Estrutura Abstrata de Dados
Árvore Binária de Pesquisa (ABP)

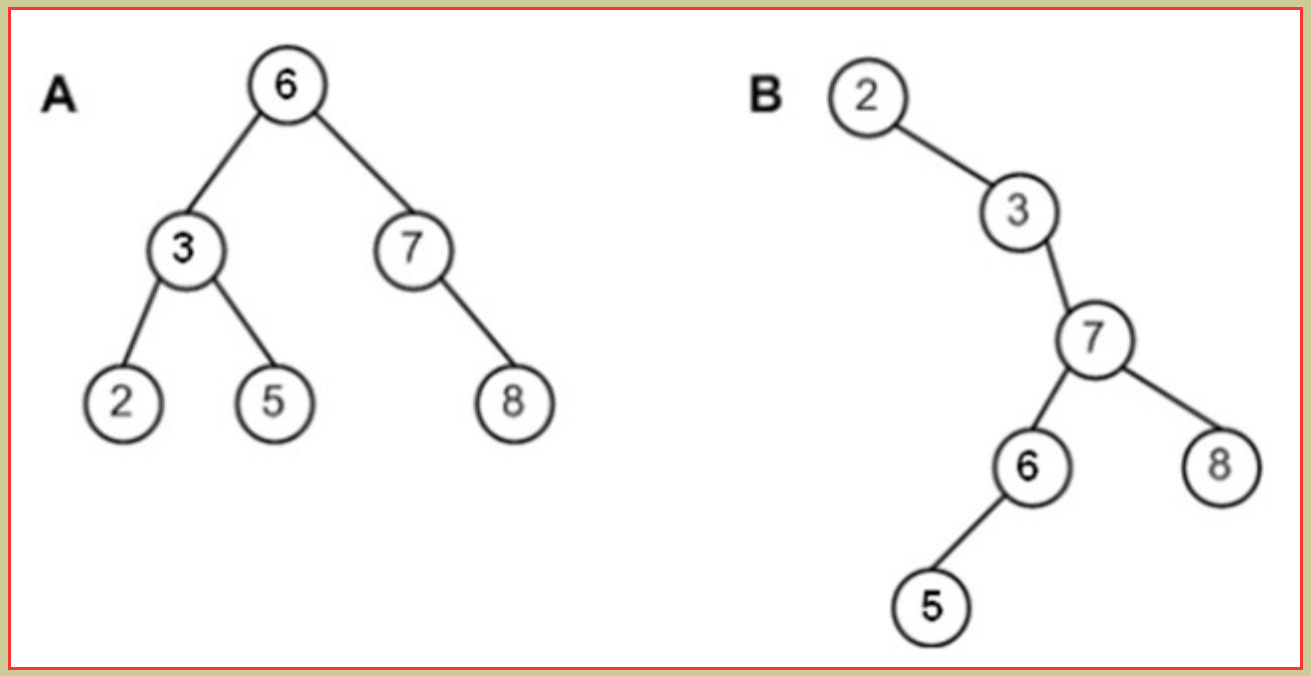
Implementação com ligações simples
(usando ponteiros e memória dinâmica)

Árvore Binária de Pesquisa

Definição

- É uma **árvore binária** em que a chave (elemento) guardada nos seus nodos
  - determina a sua posição de colocação na árvore
  - a chave de um nodo obedece às seguintes regras:
    - é maior do que a chave de qualquer nodo da sua subárvore esquerda
    - é menor do que a chave de qualquer nodo da sua subárvore direita
- Desta definição decorre que não podem existir nodos com chaves iguais
- Desta forma, ao percorrer a árvore *em ordem*, os seus elementos são visitados por *ordem crescente* das suas chaves
- Cada nodo pode ser visto como a raiz de duas árvores
  - a árvore *esquerda* com nodos cujas chaves são inferiores à chave da raiz
  - a árvore *direita* com nodos cujas chaves são superiores à chave da raiz.

Exemplos



Árvore Binária vs. Árvore Binária de Pesquisa

Árvore binária

- As chaves dos elementos de uma AB encontram-se distribuídos pela árvore sem qualquer relação de ordem

Árvore binária de pesquisa

- As chaves dos elementos de uma ABP encontram-se distribuídos pela árvore ordenados por ordem crescente, ao ser percorrida em-ordem
- O objetivo de organizar dados em ABP é facilitar a procura de um elemento
  - a partir da raiz e da chave a ser encontrada, é possível saber qual o caminho a ser percorrido até encontrar o elemento/nodo com aquela chave
  - basta verificar se a chave do elemento procurado é maior ou menor à chave do elemento na posição atual
- Praticamente todas as operações que se aplicam a uma AB também se aplicam a uma ABP

Operações básicas sobre uma ABP

Criar uma ABP

Libertar/destruir uma ABP

Verificar se uma ABP está vazia

Mostrar os elementos de uma ABP: pré-ordem, em-ordem e pós-ordem

Determinar a altura de uma ABP

Pesquisar um elemento numa ABP

Consultar um elemento de uma ABP

Atualizar um elemento de uma ABP

Inserir um elemento numa ABP

Remover um elemento de uma ABP

Nodos e ligações

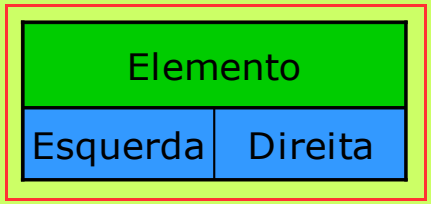
Estrutura

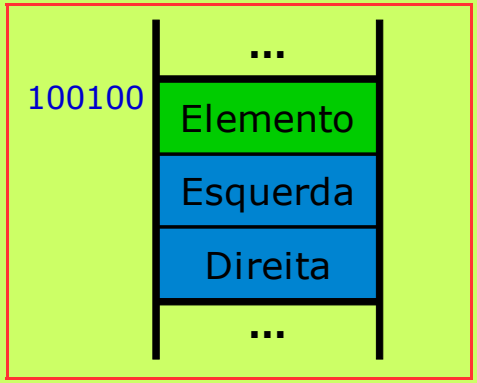
- Como uma ABP é uma AB, cada nodo pode ser representado pela mesma estrutura; ou seja, contendo os seguintes dados
  - a **informação** propriamente dita que se pretende guardar e tratar
  - as **ligações** a outros dois nodos, correspondentes
    - à raiz da **subárvore esquerda**
    - à raiz da **subárvore direita**
- As **ligações** são feitas através de **apontadores**/**ponteiros**, os quais indicam onde
  - está a raiz da subárvore **esquerda**
  - está a raiz da subárvore **direita**

Definição de um nodo

- A definição pode ser a mesma usada para a Árvore Binária, em linuagem C

```
struct  NodoABP {
    DadosABP  Elemento;
    struct NodoABP  *Esquerda;
    struct NodoABP  *Direita;
};
typedef  struct NodoABP  *PNodoABP;
```





- definição implementada através de uma estrutura composta por três campos:
  - **Elemento**: DadosABP é o tipo de dados que contém a informação a guardar e tratar
  - **Esquerda**: ligação à raiz da subárvore **esquerda**
  - **Direita**: ligação à raiz da subárvore **direita**
- definição perante um problema de circularidade, pois os campos **Esquerda** e **Direita**
  - são apontadores para elementos do tipo **NodoABP**
  - fazem parte da estrutura **NodoABP**

Operações sobre a estrutura DadosABP

Mostrar os dados de um elemento do tipo DadosABP

void **mostrarElementoABP** (DadosABP X)

- operação que mostra os dados/informação do elemento X do tipo DadosABP

Criar um elemento do tipo DadosABP

DadosABP **criarElementoABP** ()

- operação para criar um elemento do tipo DadosABP, com dados vindos de fonte adequada

Comparar dois elemento do tipo DadosABP

int **compararElementosABP** (DadosABP X, DadosABP Y)

- operação que compara dois elementos do tipo DadosABP, segundo um dos seus campos (chave)
- devolve: **-1** (X < Y), **0** (X = Y) ou **1** (X > Y)

Operações básicas sobre um nodo

Criar um nodo de uma ABP

- Adaptar a partir da implementada para Árvores Binárias

Entrada: informação/dados que se pretende guardar na ABP
Saída: um ponteiro para um nodo (informação + ligações)

```
PNodoABP criarNodoABP (DadosABP X)
{
  PNodoABP  P;
  P = (PNodoABP) malloc(sizeof(struct NodoABP));
  if (P == NULL)
     return NULL;
  P→Elemento = X;
  P→Esquerda = NULL;
  P→Direita = NULL;
  return P;
}
```

Libertar um nodo de uma ABP

- Adaptar a partir da implementada para Árvores Binárias

Entrada: um ponteiro para o nodo que se pretende destruir
Saída: o ponteiro a apontar para NULL

```
PNodoABP libertarNodoABP (PNodoABP P)
{
  P→Esquerda = NULL;
  P→Direita = NULL;
  free(P);
  P = NULL;
  return P;
}
```

Operações básicas sobre uma ABP

Criar uma ABP (vazia)

- Adaptar a partir da implementada para Árvores Binárias

Entrada: --
Saída: devolve um ABP vazia (sem elementos)

```
PNodoABP criarABP ()
{
  PNodoABP T;
  T = NULL;
  return T;
}
```

Libertar/destruir uma ABP

- Adaptar a partir da implementada para Árvores Binárias

Entrada: uma ABP T (a que se pretende libertar/destruir)
Saída: a ABP T vazia (T a apontar para NULL)

```
PNodoABP destruirABP (PNodoABP T)
{
  if (T == NULL)
      return T;
  T→Esquerda = destruirABP(T→Esquerda);
  T→Direita = destruirABP(T→Direita);
  T = libertarNodoABP(T);
  return T;
}
```

Verificar se uma ABP está vazia

- Adaptar a partir da implementada para Árvores Binárias

Entrada: uma ABP T
Saída: 1 (se T está vazia) ou 0 (se T não está vazia)

```
int ABPVazia (PNodoABP T)
{
  if (T == NULL)
      return 1;
  else
      return 0;
}
```

Mostrar os elementos de uma ABP

- Adaptar a partir das implementadas para Árvores Binárias
- Operação para fazer a travessia em **em-ordem** (esquerda, raiz e direita)

Entrada: uma ABP T
Saída: -- (mostra os elementos da árvores pela ordem indicada)

```
void mostrarEmOrdemABP (PNodoABP T)
{
  if (T != NULL) {
     mostrarEmOrdemABP(T→Esquerda);
     mostrarElementoABP(T→Elemento);
     mostrarEmOrdemABP(T→Direita);
  }
}
```

Mostrar os elementos de uma ABP

- Adaptar a partir das implementadas para Árvores Binárias
- Operação para fazer a travessia em **pré-ordem** (raiz, esquerda e direita)

Entrada: uma ABP T
Saída: -- (mostra os elementos da árvores pela ordem indicada)

```
void mostrarPreOrdemABP (PNodoABP T)
{
  if (T != NULL) {
     mostrarElementoABP(T→Elemento);
     mostrarPreOrdemABP(T→Esquerda);
     mostrarPreOrdemABP(T→Direita);
  }
}
```

Mostrar os elementos de uma ABP

- Adaptar a partir das implementadas para Árvores Binárias
- Operação para fazer a travessia em **pós-ordem** (esquerda, direita e raiz)

Entrada: uma ABP T
Saída: -- (mostra os elementos da árvores pela ordem indicada)

```
void mostrarPosOrdemABP (PNodoABP T)
{
  if (T != NULL) {
     mostrarPosOrdemABP(T→Esquerda);
     mostrarPosOrdemABP(T→Direita);
     mostrarElementoABP(T→Elemento);
  }
}
```

Determinar a altura de uma ABP

- Adaptar a partir da implementada para Árvores Binárias

Entrada: uma ABP T
Saída: a altura da ABP T (-1 se é vazia, 0 se só tem um nodo, ...)

```
int alturaABP (PNodoABP T)
{
  int  altEsq, altDir;
  if (T == NULL)
     return  -1;   // por definição, altura de uma árvore vazia é -1
  altEsq = alturaABP(T→Esquerda);
  altDir = alturaABP(T→Direita);
  if (altEsq > altDir)
     return  altEsq + 1;
  else
     return  altDir + 1;
}
```

Pesquisar um elemento numa ABP

Entrada: o elemento X a pesquisar e uma ABP T

Saída: ponteiro para o nodo com o elemento a pesquisar (existe); NULL (não existe)

```
PNodoABP pesquisarABP (DadosABP X, PNodoABP T)
{
  if (T == NULL)
     return NULL;
  if (compararElementosABP(X, T→Elemento) == 0)     //  X = T→Elemento
     return T;
  if (compararElementosABP(X, T→Elemento) == -1)    //  X < T→Elemento
     return pesquisarABP(X, T→Esquerda);
  else
     return pesquisarABP(X, T→Direita);
}
```

Consultar um elemento de uma ABP

Entrada: o elemento a consultar X (a chave) e uma ABP T
Saída: -- (mostra o elemento da árvore T com chave igual à do elemento X)

```
void consultarABP (DadosABP X, PNodoABP T)
{
  PNodoABP  P;
  P = pesquisarABP(X, T);
  if (P != NULL)
     mostrarElementoABP(P→Elemento);
  else
     printf("Nao existe elemento na AB com a mesma chave de X!\n");
}
```

Atualizar um elemento de uma ABP

Entrada: a informação atual (chave) X, a nova informação Y e uma ABP T
Saída: ponteiro para a raiz da árvore T (pode ser diferente do inicial)

```
PNodoABP atualizarABP (DadosABP X, DadosABP Y, PNodoABP T)
{
  PNodoABP  P;
  P = pesquisarABP(X, T);
  if (P != NULL) {
     T = removerABP(X, T);
     T = inserirABP(Y, T);
  }
  return T;
}
```

Inserir um elemento numa ABP

- Este processo tem que obedecer à definição de ABP
  - a chave do elemento a inserir não pode ainda existir na árvore e
  - a sua posição de inserção tem que assegurar que a chave de um elemento é
    - maior do que as chaves de todos os elementos da sua subárvore esquerda e
    - menor do que as chaves de todos os elementos da sua subárvores direita
  - o novo nodo com este elemento é sempre inserido como folha da árvore

Inserir um elemento numa ABP

- Operação só realizada se o elemento a inserir ainda não existir em T

Entrada: o elemento X a inserir e uma ABP T
Saída: a ABP T atualizada, após a inserção de X em T

```
PNodoABP inserirABP (DadosABP X, PNodoABP T)
{
  if (T == NULL) {
     T = criarNodoABP(X);
     return T;
  }
  if (compararElementosABP(X, T→Elemento) == -1)    // X < T→Elemento
     T→Esquerda = inserirABP(X, T→Esquerda);
  else
     T→Direita = inserirABP(X, T→Direita);
  return T;
}
```

Remover um elemento de uma ABP

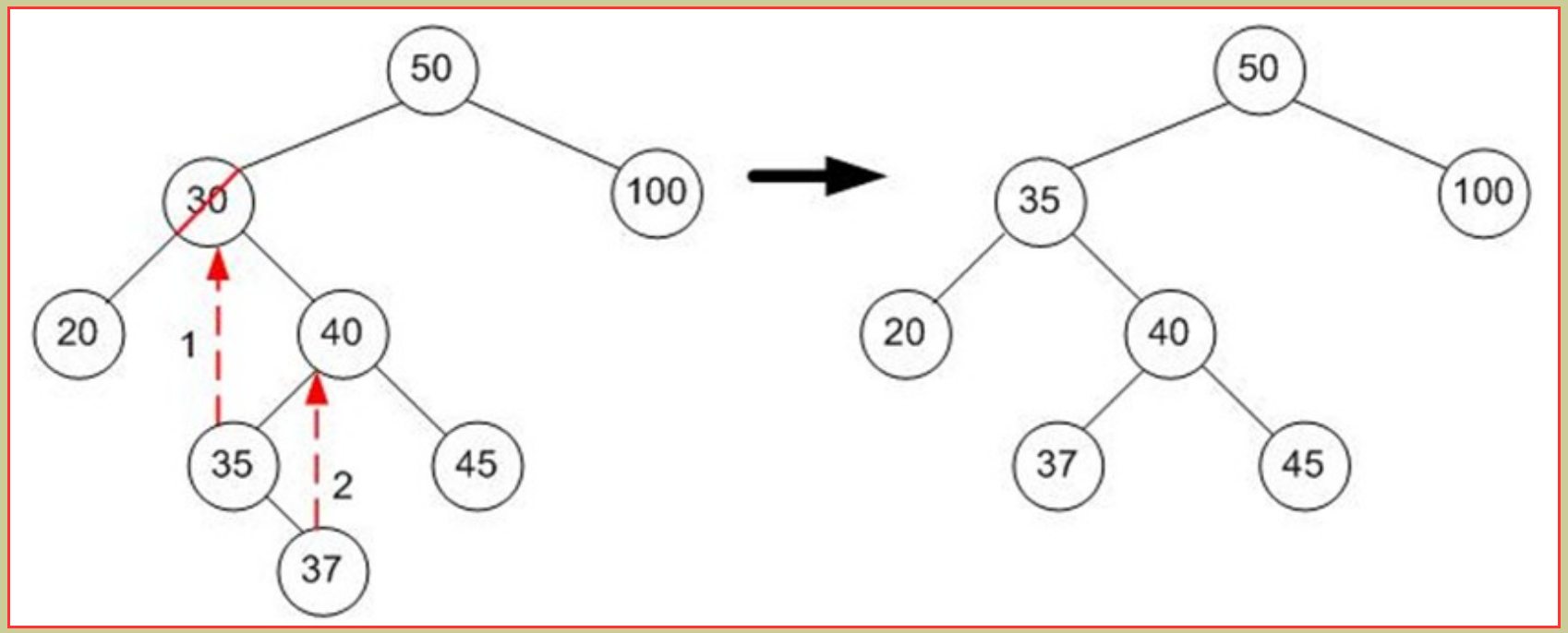
- Considerar três casos distintos, segundo as características do nodo a remover
  - se é um nodo sem filhos (folha)
  - se é um nodo com um único filho
  - se é um nodo com dois filhos
- Mecanismos para tratar os dois primeiros casos são
  - os mesmos descritos para o caso de árvores binárias genéricas
- Mecanismo para tratar o terceiro caso, remoção de um nodo com dois filhos,
  - tem em conta as características de uma ABP (os elementos estão organizados segundo um critério)

Remover um elemento de uma ABP

- Remoção de um nodo com dois filhos
  - O mecanismo de remoção implica substituir este nodo pelo nodo com chave mais "próxima" (maior ou menor) da chave do elemento a ser removido
  - Esta operação pode realizar-se de duas formas diferentes:
    - substituir a informação associada ao nodo a ser removido pela informação do seu sucessor, que é o nodo mais à esquerda da subárvore direita (é o menor elemento da subárvore direita, que é maior do que qualquer elemento da subárvore esquerda)
    - substituir a informação associada ao nodo a ser removido pelo seu antecessor, que é o nodo mais à direita da subárvore esquerda (é o maior elemento da subárvore esquerda, que é menor do que qualquer elemento da subárvore direita)

Remover um elemento de uma ABP

- Remoção de um nodo com dois filhos
  - exemplo: remover o nodo com **30** da árvore em baixo



  - substituir o elemento do nodo com 30 pelo do nodo
    - com **35** (sucessor de 30 na ABP), ou
    - com 20 (antecessor de 30 na ABP)
  - remover a folha com **35** ou com 20, conforme o caso usado

Remover um elemento de uma ABP

Requisito: operação a realizar se o elemento a remover existe em T
Entrada: o elemento X a remover e uma ABP T (X existe em T)
Saída: a ABP T atualizada, após a remoção de X em T

```
PNodoABP removerABP (INFOABP X, PNodoABP T)
{
  if (compararElementosABP(X, T→Elemento) == 0) {       //  X == T→Elemento
     T = removerNodoABP(T);
     return T;
  }
  if (compararElementosABP(X, T→Elemento) == -1)        // X < T→Elemento
     T→Esquerda = removerABP(X, T→Esquerda);
  else
     T→Direita = removerABP(X, T→Direita);
  return T;
}
```

Remover um elemento de uma ABP – operação auxiliar

Entrada: ponteiro para o nodo a remover (raiz de subárvore não vazia)
Saída: ponteiro para a raiz da subárvore após remoção do elemento

```
PNodoABP removerNodoABP (PNodoABP T) {
  PNodoABP  P;
  INFOABP  X;
  if (T→Esquerda == NULL && T→Direita == NULL) {      // T é uma folha
     T = libertarNodoABP(T);
     return T;
  }
  if (T→Esquerda == NULL) {      // T só tem um filho: o direito
     P = T;
     T = T→Direita;
     P = libertarNodoABP(P);
     return T;
  }
...
```

Remover um elemento de uma ABP – operação auxiliar

```
...
  if (T→Direita == NULL) {          // T só tem um filho: o esquerdo
      P = T;
      T = T→Esquerda;
      P = libertarNodoABP(P);
      return T;
  }
  // T tem 2 filhos: remover o nodo sucessor/antecessor e copiar a sua informação
  if (alturaABP( T→Esquerda) < alturaABP( T→Direita)
      T→Direita = substituirNodoDireita(T→Direita, &X);          // 1º caso
  else
      T→Esquerda = substituirNodoEsquerda(T→Esquerda, &X);     // 2º caso
  // substituir o elemento do nodo apontado por T pelo elemento da folha substituta
  T→Elemento = X;
  return T;
}
```

Remover um elemento de uma ABP – operação auxiliar

- Operação para substituir a informação associada ao nodo a ser removido pela informação do seu **sucessor** (o nodo mais à esquerda da subárvore direita)

Entrada: ponteiro para o nodo que se pretende excluir (raiz de subárvore não vazia)
Saída: ponteiro para a raiz da árvore e o elemento do nodo removido

```
PNodoABP substituirNodoDireita (PNodoABP T, INFOABP *X) {
  PNodoABP  P;
  if (T→Esquerda == NULL) {
     *X = T→Elemento;
     P = T;
     T = T→Direita;
     P = libertarNodoABP(P);
     return T;
  }
  T→Esquerda = substituirNodoDireita(T→Esquerda, X);
  return T;
}
```

Remover um elemento de uma ABP – operação auxiliar

- Operação para substituir a informação associada ao nodo a ser removido pela informação do seu **antecessor** (o nodo mais à direita da subárvore esquerda)

Entrada: ponteiro para o nodo a remover (raiz de subárvore não vazia)
Saída: ponteiro para a raiz da árvore e o elemento do nodo removido

```
PNodoABP substituirNodoEsquerda (PNodoABP T, INFOABP *X) {
  PNodoABP P;
  if (T→Direita == NULL) {
     *X = T→Elemento;
     P = T;
     T = T→Esquerda;
     P = libertarNodoABP(P);
     return T;
  }
  T→Direita = substituirNodoEsquerda(T→Direita, X);
  return T;
}
```