DCC013 - Estrutura de Dados / DCC107 - Laboratório de Programação II

Árvore binária

Você pode utilizar qualquer ambiente de programação para desenvolver sua atividade. Ao final, **copie e cole o seu código-fonte com a resposta aqui mesmo neste documento**, dentro dos espaços indicados para isso e **preservando a identação do código**. **Depois que terminar sua avaliação, não se esqueça de entregar sua atividade!** Fique atento ao relógio, pois as atividades entregues com atraso não serão aceitas.

Para resolver esta atividade, **clique aqui para baixar o projeto da aula de laboratório de programação 2, que contém a implementação do TAD ArvBin para árvore binária de números inteiros.** Na sua solução para a questão abaixo, você pode utilizar/chamar qualquer uma das operações que estejam disponíveis no projeto (exatamente do jeito que ele se encontra no site da disciplina). Outras operações que você venha a criar para resolver o seu exercício, inclusive as operações auxiliares, devem ser copiadas para sua resposta neste documento.

Implementar a operação `int* ArvBin::criaVetNegativos(int k, int *n);` que, dado um nível `k`, aloca (com um número apropriado, mas não necessariamente exato, de elementos), preenche e retorna um vetor com todos os **valores negativos existentes no nível `k`** de uma árvore binária. Se a árvore for vazia, retornar `NULL`. A operação deve armazenar o tamanho do vetor que foi alocado no ponteiro `n`. **Percorrer a árvore uma única vez, fazendo um percurso em pré-ordem, e não visitar nós de níveis desnecessários**. Completar as posições não utilizadas do vetor com o valor `-1`.

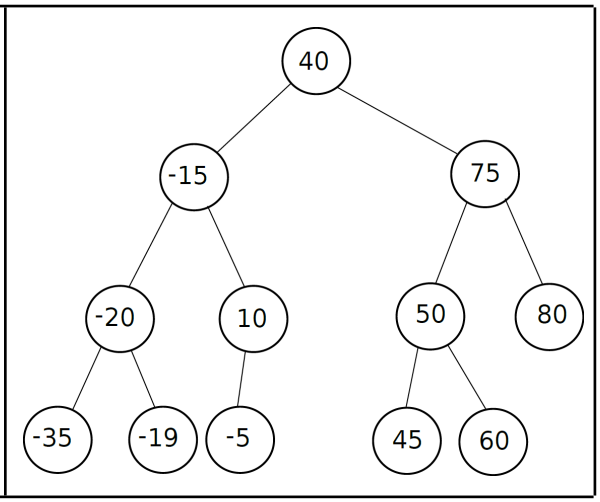
A figura a seguir exibe uma AB e exemplos de valores a serem inseridos no vetor:

`k = 0:`
- A árvore não tem valores negativos neste nível.

`k = 1:`
- `-15.`

`k = 2:`
- `-20.`

`k = 3:`
- `-35, -19, -5.`




```
int *ArvBin::criaVetNegativos(int k, int *n)
{
   if (raiz == NULL)
```

```
    {
        return NULL;
    }

    int val = 0;
    *n = pow(2, k);
    int *vet = new int[*n];

    for (int i = 0; i < *n; i++)
    {
        vet[i] = -1;
    }

    auxCriaVetNegativos(raiz, k, n, vet, &val);

    if (val == 0)
    {
          cout << "A árvore não tem valores negativos neste nível." <<
endl;
    }

    return n;
}
```

```
void ArvBin::auxCriaVetNegativos(NoArv *p, int k, int *n, int *vet, int
*elem)
{

    if (k == 0 && p != NULL)
    {
        if (p->getInfo() < 0)
        {
            vet[*elem] = p->getInfo();
            *elem = *elem + 1;
        }
        else
            return;
    }
    else if (p != NULL)
    {
        auxCriaVetNegativos(p->getEsq(), k - 1, n, vet, elem);
        auxCriaVetNegativos(p->getDir(), k - 1, n, vet, elem);
```

```
    }
}
```

O trecho de código a seguir cria a árvore (`arv`) do exemplo acima:

```
ArvBin arv, vazia, a1, a2, a3;
a1.cria(-35, &vazia, &vazia);
a2.cria(-19, &vazia, &vazia);
a3.cria(-20, &a1, &a2);
a1.cria(-5, &vazia, &vazia);
a2.cria(10, &a1, &vazia);
a1.cria(-15, &a3, &a2);
a2.cria(45, &vazia, &vazia);
a3.cria(60, &vazia, &vazia);
a2.cria(50, &a2, &a3);
a3.cria(80, &vazia, &vazia);
a2.cria(75, &a2, &a3);
arv.cria(40, &a1, &a2);
a1.anulaRaiz();
a2.anulaRaiz();
a3.anulaRaiz();
```