SKIPLIST

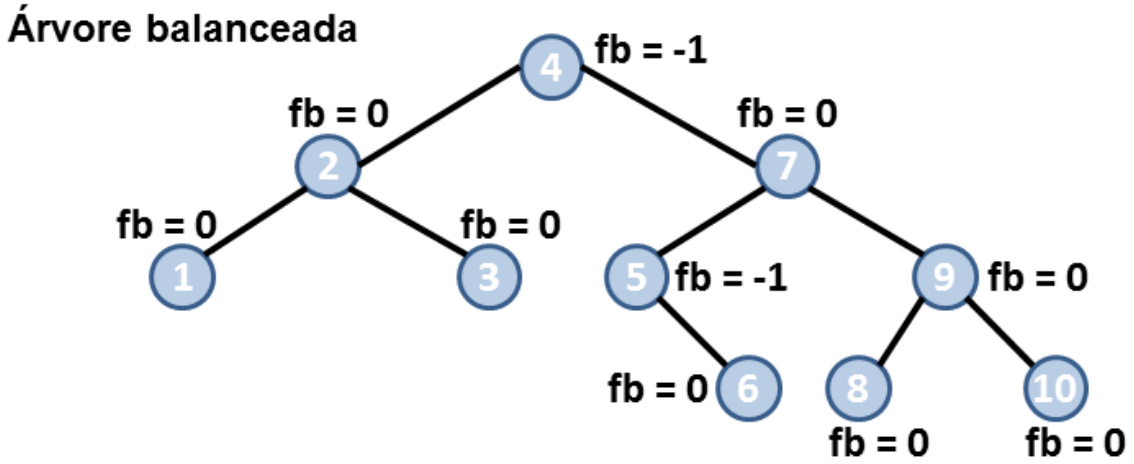
Prof. André Backes | @progdescomplicada

Relembrando...

- Lista Dinâmica Encadeada
  - Fácil implementação
  - No melhor caso
    - Inserção/remoção pode ser feita em ***O(1)***
  - No pior caso
    - Inserção/remoção/busca é feita em ***O(N)***
  - Não é possível usar Busca Binária
  - Permite percorrer todos os itens em ***O(N)***

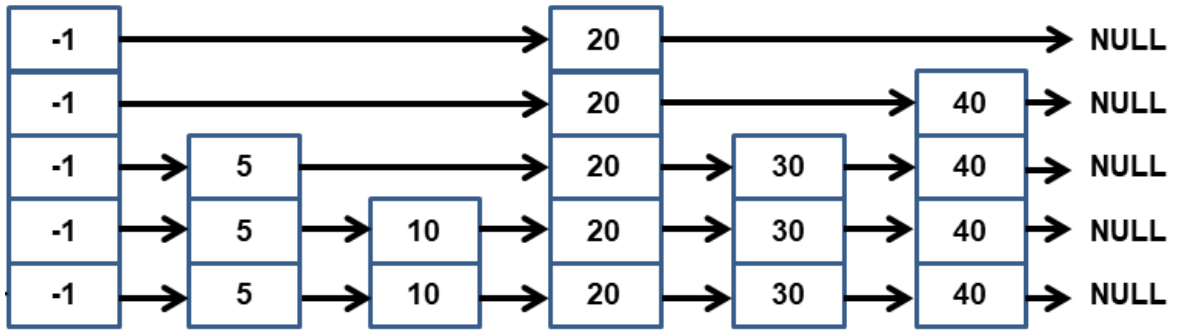
Relembrando...

- Árvore Binária de Busca Balanceada
  - Inserção/remoção/busca é feita em ***O(log N)***
  - É preciso tratar o balanceamento dos nós da árvore
  - A implementação é complexa
    - Envolve rotações e realocações de nós



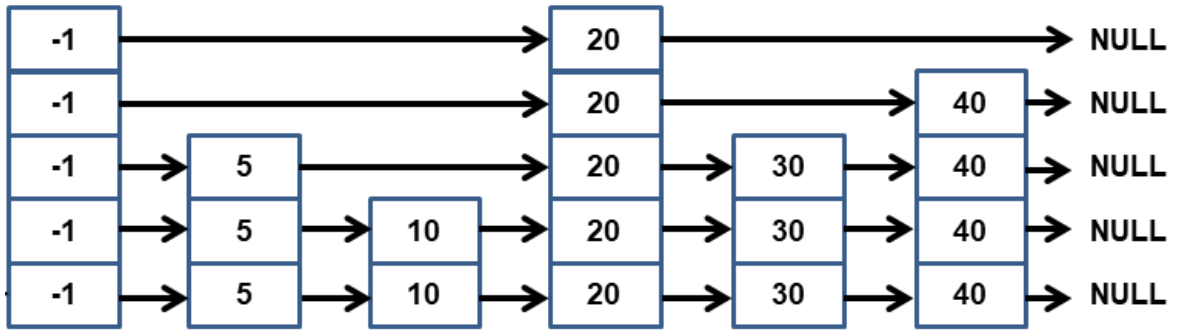
Skip List | Definição

- Proposta em 1990 por Bill Pugh
- Generalização da Lista Dinâmica Encadeada
  - Listas Encadeadas consideram nós a frente e atrás
  - Skip list considera também os nós acima e abaixo
  - Funciona como um agrupamento vertical de listas



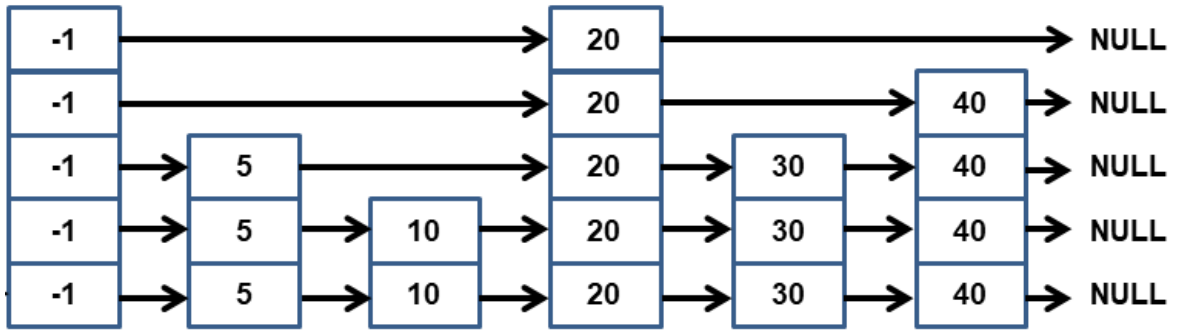
Skip List | Definição

- Usa uma estrutura hierárquica de Listas Dinâmicas
  - As listas são organizadas em diferentes níveis
  - Níveis mais altos permitem “pular” vários nós, o que acelera o processo de busca
- Estrutura aleatória
  - Usa sorteio para definir o número de níveis de um nó



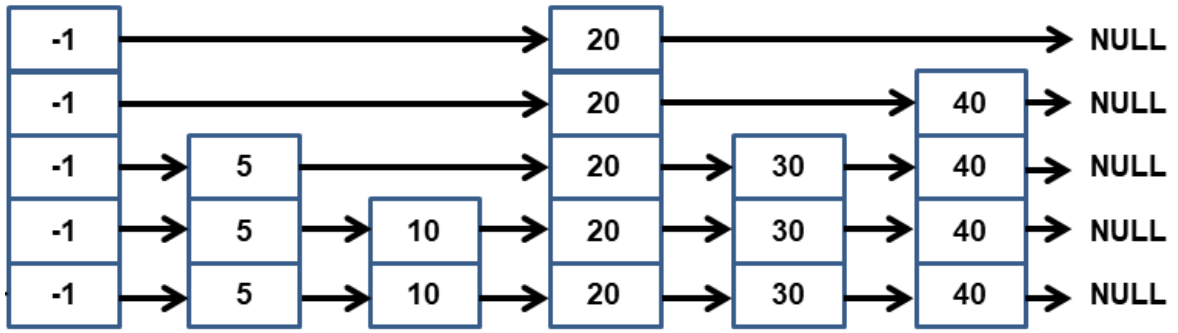
Skip List | Definição

- São chamadas de Skip Lists porque os níveis mais altos permitem pular elementos
  - Apenas o **nível mais baixo** contém todos os elementos
  - Numa Skip List perfeita, cada nó possui ½ dos nós do nível abaixo



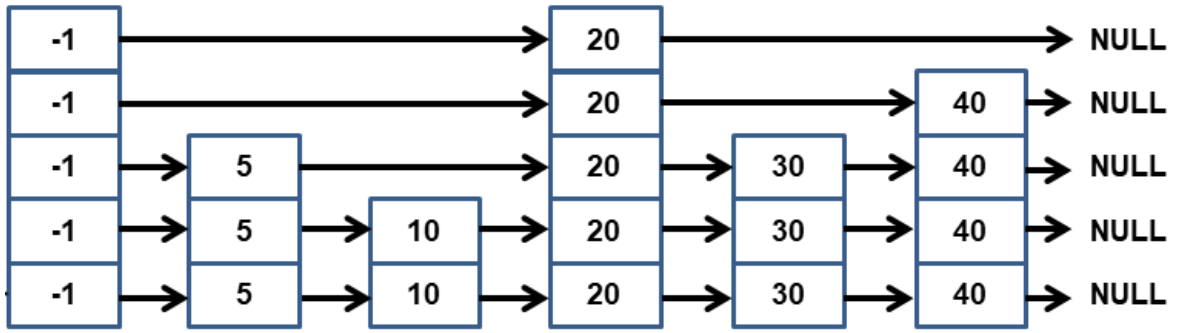
Skip List | Definição

- Skip List pode substituir uma árvore
  - Fácil Implementação
    - Não utiliza rotações e realocações de nós
  - Inserção/remoção/busca é feita em ***O(log N)***
    - Não depende da ordem de inserção dos nós
    - Não exige balancear a estrutura



Skip List | Definição

- Skip List pode substituir uma árvore
  - Próximo elemento pode ser recuperado em custo constante
  - Seu pior caso (raro) é ***O(N)***
    - A Skip List se transforma em uma lista dinâmica encadeada
    - Todos os nós estarão presentes em cada nível



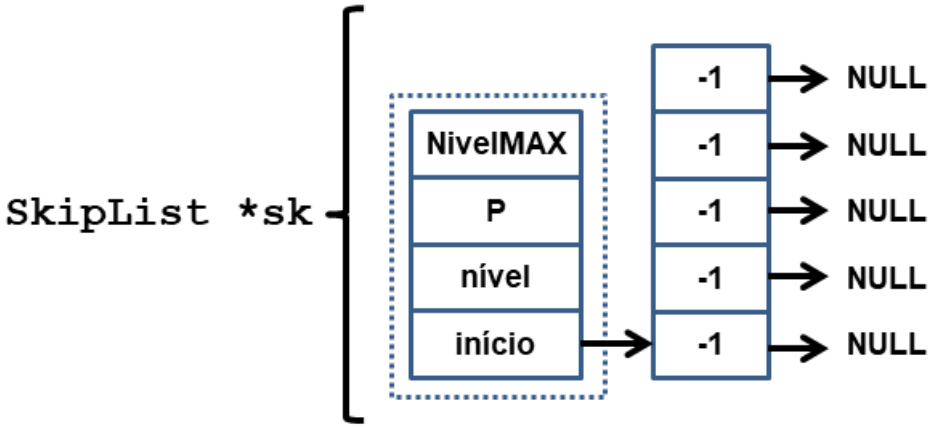
Skip List | Implementação

- Sua implementação utiliza uma estrutura similar a da Lista Dinâmica Encadeada
- Cada nó da lista possui um array de níveis
- É o array que faz as ligações entre os diferentes níveis da lista

- Desvantagem
  - Necessita que se defina previamente o número máximo de níveis da Skip List
  - Isso limita o número de níveis que podemos ter

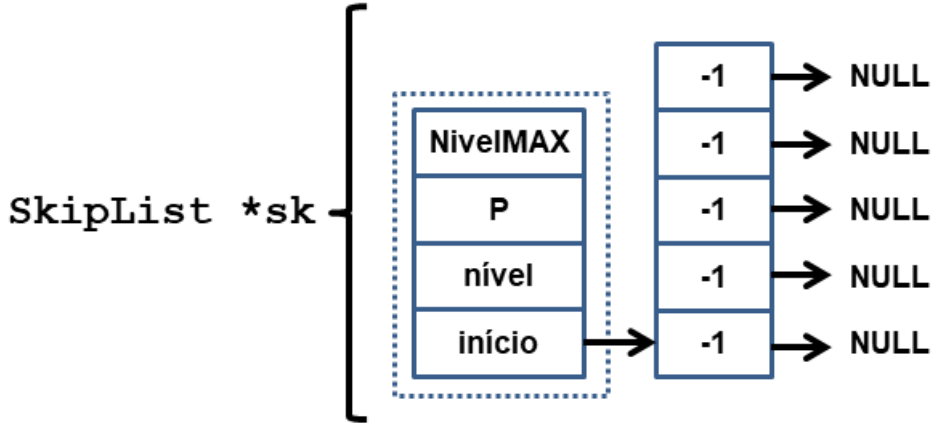
Skip List | Implementação

```c
// SkipList.h
typedef struct SkipList SkipList;

// SkipList.c
struct NO{
    int chave;
    struct NO **prox;
};

struct SkipList{
    // Nível máximo
    int NivelMAX;
    // Fração dos nós
    float P;
    // Nível atual do nó
    int nivel;
    // ponteiro para o nó cabeçalho
    struct NO *inicio;
};

// Programa Principal
SkipList* sk
```



Skip List | Implementação

- Importante
  - Por questões de desempenho, nossa **Skip List** irá armazenar a chave apenas na **struct NO**
  - Essa **struct** é quem mantém os ponteiros para os nós seguintes nos diferentes níveis



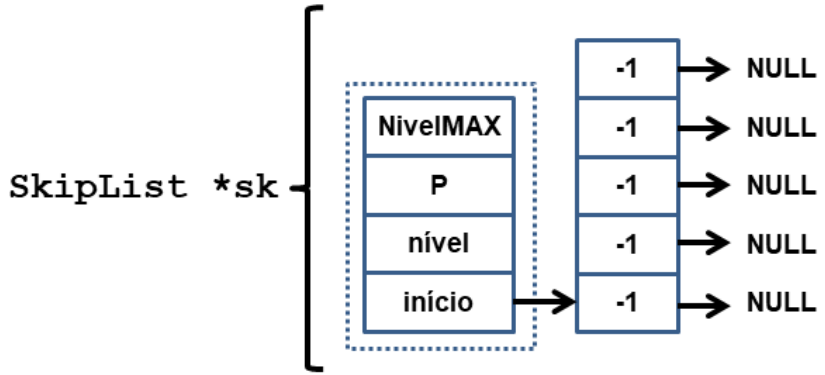
Skip List | Implementação

- Criando uma Skip List
  - Alocar espaço para o cabeçalho
  - Alocar espaço para o primeiro nó e inicializar todos os níveis com **NULL**
  - Primeiro nó define o menor valor a ser armazenado
    - -1 no nosso exemplo

Skip List | Implementação

- Criando uma Skip List

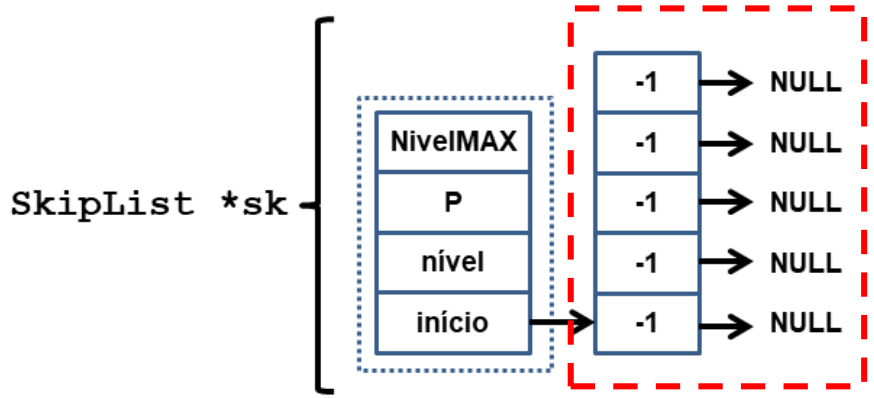
```
// Programa Principal
SkipList* sk = criaSkipList(3, 0.5);

// SkipList.h
SkipList* criaSkipList(int MAXLVL, float P);

// SkipList.c
SkipList* criaSkipList(int NivelMAX, float P){
    SkipList *sk = (SkipList*) malloc(sizeof(SkipList));
    if(sk != NULL){
        sk->NivelMAX = NivelMAX;
        sk->P = P;
        sk->nivel = 0;
        // cria o cabeçalho com chave -1
        // a SkipList armazena apenas valores positivos
        sk->inicio = novoNo(-1, NivelMAX);
    }

    return sk;
}
```



Skip List | Implementação

- Criando uma Skip List
  - Função responsável por criar um novo nó

```
struct NO* novoNo(int chave, int nivel){
    struct NO* novo = malloc(sizeof(struct NO));
    if(novo != NULL){
        // Cria um novo nó apontando para NULL
        novo->chave = chave;
        novo->prox = malloc((nivel+1)* sizeof(struct NO*));
        int i;
        for(i=0; i<(nivel+1); i++)
            novo->prox[i] = NULL;
    }
    return novo;
}
```



Skip List | Implementação

- Destruindo uma Skip List
  - Envolve percorrer todos os nós do nível 0
  - Para cada nó
    - Liberar o array de níveis
    - Liberar o próprio nó
  - Por fim, liberar o cabeçalho

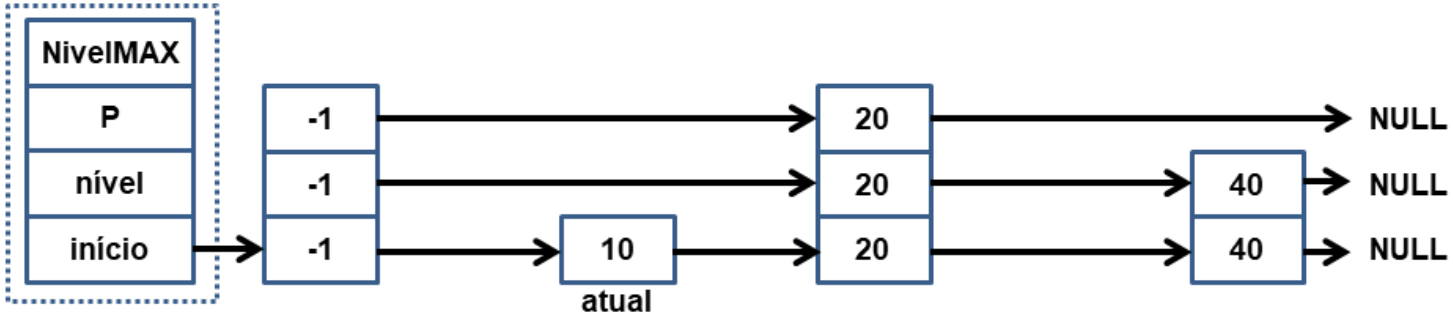
Skip List | Implementação

- Destruindo uma Skip List

```
// Programa Principal
liberaSkipList(sk);

// SkipList.h
void liberaSkipList(SkipList* sk);

// SkipList.c
void liberaSkipList(SkipList* sk){
    if(sk == NULL)
        return;

    struct NO *no, *atual;
    atual = sk->inicio->prox[0];
    while(atual != NULL){
        no = atual;
        atual = atual->prox[0];
        free(no->prox);
        free(no);
    }

    free(sk->inicio);
    free(sk);
}
```

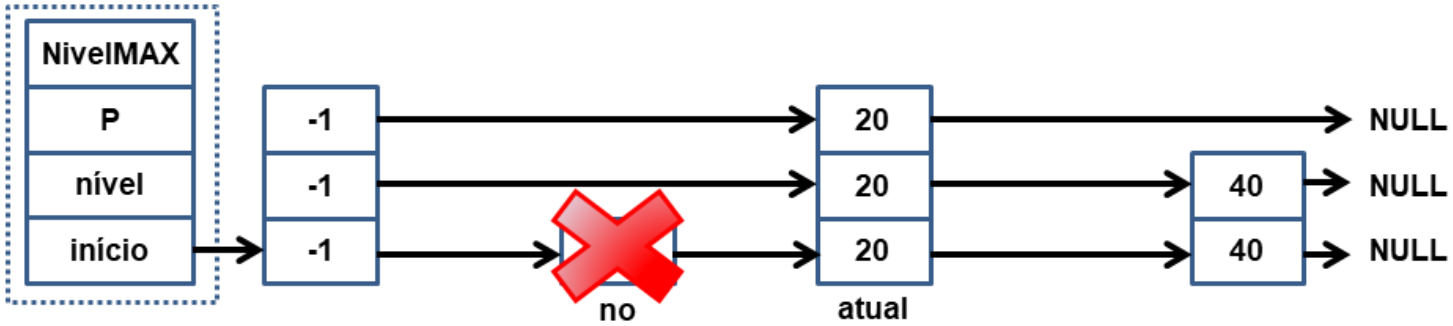
Skip List | Implementação

- Passo a passo: Destruindo uma Skip List



Skip List inicial:

```
atual = sk->inicio->prox[0];
```

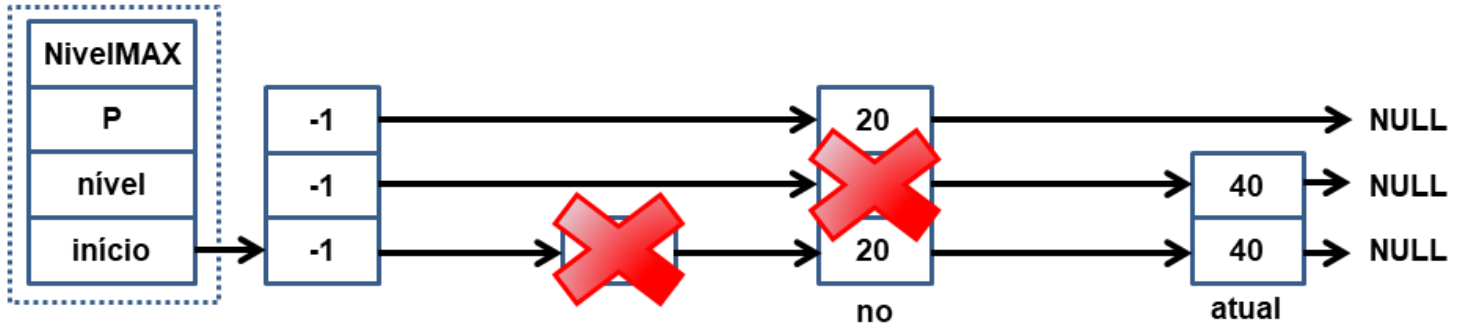


Passo 1:

```
no = atual;
atual = atual->prox[0];
free(no);
```

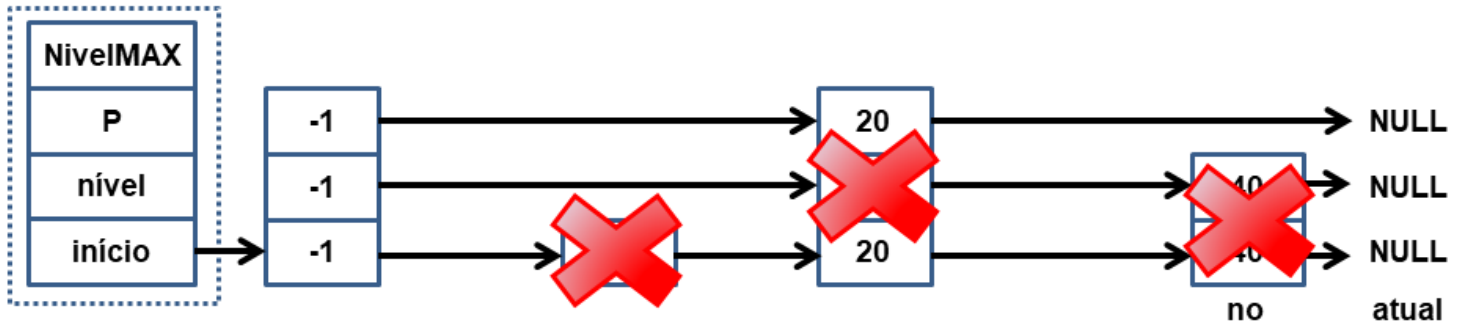
Skip List | Implementação

- Passo a passo: Destruindo uma Skip List



Passo 2:

```
no = atual;
atual = atual->prox[0];
free(no);
```

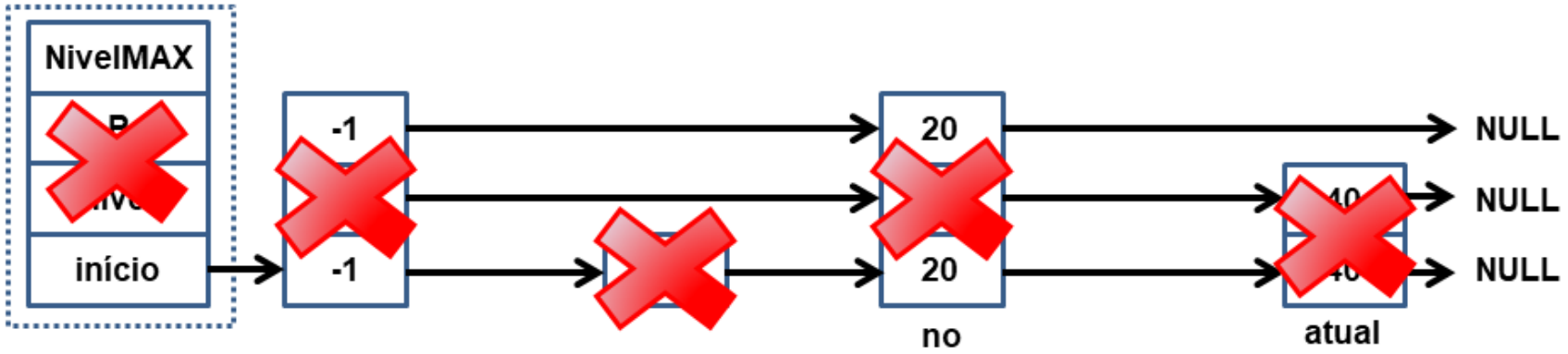


Passo 3:

```
no = atual;
atual = atual->prox[0];
free(no);
```

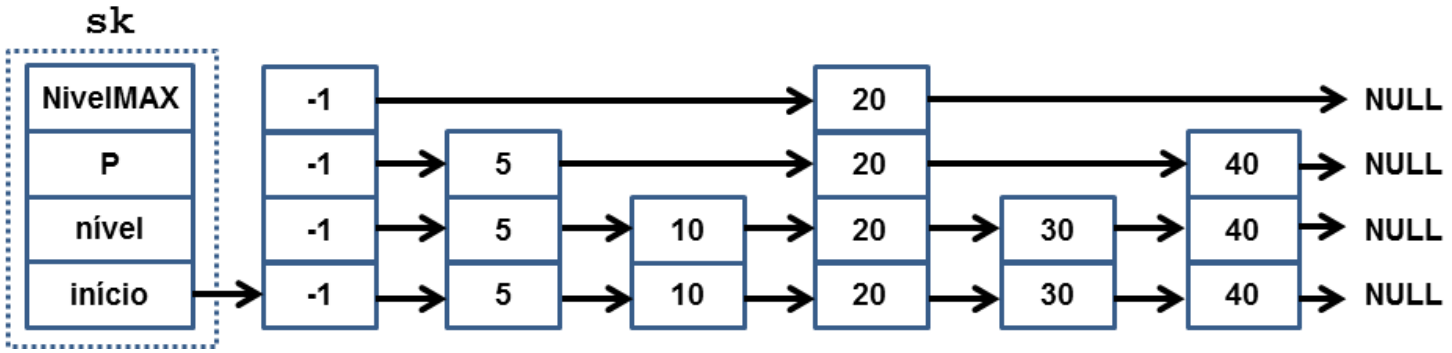
Skip List | Implementação

- Passo a passo: Destruindo uma Skip List



Skip List | Busca

- Para pesquisar um valor V em uma Skip List
  - Partindo do **nível mais alto**
    - Vá para o próximo **nó** se a chave procurada for maior do que a do próximo **nó**.
    - Caso contrário, desça um nível e continue a busca.
    - Esse processo continua até chegar ao nível ZERO.
  - Se a chave existir ela estará no próximo nó.



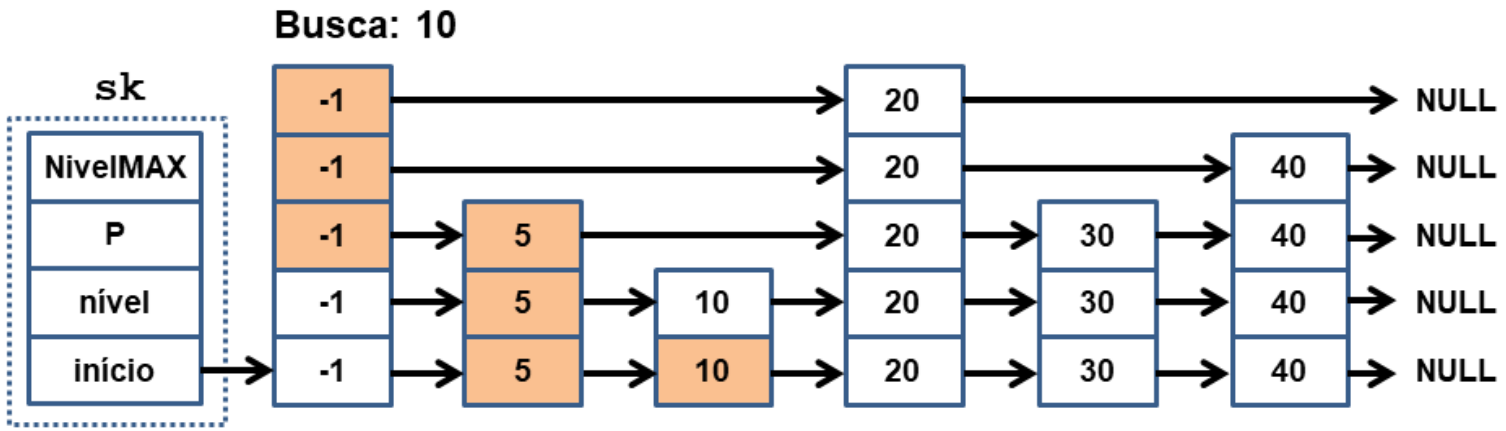
Skip List | Busca

- Buscando um valor na Skip List

```c
// SkipList.c
int buscaSkipList(SkipList *sk, int chave){
    if(sk == NULL) return 0;

    struct NO *atual = sk->inicio;
    // Partindo do maior nível, vá para o próximo nó
    // enquanto a chave for maior do que a do próximo nó
    // Caso contrário, desca um nível e continue a busca
    int i;
    for(i = sk->nivel; i >= 0; i--){
        while(atual->prox[i] != NULL &&
              atual->prox[i]->chave < chave)
            atual = atual->prox[i];
    }
    // Acesse o nível 0 do próximo nó, que é
    // onde a chave procurada deve estar
    atual = atual->prox[0];
    if(atual != NULL && atual->chave == chave)
        return 1;
    else
        return 0;
}
```

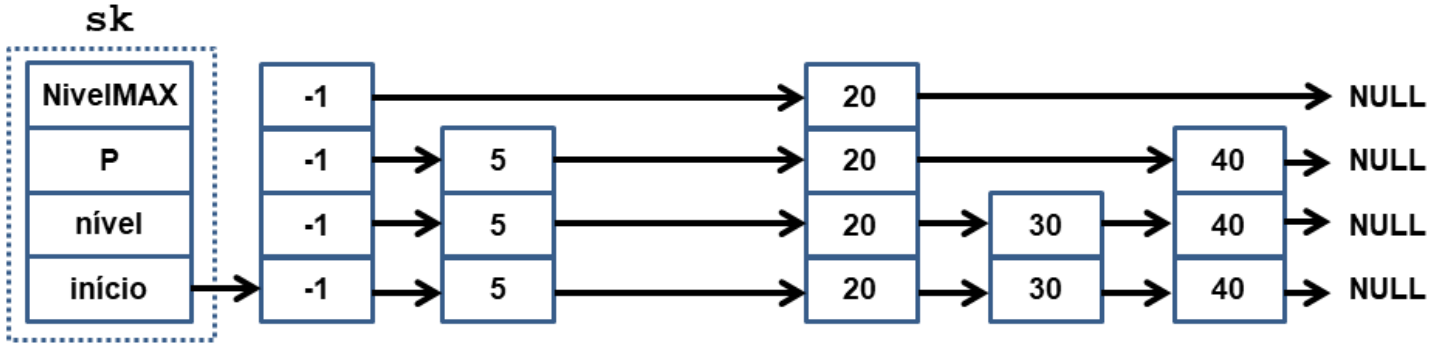
Skip List | Busca

- Buscando um valor na Skip List



Skip List | Inserção

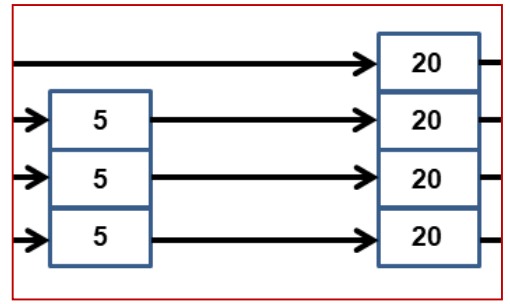
- Para inserir um valor V em uma Skip List
  - Procurar a posição de inserção em cada nível da Skip List e armazenar em um array auxiliar.
  - Alocar espaço para o novo nó e sortear quantos níveis ele terá.
  - Se ele tiver mais níveis que a Skip List, atualizar os níveis do array auxiliar.
  - Fazer a ligação entre o array auxiliar e o novo nó, similar a lista dinâmica encadeada.

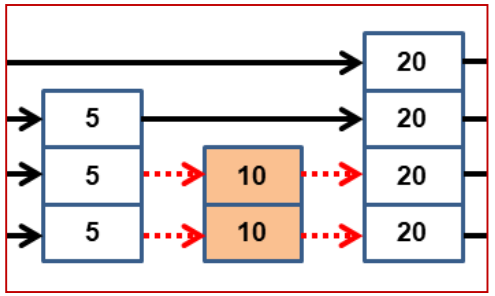


Skip List | Inserção

- Inserindo um valor na Skip List

```
// Programa Principal
int x = insereSkipList(sk, 10);
// SkipList.h
int insereSkipList(SkipList *sk, int key);
// SkipList.c
int insereSkipList(SkipList *sk, int chave){
    if(sk == NULL) return 0;

    int i;
    struct NO *atual = sk->inicio;
    // Cria um array de nós auxiliar apontando para NULL
    struct NO **aux;
    aux = malloc((sk->NivelMAX+1) * sizeof(struct NO*));
    for(i = 0; i <= sk->NivelMAX; i++)
        aux[i] = NULL;

    // Partindo do maior nível, vá para o próximo nó
    // enquanto a chave for maior do que a do próximo nó
    // Caso contrário, insira o nó no array auxiliar,
    // desca um nível e continue a busca
    for(i = sk->nivel; i >= 0; i--){
        while(atual->prox[i] != NULL &&
              atual->prox[i]->chave < chave)
            atual = atual->prox[i];
        aux[i] = atual;
    }
    //CONTINUA...
```



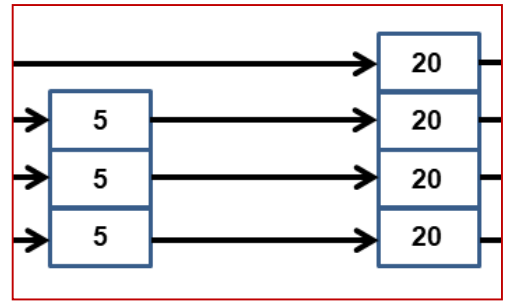


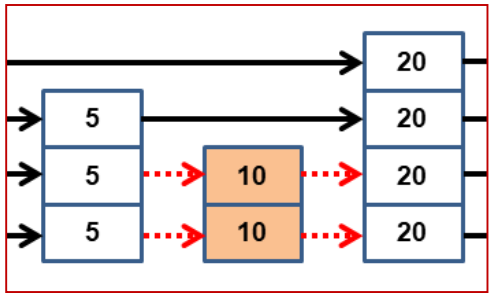
aux	
aux[3]	-1
aux[2]	5
aux[1]	5
aux[0]	5

Skip List | Inserção

- Inserindo um valor na Skip List

```
// Acesse o nível 0 do próximo nó, que é
// onde a chave deve ser inserida
atual = atual->prox[0];
// Cria e insere um novo nó se a chave não existir
// Final da lista (atual == NULL) ou
// entre auxiliar[0] e atual
if(atual == NULL || atual->chave != chave){
    // Sorteia o nível
    int novo_nivel = sorteiaNivel(sk);
    // Cria um novo nó apontando para NULL
    struct NO* novo = novoNo(chave, novo_nivel);
    if(novo == NULL){
        free(aux);
        return 0;
    }
    // Se o nível sorteado for maior do que o nível
    // atual da SkipList, atualizar os novos níveis
    // do array auxiliar.
    if(novo_nivel > sk->nivel){
        for(i = sk->nivel+1; i <= novo_nivel; i++)
            aux[i] = sk->inicio;
        // Atualiza o nível da SkipList
        sk->nivel = novo_nivel;
    }
    // Insere o nó, arrumando os ponteiros
    for(i = 0; i <= novo_nivel; i++){
        novo->prox[i] = aux[i]->prox[i];
        aux[i]->prox[i] = novo;
    }
    free(aux);
    return 1;
}
```





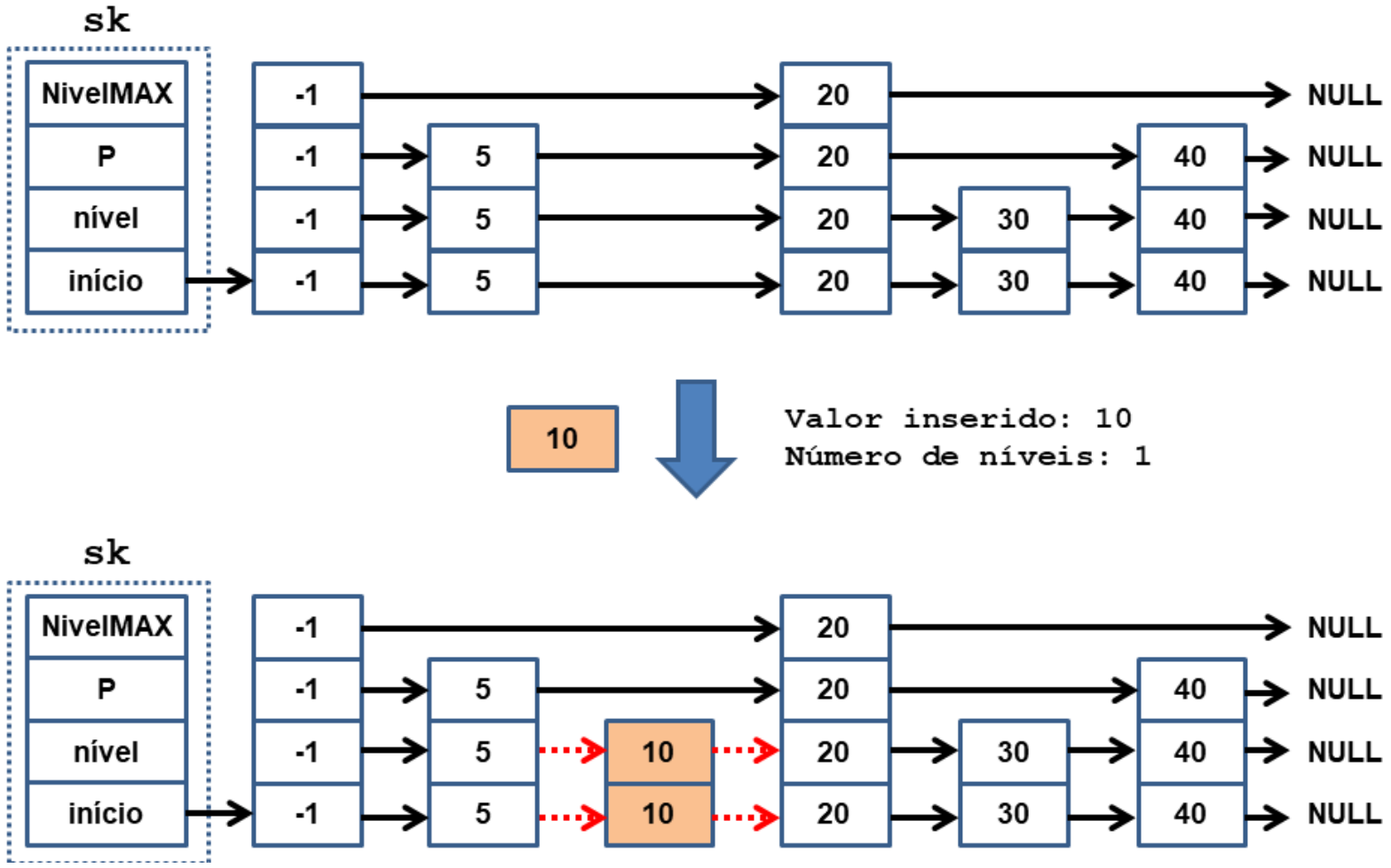
aux	
aux[3]	-1
aux[2]	5
aux[1]	5
aux[0]	5

Skip List | Inserção

- Inserindo um valor na Skip List
  - Funções auxiliares

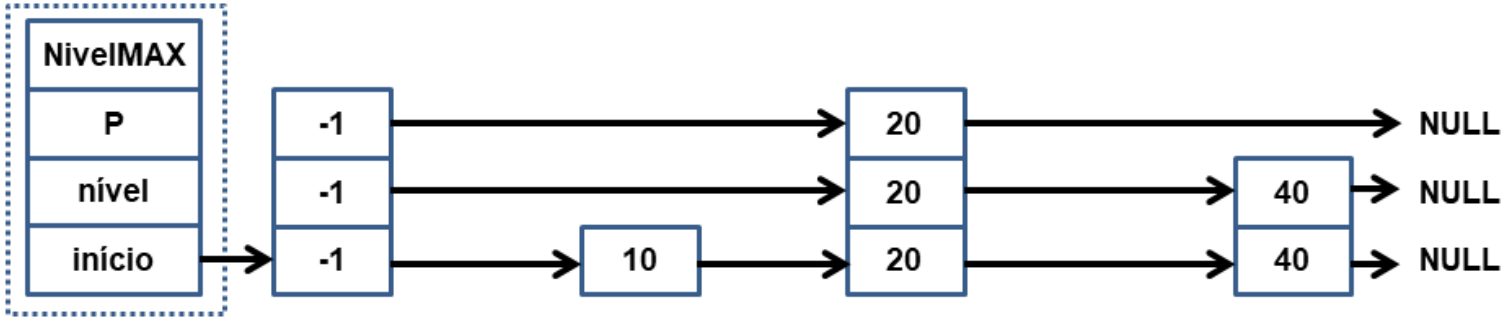
```c
int sorteiaNivel(SkipList *sk){
    // Sorteia o nível para o nó
    float r = (float)rand()/RAND_MAX;
    int nivel = 0;
    while(r < sk->P && nivel < sk->NivelMAX){
        nivel++;
        r = (float)rand()/RAND_MAX;
    }
    return nivel;
}

struct NO* novoNo(int chave, int nivel){
    struct NO* novo = malloc(sizeof(struct NO));
    if(novo != NULL){
        // Cria um novo nó apontando para NULL
        novo->chave = chave;
        novo->prox = malloc((nivel+1)* sizeof(struct NO*));
        int i;
        for(i=0; i<(nivel+1); i++)
            novo->prox[i] = NULL;
    }
    return novo;
}
```

Skip List | Inserção passo a passo



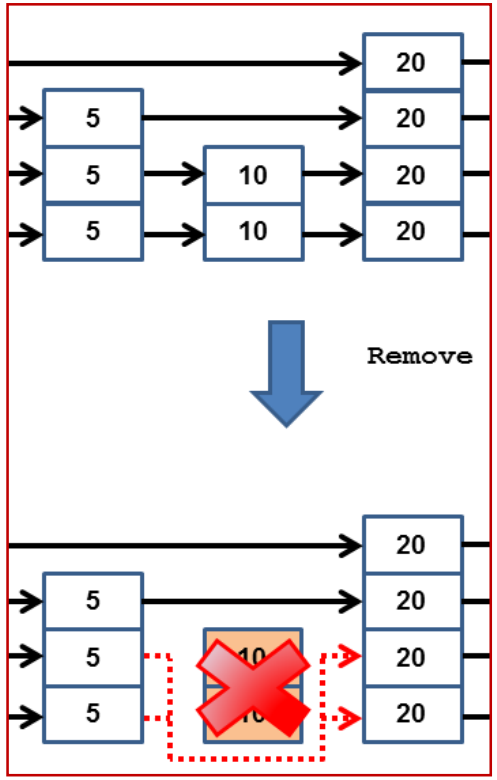
Skip List | Remoção

- Para remover um valor V de uma Skip List
  - Procurar a posição do valor em cada nível da Skip List e armazenar em um array auxiliar.
  - Começando no nível ZERO
    - Se o array auxiliar aponta para o nó a ser removido, faça ele apontar para o próximo nó
    - Similar a remoção de lista dinâmica encadeada.
  - Remova os níveis sem elemento



Skip List | Remoção

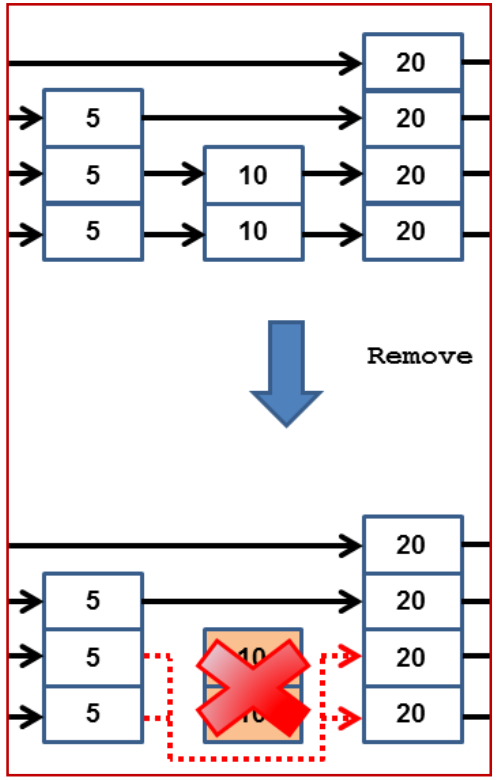
```
// Programa Principal
int x = removeSkipList(sk, 15);
// SkipList.h
int removeSkipList(SkipList *sk, int key);
// SkipList.c
int removeSkipList(SkipList *sk, int chave){
    if(sk == NULL) return 0;

    int i;
    struct NO *atual = sk->inicio;
    // Cria um array de nós auxiliar apontando para NULL
    struct NO **aux;
    aux = malloc((sk->NivelMAX+1) * sizeof(struct NO*));
    for(i = 0; i <= sk->NivelMAX; i++)
        aux[i] = NULL;

    // Partindo do maior nível, vá para o próximo nó
    // enquanto a chave for maior do que a do próximo nó
    // Caso contrário, insira o nó no array auxiliar,
    // desca um nível e continue a busca
    for(i = sk->nivel; i >= 0; i--){
        while(atual->prox[i] != NULL &&
              atual->prox[i]->chave < chave)
            atual = atual->prox[i];
        aux[i] = atual;
    }
    //CONTINUA...
```



aux	
aux[3]	-1
aux[2]	5
aux[1]	5
aux[0]	5

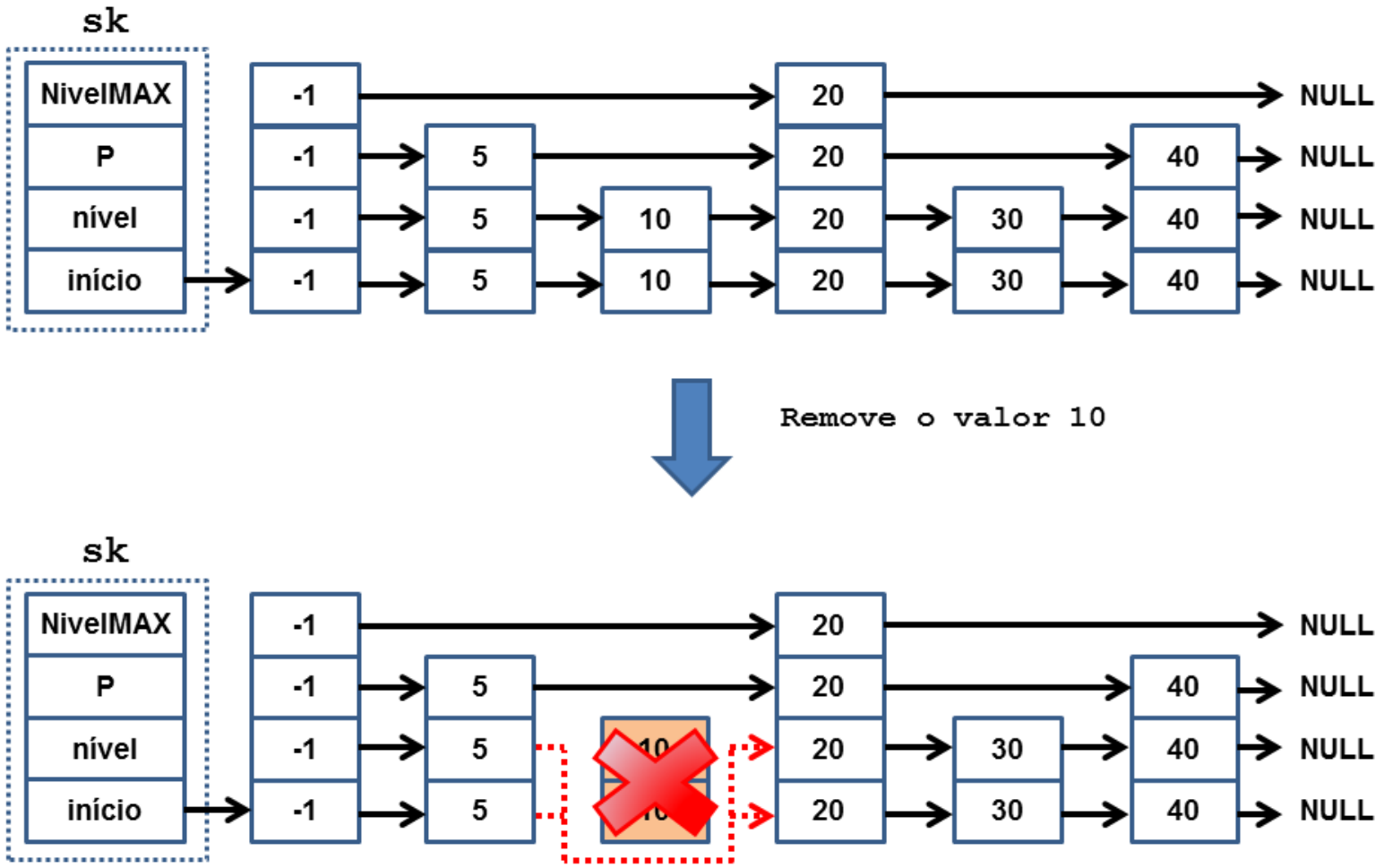
Skip List | Remoção

```
    // Acesse o nível 0 do próximo nó, que é
    // onde a chave a ser removida deve estar
    atual = atual->prox[0];

    // Achou a chave a ser removida?
    if(atual != NULL && atual->chave == chave){
        // Começando no nível 0, se o array auxiliar
        // aponta para o nó a ser removido, faça ele
        // apontar para o próximo nó (remoção de lista encadeada)
        for(i = 0; i <= sk->nivel; i++){
            if(aux[i]->prox[i] != atual)
                break;

            aux[i]->prox[i] = atual->prox[i];
        }
        // Remova os níveis sem elemento
        while(sk->nivel > 0 &&
              sk->inicio->prox[sk->nivel] == NULL)
            sk->nivel--;

        free(atual->prox);
        free(atual);
        free(aux);
        return 1;
    }

    free(aux);
    return 0;
}
```

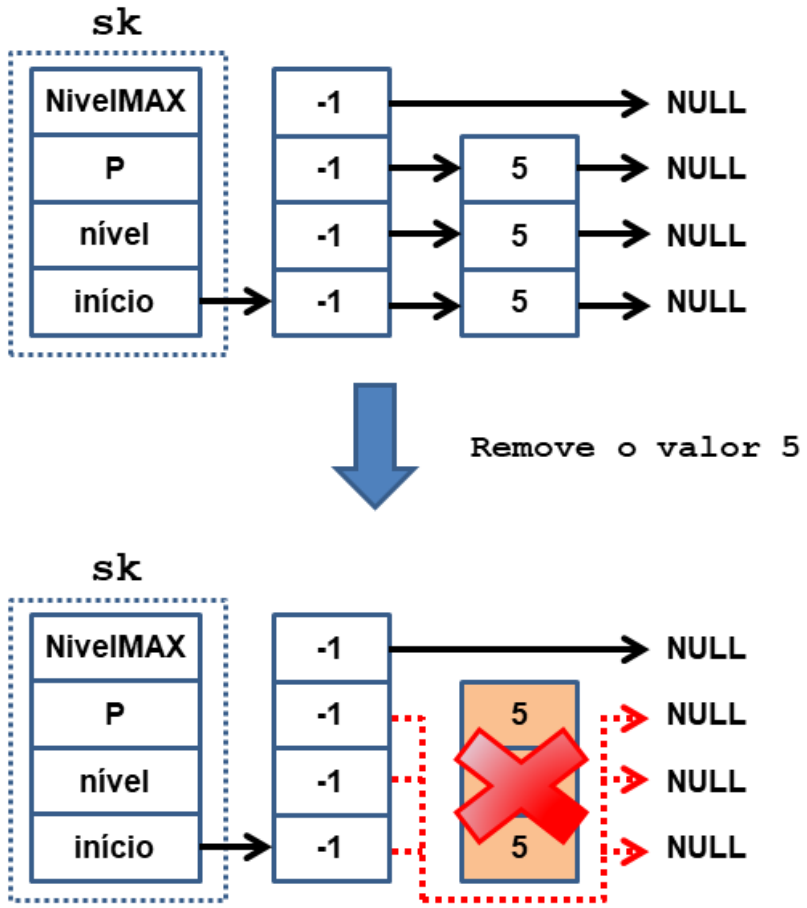


aux	
aux[3]	-1
aux[2]	5
aux[1]	5
aux[0]	5

Skip List | Remoção passo a passo



Skip List | Remoção passo a passo



Material Complementar | Vídeo Aulas

- [ED] Aula 125 - SkipList: Definição
  - https://youtu.be/rHWO_dFMSOM
- [ED] Aula 126 - SkipList: Implementação
  - https://youtu.be/WvcoyI5FIhA
- [ED] Aula 127 - SkipList: Busca
  - https://youtu.be/NEDfJCNisC8
- [ED] Aula 128 - SkipList: Inserção
  - https://youtu.be/RVrNUHe6pSA
- [ED] Aula 129 - SkipList: Remoção
  - https://youtu.be/D7RiVcZOje0

Material Complementar | GitHub

- https://github.com/arbackes