UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Corso di Introduzione agli algoritmi

Prof.ssa Tiziana Calamoneri

Strutture dati fondamentali: Alberi

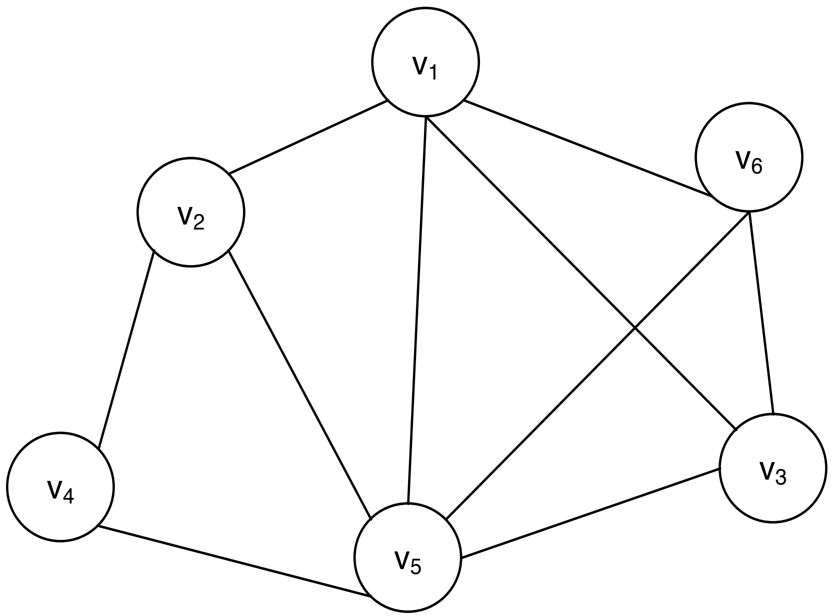
Alberi (1)

- L'*albero* è una struttura dati estremamente versatile, utile per modellare una grande quantità di situazioni reali e progettare le relative soluzioni algoritmiche.
- Abbiamo già incontrato la struttura ad albero (in particolare ad albero binario) varie volte, ma l'abbiamo sempre considerata in modo intuitivo.

Alberi (2)

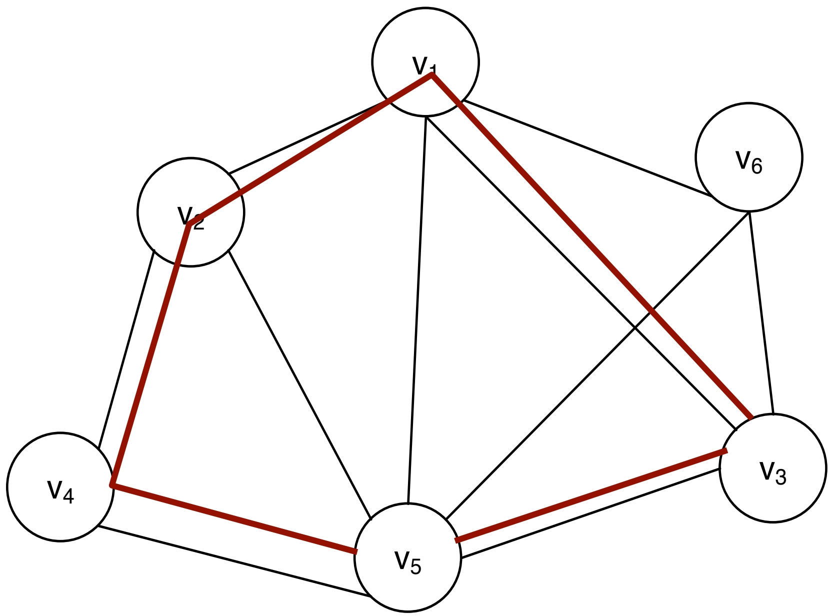
Per dare la definizione formale di albero è necessario prima fornire alcune definizioni relative ad un'altra stuttura dati, il ***grafo***:

- Un ***grafo*** *G = (V, E)* è costituito da una coppia di insiemi:
  - un insieme finito *V* dei ***nodi***, o ***vertici***;
  - un insieme finito $E \subseteq V x V$ di ***coppie non ordinate di nodi***, dette ***archi*** o ***spigoli***.



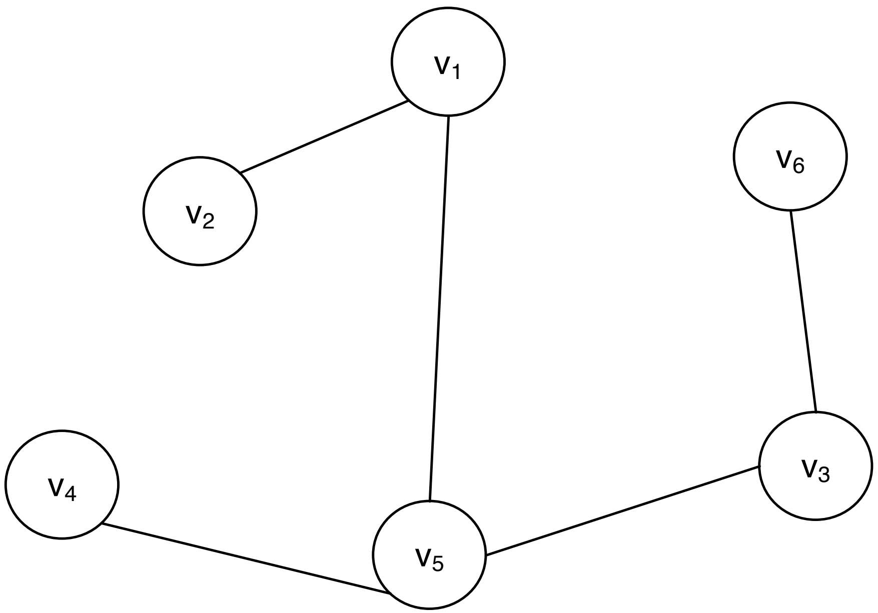
Alberi (2)

- Un ***cammino*** in un grafo $G = (V, E)$ è una sequenza $(v_1, v_2, \ldots, v_k)$ di nodi distinti di V tale che (v_i, v_{i+1}) sia un arco di E per ogni $1 \leq i \leq k\text{-}1$.
- Se ad un cammino $(v_1, v_2, \ldots, v_k)$ si aggiunge l'arco (v_k, v_1) si parla di ***ciclo***.
- Un grafo G è ***connesso*** se, per ogni coppia di nodi (u, v), esiste un cammino tra u e v. Un grafo G è ***aciclico*** se non contiene cicli.

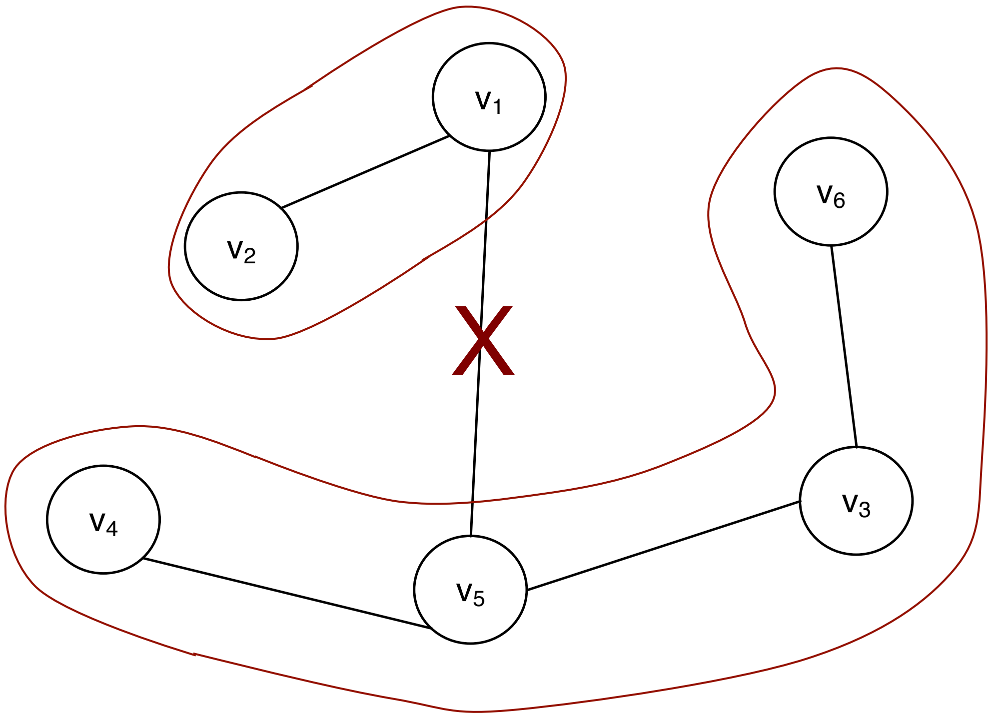


Alberi (3)

Definzione. *Un albero è un grafo G = (V, E) connesso e aciclico.*



Lemma. *Sia G=(V,E) un grafo connesso aciclico; eliminando da G un arco qualsiasi, G si disconnette, cioè si suddivide in due grafi* $G_1 = (V_1, E_1)$ *e* $G_2 = (V_2, E_2)$*, entrambi connessi e aciclici.*



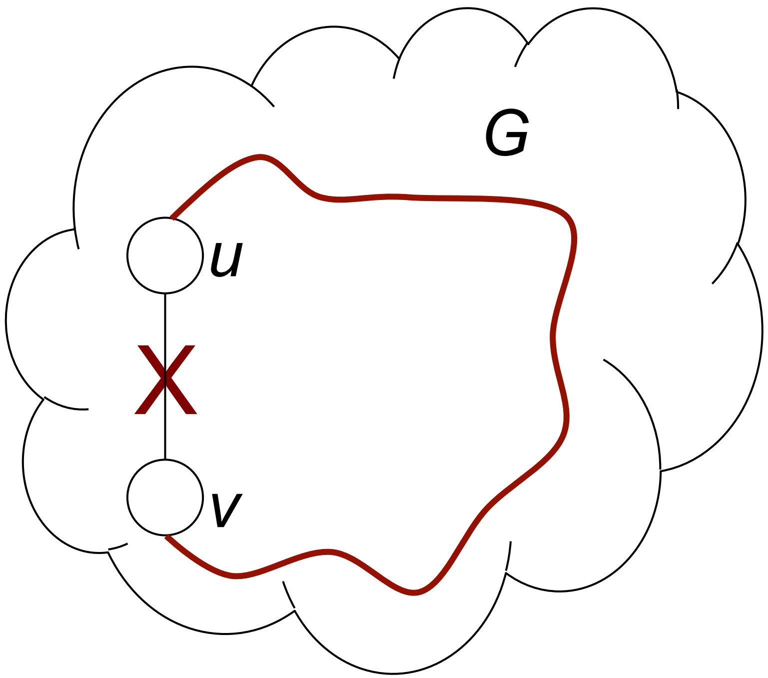
Alberi (5)

Dimostrazione. Per assurdo, dopo l'eliminazione dell'arco *e=(u,v)* il grafo rimane connesso.

Cioè, nel nuovo grafo esiste un cammino da *u* a *v*.

Ma allora, nel grafo originario *G*, tale cammino, con *e*, forma un ciclo, contro l'ipotesi che *G* sia aciclico.

Infine, banalmente entrambe le componenti generatesi devono rimanere connesse e acicliche.



Teorema. *Sia $G=(V,E)$ un grafo. Le seguenti due affermazioni sono equivalenti:*

- *G* è connesso e aciclico (in altre parole, *G* è un albero).
- *G* è connesso ed $|E| = |V| - 1$.

Dim. 1. ⇒ 2. Dimostreremo per induzione che, se *G* è aciclico, allora $|E| = |V| - 1$.

Passo base: se $|V| = 1$ oppure $|V| = 2$ l'affermazione è banalmente vera.

...

Caratterizzazione per gli alberi (2)

segue dim. G è connesso e aciclico $\Rightarrow$ G è connesso ed $|E| = |V| - 1$

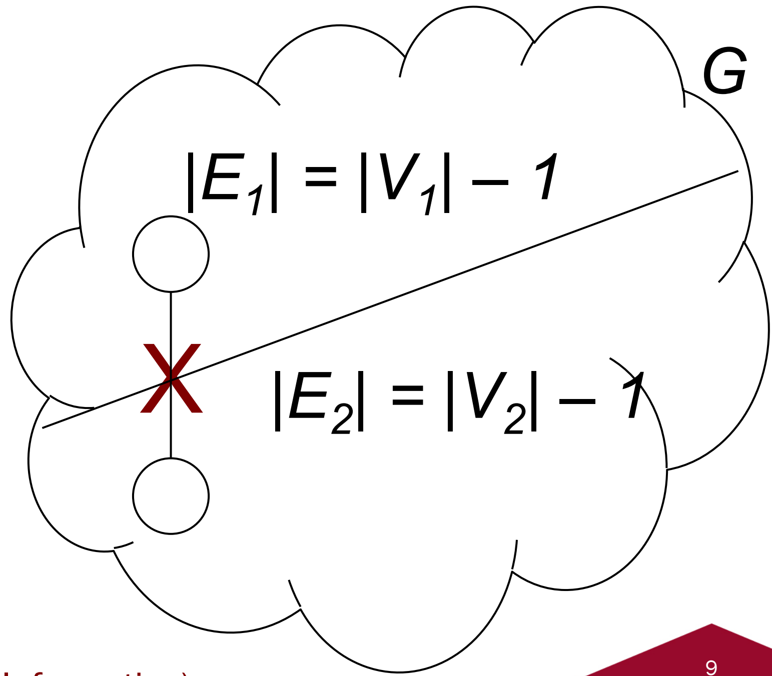
Passo induttivo: rimuovendo un arco qualsiasi, per il lemma provato precedentemente, il grafo G si disconnette in due grafi $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, entrambi connessi e aciclici.

Per essi vale l'hp induttiva:

$$|V| = |V_1| + |V_2|$$

$$|E| = |E_1| + |E_2| + 1 =$$

$$= |V_1| - 1 + |V_2| - 1 + 1 = |V| - 1$$



segue dim. G è connesso ed $|E| = |V| - 1 \Rightarrow G$ è connesso e aciclico

2. ⇒ 1. $G = (V, E)$ connesso, con $|V|=n$ e con $|E| = |V| - 1$
per assurdo contenga un ciclo $v_1, v_2, \ldots, v_k, v_1$.

Consideriamo il grafo $G_k = (V_k, E_k)$ costituito dal solo ciclo.

In esso abbiamo: $|E_k| = |V_k|$

Se $k=n$ assurdo.

Se $k<n$ esiste un nodo v_{k+1} connesso a G_k tramite un arco

$\Rightarrow G_{k+1}$ con $|E_{k+1}| = |V_{k+1}|$.

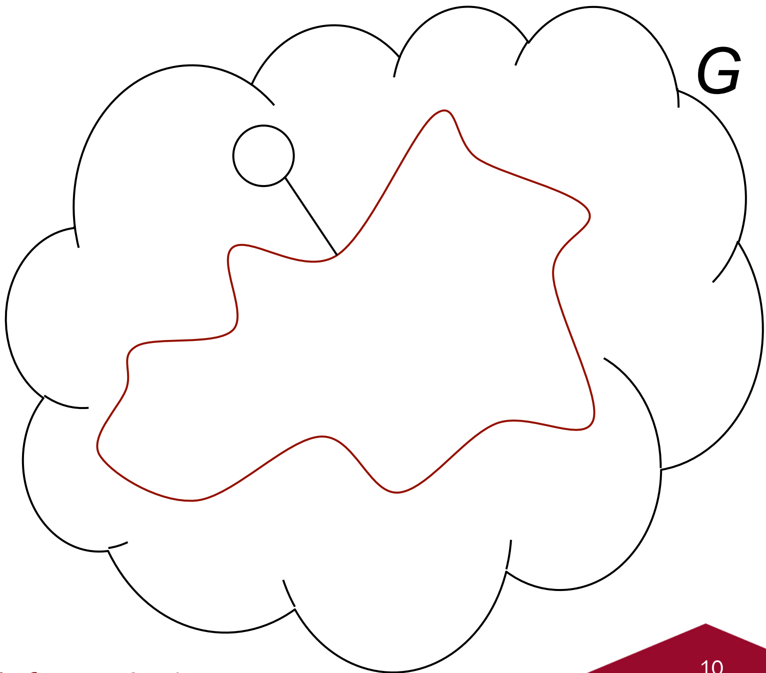
Si prosegue fino a G_n per cui:

$V = V_n$ ed $E \supseteq E_n \Rightarrow |E| \geq |E_n|$

Per ipotesi $|E| = |V| - 1$, per cui:

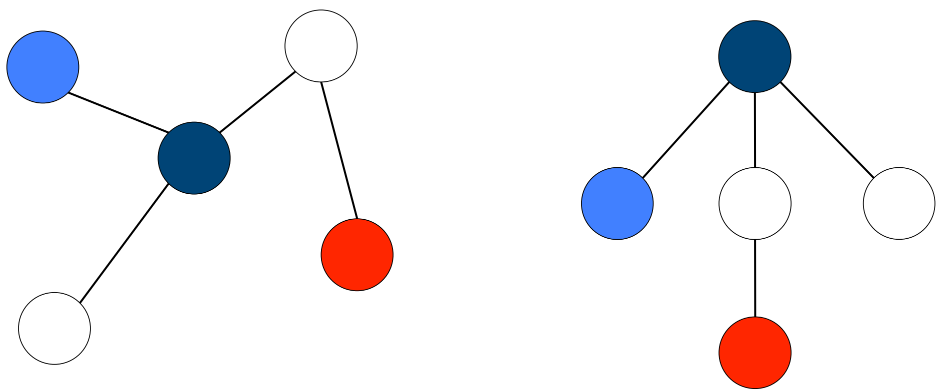
$|V| - 1 = |E| \geq |E_n| = |V_n| = |V|$.

Assurdo.



alberi radicati: in cui si distingue un nodo particolare tra gli altri, detto *radice*.

L'albero radicato si può rappresentare in modo tale che i cammini da ogni nodo alla radice seguano un percorso dal basso verso l'altro, come se l'albero venisse, in qualche modo, "appeso" per la radice.



Alberi radicati (2)

In un albero radicato:

- i nodi sono organizzati in ***livelli***, numerati in ordine crescente allontanandosi dalla radice (di norma la radice è posta a livello zero);
- l'***altezza*** di un albero radicato è la lunghezza del più lungo cammino dalla radice ad una foglia; un albero di altezza h contiene $(h + 1)$ livelli, di norma numerati *da 0 ad h*.

Alberi radicati (3)

In un albero radicato:

- Dato un qualunque nodo *v* di una albero radicato che non sia la radice, il primo nodo che si incontra sul (unico) cammino da *v* alla radice viene detto ***padre di v***
- nodi che hanno lo stesso padre sono detti ***fratelli*** e la radice è l'unico nodo che non ha padre
- ogni nodo sul cammino da *v* alla radice viene detto ***antenato di v***
- tutti i nodi che ammettono *v* come padre sono detti ***figli di v***, ed i nodi che non hanno figli sono detti ***foglie***;
- tutti i nodi che ammettono *v* come antenato vengono detti ***discendenti di v***.

Alberi radicati (4)

Un albero radicato si dice ***ordinato*** se attribuiamo un qualche ordine ai figli di ciascun nodo, nel senso che se un nodo ha *k* figli, allora vi è un figlio che viene considerato primo, uno che viene considerato secondo, …, uno che viene considerato *k*-esimo.

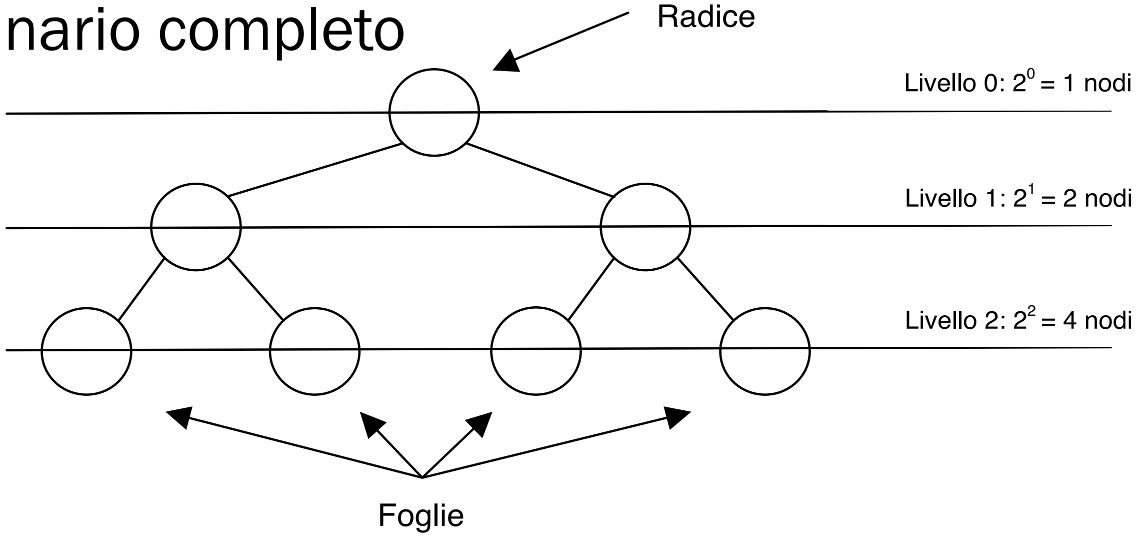
Una particolare sottoclasse di alberi radicati e ordinati è quella degli ***alberi binari***, che hanno la particolarità che ogni nodo ha al più ***due figli***. Poiché sono alberi ordinati, i due figli di ciascun nodo si distinguono in ***figlio sinistro*** e ***figlio destro.***

Alberi binari (1)

Un albero binario nel quale tutti i livelli contengono il massimo numero possibile di nodi è chiamato ***albero binario completo***.

Se invece tutti i livelli tranne l'ultimo contengono il massimo numero possibile di nodi mentre l'ultimo livello è riempito completamente da sinistra verso destra solo fino ad un certo punto, l'albero è chiamato ***albero binario quasi completo***.

In un albero binario completo di altezza *h*:



- il numero delle foglie è 2^h
- il numero dei nodi interni è $\sum_{i=0}^{h-1} 2^i = \frac{2^h - 1}{2 - 1} = 2^h - 1$
- il numero totale dei nodi è $2^h + 2^h - 1 = 2^{h+1} - 1$.

Alberi binari (3)

altezza h di un albero binario completo:

numero totale dei nodi: $n = 2^{h+1} - 1$

da cui: $\log(n+1)=h+1$

cioè: $h=\log(n+1)-1=\log((n+1)/2)$

UNITELMA SAPIENZA
SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Memorizzazione tramite record e puntatori:

Il modo più naturale di rappresentare e gestire gli alberi binari è per mezzo dei puntatori. Ogni singolo nodo è costituito da un record contenente:

key: le opportune informazioni pertinenti al nodo stesso;

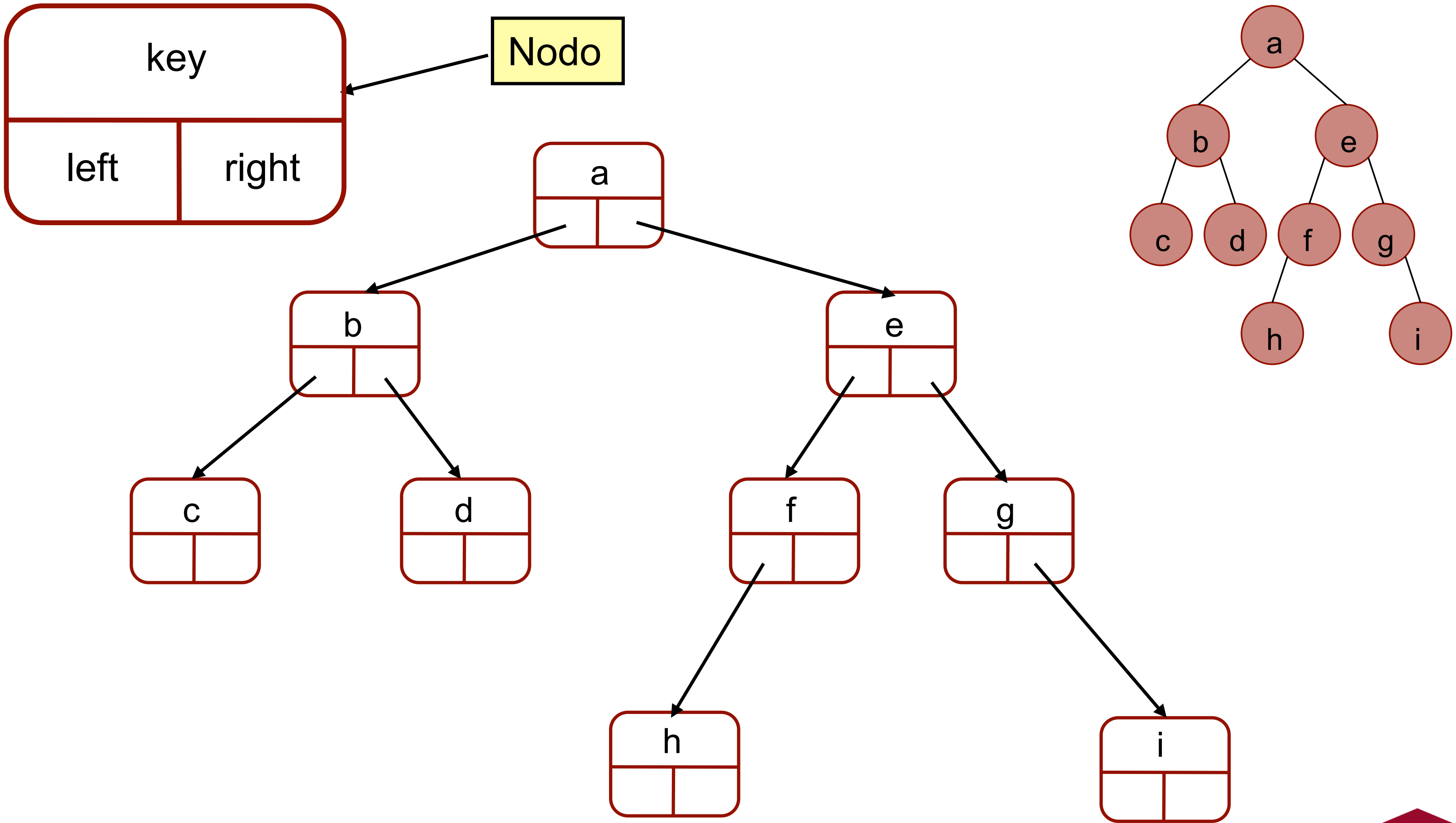
left: il puntatore al figlio sinistro (oppure *NULL* se il nodo non ha figlio sinistro);

right: il puntatore al figlio destro (oppure *NULL* se il nodo non ha figlio destro);

L'albero viene acceduto per mezzo del puntatore alla radice.

UNITELMA SAPIENZA
SAPIENZA Università di Roma
Dipartimento di Informatica

Memorizzazione tramite record e puntatori (segue):



Memorizzazione tramite record e puntatori (segue):

ha tutti i vantaggi e l'elasticità delle strutture dinamiche basate sui puntatori (si possono inserire nuovi nodi, spostare dei nodi ecc.), ma ne presenta svantaggi moltiplicati: l'unico modo per accedere all'informazione memorizzata in un nodo è scendere verso di esso partendo dalla radice e poi spostandosi di padre in figlio, ma non è chiaro se ad ogni passo si debba andare verso il figlio sinistro o verso il figlio destro.

Rappresentazione posizionale:

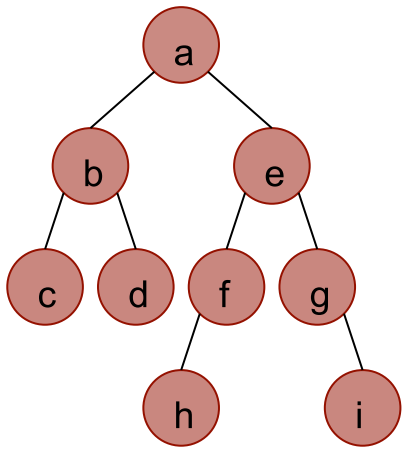
(già discusso in merito all'heap) i nodi vengono memorizzati in un vettore, nel quale la radice occupa la posizione di indice *1* ed i figli sinistro e destro del nodo in posizione *i* si trovano rispettivamente nelle posizioni $2i$ e $2i + 1$.

Svantaggi rispetto alla gestione mediante puntatori:

- richiede di conoscere in anticipo la massima altezza *h* dell'albero e, una volta noto tale valore, richiede l'allocazione di un vettore in grado di contenere un albero binario completo di altezza *h*;
- a meno che l'albero non sia abbastanza "denso" di nodi, si verifica uno spreco di memoria.

Rappresentazione in memoria (5)

Rappresentazione posizionale (segue):



a	b	e	c	d	f	g	0	0	0	0	h	0	0	i

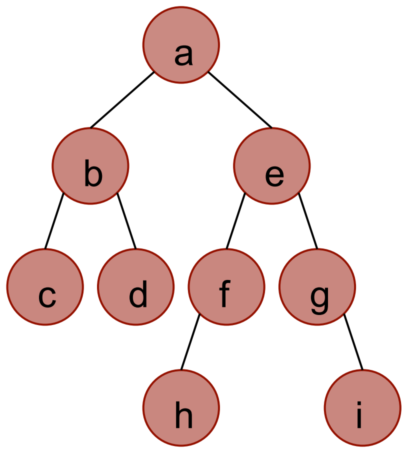
Vettore dei padri:

Costituito da un vettore in cui ogni elemento è associato ad un nodo dell'albero.

Introducendo una biezione tra gli n nodi dell'albero e gli indici $1, \ldots, n$, l'elemento i del vettore contiene l'indice del padre del nodo i nell'albero.

Questo metodo di memorizzazione funziona senza alcuna modifica anche per alberi non necessariamente binari, in cui cioè ogni nodo può avere un numero qualunque di figli.

Vettore dei padri (segue):



1	2	3	4	5	6	7	8	9
a	b	c	d	e	f	g	h	i
/	1	2	2	1	5	5	6	7

Confronto delle strutture dati:

Trovare il padre di un nodo

- struttura a puntatori: al momento non siamo in grado: dobbiamo accedere all'albero tramite il puntatore alla sua radice, ma poi non possiamo scorrere la struttura come fosse una lista, perché non sappiamo se dirigerci a destra o a sinistra ➔ visite
- rappresentazione posizionale: il padre del nodo *i* è banalmente *i/2*
- vettore dei padri: memorizzato in posizione *i*

Confronto delle strutture dati (segue):

Determinare se il nodo abbia 0,1 o 2 figli

- struttura a puntatori: verificare se i campi left e right siano settati a NULL oppure no
- rappresentazione posizionale: vedere se gli elementi di indice *2i* e *2i+1* sono settati a 0 oppure no
- vettore dei padri: scorrere l'intero vettore e contarvi il numero di occorrenze dell'elemento *i*

Confronto delle strutture dati (segue):

Determinare la distanza dalla radice di un nodo

- struttura a puntatori: come nel caso della ricerca del padre di un nodo ➔ visite
- rappresentazione posizionale: il livello del nodo *i* è banalmente la parte intera inf. di *log i*
- vettore dei padri: a partire da *P[i]* risaliamo di padre in padre passando per *P[P[i]], P[P[P[i]]],* ecc. fino a giungere alla radice; ciò richiede tempo proporzionale ad *h.*

- Dire quant'è la massima lunghezza di un vettore che è necessario allocare per poter memorizzare sempre un albero con la rappresentazione posizionale.
- Progettare un algoritmo che, dato un albero binario memorizzato tramite vettore dei padri, restituisca il vettore relativo alla rappresentazione posizionale dello stesso albero. Calcolare il costo computazionale.
  - Progettare un algoritmo che, dato un albero binario memorizzato tramite rappresentazione posizionale, restituisca il vettore dei padri dello stesso albero. Calcolare il costo computazionale.