Strutture dati - 2 -

Alberi binari

Definizione

- L’albero è un insieme di elementi (nodi), sui quali è definita una relazione di discendenza con due proprietà:
  - esiste un solo nodo radice senza predecessori
  - ogni altro nodo ha un unico predecessore

Definizione

- tree= $\langle V, E \rangle$ dove V è un insieme di valori,

  $E \subseteq V \times V$ è una relazione su V

  $e = \langle v_{parent}, v_{child} \rangle \quad e \in E$

- Grado di uscita: numero dei successori diretti di un nodo
- Profondità: distanza di un nodo dalla radice; la radice ha profondità 0; ogni nodo≠radice ha profondità pari a quella del predecessore + 1

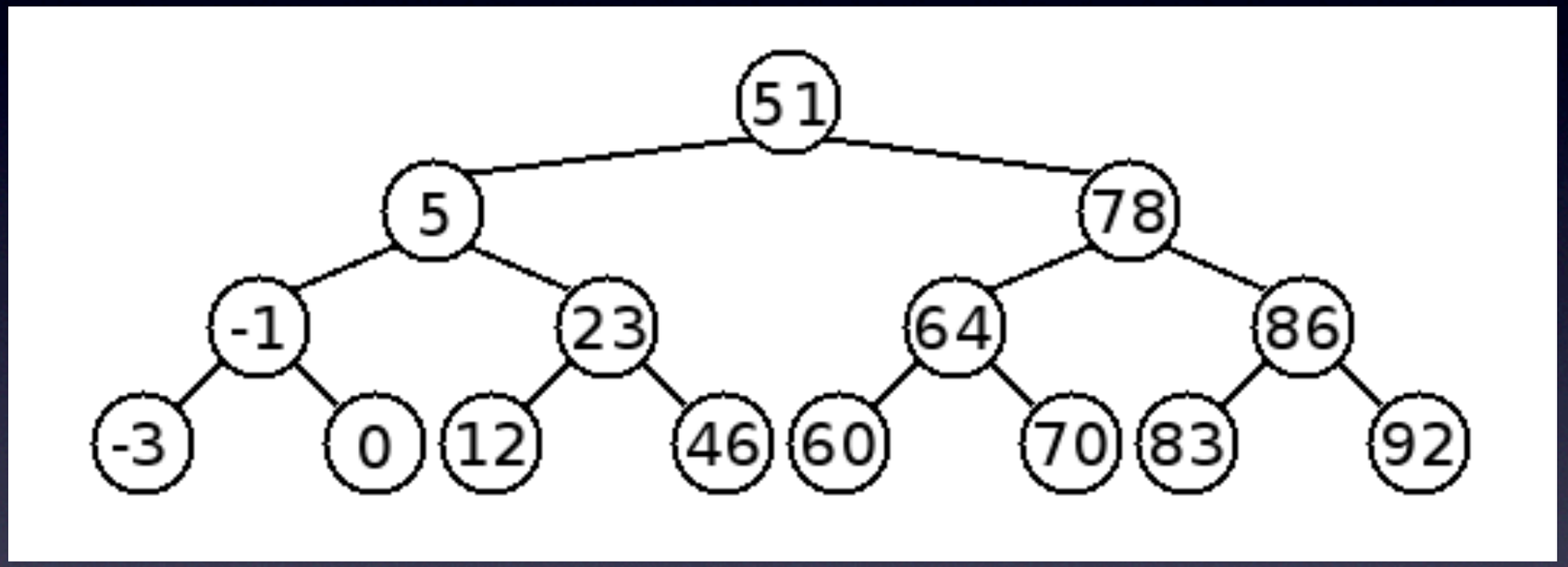
Operazioni

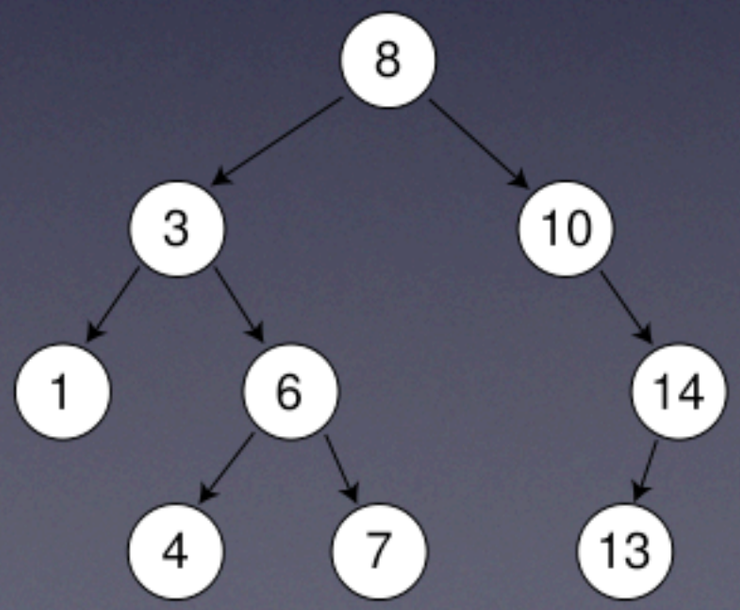
- Si eseguono le stesse operazioni effettuate sulle liste ma:
  - esistono più nodi terminali (le foglie): si deve decidere un criterio per l'inserimento in coda
  - l'inserimento in testa o nodi intermedi fa degenerare l'albero su una forma sequenziale: è da evitare

Alberi binari di ricerca

- Ogni nodo a grado di uscita ≤ 2
- Il valore codificato su un nodo è $\geq$ del valore codificato sul figlio sinistro e < del valore codificato sul figlio destro
- Un albero è bilanciato se ogni nodo su un livello diverso dall'ultimo o penultimo ha due figli

Alberi binari di ricerca: esempi





Numero di nodi

- Un albero di profondità D è bilanciato sse contiene tutti i nodi che posso essere ospitati sui livelli D+1, da cui se esiste almeno un nodo al livello D il numero N dei nodi è:

$$N \geq 1 + \sum_{d=0}^{D-1} 2^d = 2^D$$

- da cui $D \leq \ln_2 N$

Implementazione

- Implementazione con puntatori
  - i puntatori sono ai nodi figli, se non esistono hanno valore NULL

```
struct btree{
   float value;
   struct btree *left;
   struct btree *right;
}
```

Attraversamento dell'albero

- Preorder: visita radice, albero sinistro, albero destro
- Inorder: visita albero sinistro, radice, albero destro
- Postorder: visita albero sinistro, albero destro, radice
- Il carico sullo stack è sempre $O(\ln_2 N)$

Visita

- L'implementazione ricorsiva è simile a quella della lista:

```
void visit_r(struct btree *ptr)
{
   if ( ptr!=NULL )   {
      visit_r( ptr->left );
      printf("%f",ptr->value);
      visit_r( ptr->right );
   }
}
```

- E' un attraversamento inorder

Visita - 2 -

```
void visit_r(struct btree *ptr)
{
   if ( ptr!=NULL )  {
        visit_r( ptr->left );
        visit_r( ptr->right );
        //postorder
        printf("%f",ptr->value);
    }
}
```

- E' una visita postorder, ovvero depth-first

Ricerca

- In un albero bilanciato la ricerca è $O(\ln_2 N)$, se l'albero degenera diventa O(N)
- Algoritmo: se la radice contiene il target ci si ferma con successo; se la radice è > target si deve cercare sul sotto-albero di sinistra, altrimenti su quello di destra

Ricerca: implementazione ricorsiva

```
Boolean search_r(struct btree *ptr, float target)
{
  if ( ptr!=NULL ) {
    if ( ptr->value == target )
      return TRUE;
    else {
      if ( ptr->value > target )
        search_r( ptr->left, target );
      else
        search_r( ptr->right, target );
    }
  } else
    return FALSE;
}
```

Inserimento

- Si deve stabilire il criterio di inserimento in coda
  - in un albero binario di ricerca si inserisce il valore nell'unica foglia che mantiene l'albero in ordine
  - si scende dalla radice e si opera sul sottoalbero appropriato
  - raggiunta una foglia si effettua l'inserimento
- E' $O(\ln_2 N)$

Inserimento: implementazione ricorsiva

```
void insert_inorder_r(struct btree **pptr, value)
{
  if ( *pptr != NULL ) {
    if ( (*pptr)->value > value )
      insert_inorder_r( (*pptr)->left, value );
    else
      insert_inorder_r( (*pptr)->right, value );
  } else {
    (*pptr)=(struct btree *)malloc(sizeof(struct btree));
    (*pptr)->value = value;
    (*pptr)->left = NULL;
    (*pptr)->right = NULL;
  }
}
```

Cancellazione

- Dipende dal numero di figli di un nodo:
  - 0 (foglia): si cancella il nodo
  - 1: si cambia il puntatore al nodo cancellato, facendolo puntare al figlio
  - 2: ci si riconduce al caso precedente spostando uno dei sotto-alberi

Cancellazione - 2 -

- Si deve mantenere l'ordine:
  - Se si sposta il sottoalbero di sinistra, la sua radice diventa figlia di sinistra del nodo più a sinistra (che è il minimo)
  - Se si sposta il sottoalbero di destra, la sua radice diventa figlia di destra del nodo più a destra (che è il massimo)

Cancellazione: implementazione

```
void delete(struct btree **pptr)
{
  struct btree *tmp;

  if ( *pptr != NULL ) {
    tmp = *pptr;
    *pptr = (*pptr)->right;

    pptr = &((*pptr)->right);
    while ( *pptr != NULL )
      pptr = & ((*pptr)->left);

    *pptr = tmp->left;

    free( tmp ); // cancellazione del nodo
  }
}
```

Inserimenti e cancellazioni: esempi