INSTITUTO FEDERAL
Santa Catarina

Banco de dados em C usando Btree

Leonardo Benitez

INSTITUTO FEDERAL
Santa Catarina

Motivação

- Como armazenar dados de forma segura e persistente?
- Como buscar, inserir e deletar esses dados em O(log n)?
- Como otimizar esse sistema para o armazenamento em disco?

INSTITUTO FEDERAL
Santa Catarina

Banco de dados

I see data, all the time

INSTITUTO FEDERAL
Santa Catarina

Banco de dados

- Integridade, consistência e disponibilidade
- Dados organizados em tabelas (para banco de dados relacionais)
- Interação por meio da linguagem SQL

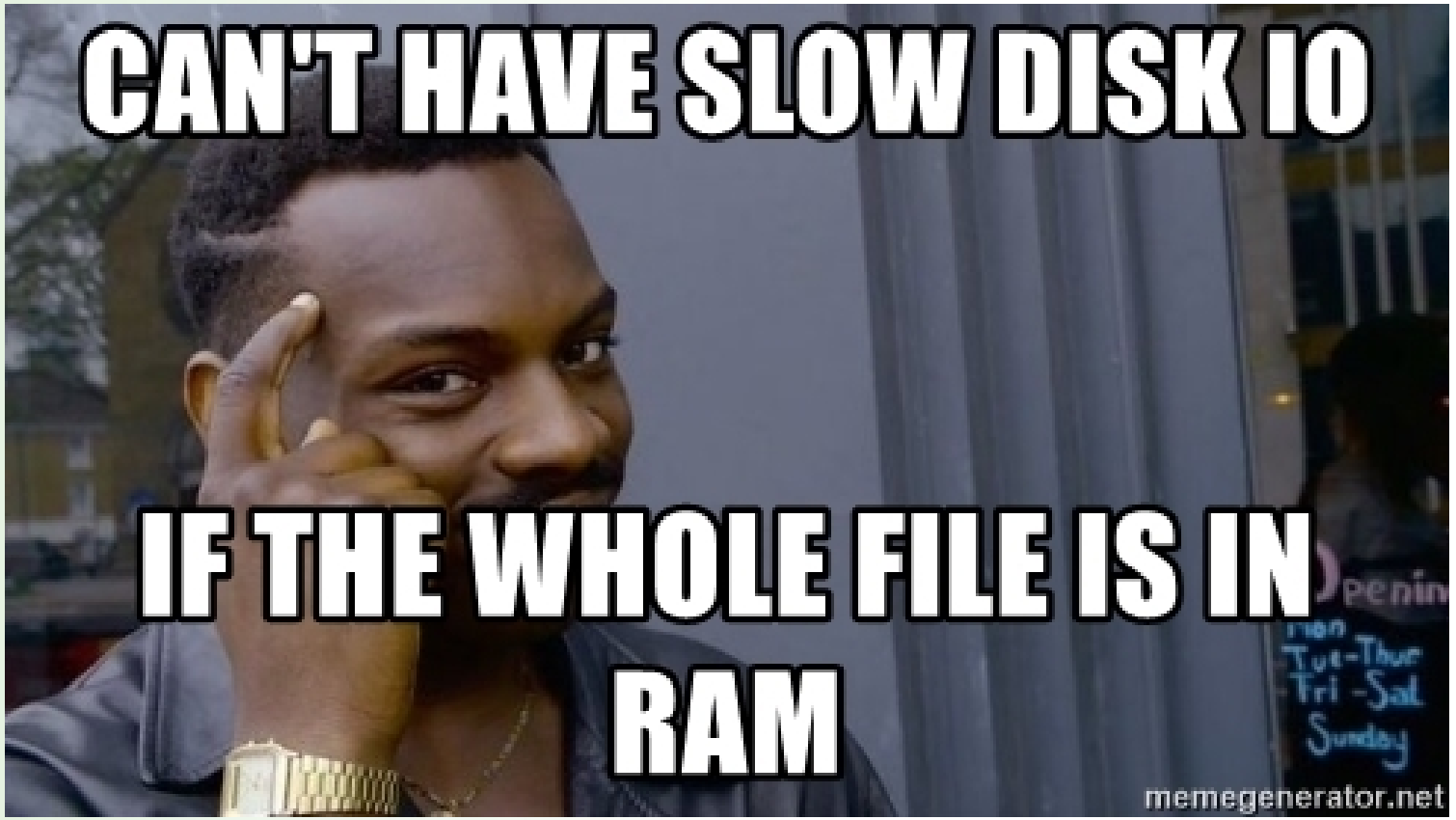
INSTITUTO FEDERAL
Santa Catarina

Santo Graal: O(log n)

- Binary search tree
- N-ary tree
- Red black tree
- Btree

INSTITUTO FEDERAL
Santa Catarina

E o disco?



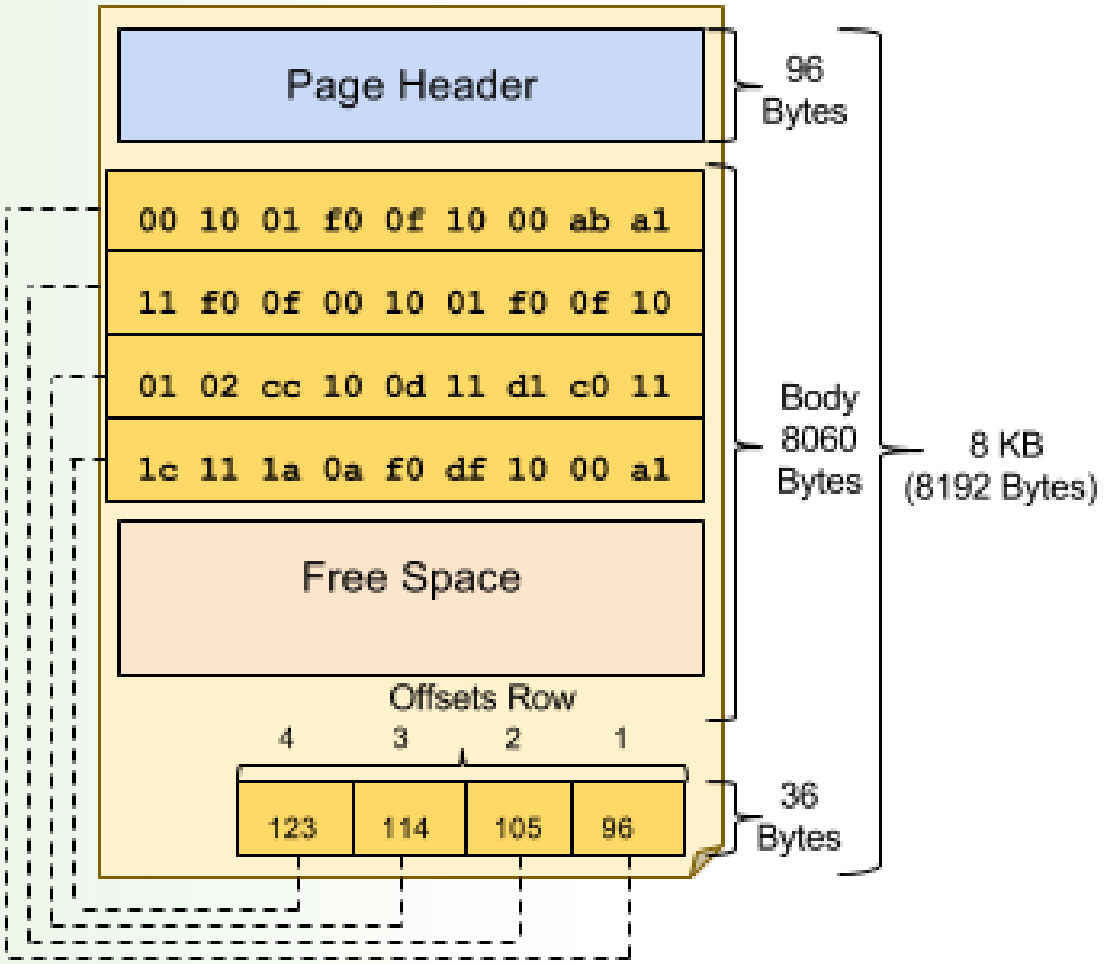
INSTITUTO FEDERAL
Santa Catarina

E o disco?

- Diminuir o numero de acessos ao disco (árvore com baixa altura)
- Otimizar para o bloco de memória
- Page abstraction

INSTITUTO FEDERAL
Santa Catarina

E o disco?



INSTITUTO FEDERAL
Santa Catarina

Btree

- Auto balanceável
- Muitas Keys por nó (nº variável)
- Minimun degree (t): no mínimo t filhos, no máximo 2t filhos

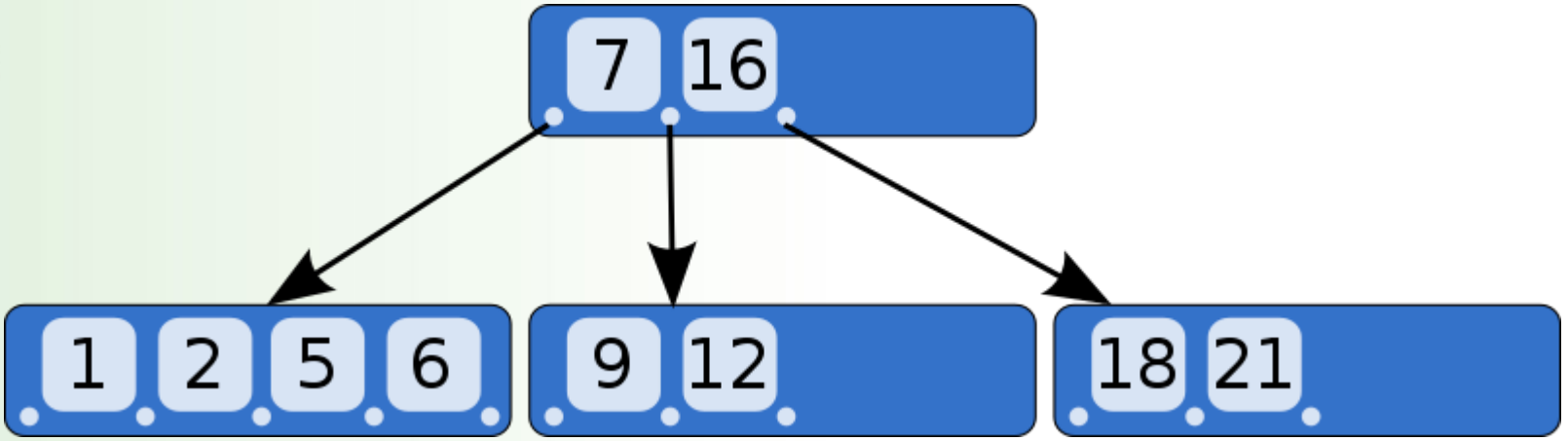
Theorem 18.1

If $n \geq 1$, then for any n-key B-tree T of height h and minimum degree $t \geq 2$,

$$h \leq \log_t \frac{n+1}{2} .$$

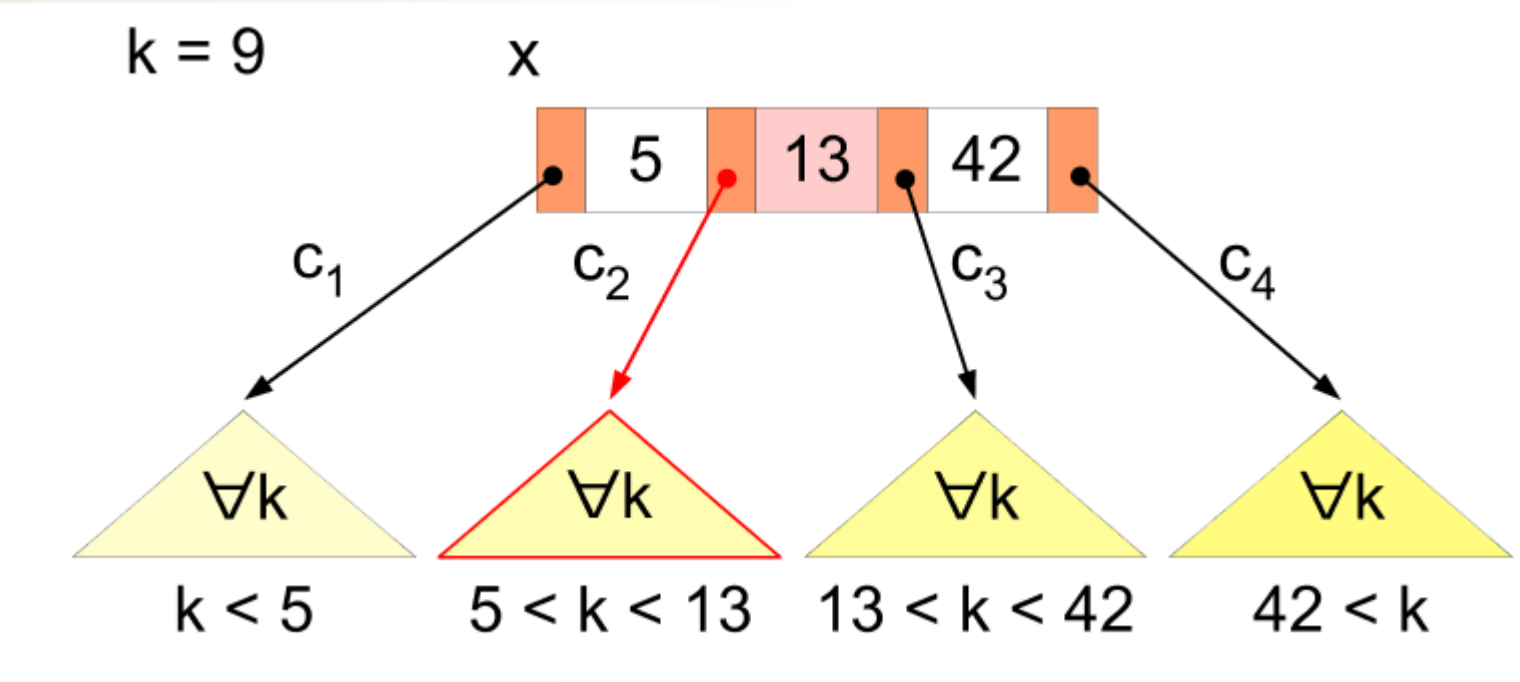
INSTITUTO FEDERAL
Santa Catarina

Btree



Btree - Busca

- Busca linear no nó
- Ou é
- Ou ainda pode ser
- Ou nunca será



INSTITUTO FEDERAL
Santa Catarina

Btree - Inserção

- Se a **folha** estiver cheia, corte-a ao redor key mediana
- A key mediana vai pro pai
- Já temos que ir cortando os nós cheios enquanto descemos (pois não podemos subir a tree)

INSTITUTO FEDERAL
Santa Catarina

Implementação

"Vocês nunca vão pegar um projeto do zero, e já devem ir se acostumando a lidar com o código dos outros"

Renan Starke

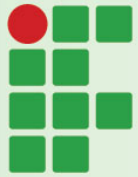
[Carece de fontes]

INSTITUTO FEDERAL
Santa Catarina

Testes

- Força bruta + valgrind
- Inseri várias linhas, procurei por bugs, testei combinações diferentes
- Tempo para selecionar 1000 linhas aleatórias, em um banco de dados com n linhas:

N	10k	100k	1M	10M
Tempo (us)	13	12	13	14



INSTITUTO

Comparação

```
node_position btree_insert(BTree* bt, int key, void *value) {
        if (bt == NULL){
                printf ("btree_insert: tree invalid");
                exit (EXIT_FAILURE);
        }
        #ifdef DEBUG
        printf("inserting key %d\n", key);
        #endif
        node_t *root = bt->root;
        // Esse pair será enviado durante as chamadas recursivas de inserção,
        // e é o que realmente será inserido na B-Tree
        pair_t *pair = _pair_new(key, value);
        if (root->n_keys == 2*bt->order -1) {
                // Caso a raiz da B-Tree já esteja cheia,
                // devemos executar o procedimento de split
                // e deixá-la com apenas uma chave.
                // Esse é o único caso em que a altura da B-Tree aumenta
                #ifdef DEBUG
                printf("root full - spliting up\n");
                #endif
                node_t *new_root = _node_new(bt->order, FALSE);
                new_root->children[0] = root;
                _btree_split(new_root, 0, bt->order);
                bt->root = new_root;
                // Podemos prosseguir com a inserção
                return _btree_insert_nonfull(new_root, pair, bt->order);
        }
        else {
                // A raiz respeita a restrição de não estar cheia
                #ifdef DEBUG
                printf("root not full - calling _btree_insert_nonfull()\n");
                #endif
                return _btree_insert_nonfull(bt->root, pair, bt->order);
        }
}
```

B-TREE-INSERT(T, k)

```
1   r = T.root
2   if r.n == 2t − 1
3       s = ALLOCATE-NODE()
4       T.root = s
5       s.leaf = FALSE
6       s.n = 0
7       s.c_1 = r
8       B-TREE-SPLIT-CHILD(s, 1)
9       B-TREE-INSERT-NONFULL(s, k)
10  else B-TREE-INSERT-NONFULL(r, k)
```

Comparação

```
node_position btree_insert(BTree* bt, int key, void *va
        if (bt == NULL){
                printf ("btree_insert: tree invalid");
                exit (EXIT_FAILURE);
        }
        #ifdef DEBUG
        printf("inserting key %d\n", key);
        #endif
        node_t *root = bt->root;
        // Esse pair será enviado durante as ch
        // e é o que realmente será inserido na B-
        pair_t *pair = _pair_new(key, value);
        if (root->n_keys == 2*bt->order -1) {
                // Caso a raiz da B-Tree já esteja
                // devemos executar o procedimento
                // e deixá-la com apenas uma chave.
                // Esse é o único caso em que a altura da B-Tree aumenta
                #ifdef DEBUG
                printf("root full - spliting up\n");
                #endif
                node_t *new_root = _node_new(bt->order, FALSE);
                new_root->children[0] = root;
                _btree_split(new_root, 0, bt->order);
                bt->root = new_root;
                // Podemos prosseguir com a inserção
                return _btree_insert_nonfull(new_root, pair, bt->order);
        }
        else {
                // A raiz respeita a restrição de não estar cheia
                #ifdef DEBUG
                printf("root not full - calling _btree_insert_nonfull()\n");
                #endif
                return _btree_insert_nonfull(bt->root, pair, bt->order);
        }
}
```

Sanity Checks

Debugs

Mallocs abstratos

Manipulação de ponteiros

Comentários inline

```
EE-INSERT(T, k)
 = T.root
2  if r.n == 2t − 1
3      s = ALLOCATE-NODE()
4      T.root = s
5      s.leaf = FALSE
6      s.n = 0
7      s.c_1 = r
8      B-TREE-SPLIT-CHILD(s, 1)
9      B-TREE-INSERT-NONFULL(s, k)
10 else B-TREE-INSERT-NONFULL(r, k)
```

Comparação

```
BTree* btree_new(char* name, int order) {
        BTree* bt = malloc(sizeof(BTree));
        assert(bt != NULL);

        #ifdef DEBUG
        printf("allocated new b-tree %s of order %d\n", name, order);
        #endif

        // Após alocar, temos que inicializar a B-Tree
        strcpy (bt->name, name);
        bt->order = order;
        bt->root = _node_new(order, TRUE);

        return bt;
}
```

B-TREE-CREATE(T)

1 $x = \text{ALLOCATE-NODE}()$
2 $x.leaf = \text{TRUE}$
3 $x.n = 0$
4 DISK-WRITE(x)
5 $T.root = x$

Manipulação explícita da memória

Funções getter/setter

```
char* btree_get_name (BTree* bt){
    return bt->name;
}
```

INSTITUTO FEDERAL
Santa Catarina

Btree - Conclusão

- A implementação acaba sendo bem mais trabalhosa do que o algoritmo
- Mesmo que se siga ao pé da letra o pseudo código, sempre surgem bizarrices (ex: x->y->z->w)
- Ficou longe de um banco de dados real, mas serviu para explorar a estrutura básica