```
void excluir(TipoChave ch, ArvoreB *arvore)
{
  int diminuiu;
  No *aux;
  excluirAux(ch, arvore, &diminuiu);
  if (diminuiu && (*arvore)->numChaves == 0)
  {
    aux = *arvore;
    *arvore = aux->filhos[0];
    free(aux);
  }
}
```

```
void excluirAux(TipoChave ch, No **ap, int *diminuiu) {
  int ind, j;
  No *WITH;
  if (*ap == NULL) {
    printf("\nErro: registro nao esta na arvore.\n");
    *diminuiu = 0;
    return;
  }
  WITH = *ap;
  ind = 1;
  while (ind < WITH->numChaves && ch > WITH->registros[ind -
    1].chave)
    ind++;
  if (ch == WITH->registros[ind - 1].chave) {
    if (WITH->filhos[ind - 1] == NULL) {
      WITH->numChaves--;
      *diminuiu = WITH->numChaves < minimoDeChaves;
      for (j = ind; j <= WITH->numChaves; j++) {
        WITH->registros[j - 1] = WITH->registros[j];
        WITH->filhos[j] = WITH->filhos[j + 1];
      }
      return;
    }
```

```
    antecessor (*ap, ind, WITH->filhos[ind - 1], diminuiu);
    if (*diminuiu)
      reconstituir (WITH->filhos[ind - 1], *ap, ind - 1, diminuiu);
    return;
  }
  if (ch > WITH->registros[ind - 1].chave)
    ind++;
  excluirAux (ch, &WITH->filhos[ind - 1], diminuiu);
  if (*diminuiu)
    reconstituir (WITH->filhos[ind - 1], *ap, ind - 1, diminuiu);
}
```

```
void antecessor(No *ap, int ind, No *apPai, int *diminuiu)
{
  if (apPai->filhos[apPai->numChaves] != NULL)
  {
    antecessor(ap, ind, apPai->filhos[apPai->numChaves], diminuiu);
    if (*diminuiu)
      reconstituir(apPai->filhos[apPai->numChaves], apPai, apPai-
    >numChaves, diminuiu);
    return;
  }
  ap->registros[ind - 1] = apPai->registros[apPai->numChaves - 1];
  apPai->numChaves--;
  *diminuiu = apPai->numChaves < minimoDeChaves;
}
```

```
void reconstituir(No *apNo, No *apPai, int posPai, int *diminuiu)
{
  No *aux;
  int dispAux, j;
  if (posPai < apPai->numChaves) {
    aux = apPai->filhos[posPai + 1];
    dispAux = (aux->numChaves - minimoDeChaves + 1) / 2;
    apNo->registros[apNo->numChaves] = apPai->registros[posPai];
    apNo->filhos[apNo->numChaves + 1] = aux->filhos[0];
    apNo->numChaves++;
    if (dispAux > 0) {
      for (j = 1; j < dispAux; j++)
        inserirChaveNoNo(apNo, aux->registros[j - 1], aux->filhos[j]);
      apPai->registros[posPai] = aux->registros[dispAux - 1];
      aux->numChaves -= dispAux;
      for (j = 0; j < aux->numChaves; j++)
        aux->registros[j] = aux->registros[j + dispAux];
      for (j = 0; j <= aux->numChaves; j++)
        aux->filhos[j] = aux->filhos[j + dispAux];
      *diminuiu = 0;
    }
```

```
  else
  {
    for (j = 1; j <= minimoDeChaves; j++)
      inserirChaveNoNo(apNo, aux->registros[j - 1], aux->filhos[j]);
    free(aux);
    for (j = posPai + 1; j < apPai->numChaves; j++)
    {
      apPai->registros[j - 1] = apPai->registros[j];
      apPai->filhos[j] = apPai->filhos[j + 1];
    }
    apPai->numChaves--;
    if (apPai->numChaves >= minimoDeChaves)
      *diminuiu = 0;
  }
}
else
{
  aux = apPai->filhos[posPai - 1];
  dispAux = (aux->numChaves - minimoDeChaves + 1) / 2;
  for (j = apNo->numChaves; j >= 1; j--)
    apNo->registros[j] = apNo->registros[j - 1];
```

```
apNo->registros[0] = apPai->registros[posPai - 1];
for (j = apNo->numChaves; j >= 0; j--)
  apNo->filhos[j + 1] = apNo->filhos[j];
apNo->numChaves++;
if (dispAux > 0) {
  for (j = 1; j < dispAux; j++)
    inserirChaveNoNo(apNo, aux->registros[aux->numChaves - j],
aux->filhos[aux->numChaves - j + 1]);
  apNo->filhos[0] = aux->filhos[aux->numChaves - dispAux + 1];
  apPai->registros[posPai - 1] = aux->registros[aux->numChaves
- dispAux];
  aux->numChaves -= dispAux;
  *diminuiu = 0;
}
else {
  for (j = 1; j <= minimoDeChaves; j++)
    inserirChaveNoNo(aux, apNo->registros[j - 1], apNo->filhos[j]);
  free(apNo);
  apPai->numChaves--;
  if (apPai->numChaves >= minimoDeChaves)
    *diminuiu = 0;
  } }
```

Árvores B

Exercício:

Os alunos desta disciplina, visando obter presença na aula que não ocorreu no dia 4 de dezembro devido à realização do SCIENTEX, deverão se reunir em trios e preparar uma apresentação para a aula do dia 18 de dezembro. A qual apresentará uma explicação das funções apresentadas nos slides compreendidos entre o 100 e 106. Em tempo, externo que as equipes também podem optar por apresentar uma função equivalente que efetue a remoção de uma chave em uma árvore B.

UNIVASF

Árvores B

Em muitas aplicações pode ser necessário apresentar as chaves em ordem crescente.

Com esta operação pode ser desenvolvida?

A árvore pode ser percorrida em in-ordem.

Visando consolidar sua visão da estruturação dos dados em uma árvore B, elabore uma função, na linguagem C, que execute o percurso in-ordem em uma árvore B, apresentando as chaves visitadas na saída padrão.

UNIVASF

```
void percursoInOrdem(ArvoreB arvore)
{
  if (arvore != NULL)
  {
    int i;
    for (i=0; i<arvore->numChaves; i++)
    {
      percursoInOrdem(arvore->filhos[i]);
      printf("%d ", arvore->registros[i].chave);
    }
    percursoInOrdem(arvore->filhos[i]);
  }
}
```

Árvores B

As árvores B, de acordo com sua definição estão garantidas de serem pelo menos 50% cheias, por isso pode acontecer de 50% de espaço ser basicamente perdido.

Com que frequência isto acontece?

Se muito frequentemente a definição precisa ser reconsiderada ou algumas outras restrições impostas a esta árvore B.

As analises e simulações, no entanto, indicam que, depois de uma série de numerosas inserções e remoções aleatória, a árvore B está aproximadamente 69% cheia (Yao, 1968), depois deste limiar as mudanças na porcentagem de células ocupadas são muito pequenas.

UNIVASF

Árvores B*

Uma vez que cada nó de uma árvore B representa um bloco de memória secundária, acessar um nó significa um acesso da memória secundária, que é dispendioso se comparado a acessar chaves no nó que reside na memória primária. Em consequência, quanto menos nós forem criados, melhor.

Uma árvore B* é uma variante da árvore B introduzida por Donald Knuth e batizada por Douglas Comer. Em uma árvore B*, exige-se que todos os nós, exceto a raiz, estejam pelo menos 2/3 cheios, não apenas metade cheios, como nas árvores B.

UNIVASF

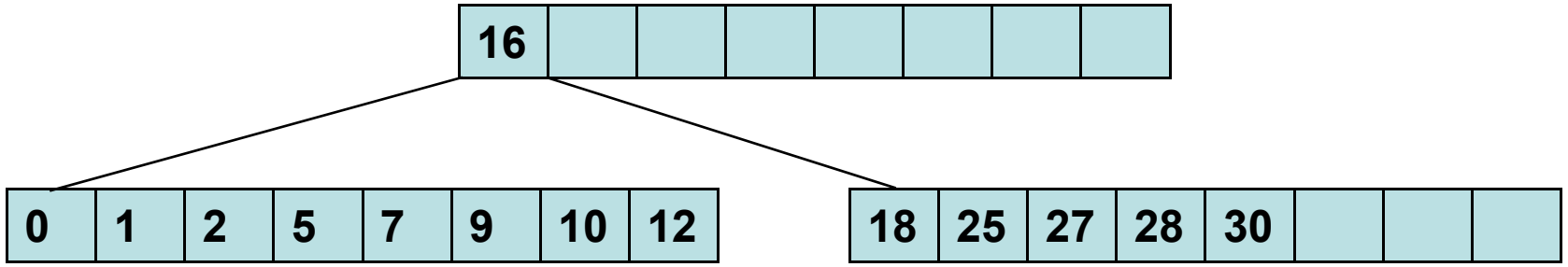
Árvores B*

Mas precisamente, o número de chaves em todos os nós não raiz em uma árvore B de ordem m é agora k para $\lfloor (2m-1)/3 \rfloor \leq k \leq m-1$.

A frequência de divisão de nós é diminuída atrasando-se uma divisão e, quando o momento vem, dividindo-se dois nós em três, não um em dois. A utilização média da árvore B* é de 81% (Leung, 1984).

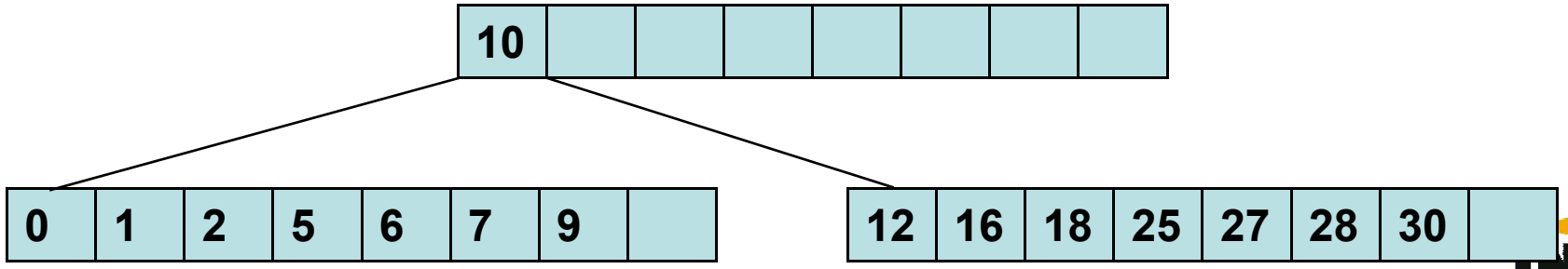
Uma divisão em uma árvore B* é atrasada tentando-se redistribuir as chaves entre um nó e seu irmão quando o nó transborda. Veremos agora um exemplo deste processo na árvore B* de ordem 9 apresentada no slide a seguir.

Árvores B*



Inserir 6

A chave 6 é para ser inserida no nó esquerdo, que já está cheio. Em vez de dividir o nó, todas as chaves desse nó e de seu irmão são homogeneamente divididas e a chave mediana, a chave 10, é colocada no ascendente.



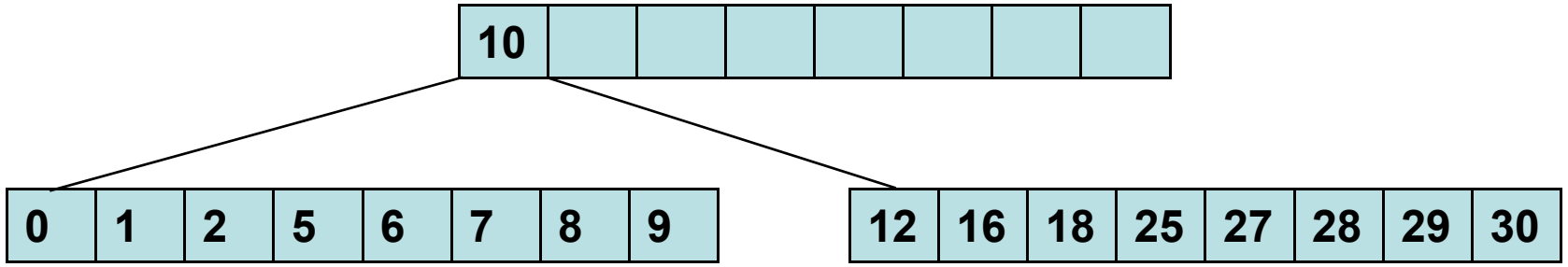
Árvores B*

Note que isso não apenas divide homogeneamente as chaves, mas também o espaço livre, de modo que o nó que estava cheio é agora capaz de acomodar uma chave a mais.

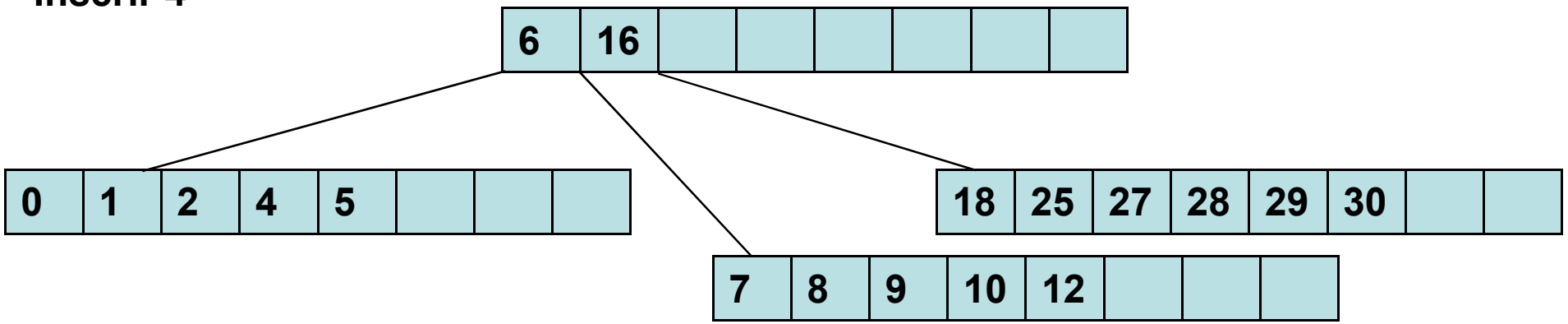
Se o irmão também estiver cheio, uma divisão ocorre: um novo nó é criado, as chaves do nó e de seu irmão (junto com a chave de separação do ascendente) são homogeneamente divididas entre três nós e duas chaves de separação são colocadas no ascendente (veja o processo no próximo slide). Todos os três nós que participam da divisão são garantidos de estarem 2/3 cheios.

UNIVASF

Árvores B*



Inserir 4



Note que, como se pode esperar, esse aumento de um fator de enchimento pode ser feito em uma variedade de modos, e alguns sistemas de banco de dados permitem ao usuário escolher um fator de enchimento entre 0,5 e 1.

Árvores B*

Em particular, uma árvore B cujos nós devem estar obrigatoriamente 75% cheios, no mínimo, é chamada de árvore B** (McCreight, 1977).

Está observação sugere uma generalização: uma árvore B^n é uma árvore B cujos nós precisam estar (n+1)/(n+2) cheios.