

## AED2 - Aula 06

### Skip lists

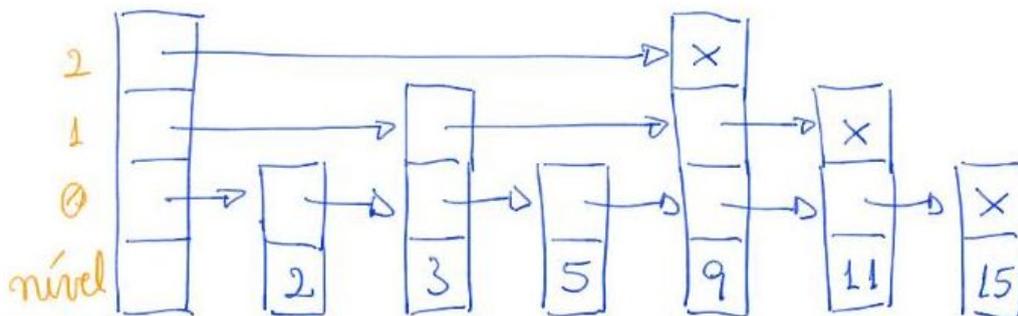
Considere usar uma lista ligada ordenada

- para implementar uma tabela de símbolos.
  - Essa abordagem não é eficiente, pois localizar um registro
    - leva tempo proporcional ao tamanho da lista.

A ideia da estrutura de dados Skip List é

- usar uma hierarquia de listas ligadas ordenadas,
  - em que cada lista tem uma densidade de itens própria,
- sendo que as listas estão conectadas entre si.

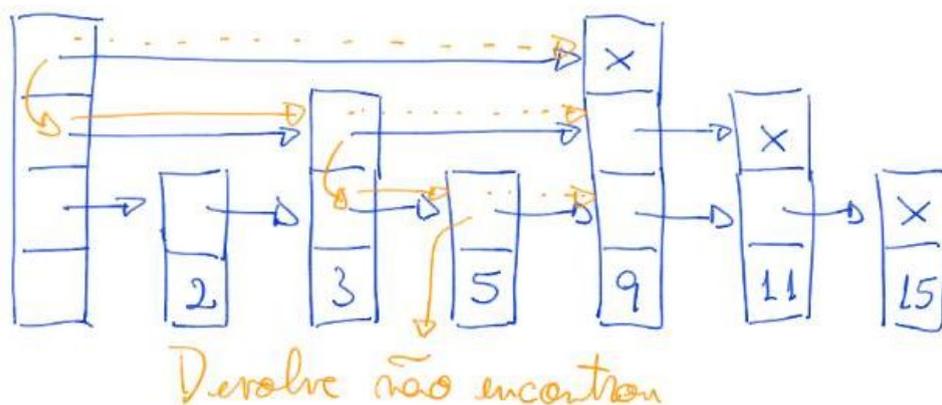
Exemplo:



Ideia da busca:

- Começar a percorrer a lista de maior nível,
  - que é a que tem menos itens.
- Quando encontrar o fim da lista
  - ou um item com chave maior do que a buscada,
    - continua a busca no nível abaixo,
    - que tem mais itens.

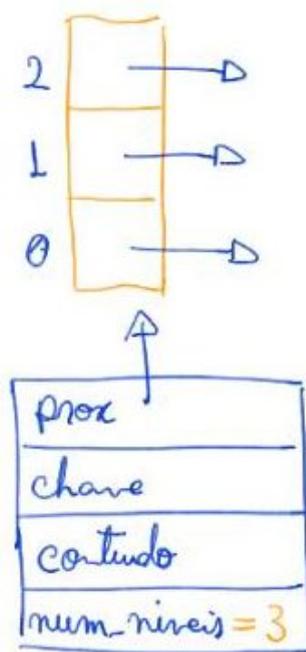
Exemplo de busca pela chave 8:



Antes de analisarmos o código da busca,

- vamos entender a estrutura dos nós:

Noh



```
#define num_max_niveis 100

typedef int Chave;
typedef int Item;

typedef struct noh {
    Chave chave;
    Item conteudo;
    struct noh **prox;
    int num_niveis;
} Noh;

static Noh *lista;
static int num_itens, nivel_max;
```

Código da busca:

```
Noh *buscaR(Noh *t, Chave chave, int nivel) {
    if (t != lista && chave == t->chave) return t;
    if (t->prox[nivel] == NULL || chave < t->prox[nivel]->chave) {
        if (nivel == 0) return NULL;
        return buscaR(t, chave, nivel - 1);
    }
    return buscaR(t->prox[nivel], chave, nivel);
}

Noh *Tbusca(Chave chave) {
    return buscaR(lista, chave, nivel_max);
}
```

### Eficiência de tempo:

- Busca em skip lists leva, em média,
  - $(t \log_t n) / 2 = O(\log n)$  comparações,
    - sendo  $t > 2$  o fator de dispersão da skip list,
      - i.e., o número de nós do nível  $i$  para o nível  $i + 1$ 
        - cai, em média, de  $1 / t$ .
  - Por ser uma estrutura probabilística, falamos de eficiência média.
    - No entanto, note que essa média depende apenas
      - das escolhas aleatórias da própria estrutura,
        - e não dos valores da entrada.
  - Para entender de onde vem o valor  $(t \log_t n) / 2$ , observe que
    - uma skip list com  $n$  itens deve ter  $\log_t n$  níveis,
      - já que o número de itens cai de  $1 / t$  por nível.
  - Além disso, entre dois valores do nível  $i + 1$  devem existir,
    - em média,  $t$  valores no nível  $i$ .
  - Por isso, esperamos dar  $t / 2$  saltos por nível, em média,
    - antes de descer para o nível seguinte.
  - O resultado deriva do produto do número esperado de níveis
    - pelo número esperado de saltos por nível.

### Eficiência de espaço:

- Skip lists tem, em média,
  - $n (t / (t - 1)) = O(n)$  nós.
- Para entender de onde vem esse valor, observe que,
  - o primeiro nível tem  $n$  nós,
  - o segundo tem  $n/t$ ,
  - o terceiro  $n/t^2$ .
- Assim, o número esperado de nós corresponde
  - à soma dos termos de uma Progressão Geométrica (PG)
    - que começa em  $n$  e tem razão  $1/t$ .
- Deduzindo a soma dos termos de uma PG de razão  $< 1$  temos
  - $\text{SomaPG}(q) = 1 + q + q^2 + q^3 + \dots$
  - $q \text{SomaPG}(q) = q + q^2 + q^3 + \dots$
  - $(1 - q) \text{SomaPG}(q) = (1 + q + q^2 + q^3 + \dots) - (q + q^2 + q^3 + \dots)$
  - $\text{SomaPG}(q) = 1 / (1 - q)$ .
- Como na nossa PG o primeiro termo é  $n$  e a razão é  $1/t$ ,
  - sua soma converge para
    - $n * 1 / (1 - 1/t) = n / ((t - 1) / t) = n (t / (t - 1))$ .

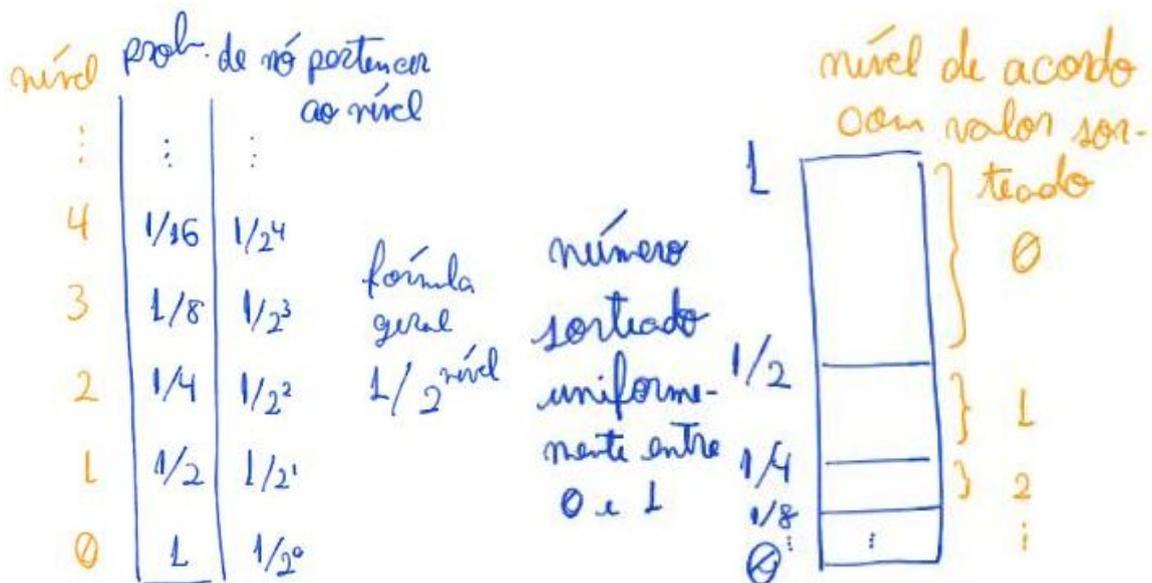
### Relação entre tempo e espaço:

- Comparando a eficiência de tempo e de espaço das skip lists, vale notar que
  - a eficiência de tempo  $(t \log_t n) / 2$  é  $O(\log n)$
  - e a eficiência de espaço  $n (t / (t - 1))$  é  $O(n)$ ,
    - por considerarmos que o fator de dispersão  $t$  é fixo.
- Agora, vamos comparar skip lists com diferentes fatores de dispersão.
  - Como exemplo, tome  $t' = 2$ 
    - tempo =  $(t' \log_{t'} n) / 2 = (2 \lg n) / 2 = \lg n$
    - espaço =  $n (t' / (t' - 1)) = n (2 / (2 - 1)) = 2 n$
  - e compare com  $t'' = 10$

- tempo =  $(t'' \log_{t''} n) / 2 = (10 \log_{10} n) / 2 = 5 (\lg n / \lg 10) \approx 1,5 \lg n$
    - espaço =  $n (t'' / (t'' - 1)) = n (10 / (10 - 1)) \approx 1,11 n$
  - Podemos perceber que, quanto maior o t,
    - mais lenta a busca e menos espaço ocupa a skip list,
      - o que é consistente com uma dispersão maior.

Probabilidade:

- A ideia central das skip lists é que
  - a cada novo nível temos menos nós,
    - mais especificamente  $1/t$  do número do nível anterior,
    - e estes devem estar homogeneamente espaçados.
  - Para obter tal resultado precisamos utilizar escolhas aleatórias,
    - de modo que um nó pertença ao nível i com probabilidade  $(1/t)^i$ .
  - Uma ideia para fazer isso é sortear um valor entre 0 e 1
    - e atribuir um nível de acordo com o valor sorteado,
      - como sugerem as seguintes figuras, i.e.,
        - nível 0 se o valor estiver entre 1 e  $\frac{1}{2}$
        - nível 1 se estiver entre  $\frac{1}{2}$  e  $\frac{1}{4}$ ,
        - nível 2, se estiver entre  $\frac{1}{4}$  e  $\frac{1}{8}$
        - ...



- A seguinte função implementa essa ideia

```
int nivelAleatorio() {
    int nivel, disp_acum, t = 2, v = rand();
    disp_acum = t;
    for (nivel = 0; nivel < num_max_niveis; nivel++) {
        if (v > RAND_MAX / disp_acum)
            break;
        disp_acum *= t;
    }
    if (nivel > nivel_max) nivel_max = nivel;
    return nivel;
}
```

- Observe que, essa função sorteia um valor  $v$  e
  - quanto menor tal valor maior será o nível do nó.
- Note que, no início do laço valem os invariantes
  - $\text{disp\_acum} = t^{(\text{nível} + 1)}$  e
  - $v / \text{RAND\_MAX} \leq t^{\text{nível}}$ .
- Como  $v = \text{rand}()$  recebe um valor
  - aleatório e uniforme entre 0 e  $\text{RAND\_MAX}$ ,
    - o que estamos fazendo é sortear um valor entre 0 e 1
  - e colocando o nó num determinado nível
    - se o valor sorteado  $\leq 1 / t^{\text{nível}}$ ,
      - o que ocorre com probabilidade  $1 / t^{\text{nível}}$ .

Ideia da inserção:

- Sortear um nível para o novo nó,
  - percorrer um caminho semelhante à busca até
    - chegar na posição que o nó deveria ocupar no nível sorteado,
  - então inserir o nó em todas as listas
    - com nível menor ou igual ao dele.

Código da inserção:

```
Noh *novo(Chave chave, Item conteudo, int num_niveis) {
    int i;
    Noh *p = (Noh *)malloc(sizeof *p);
    p->chave = chave;
    p->conteudo = conteudo;
    p->prox = malloc(num_niveis * sizeof(Noh *));
    p->num_niveis = num_niveis;
    for (i = 0; i < num_niveis; i++)
        p->prox[i] = NULL;
    return p;
}

void insereR(Noh *t, Noh *novoNoh, int nivel) {
    Chave chave = novoNoh->chave;
    if (t->prox[nivel] == NULL || chave < t->prox[nivel]->chave) {
        if (nivel < novoNoh->num_niveis) {
            novoNoh->prox[nivel] = t->prox[nivel];
            t->prox[nivel] = novoNoh;
        }
        if (nivel > 0) insereR(t, novoNoh, nivel - 1);
        return;
    }
    insereR(t->prox[nivel], novoNoh, nivel);
}
```

```

void TSinsere(Chave chave, Item conteudo) {
    int nivelAleat = nivelAleatorio();
    Noh *novoNoh = novo(chave, conteudo, nivelAleat + 1);
    insereR(lista, novoNoh, nivel_max);
    num_itens++;
}

```

Eficiência de tempo:

- Inserção leva, em média,  $(t \log_t n) / 2 = O(\log n)$  comparações,
  - sendo  $t > 2$  o fator de dispersão da skip list.

Ideia da remoção:

- A ideia é buscar o nó normalmente,
  - removê-lo de todas as listas com nível menor ou igual ao dele,
- então liberar o nó em si.

Código da remoção:

```

int removeR(Noh *t, Chave chave, int nivel) {
    Noh *p = t->prox[nivel];
    if (p == NULL || chave <= p->chave) {
        if (p != NULL && chave == p->chave) {
            t->prox[nivel] = p->prox[nivel];
            if (nivel == 0) {
                free(p->prox);
                free(p);
                return 1;
            }
        }
        if (nivel == 0) return 0;
        return removeR(t, chave, nivel - 1);
    }
    return removeR(t->prox[nivel], chave, nivel);
}

void TSremove(Chave chave) {
    if (removeR(lista, chave, nivel_max))
        num_itens--;
}

```

Eficiência de tempo:

- Inserção leva, em média,  $(t \log_t n) / 2 = O(\log n)$  comparações,
  - sendo  $t > 2$  o fator de dispersão da skip list.

Bônus:

- Observe que, nas várias funções as chamadas recursivas

- são feitas por último, o que caracteriza recursão caudal.
- Neste caso, é relativamente simples substituir essas chamadas recursivas
  - por um laço, que envolve a função e tem condições de parada
    - complementares ao caso base da recursão,
  - junto da reatribuição de valores para os parâmetros da função
    - nos pontos em que ocorriam as chamadas recursivas.
- Implementar a versão iterativa das funções de manipulação da skip list
  - é um bom exercício.