AULA e04
Algoritmos e Estruturas de Dados I

Múltiplas Pilhas - implementação estática

Luciano Antonio Digiampietri

Já vimos como gerenciar duas pilhas em um único arranjo

Já vimos como gerenciar duas pilhas em um único arranjo

E se precisarmos organizar nossas informações em mais do que dois grupos?

Gerenciaremos múltiplas pilhas dentro de um único arranjo

Gerenciaremos múltiplas pilhas dentro de um único arranjo

Precisaremos controlar o índice da base e do topo de cada uma das pilhas

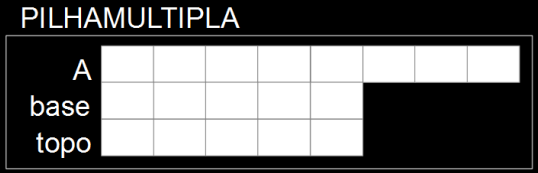
Estrutura PILHAMULTIPLA

Estrutura PILHAMULTIPLA

```
#define true 1
#define false 0
#define ERRO -1
#define MAX 8
#define NP 4

typedef int TIPOCHAVE;

typedef int bool;

typedef struct {
  TIPOCHAVE A[MAX];
  int base[NP+1];
  int topo[NP+1];
} PILHAMULTIPLA;
```

Estrutura PILHAMULTIPLA

```c
#define true 1
#define false 0
#define ERRO -1
#define MAX 8
#define NP 4

typedef int TIPOCHAVE;

typedef int bool;

typedef struct {
  TIPOCHAVE A[MAX];
  int base[NP+1];
  int topo[NP+1];
} PILHAMULTIPLA;
```



Inicialização da estrutura

Inicialização da estrutura

Para a inicialização precisaremos acertar os valores de todas as *variáveis de controle*

Inicialização da estrutura

Para a inicialização precisaremos acertar os valores de todas as *variáveis de controle*

Inicialmente, dividiremos o espaço de maneira igual entre todas as pilhas

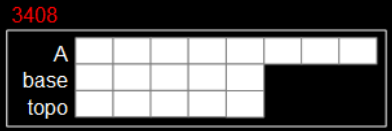
Inicialização da estrutura

Para a inicialização precisaremos acertar os valores de todas as *variáveis de controle*

Inicialmente, dividiremos o espaço de maneira igual entre todas as pilhas

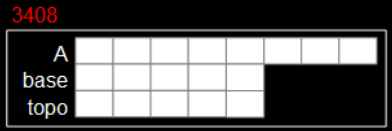
Assim como em nossas outras implementações estáticas de pilhas, o valor do topo iniciará anterior à base

Inicialização da estrutura



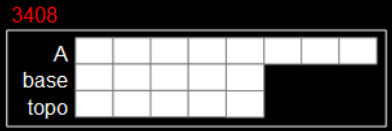
Inicialização da estrutura

```
void inicializarPilhaMultipla(PILHAMULTIPLA* p){



}
```



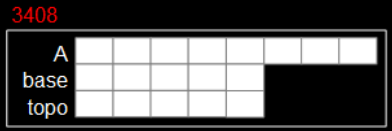
Inicialização da estrutura

```
void inicializarPilhaMultipla(PILHAMULTIPLA* p){
  int i;
  for(i = 0; i <= NP ; i++){


  }
}
```



Inicialização da estrutura

```
void inicializarPilhaMultipla(PILHAMULTIPLA* p){
  int i;
  for(i = 0; i <= NP ; i++){
    p->base[i] = (i * (MAX / NP));
    p->topo[i] = p->base[i] - 1;
  }
}
```



Inicialização da estrutura

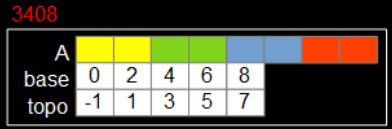
```
void inicializarPilhaMultipla(PILHAMULTIPLA* p){
  int i;
  for(i = 0; i <= NP ; i++){
    p->base[i] = (i * (MAX / NP));
    p->topo[i] = p->base[i] - 1;
  }
}
```

3408

A							
base	0	2	4	6	8		
topo	-1	1	3	5	7		

Inicialização da estrutura

```
void inicializarPilhaMultipla(PILHAMULTIPLA* p){
  int i;
  for(i = 0; i <= NP ; i++){
    p->base[i] = (i * (MAX / NP));
    p->topo[i] = p->base[i] - 1;
  }
}
```



Número de elementos de uma pilha

Número de elementos de uma pilha

```
int tamanhoPilhaK(PILHAMULTIPLA* p, int k) {


}
```

Número de elementos de uma pilha

```
int tamanhoPilhaK(PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return ERRO;
  return p->topo[k] - p->base[k] + 1;
}
```

3408

A								
base	0	2	4	6	8			
topo	-1	1	3	5	7			

Número total de elementos

Número total de elementos

```
int tamanhoTotalPilhas(PILHAMULTIPLA* p) {
  int tamanho = 0;


  return tamanho;
}
```

Número total de elementos

```
int tamanhoTotalPilhas(PILHAMULTIPLA* p) {
  int tamanho = 0;
  int i;
  for (i=0;i<NP;i++) tamanho += tamanhoPilhaK(p,i);
  return tamanho;
}
```

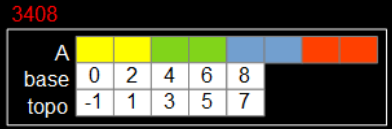
Verificando se uma pilha está *cheia*

Verificando se uma pilha está *cheia*

```
bool pilhaKcheia(PILHAMULTIPLA* p, int k) {


}
```

Verificando se uma pilha está *cheia*

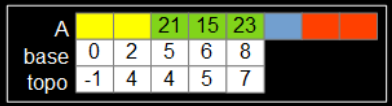
```
bool pilhaKcheia(PILHAMULTIPLA* p, int k) {
  if(p->topo[k] == p->base[k + 1] - 1) return true;
  else return false;
}
```



Verificando se uma pilha está *cheia*

```
bool pilhaKcheia(PILHAMULTIPLA* p, int k) {
  if(p->topo[k] == p->base[k + 1] - 1) return true;
  else return false;
}
```

3408

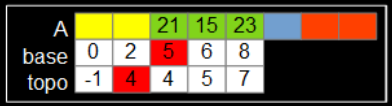


Para k = 1

Verificando se uma pilha está *cheia*

```
bool pilhaKcheia(PILHAMULTIPLA* p, int k) {
  if(p->topo[k] == p->base[k + 1] - 1) return true;
  else return false;
}
```

3408



Para k = 1

Exclusão de um elemento

Exclusão de um elemento

O usuário indicará de qual pilha deseja excluir

Exclusão de um elemento

O usuário indicará de qual pilha deseja excluir

Se a pilha não estiver vazia, o elemento do topo será copiado para o endereço de memória passado pelo usuário e então *excluído* da estrutura

Exclusão

Exclusão

```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){




  return false;
}
```

Exclusão

```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){
  if (k<0 || k>=NP) return false;




  return false;
}
```

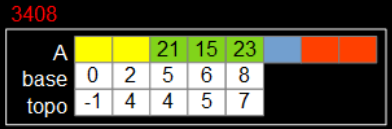
Exclusão

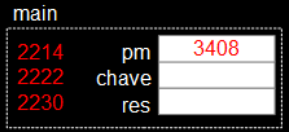
```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){
  if (k<0 || k>=NP) return false;
  if(p->topo[k] >= p->base[k]) {



  }
  return false;
}
```

Exclusão

```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){
  if (k<0 || k>=NP) return false;
  if(p->topo[k] >= p->base[k]) {
    *ch = p->A[p->topo[k]];
    p->topo[k]--;
    return true;
  }
  return false;
}
```

Exclusão

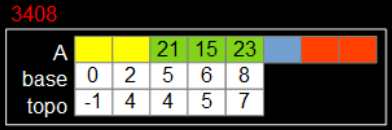





```
int main(){
  PILHAMULTIPLA* pm;
     ...
  TIPOCHAVE chave;
  bool res
```

Exclusão

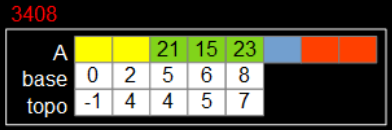
```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){




  return false;
}
```




```
int main(){
  PILHAMULTIPLA* pm;
     ...
  TIPOCHAVE chave;
  bool res = popK(pm, &chave, 1);
```

Exclusão

```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){
  if (k<0 || k>=NP) return false;




  return false;
}
```



popK

4400	p	3408
4408	ch	2222
4416	k	1

main

2214	pm	3408
2222	chave	
2230	res	

```
int main(){
  PILHAMULTIPLA* pm;
     ...
  TIPOCHAVE chave;
  bool res = popK(pm, &chave, 1);
```

Exclusão

```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){
  if (k<0 || k>=NP) return false;
  if(p->topo[k] >= p->base[k]) {



  }
  return false;
}
```

3408

A			21	15	23			
base	0	2	5	6	8			
topo	-1	4	4	5	7			

popK

4400	p	3408
4408	ch	2222
4416	k	1

main

2214	pm	3408
2222	chave	
2230	res	

```
int main(){
  PILHAMULTIPLA* pm;
     ...
  TIPOCHAVE chave;
  bool res = popK(pm, &chave, 1);
```

Exclusão

```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){
  if (k<0 || k>=NP) return false;
  if(p->topo[k] >= p->base[k]) {
    *ch = p->A[p->topo[k]];


  }
  return false;
}
```

3408

A			21	15	23			
base	0	2	5	6	8			
topo	-1	4	4	5	7			

popK

4400	p	3408
4408	ch	2222
4416	k	1

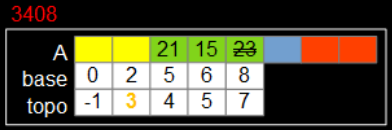
main

2214	pm	3408
2222	chave	23
2230	res	

```
int main(){
  PILHAMULTIPLA* pm;
     ...
  TIPOCHAVE chave;
  bool res = popK(pm, &chave, 1);
```

Exclusão

```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){
  if (k<0 || k>=NP) return false;
  if(p->topo[k] >= p->base[k]) {
    *ch = p->A[p->topo[k]];
    p->topo[k]--;

  }
  return false;
}
```




```
int main(){
  PILHAMULTIPLA* pm;
     ...
  TIPOCHAVE chave;
  bool res = popK(pm, &chave, 1);
```

Exclusão

```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){
  if (k<0 || k>=NP) return false;
  if(p->topo[k] >= p->base[k]) {
    *ch = p->A[p->topo[k]];
    p->topo[k]--;
    return true;
  }
  return false;
}
```

3408

A			21	15	~~23~~			
base	0	2	5	6	8			
topo	-1	3	4	5	7			

popK

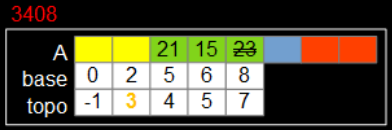
4400	p	3408
4408	ch	2222
4416	k	1

main

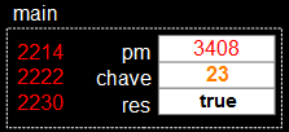
2214	pm	3408
2222	chave	23
2230	res	

```
int main(){
  PILHAMULTIPLA* pm;
     ...
  TIPOCHAVE chave;
  bool res = popK(pm, &chave, 1);
```

Exclusão




```
bool popK(PILHAMULTIPLA* p, TIPOCHAVE* ch, int k){
  if (k<0 || k>=NP) return false;
  if(p->topo[k] >= p->base[k]) {
    *ch = p->A[p->topo[k]];
    p->topo[k]--;
    return true;
  }
  return false;
}
```




```
int main(){
  PILHAMULTIPLA* pm;
     ...
  TIPOCHAVE chave;
  bool res = popK(pm, &chave, 1);
```

Inserção

Inserção

A inserção ocorre *no topo* de uma pilha (após o elemento que está no topo)

Inserção

A inserção ocorre *no topo* de uma pilha (após o elemento que está no topo)

Porém, pode ser que a pilha atual já esteja *cheia*

Inserção

A inserção ocorre *no topo* de uma pilha (após o elemento que está no topo)

Porém, pode ser que a pilha atual já esteja *cheia*

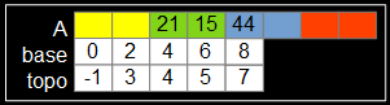
Neste caso procuraremos espaços livre à direita ou à esquerda da pilha atual (**deslocando as demais pilhas** para liberação do espaço)

Deslocando uma pilha para a direita

```
bool paraDireita(PILHAMULTIPLA* p, int k) {




  return false;
}
```

Para k = 2

3408

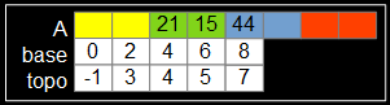


Deslocando uma pilha para a direita

```
bool paraDireita(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;




  return false;
}
```

Para k = 2

3408

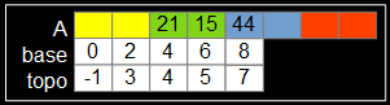


Deslocando uma pilha para a direita

```
bool paraDireita(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k] < p->base[k + 1] - 1){









  }
  return false;
}
```

Para k = 2

3408

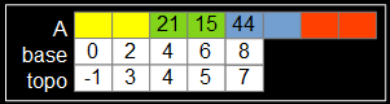


Deslocando uma pilha para a direita

```
bool paraDireita(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k] < p->base[k + 1] - 1){
    for(i = p->topo[k]; i >= p->base[k]; i--)
            p->A[i+1] = p->A[i];



  }
  return false;
}
```

Para k = 2

3408



Deslocando uma pilha para a direita

```
bool paraDireita(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k] < p->base[k + 1] - 1){
    for(i = p->topo[k]; i >= p->base[k]; i--)
            p->A[i+1] = p->A[i];



  }
  return false;
}
```

Para k = 2

3408

A			21	15	44	**44**		
base	0	2	4	6	8			
topo	-1	3	4	5	7			

Deslocando uma pilha para a direita

```
bool paraDireita(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k] < p->base[k + 1] - 1){
    for(i = p->topo[k]; i >= p->base[k]; i--)
            p->A[i+1] = p->A[i];
    p->topo[k]++;
    p->base[k]++;
    return true;
  }
  return false;
}
```

Para k = 2

3408

A			21	15	44	**44**		
base	0	2	4	6	8			
topo	-1	3	4	5	7			

Deslocando uma pilha para a direita

```
bool paraDireita(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k] < p->base[k + 1] - 1){
    for(i = p->topo[k]; i >= p->base[k]; i--)
            p->A[i+1] = p->A[i];
    p->topo[k]++;
    p->base[k]++;
    return true;
  }
  return false;
}
```

Para k = 2

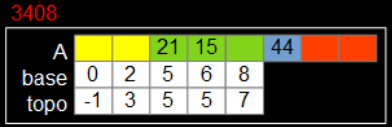
3408

A			21	15	44	44		
base	0	2	**5**	6	8			
topo	-1	3	**5**	5	7			

Deslocando uma pilha para a esquerda

```
bool paraEsquerda(PILHAMULTIPLA* p, int k) {




  return false;
}
```

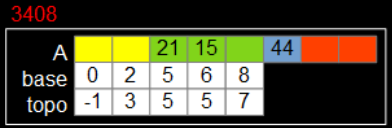
Para k = 1



Deslocando uma pilha para a esquerda

```
bool paraEsquerda(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;




  return false;
}
```

Para k = 1

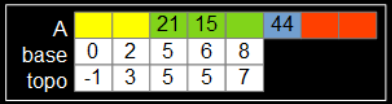


Deslocando uma pilha para a esquerda

```
bool paraEsquerda(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k-1] < p->base[k] - 1){









  }
  return false;
}
```

Para k = 1

3408

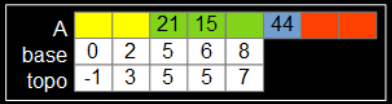


Deslocando uma pilha para a esquerda

```
bool paraEsquerda(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k-1] < p->base[k] - 1){
    for(i = p->base[k]; i <= p->topo[k]; i++)
            p->A[i-1] = p->A[i];




  }
  return false;
}
```

Para k = 1

3408

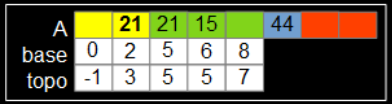


Deslocando uma pilha para a esquerda

```
bool paraEsquerda(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k-1] < p->base[k] - 1){
    for(i = p->base[k]; i <= p->topo[k]; i++)
            p->A[i-1] = p->A[i];



  }
  return false;
}
```

Para k = 1

3408



Deslocando uma pilha para a esquerda

```
bool paraEsquerda(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k-1] < p->base[k] - 1){
    for(i = p->base[k]; i <= p->topo[k]; i++)
            p->A[i-1] = p->A[i];




  }
  return false;
}
```

Para k = 1

3408

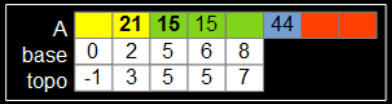
A		21	15	15		44		
base	0	2	5	6	8			
topo	-1	3	5	5	7			

Deslocando uma pilha para a esquerda

```
bool paraEsquerda(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k-1] < p->base[k] - 1){
    for(i = p->base[k]; i <= p->topo[k]; i++)
            p->A[i-1] = p->A[i];
    p->topo[k]--;
    p->base[k]--;
    return true;
  }
  return false;
}
```

Para k = 1

3408

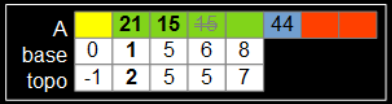


Deslocando uma pilha para a esquerda

```
bool paraEsquerda(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(p->topo[k-1] < p->base[k] - 1){
    for(i = p->base[k]; i <= p->topo[k]; i++)
            p->A[i-1] = p->A[i];
    p->topo[k]--;
    p->base[k]--;
    return true;
  }
  return false;
}
```

Para k = 1

3408



Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {




}
```

Para k = 1 e ch = 11

3408

A			21	15	23	44		
base	0	2	5	6	8			
topo	-1	4	5	5	7			

Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;









}
```

Para k = 1 e ch = 11

3408

A			21	15	23	44		
base	0	2	5	6	8			
topo	-1	4	5	5	7			

Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  int j;
  if (pilhaKcheia(p,k)){









  }



}
```

Para k = 1 e ch = 11

3408

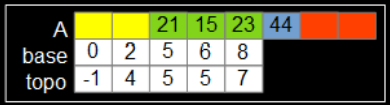
A			21	15	23	44		
base	0	2	5	6	8			
topo	-1	4	5	5	7			

Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  int j;
  if (pilhaKcheia(p,k)){
    for(j = NP-1; j > k; j--) paraDireita(p, j);




  }



}
```

Para k = 1 e ch = 11

3408

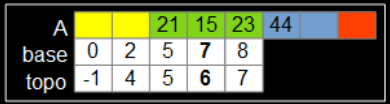


Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  int j;
  if (pilhaKcheia(p,k)){
    for(j = NP-1; j > k; j--) paraDireita(p, j);




  }



}
```

Para k = 1 e ch = 11

3408

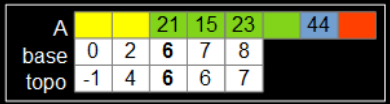


Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  int j;
  if (pilhaKcheia(p,k)){
    for(j = NP-1; j > k; j--) paraDireita(p, j);




  }



}
```

Para k = 1 e ch = 11

3408

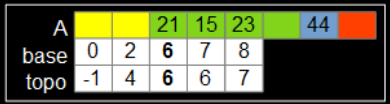


Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  int j;
  if (pilhaKcheia(p,k)){
    for(j = NP-1; j > k; j--) paraDireita(p, j);
    if(pilhaKcheia(p, k)){
        for( j = 1; j <= k; j++) paraEsquerda(p, j);

    }
  }



}
```

Para k = 1 e ch = 11

3408



Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  int j;
  if (pilhaKcheia(p,k)){
    for(j = NP-1; j > k; j--) paraDireita(p, j);
    if(pilhaKcheia(p, k)){
        for( j = 1; j <= k; j++) paraEsquerda(p, j);
        if(pilhaKcheia(p, k)) return false;
    }
  }


}
```

Para k = 1 e ch = 11

3408

A			21	15	23		44
base	0	2	**6**	7	8		
topo	-1	4	**6**	6	7		

Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  int j;
  if (pilhaKcheia(p,k)){
    for(j = NP-1; j > k; j--) paraDireita(p, j);
    if(pilhaKcheia(p, k)){
        for( j = 1; j <= k; j++) paraEsquerda(p, j);
        if(pilhaKcheia(p, k)) return false;
    }
  }
  p->topo[k]++;
  p->A[p->topo[k]] = ch;
  return true;
}
```

Para k = 1 e ch = 11

3408

A			21	15	23		44
base	0	2	**6**	7	8		
topo	-1	4	**6**	6	7		

Inserção

```
bool pushK(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  int j;
  if (pilhaKcheia(p,k)){
    for(j = NP-1; j > k; j--) paraDireita(p, j);
    if(pilhaKcheia(p, k)){
        for( j = 1; j <= k; j++) paraEsquerda(p, j);
        if(pilhaKcheia(p, k)) return false;
    }
  }
  p->topo[k]++;
  p->A[p->topo[k]] = ch;
  return true;
}
```

Para k = 1 e ch = 11

3408

A			21	15	23	**11**	44
base	0	2	6	7	8		
topo	-1	**5**	6	6	7		

Inserção (versão recursiva)

Inserção (versão recursiva)

A inserção ocorre *no topo* de uma pilha (após o elemento que está no topo)

Porém, pode ser que a pilha atual já esteja *cheia*

Neste caso procuraremos espaços livre à direita ou à esquerda da pilha atual (**deslocando as pilhas necessárias** para liberação do espaço)

Deslocando recursivamente para a direita

```
bool paraDireitaRec(PILHAMULTIPLA* p, int k) {









  return false;
}
```

Deslocando recursivamente para a direita

```
bool paraDireitaRec(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;









  return false;
}
```

Deslocando recursivamente para a direita

```
bool paraDireitaRec(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(!pilhaKcheia(p, k) || paraDireitaRec(p,k+1)){




  }
  return false;
}
```

Deslocando recursivamente para a direita

```
bool paraDireitaRec(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(!pilhaKcheia(p, k) || paraDireitaRec(p,k+1)){
    for(i = p->topo[k]; i >= p->base[k]; i--) p->A[i+1] = p->A[i];
    p->topo[k]++;
    p->base[k]++;
    return true;
  }
  return false;
}
```

Deslocando recursivamente para a esquerda

```
bool paraEsquerdaRec(PILHAMULTIPLA* p, int k) {









  return false;
}
```

Deslocando recursivamente para a esquerda

```
bool paraEsquerdaRec(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;









  return false;
}
```

Deslocando recursivamente para a esquerda

```
bool paraEsquerdaRec(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(!pilhaKcheia(p, k-1) || paraEsquerdaRec(p,k-1)){




  }
  return false;
}
```

Deslocando recursivamente para a esquerda

```
bool paraEsquerdaRec(PILHAMULTIPLA* p, int k) {
  if( (k < 1) || (k > NP-1) ) return  false;
  int i;
  if(!pilhaKcheia(p, k-1) || paraEsquerdaRec(p,k-1)){
    for(i = p->base[k]; i <= p->topo[k]; i++) p->A[i-1] = p->A[i];
    p->topo[k]--;
    p->base[k]--;
    return true;
  }
  return false;
}
```

Inserção (segunda versão)

```
bool pushK2(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
```

Inserção (segunda versão)

```
bool pushK2(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
```

Inserção (segunda versão)

```
bool pushK2(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  if (pilhaKcheia(p,k)){









}
```

Inserção (segunda versão)

```
bool pushK2(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  if (pilhaKcheia(p,k)){
    if (!paraDireitaRec(p, k+1)){


  }



}
```

Inserção (segunda versão)

```
bool pushK2(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  if (pilhaKcheia(p,k)){
    if (!paraDireitaRec(p, k+1)){
      if (!paraEsquerdaRec(p, k)) return false;
    }
  }



}
```

Inserção (segunda versão)

```
bool pushK2(TIPOCHAVE ch, PILHAMULTIPLA* p, int k) {
  if (k<0 || k>=NP) return false;
  if (pilhaKcheia(p,k)){
    if (!paraDireitaRec(p, k+1)){
      if (!paraEsquerdaRec(p, k)) return false;
    }
  }
  p->topo[k]++;
  p->A[p->topo[k]] = ch;
  return true;
}
```

AULA e04
Algoritmos e Estruturas de Dados I

Múltiplas Pilhas - implementação estática

Luciano Antonio Digiampietri