Sobre os Percursos de Árvores Binárias

Percurso em Profundidade

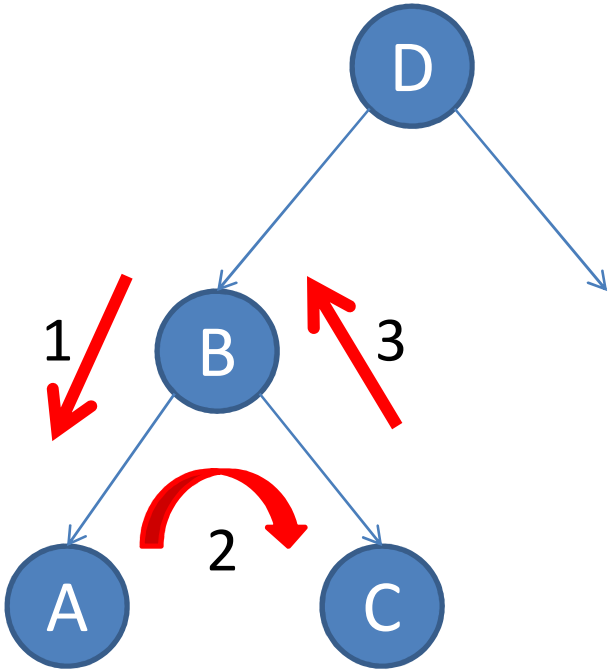
- As 3 funções padrão de percurso em profundidade dadas em sala podem ser resumidas numa única função:

```
typedef enum tipoP = { PRE,IN,POS}

void percorre(apno t, tipoP v){
  if(v == PRE) visita(t);
  visita(t->esq);
  if(v == IN) visita(t);
  visita(t->dir);
  if(v == POS) visita(t);
}
```

Idéia Geral

- A função anterior mostra que cada nó da árvore é ‘examinado’ exatamente 3 vezes.



Percurso sem uso de pilha

- Usando essa característica do percurso e colocando em cada nó campos auxiliares, é possível fazer o mesmo tipo de percurso de forma não recursiva e sem o uso de uma pilha auxiliar.

```
typedef struct tnode*  tnodeptr;
typedef struct tnode{
                        char info;
                        tnodeptr esq;
                        tnodeptr dir;
                        int aux;
                        tnodeptr pai;
                    } tnode;
```

Percurso sem uso de pilha

```
void percorre(tnodeptr p, int tt){
  while(p != NULL){
    switch (p->aux){
      case 0:   if(tt == PRE) visita(p);
                p->aux++;
                if(p->esq != NULL) p = p->esq;
                break;
      case 1:  if(tt == IN) visita(p);
                p->aux++;
                if(p->dir != NULL) p = p->dir;
                break;
      case 2:   if(tt == POS) visita(p);
                p->aux = 0;
                p = p->pai;
                break;
    }
  }
}
```

Percurso sem o uso de pilha

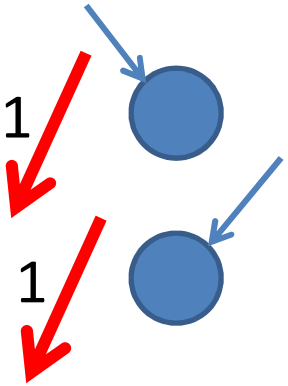
A função apresentada

- Supõe a existência de um campo adicional (aux) e conta com o fato de que seu valor seja igual a zero para todos os nós da árvore.
- Durante o percurso o valor desse campo é alterado e só volta a zero em todos os campos ao final do percurso. Sendo assim, a função de percurso deve sempre percorrer todos os nós da árvore.

Percurso sem o uso de pilha

- O valor do campo aux indica a ‘fase’ em que o nó correspondente é acessado.
- Essa ‘fase’ pode ser obtida a partir da forma como se chega ao nó.

Percurso sem o uso de pilha

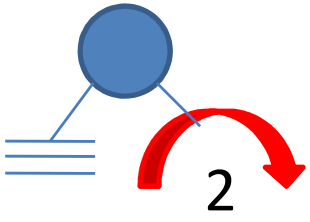


- Caso A: chegamos ao nó a partir do seu pai, fazendo `p = p->esq;` `ou p = p->dir;`

  Nesse caso, vamos iniciar o percurso de uma nova sub-árvore, portanto estamos na 'fase 1'.

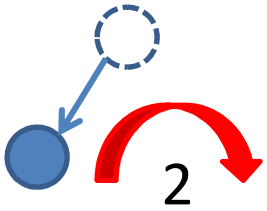
- Caso B: na 'fase 1', **`p->esq == NULL`**

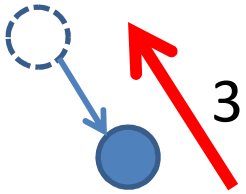
  Nesse caso, devemos passar a visitar a sub-árvore direita, ou seja passamos à 'fase 2'.



- Caso C: fazemos p = p->pai;
  - Caso C.1: o nó atual é filho esquerdo: nesse caso passamos à 'fase 2'.
  - Caso C.2: o nó atual é filho direito:

    nesse caso passamos à 'fase 3'.





A função de percurso

```
void percorre(tnodeptr p, int tt){
  int d = 0; tnodeptr q;
  while(p != NULL){
    switch (d){
      case 0:
              if(tt == PRE) visita(p);
              if(p->esq != NULL) p = p->esq;
              else d = 1;
              break;
      case 1:
              if(tt == IN) visita(p);
              if(p->dir != NULL) { p = p->dir; d = 0; }
              else d = 2;
              break;
      case 2:
              if(tt == POS) visita(p);
              q = p;
              p = p->pai;
              if(p != NULL) if(p->esq == q) d = 1;
              break;
    }
  }
}
```

Percurso iterativo

- A função de percurso apresentada visita todos os nós da árvore.
- A função de visita propriamente dita é chamada dentro da função de percurso. Em alguns casos isso pode ser uma limitação.
- A idéia: uma função que devolve o ‘próximo nó a ser visitado’ para um certo tipo de percurso.

Percurso iterativo

```
tnodeptr proximo(tnodeptr p, int tt,int* d){
  tnodeptr q;
  do{
    switch (*d){
      case 0:
              if(p->esq != NULL) p = p->esq;
              else *d = 1;
              break;
      case 1:
              if(p->dir != NULL) { p = p->dir; *d = 0; }
              else *d = 2;
              break;
      case 2:
              q = p;
              p = p->pai;
              if(p != NULL) if(p->esq == q) *d = 1;
              break;
    }
  } while((p != NULL) && (*d != tt));
  return p;
}
```

Percurso iterativo – uso p/ pré-ordem

```
void preordem(tnodeptr p){
  int d = 0;
  while (p != NULL){
    visita(p);
    p = proximo(p,PRE,&d);
  }
}
```

Percurso iterativo – uso p/ in-ordem

```
void inordem(tnodeptr p){
  int d = 0;
  while (p != NULL){
    p = proximo(p,IN,&d);
    if(p!= NULL) visita(p);
  }
}
```

Percurso iterativo – uso p/ pós-ordem

```
void posordem(tnodeptr p){
  int d = 0;
  while (p != NULL){
    p = proximo(p,POS,&d);
    if(p!= NULL) visita(p);
  }
}
```