Hash – Implementação

maio de 2019

Implementação de Tabelas de Hash

- tratamento de colisão
- estouro da tabela

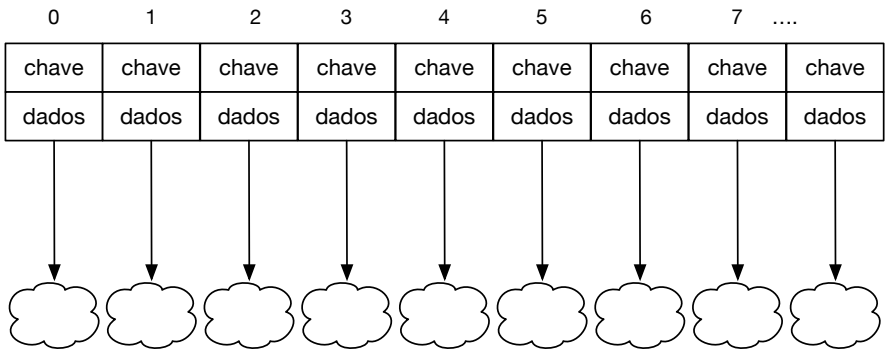
Implementação de Tabelas de Hash

tratamento de colisão

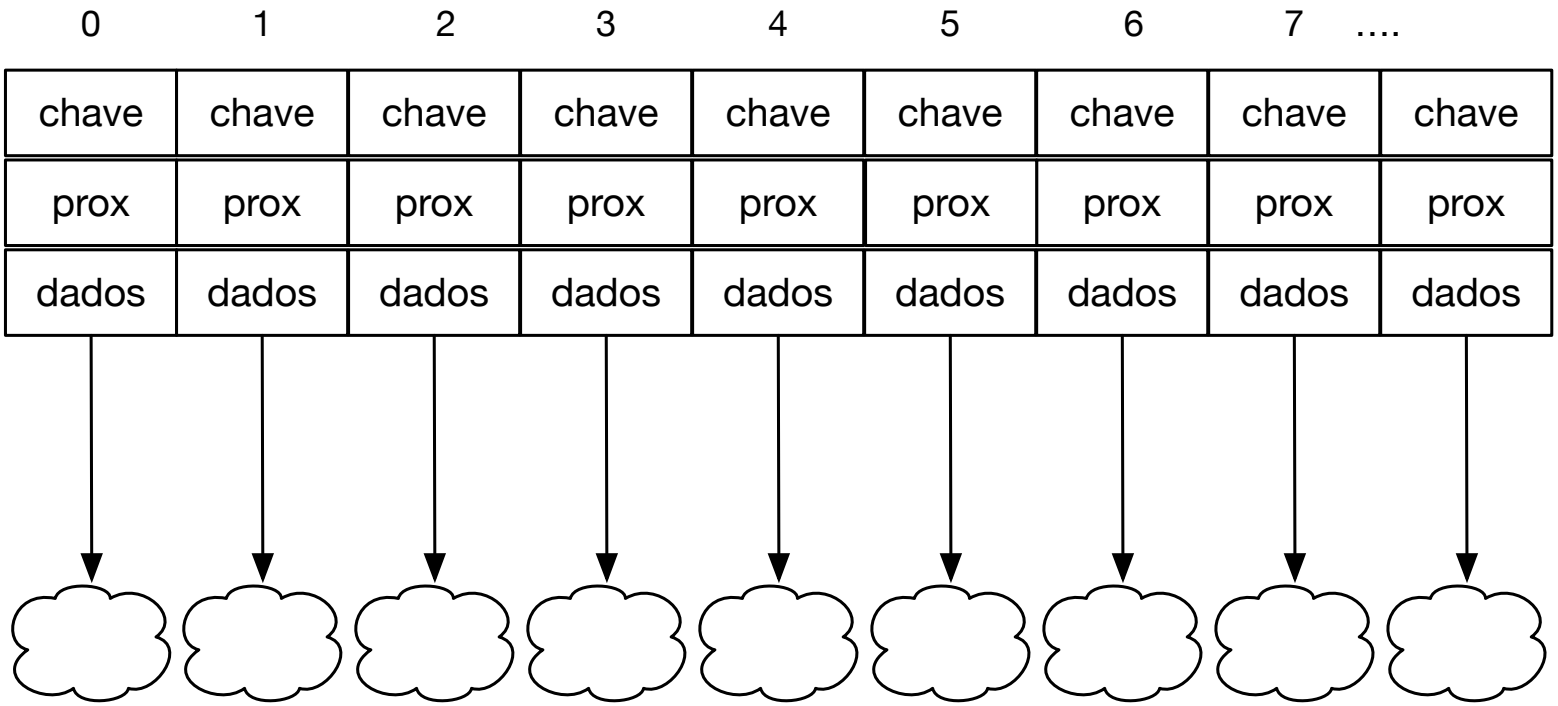
- encadeamento interno
- encadeamento externo

Implementação de Tabelas de Hash

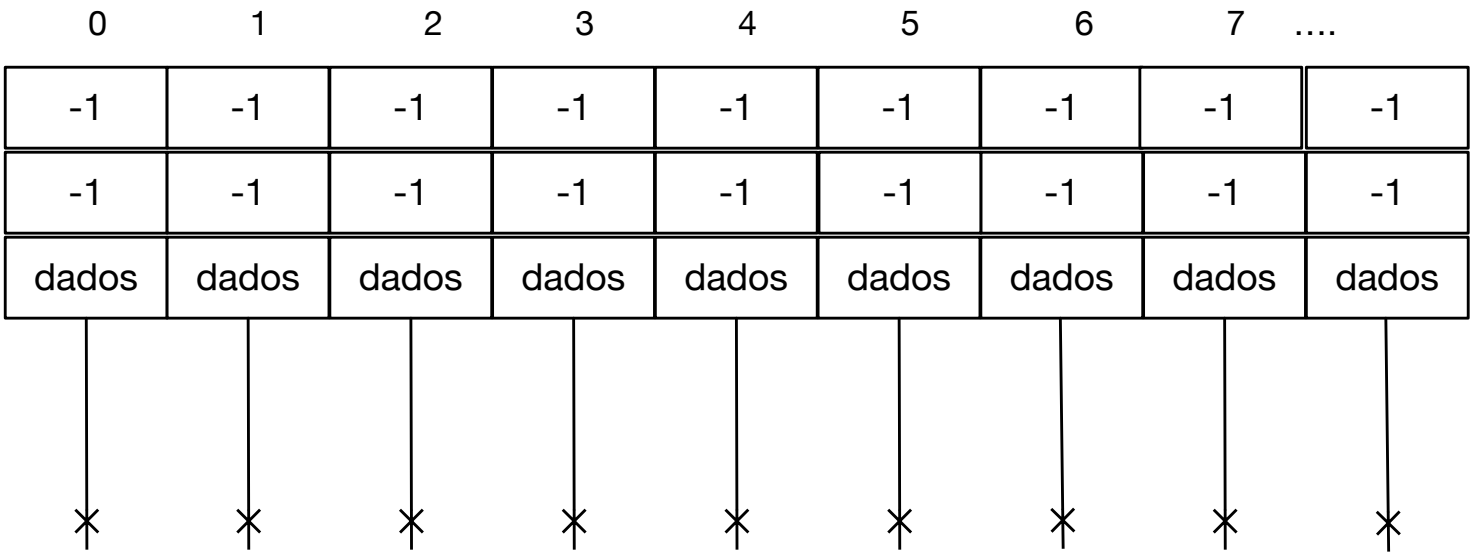
- encadeamento interno
  - itens com mesmo hash "sobram" para outras posições da tabela
  - clusterização (e busca): problemas



mais uma vez podemos trocar eficiência por espaço...



Estrutura que usaremos no lab



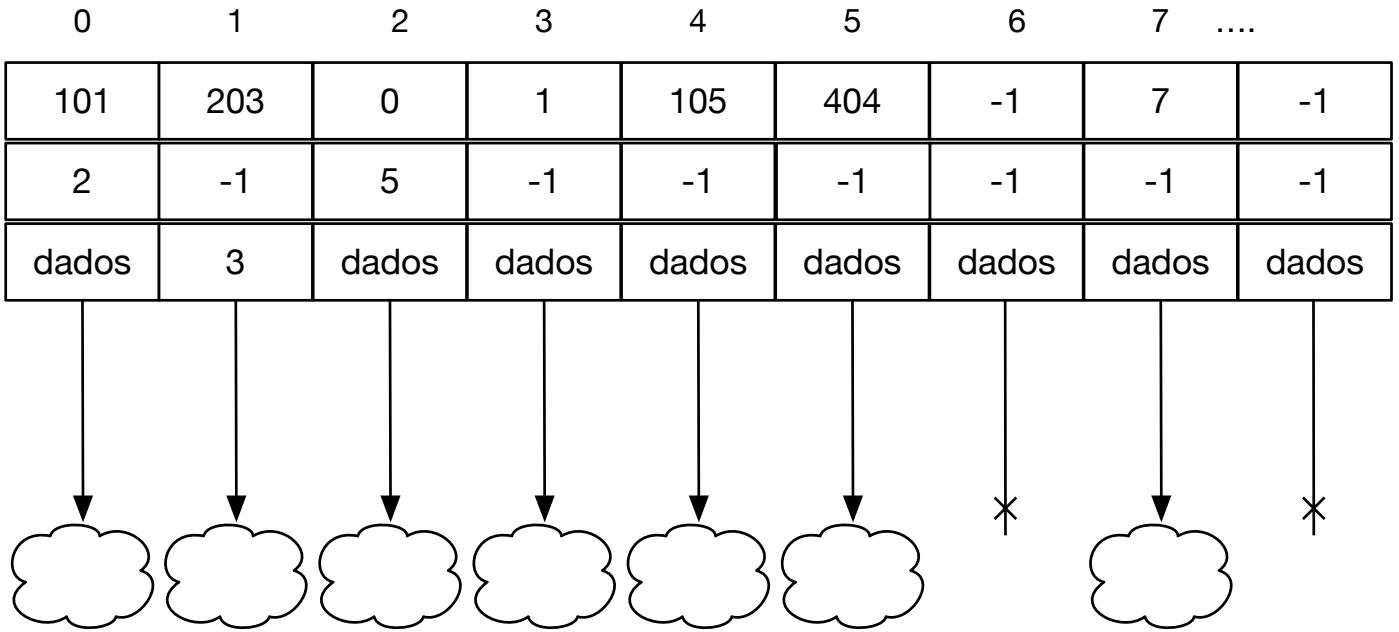
Estrutura que usaremos no lab

```
typedef struct {
  int chave;
  int dados;
  int prox;
} ttabpos;

struct smapa {
  int tam;
  ttabpos *tabpos;
};
```

Exemplo

exemplo: hash(c) = c%101



Inserção

insert(th, c, dados)

- hash(c): posição livre ou ocupada?
- livre: insere nessa posição
- ocupada: trata conflito com chave c1 já presente
  - conflito primário
  - conflito secundário

Inserção

conflito primário

- $hash(c1) = hash(c)$

Inserção

conflito primário

- $hash(c1) = hash(c)$
- encadeia item na mesma cadeia de c1

Inserção

conflito secundário

- $hash(c1) \neq hash(c)$

Inserção

conflito secundário

- $hash(c1) \neq hash(c)$
- tem que achar cadeia de c1


PUC
RIO

E quando a tabela fica cheia?

E quando a tabela fica cheia?

```
typedef struct {
  int chave;
  int dados;
  int prox;
} ttabpos;

struct smapa {
  int tam;
  int ocupadas; /*!!! idealmente mantem maximo de ocupação */
  ttabpos *tabpos;
};
```

E quando a tabela fica cheia?

- realocação e *rehash*

```
Mapa* insere (Mapa* m, int chave, int dados){
  if (m->ocupadas > (m->tam)*0.75)
    redimensiona (m);
  int h = hash (m, chave); /* hash tem que saber tam */
  ...
```