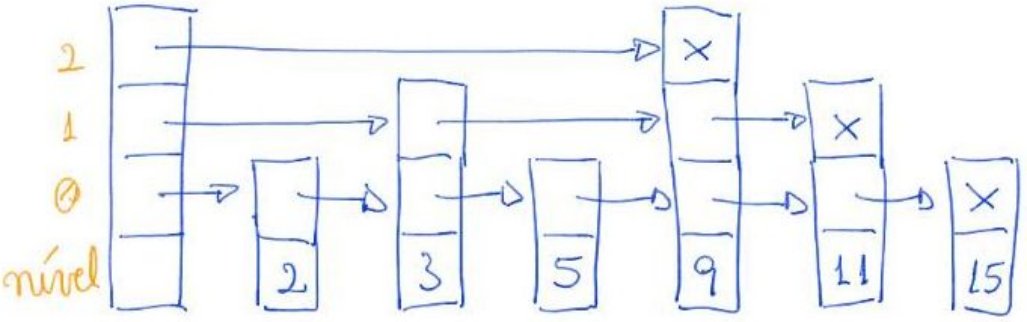
AED2 - Aula 08
Skip lists

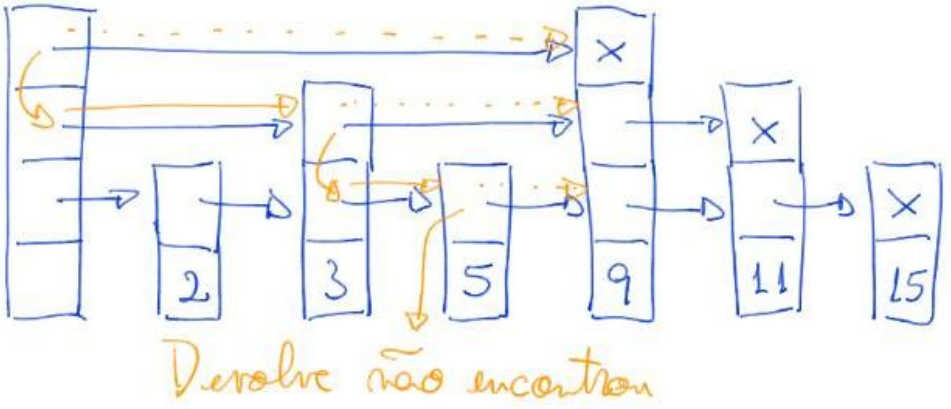
Ideia:

- listas hierárquicas, com diferentes densidades de itens e conectadas, possibilitando busca (e outras operações) eficientes.
- exemplo



Busca:

- exemplo de busca pelo 8



- código

```
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 *busca(Chave chave)
{
```

```
    return buscaR(lista, chave, lgN);
}
```

Eficiência de tempo:

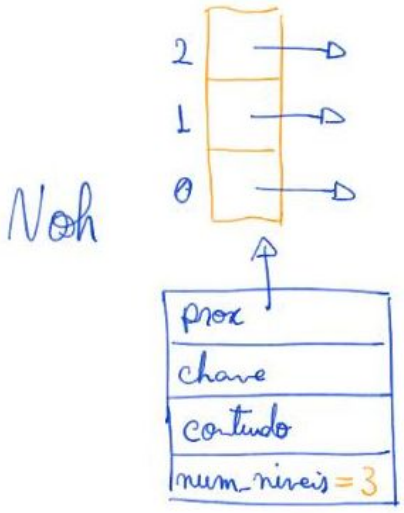
- busca em skip lists leva, em média, (t log_t N) / 2 = O(log n) comparações,
    - sendo t > 2 o fator de dispersão da skip list, i.e., o número de nós do nível i para o nível i + 1 cai, em média, de 1/t.
    - por ser uma estrutura probabilística, falamos de eficiência média
        - no entanto, note 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.
- 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.
    - Como toda PG de razão < 1, sua soma converge para
      N * 1 / (1 - 1/t) = N / ((t - 1) / t) = N (t / (t -1))

Nós e definições:

- esquema

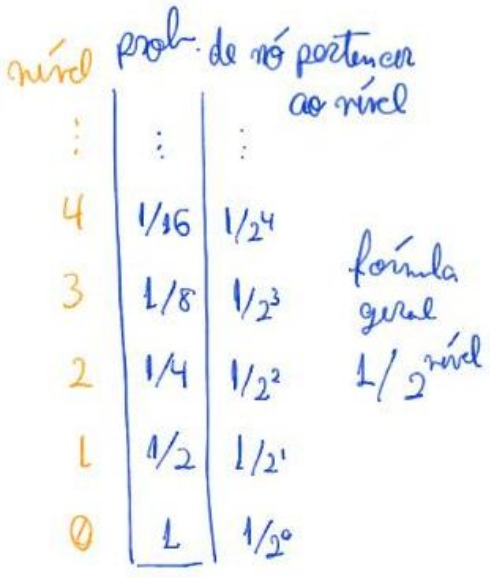


- código

```
#define lgNmax 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 N, lgN; // numero de nós e nível do nó mais alto
```

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 que estes estão homogeneamente espaçados
- para obter tal resultado precisamos utilizar escolhas aleatórias, de modo que
  - um nó qualquer pertença ao nível i com probabilidade (1/t)^i
    - lembrando que t > 2 é o fator de dispersão da skip list

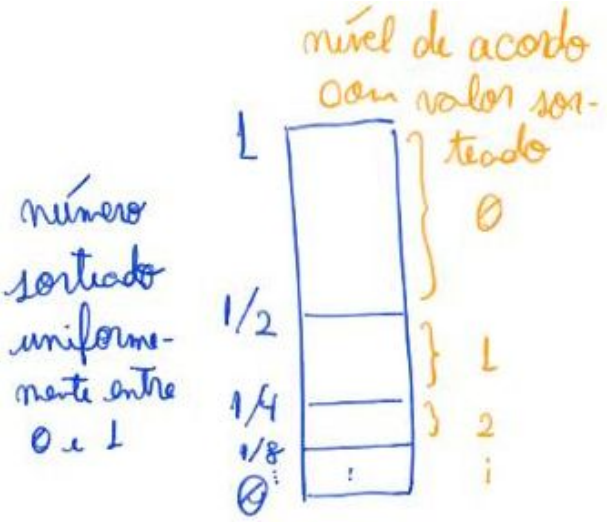


- a seguinte função implementa essa ideia

```
int nivelAleatorio()
{
   int i, j, t = 2, v = rand();
   for (i = 0, j = t; i < lgNmax; i++, j *= t)
       if (v > RAND_MAX / j)
           break;
```

```
    if (i > lgN)
        lgN = i;
    return i;
}
```

- observe que essa função sorteia um valor v e
    - quanto menor tal valor maior será o nível do nó
- note que a comparação v > RAND_MAX / j
    - equivale a v / RAND_MAX > 1 / j
- assim, podemos pensar que estamos sorteando um valor entre 0 e 1
    - e colocando o nó no nível i se tal valor <= 1/t^i



Inserção:

- código

```
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 insere(Chave chave, Item conteudo)
{
    int nivelAleat = nivelAleatorio();
```

```
    Noh *novoNoh = novo(chave, conteudo, nivelAleat + 1);
    insereR(lista, novoNoh, lgN);
    N++;
}
```

Remoção:

- código

```
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, lgN))
        N--;
}
```