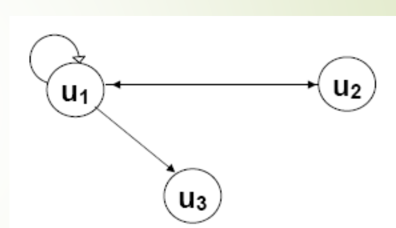
Programmazione e Strutture Dati (PR&SD)
I° ANNO – Informatica
Prof. V. Fuccella

ADT *Albero binario*

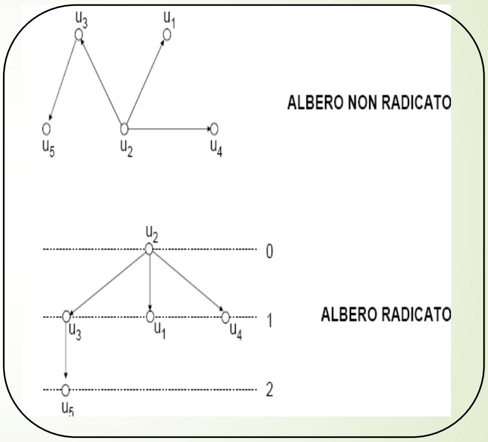
I Grafi

- Un grafo orientato G e' una coppia <N,A> dove:
  - N è un insieme finito non vuoto (insieme di nodi) e
  - A ⊆ NxN è un insieme finito di coppie ordinate di nodi, detti archi (o spigoli o linee).
- Se $< u_i, u_j > \in A$, nel grafo vi è un arco da u_i ad u_j
- Nell'esempio
  - $N = \{u_1,u_2,u_3\}$,
  - $A = \{(u_1,u_1),(u_1,u_2),(u_2,u_1),(u_1,u_3)\}$.



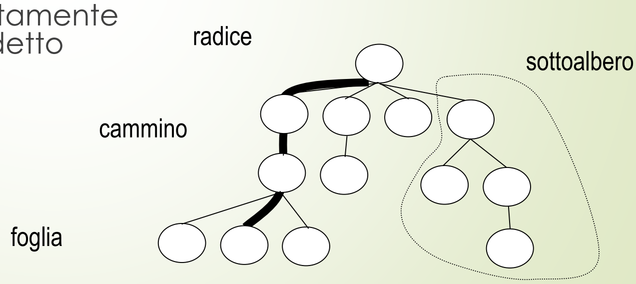
Gli Alberi

- Il GRAFO è una struttura dati alla quale si possono ricondurre strutture più semplici: LISTE ed ALBERI
- L'ALBERO è una struttura informativa per rappresentare:
  - organizzazioni gerarchiche di dati
  - partizioni successive di un insieme in sottoinsiemi disgiunti
  - procedimenti decisionali enumerativi



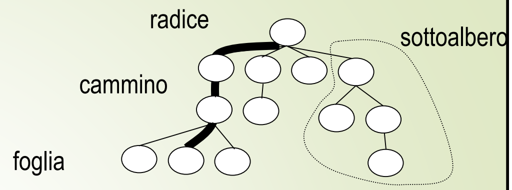
Alberi
Proprietà

- Ogni nodo ha un unico arco entrante, tranne la radice, che non ha archi entranti;
- Ogni nodo può avere zero o più archi uscenti
  - I nodi senza archi uscenti sono detti foglie
- Un arco nell'albero induce una relazione padre-figlio
- A ciascun nodo è solitamente associato un valore, detto *etichetta* del nodo



Alberi

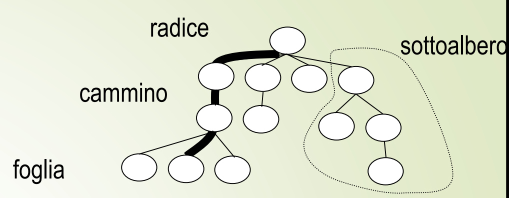
Alcuni concetti



- Grado di un nodo: numero di figli del nodo
  - Ordine dell'albero: grado max tra tutti i nodi
- Cammino: sequenza di nodi $<n_0, n_1, \ldots, n_k>$ dove il nodo n_i è padre del nodo n_{i+1}, per $0 \leq i < k$
  - La lunghezza del cammino è k
- Livello di un nodo: lunghezza del cammino dalla radice al nodo
  - Definizione ricorsiva: il livello della radice è 0, il livello di un nodo non radice è 1 + il livello del padre
- Altezza dell'albero: la lunghezza del più lungo cammino nell'albero
  - Parte dalla radice e termina in una foglia

6

Alberi VS Grafi

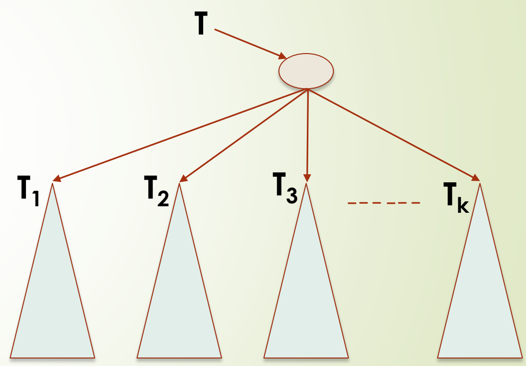


- Un albero e' un grafo diretto aciclico, in cui per ogni nodo esiste un solo arco entrante (tranne che per la radice che non ne ha nessuno)
- Se esiste un cammino che va da un nodo u ad un altro nodo v, tale cammino è unico
- In un albero esiste un solo cammino che va dalla Radice a qualunque altro nodo
- Dato un nodo u, i suoi discendenti costituiscono un albero detto sottoalbero di radice u

Gli Alberi

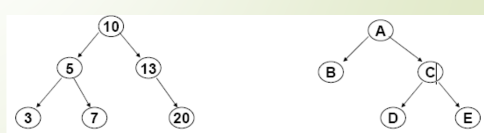
Struttura ricorsiva

- Un albero e' un insieme di nodi ai quali sono associate delle informazioni
- Tra i nodi esiste un nodo particolare che e' la radice (livello 0)
- Gli altri nodi sono partizionati in sottoinsiemi che sono a loro volta alberi (livelli successivi):
  - *Vuoto o costituito da un solo nodo (detto radice)*
  - *Oppure è una radice cui sono connessi altri alberi*



Alberi binari

- Particolari alberi n-ari: ogni nodo può avere al più due figli
  - sottoalbero sinistro e sottoalbero destro
- Definizione ricorsiva:
  - un albero binario è vuoto
  - oppure è una terna (s, r, d), dove r è un nodo (la radice), s e d sono alberi binari
- Alberi binari semplificati
  - Costruttore bottom-up
  - Operatori di selezione
  - Operatori di visita



ADT: Albero Binario

Sintattica	Semantica
Nome del tipo: BTree Tipi usati: Item, boolean	Dominio: T= nil | T = <N, T1, T2> N ϵ NODO, T1 e T2 sono BTree
newBTree() → BTree	newBTree() → T • Post: T = nil
isEmpty(BTree) → boolean	isEmpty(T) → b • Post: se T=nil allora b = true altrimenti b = false
buildBTree(Btree, Btree, Item) → BTree	buildBTree(T1,T2, e) → T • Pre: e != nil • Post: T = <N, T1, T2>; N ha etichetta e
getBTreeRoot(BTree) → Item	getBTreeRoot(T) → e • Pre: T = <N, T_{left}, T_{right}> non è vuoto • Post: N ha etichetta e
getLeft(BTree) → Btree getRight(BTree) → BTree	getLeft(T) → T' • Pre: T = <N, T_{left}, T_{right}> non è vuoto • Post: T' = T_{left}

Alberi binari: realizzazione

- Realizzazione più diffusa: struttura a puntatori con nodi doppiamente concatenati
- Ogni nodo è una struttura con 3 componenti:
  - Puntatore alla radice del sottoalbero sinistro
  - Puntatore alla radice del sottoalbero destro
  - Etichetta (useremo il tipo generico *Item* per questo campo)
- Un albero binario è definito come puntatore ad un nodo:
  - Se l'albero binario è vuoto, puntatore nullo
  - Se l'albero binario non è vuoto, puntatore al nodo radice

Dichiarazione del tipo nodo

- Per usare un albero binario serve una struttura che rappresenti i nodi
- La struttura conterrà i dati necessari (un Item) e due puntatori ai sottoalberi:

```
struct node {
   Item value;              /* etichetta del nodo*/
   struct node *left;       /* puntatore al sottoalbero sinistro*/
   struct node *right;      /* puntatore al sottoalbero destro*/
};
```

Dichiarazione del tipo Btree

- Il passo successivo è quello di dichiarare il tipo Btree

```
typedef struct node *Btree;
```

- una variabile di tipo Btree punterà nodo radice dell’albero
- Assegnare a T il valore NULL indica che l’albero è inizialmente vuoto

```
Btree T = NULL;
```

Creare un nodo dell'albero

- Un albero binario viene costruito in maniera bottom-up
- Man mano che costruiamo l'albero, creiamo dei nuovi nodi da aggiungere come nodo radice
- I passi per creare un nodo sono:
  1. Allocare la memoria necessaria
  2. Memorizzare i dati nel nodo
  3. Collegare il sottoalbero sinistro e il sottoalbero destro, già costruiti in precedenza

Alcune note sull'ADT Btree

- L'insieme degli operatori così definiti costituisce l'insieme degli operatori di base (il minimo insieme di operatori) di un albero binario
- Ogni altro operatore che si volesse aggiungere all'ADT albero binario potrebbe essere implementato utilizzando gli operatori dell'insieme di base
- E' frequente la pratica di arricchire il tipo Btree con l'aggiunta di operatori per inserire o cancellare nodi in determinate posizioni dell'albero

15

Algoritmi di Visita

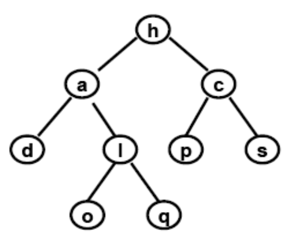
- La visita di un albero consiste nel seguire una rotta di viaggio che consenta di esaminare ogni nodo dell'albero esattamente una volta.
  - Visita in **pre-ordine**: si applica ad un albero non vuoto e richiede dapprima l'analisi della radice dell'albero e, poi, la visita, effettuata con lo stesso metodo, dei due sottoalberi, prima il sinistro, poi il destro
  - Visita in **post-ordine**: si applica ad un albero non vuoto e richiede dapprima la visita, effettuata con lo stesso metodo, dei sottoalberi, prima il sinistro e poi il destro, e, in seguito, l'analisi della radice dell'albero
  - Visita **simmetrica**: richiede prima la visita del sottoalbero sinistro (effettuata sempre con lo stesso metodo), poi l'analisi della radice, e poi la visita del sottoalbero destro

16

Algoritmi di Visita

ESEMPIO:

SIA UN ALBERO BINARIO CHE HA DEI CARATTERI NEI NODI



LA VISITA IN PREORDINE: h a d l o q c p s

LA VISITA IN POSTORDINE: d o q l a p s c h

LA VISITA SIMMETRICA: d a o l q h p c s

29

Algoritmi di Visita

visita in preordine l'albero binario t

{

se l'albero non è vuoto

allora

visita la radice di t

visita in preordine il sottoalbero sinistro di t

visita in preordine il sottoalbero destro di t

fine

}