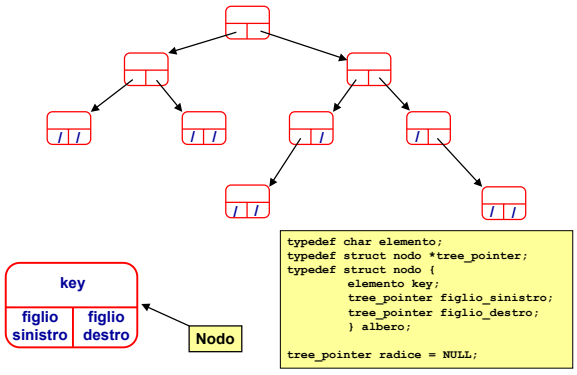
Strutture Dati

Lezione 11
Gli alberi binari

Oggi parleremo di …

- Rappresentazione
- Operazioni
  - visita inorder (o in ordine simmetrico)
  - visita preorder (o in ordine anticipato)
  - visita postorder (o in ordine posticipato)
  - visita inorder iterativa
  - visita in ordine di livello
- Altre operazioni
  - creazione
  - copia
  - equivalenza

Rappresentazione di un albero binario




```
typedef char elemento;
typedef struct nodo *tree_pointer;
typedef struct nodo {
        elemento key;
        tree_pointer figlio_sinistro;
        tree_pointer figlio_destro;
        } albero;

tree_pointer radice = NULL;
```

Attraversamento di un albero binario

- Una delle operazione più frequenti riguardanti gli alberi è la visita dei loro nodi una e una sola volta, detta anche attraversamento.
- Consideriamo solo tre tipologie:
  - visita inorder (SVD)
  - visita preorder (VSD)
  - visita postorder (SDV)
- Esiste una naturale corrispondenza tra queste visite e la produzione dei formati infisso, prefisso e postfisso di una espressione rappresentata da un albero binario.

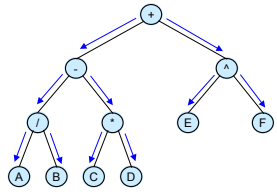
Attraversamento di un albero binario

Visita inorder (SVD) Un attraversamento inorder consiste in

1. uno spostamento verso il basso e a sinistra lungo un albero finché non viene raggiunto un nodo nullo;
2. una “visita” del padre del nodo nullo;
3. un attraversamento del nodo che si trova una posizione a destra; l’attraversamento continua con l’ultimo nodo non visitato che si trova ad un livello superiore nell’albero.

Attraversamento di un albero binario

visita inorder (SVD)



albero binario contenente un’espressione aritmetica

A / B - C * D + E ^ F

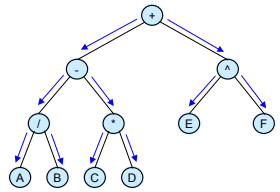
Attraversamento di un albero binario

Visita preorder (VSD) Un attraversamento preorder consiste in

1. una "visita" del nodo;
2. uno spostamento a sinistra, visitando tutti i nodi incontrati;
3. l'attraversamento continua finché non si raggiunge un nodo nullo; si torna indietro al nodo antenato più vicino che ha un figlio destro e si prosegue la visita di questo nodo.

Attraversamento di un albero binario

visita preorder (VSD)



albero binario contenente un'espressione aritmetica

+ - / A B * C D ^ E F

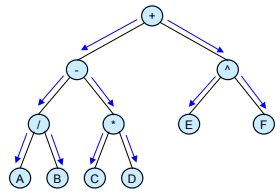
Attraversamento di un albero binario

Visita postorder (SDV) Un attraversamento postorder consiste in

1. uno spostamento a sinistra, visitando tutti i nodi incontrati;
2. uno spostamento a destra, visitando tutti i nodi incontrati;
3. una "visita" del nodo; si torna indietro al nodo antenato più vicino che ha un figlio sinistro e si prosegue la visita di questo nodo.

Attraversamento di un albero binario

visita postorder (SDV)



albero binario contenente un'espressione aritmetica

A B / C D * - E F ^ +

Attraversamento di un albero binario

visita inorder (SVD)

```
void inorder(tree_pointer ptr)
{
 if (ptr) {
     inorder(ptr->figlio_sinistro);
     printf("%4c", ptr->key);
     inorder(ptr->figlio_destro);
    }
}
```

visita preorder (VSD)

```
void preorder(tree_pointer ptr)
{
 if (ptr) {
     printf("%4c", ptr->key);
     preorder(ptr->figlio_sinistro);
     preorder(ptr->figlio_destro);
    }
}
```

visita postorder (SDV)

```
void postorder(tree_pointer ptr)
{
 if (ptr) {
     postorder(ptr->figlio_sinistro);
     postorder(ptr->figlio_destro);
     printf("%4c", ptr->key);
    }
}
```

Attraversamento di un albero binario

- **Visita inorder iterativa** Si aggiungono e si eliminano nodi dallo stack nello stesso modo in cui l'algoritmo ricorsivo gestisce lo stack di sistema
  - un nodo viene inserito nello stack se ad esso non è associata alcuna azione;
  - un nodo viene eliminato dallo stack se ad esso è associata l'azione **`printf`**.
- I nodi a sinistra vengono inseriti nello stack finché non viene raggiunto un nodo nullo, che viene eliminato.
- Il figlio a destra viene inserito nello stack.

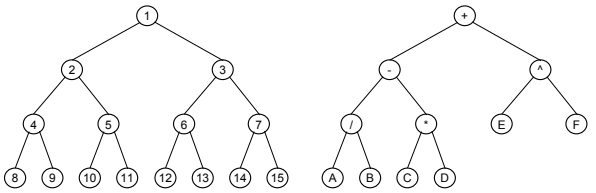
```
void inorder_iter(tree_pointer ptr)
{
 int top = -1;
 tree_pointer stack[MAX_STACK_SIZE];

 for (; ;) {
   for(; ptr; ptr=ptr>figlio_sinistro)
       add(&top, ptr);
   ptr = del(&top);
   if(!ptr)  break;
   printf("%4c", ptr->key);
   ptr = ptr->figlio_destro;
  }
}
```

$O(n)$

```
void inorder(tree_pointer ptr)
{
 if (ptr) {
     inorder(ptr->figlio_sinistro);
     printf("%4c", ptr->key);
     inorder(ptr->figlio_destro);
    }
}
```

Attraversamento di un albero binario

- **Visita in ordine di livello**
  - Si visita la radice, poi il figlio sinistro della radice, seguito dal figlio destro della radice;
  - si continua visitando i nodi ad ogni nuovo livello partendo dal nodo più a sinistra fino a quello più a destra.



+ - ^ / * E F A B C D

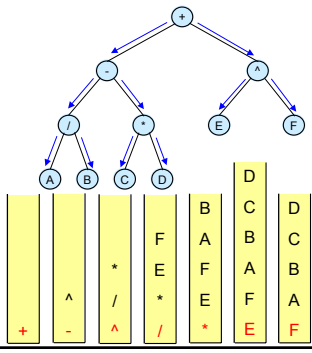
Attraversamento di un albero binario

- La visita per livello utilizza una coda circolare
  - si inserisce la radice nella coda;
  - si cancella il nodo in testa alla coda e si visualizza il campo dato;
  - si inseriscono nella coda i figli a sinistra e a destra del nodo visitato.

```
void level_order(tree_pointer ptr)
{
 int davanti = dietro = 0;
 tree_pointer queue[MAX_SIZE_CODA];

 if (!ptr)   return;
 addc(davanti, &dietro, ptr);
 for (; ;) {
   ptr = delc(&davanti, dietro);
   if(ptr) {
      printf("%4c", ptr->key);
      if (ptr->figlio_sinistro)
        addc(davanti, &dietro, ptr->figlio_sinistro);
      if(ptr->figlio_destro)
        addc(davanti, &dietro, ptr->figlio_destro);
      }
   else break;
  }
}
```

Attraversamento di un albero binario

visita in ordine di livello



+ - ^ / * E F A B C D

Creazione di un albero binario

```
void crea(tree_pointer *radice)
{
 tree_pointer ptr_sin, ptr_des;
 elemento item;
 char risp;

 printf("Inserisci il dato : ");
 scanf("%c", &item);
 (*radice)->key = item;
 printf("\n%c ha figlio sinistro? (s/n)", (*radice)->key );
 risp=getchar();
 if(risp == 'n') (*radice)->figlio_sinistro = NULL;
 else {
       ptr_sin = (tree_pointer)malloc(sizeof(nodo));
       (*radice)->figlio_sinistro = ptr_sin;
       crea(&ptr_sin);
      }
 printf("\n%c ha figlio destro? (s/n)", (*radice)->key);
 risp=getchar();
 if(risp == 'n') (*radice)->figlio_destro = NULL;
 else {
       ptr_des = (tree_pointer)malloc(sizeof(nodo));
       (*radice)->figlio_destro = ptr_des;
       crea(&ptr_des);
      }
}
```

Copia di un albero binario

```
tree_pointer tree_copy(tree_pointer orig)
{
  tree_pointer temp;

  if(orig) {
        temp = (tree_pointer)malloc(sizeof(nodo));
        temp->key = orig->key;
        temp->figlio_sinistro = tree_copy(orig->figlio_sinistro);
        temp->figlio_destro = tree_copy(orig->figlio_destro);
        return temp;
        }
  return NULL;
}
```

Equivalenza di due alberi binari

```
int tree_equiv(tree_pointer primo, tree_pointer secondo)
{
 return ((!primo && !secondo) || (primo && secondo &&
        (primo->key == secondo->key) &&
        tree_equiv(primo->figlio_sinistro, secondo->figlio_sinistr) &&
        tree_equiv(primo->figlio_destro, secondo->figlio_destro));
}
```