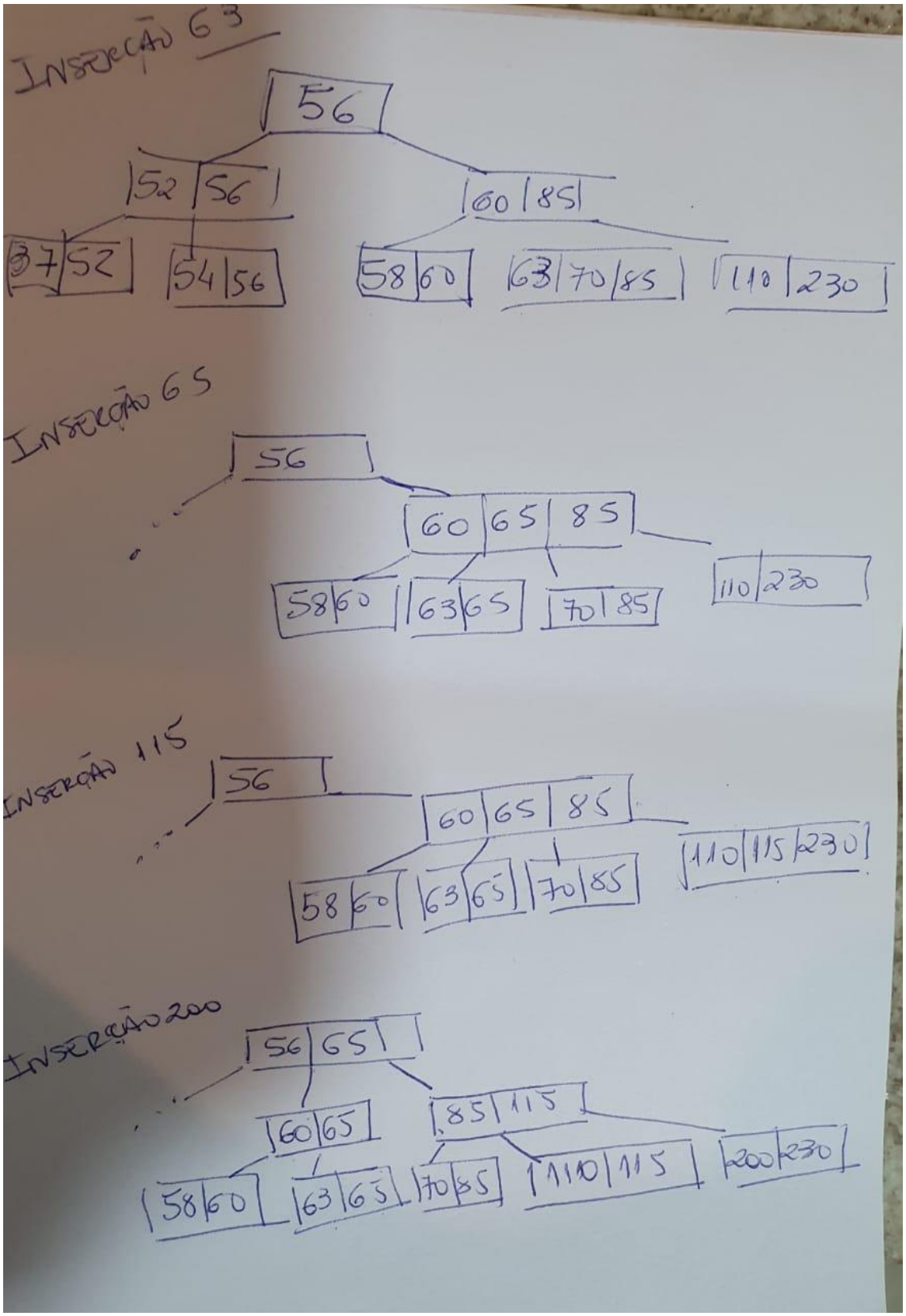
Pré-ordem: 8, 3, 1, 6, 4, 7, 10, 14, 13

Em-ordem: 1, 3, 4, 6, 7, 8, 10, 13, 14

Pós-ordem: 1, 4, 7, 6, 3, 13, 14, 10, 8

(2.0) Construa uma árvore B+, a partir da árvore abaixo, inserindo os valores 63, 65, 115, 200, conforme o algoritmo visto na aula teórica. Apresente cada passo.




INSERÇÃO 63
56
52 56
60 85
37 52
54 56
58 60
63 70 85
110 230
INSERÇÃO 65
56
60 65 85
58 60
63 65
70 85
110 230
INSERÇÃO 115
56
60 65 85
58 60
63 65
70 85
110 115 230
INSERÇÃO 200
56 65
60 65
85 115
58 60
63 65
70 85
110 115
200 230

(3.0) Utilizando o conceito de probing linear, com função hash calculada por: n%tam do vetor, faça um método que busque um elemento no vetor (circular).

```
int Hash::busca (int elemento){
   int x;
   int i = 0;
   int indice = elemento%tamVetor;
   for (i = 1; i<=tamVetor; i++){
          if (vetor[índice]==NULL)
                  return -1;
          if (vetor[índice]== elemento)
          Return índice;
       if(índice+1 == tamVetor)
         Índice = 0;
      else
         indice ++;
  }
  return -1;
}
```

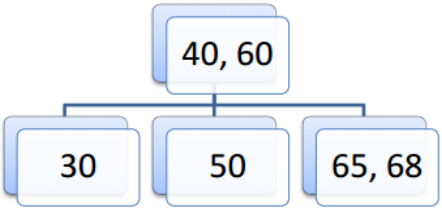
(2.0) Dado o código sobre busca em árvore B e dada a árvore B abaixo, responda:

a) Qual o valor devolvido para pos na segunda vez que a função busca_binaria é chamada para se encontrar o valor 63?
Resposta: Retorna 0

b) Por que a função busca_binaria retorna i caso não tenha encontrado o elemento, ao invés de -1?
Resposta: Porque é necessário que se busque nos nós filhos também, se o elemento não foi encontrado naquele nó. Neste caso, i corresponde ao índice do vetor de filhos que se deve buscar o elemento.

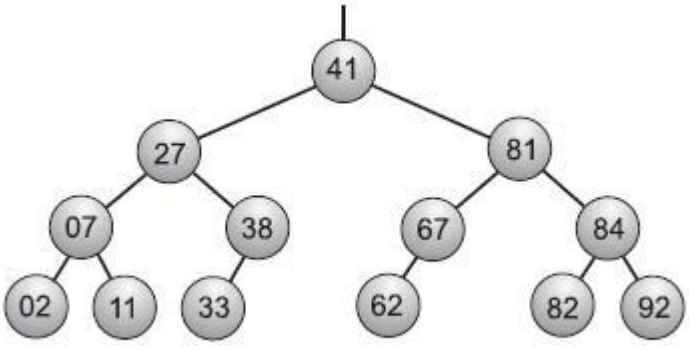
```
bool busca(arvoreB *raiz, int info)
{
 arvoreB *no;
 int pos;  //posição retornada pela busca binária.

 no =      raiz;
 while     (no!= NULL)
 {
   pos = busca_binaria(no, info);
   if (pos < no->num_chaves &&  no->chaves[pos] == info)
            return(true);
   else
            no =  no->filhos[pos];
 }
 return(false);
}
```

```
int busca_binaria(arvoreB *no, int info)
{
   int meio, i, f;
   i = 0;
   f = no->num_chaves-1;

   while (i<=f)
  {
      meio = (i + f)/2;
      if (no->chaves[meio]==info)
           return(meio);
       else if (no->chave[meio] > info)
            f = meio - 1;
       else
           i = meio + 1;
   }
   return (i);
}
```



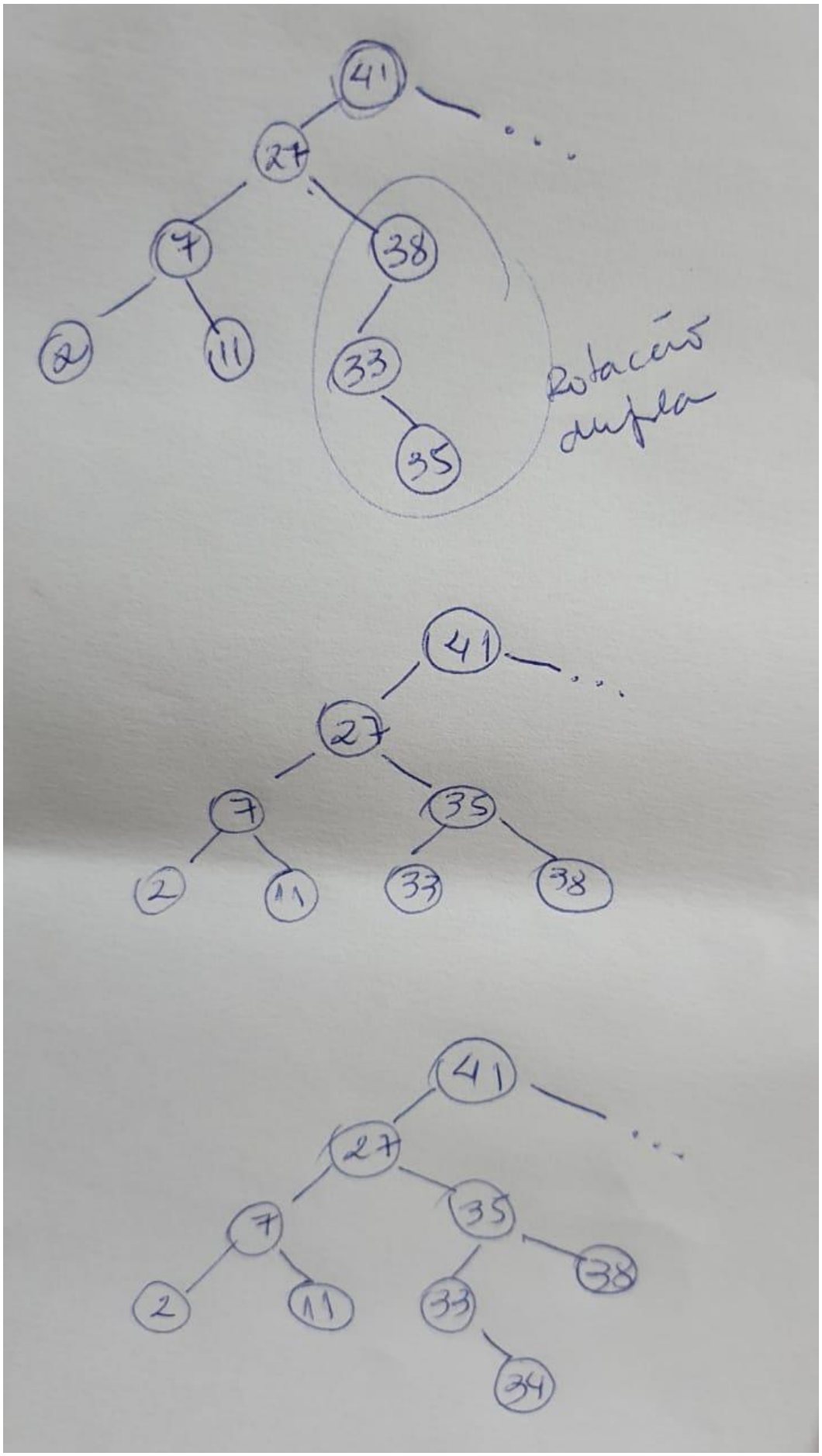

```
const t = 2;
typedef struct no_arvoreB arvoreB;

struct no_arvoreB {
  int num_chaves;
  char chaves[2*t-1];
  arvoreB *filhos[2*t];
  bool folha;
};
```

(2) Dada a árvore AVL abaixo:



Acrescente os valores 35 e 34 (faça rotações, caso necessário)

Dada a árvore AVL abaixo, acrescente os valores 35 e 34 (faça rotações, caso necessário)


41
27
7
38
2
11
33
35
Rotación doble
41
27
7
35
2
11
33
38
41
27
7
35
2
11
33
38
34