Árvores B

Anderson Fernandes
Daniel Carvalho
Gabriel Fabrício
Myron David

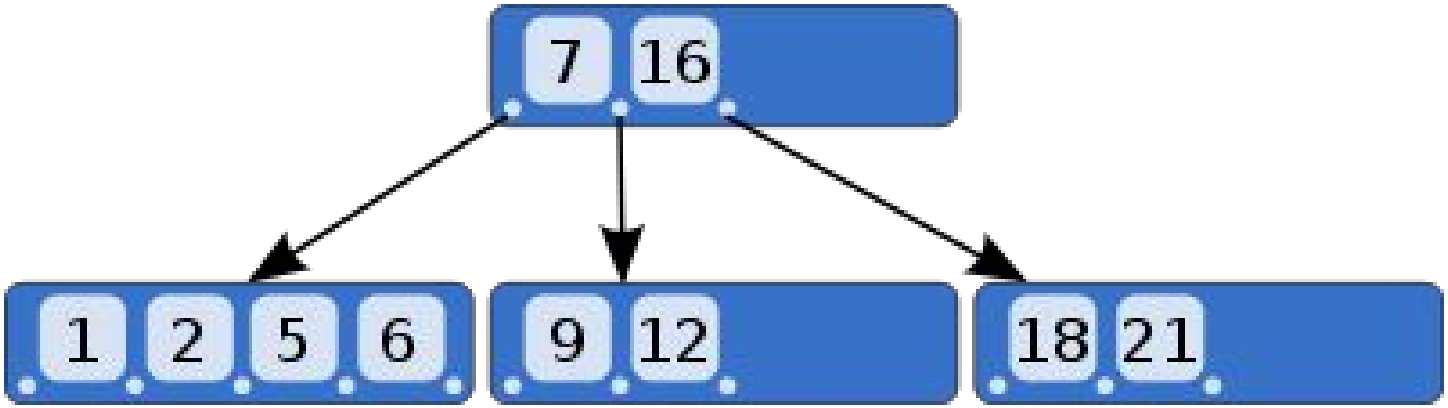
Árvores B - Motivação

- **Armazenamento de um grande volume de informação.**
  - **Memória primária vs memória secundária**

- **Balanceamento da Árvore**

Árvores B

- Desenvolvida por Rudolf Bayer e Edward Meyers;
- Baseada na árvore de busca binária, onde cada nó da árvore pode armazenar mais de uma chave de busca;



Fonte: https://goo.gl/6MHupp

Definições da Estrutura

- **Principais definições:**
  - Páginas
    - Páginas Filhas
    - Páginas Internas
  - Chaves
  - Ordem

Definições da Estrutura

- Assumindo a ordem da árvore como “R”
- Regras (De acordo com a definição de Donald Knuth):
  - Uma página interna com X páginas filhas contém X-1 chaves.
  - O número máximo de páginas filhas de uma página é R.
  - O número mínimo de páginas filhas das páginas internas é R/2.
  - As páginas folha, tem no mínimo [(R/2)-1] chaves, e no máximo (R-1) chaves.
  - A raiz tem pelo menos duas páginas filhas, a menos que ela seja uma folha.
  - Todas as páginas folha possuem a mesma profundidade.
  - As chaves de uma página sempre são ordenadas de forma crescente

ADT

```
typedef struct bt_node BTree;

/* Cria uma árvore B vazia */
BTree *btCreate();

/* Da free na árvore */
void btFree(BTree *t);

/* Retorna 1 se a chave estiver na árvore */
int btSearch(BTree *t, int key);

/* Insere um novo elemento na árvore */
void btInsert(BTree *t, int key);

/* Remove um elemento da árvore */
void btRemove(BTree *t, int key);
```

ADT

```
#define MAX_KEYS (1024) /* Indica a ordem da árvore */

struct btNode {
    int isLeaf;     /* Indica se a página é folha */
    int numKeys;    /* Indica o número de chaves atual da página */
    int keys[MAX_KEYS];
    struct btNode *kids[MAX_KEYS+1];
   //kids[i] contém páginas com chaves menores que keys[i]
};
```

Busca na Árvore B

```
1.  int btSearch(BTree *root, int key){
2.    BTree *aux;
3.    int pos; //posição retornada pela busca binária.
4.
5.    aux = root;
6.    while(aux != NULL){
7.       pos = binary_search(aux, key);
8.       if(pos < aux->numKeys && aux->keys[pos] == key){
9.         return 1;
10.      }
11.      else aux = aux->kids[pos];
12.   }
13.   return 0;
14. }
```

Busca na Árvore B

```
1.  int binary_search(BTree *page, int key) {
2.    int mid, start = 0, end;
3.    end = page->numKeys-1;
4.
5.    while(start <= end){
6.            mid = (start + end)/2;
7.            if(page->keys[mid] == key){
8.               return mid;
9.               //Encontrou. Retorna a posição em que a chave está.
10.           }
11.           else if(page->keys[mid] > key) end = mid - 1;
12.
13.           else start = mid + 1;
14.   }
15.   return start;
16.   //Não encontrou. Retorna a posição do ponteiro para o filho.
17. }
```

De volta à motivação…

- Como vimos, a árvore B faz com que seja possível que o acesso a memória secundária seja menor, possibilitando que a memória principal carregue apenas os índices necessários, sem que aconteça muitos desperdícios.

Aplicações

- Sistema de arquivos NTFS (windows)
- Sistema de arquivos HFS (mac)
- Sistemas de arquivos ReiserFS, XFS, EXT#FS, JFS (linux)
- Banco de dados Oracle, SQLserver, Postgres, etc
- Kernel do Linux <https://github.com/torvalds/linux/blob/master/lib/btree.c>

Referências

- https://en.wikipedia.org/wiki/B-tree
- http://www.ime.usp.br/~pf/estruturas-de-dados/aulas/B-trees.html
- https://www.cs.usfca.edu/~galles/visualization/BTree.html
- http://www.cs.yale.edu/homes/aspnes/pinewiki/BTrees.html
- https://web.archive.org/web/20160327132931/http://www.lcad.icmc.usp.br/~nonato/ED/B_arvore/btreebusca.htm