Aula 2 de Estruturas de Dados 2 com Bruno Ribas

Aula 2 - Tabela de Símbolo e Árvore Binária de Busca(bst)

Árvore Binária de Busca (BST - Binary Search Tree) em C é uma estrutura de dados usada para armazenar e organizar itens de forma que possam ser encontrados rapidamente. Vamos por partes:

1. Estrutura básica

```
typedef struct STNode *link;

struct STNode {
   Item item;
   link l, r;
   int N;
};

link h, z;
```

O que faz: Define a estrutura de um nó da árvore e algumas variáveis globais.
struct STNode: Cada nó da árvore tem:

- Item item: O dado que está sendo armazenado (pode ser um número, string, etc., dependendo de como Item for definido).
- link l, r: Ponteiros para os filhos esquerdo (l) e direito (r) do nó.
- int N: Um contador que guarda o número de nós na subárvore que tem esse nó como raiz.

link h, z:

- h é a raiz da árvore (o nó inicial).
- z é um nó sentinela especial que representa o "final" da árvore (como um marcador de folhas nulas).

2. Função NEW

```
link NEW(Item item, link l, link r, int N) {
   link x = malloc(sizeof(*x));
   x->item = item;
   x->l = l;
   x->r = r;
   x->N = N;
```

```
    return x;
}
```

O que faz: Cria um novo nó na árvore.
Como funciona:

- Aloca memória para um novo nó usando malloc.
- Define o item (dado), os ponteiros para os filhos l e r, e o contador N.
- Retorna o ponteiro para esse novo nó.

Exemplo de uso: Quando você quer adicionar um novo elemento, essa função cria o nó que vai armazená-lo.

3. Função STinit

```
void STinit() {
    h = (z = NEW(NULLitem, NULL, NULL, 0));
}
```

O que faz: Inicializa a árvore.
Como funciona:

- Cria um nó sentinela z com:
    - NULLitem (um valor que representa "nada", como um marcador de vazio).
    - Filhos esquerdo e direito apontando para NULL.
    - Contador N = 0 (não há nós na subárvore ainda).
- Define h (a raiz) como sendo igual a z, ou seja, a árvore começa "vazia" (apenas com o sentinela).

Por que usar um sentinela?: Simplifica o código ao evitar verificações constantes de ponteiros NULL.

4. Função insertR (inserção recursiva)

```
link insertR(link r, Item item) {
    if (r == z) return NEW(item, z, z, 1);

    Key k = key(item);
    Key t = key(r->item);

    if (less(k, t))
        r->l = insertR(r->l, item);
    else
        r->r = insertR(r->r, item);
```

```
    (r->N)++;
    return r;
}
```

O que faz: Insere um novo item na árvore de forma recursiva.
Como funciona:

- Se o nó atual r é o sentinela z, cria um novo nó com o item e filhos apontando para z.
- Extrai a chave (k) do item a ser inserido e a chave (t) do nó atual usando a função key().
- Compara as chaves:
    - Se k < t, vai para o filho esquerdo (r->l) e chama insertR novamente.
    - Se k >= t, vai para o filho direito (r->r) e chama insertR novamente.
- Após inserir, incrementa o contador N do nó atual.
- Retorna o nó atualizado.

Propriedade da BST: Itens menores vão para a esquerda, maiores ou iguais vão para a direita.

5. Função STinsert

```
void STinsert(Item item) {
    h = insertR(h, item);
}
```

O que faz: Função principal para inserir um item na árvore.
Como funciona:

- Chama insertR começando da raiz h.
- Atualiza h com o resultado (caso a raiz mude, como na primeira inserção).

Exemplo: Se você chamar STinsert(5), o número 5 será inserido na posição correta na árvore.

6. Função searchR (busca recursiva)

```
Item searchR(link r, Key k) {
    if (r == z) return NULLitem;

    Key t = key(r->item);
    if (eq(k, t))
        return r->item;
```

```
    if (less(k, t))
        return searchR(r->l, k);
    return searchR(r->r, k);
}
```

O que faz: Busca um item na árvore pela sua chave.
Como funciona:

- Se chega no sentinela z, retorna NULLitem (não encontrou).
- Pega a chave t do nó atual.
- Compara com a chave procurada k:
    - Se k == t (usando eq()), retorna o item do nó atual.
    - Se k < t, busca no filho esquerdo.
    - Se k > t, busca no filho direito.

Exemplo: Se a árvore tem [3, 1, 5] e você busca a chave 1, ele vai para a esquerda de 3 e encontra 1.

7. Função STsearch

```
Item STsearch(Key k) {
    return searchR(h, k);
}
```

- **O que faz**: Função principal para buscar um item pela chave.
- **Como funciona**:
    - Chama searchR começando da raiz h.
    - Retorna o item encontrado ou NULLitem se não achar.

Resumo do que a árvore faz

- **Armazenamento**: Organiza itens de forma hierárquica baseada em suas chaves.
- **Inserção**: Adiciona novos itens mantendo a ordem (menores à esquerda, maiores à direita).
- **Busca**: Encontra itens rapidamente seguindo a ordem das chaves.
- **Contagem**: Mantém o número de nós em cada subárvore (útil para estatísticas ou balanceamento).

Exemplo prático

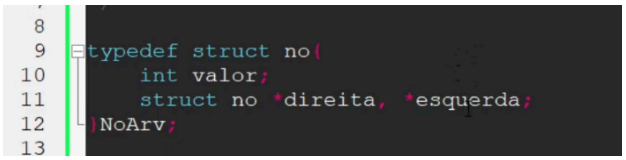
Imagine que Item é um inteiro e Key é o próprio número:

1. STinit(): Árvore vazia.

2. STinsert(3): Árvore fica [3].
3. STinsert(1): Árvore fica [3, 1 à esquerda].
4. STinsert(5): Árvore fica [3, 1 à esquerda, 5 à direita].
5. STsearch(1): Retorna 1 (encontrado).
6. STsearch(2): Retorna NULLitem (não encontrado).

Essa estrutura é eficiente para operações de busca, inserção e remoção (embora remoção não esteja implementada aqui), com complexidade média de O(log n) em árvores balanceadas.

Extras:



```
8
9   typedef struct no{
10      int valor;
11      struct no *direita, *esquerda;
12  }NoArv;
13
```

Buscar um elemento:

Curso de Programação C | Como buscar um elemento em uma ÁRVORE BINÁRIA? V…

```
    aux->direita = NULL;
    *raiz = aux;
}

NoArv* buscar_versao_1(NoArv *raiz, int num){
    if(raiz){
        if(num == raiz->valor)
            return raiz;
        else if(num < raiz->valor)
            buscar_versao_1(raiz->esquerda, num);
        else
            buscar_versao_1(raiz->direita, num);
    }
    return NULL;
}
```