Laboratorio di Algoritmi

Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano

Lezioni:	Martedì 8.30 - 10.30 in aula 3	Mercoledì 10.30 - 13.30 in aula 2
	Giovedì 15.30 - 18.30 in aula 2	Venerdì 10.30 - 12.30 in aula 3
Ricevimento:	su appuntamento (Dipartimento di Informatica)	
Tel.:	02 503 16235	
E-mail:	roberto.cordone@unimi.it	
Pagina web:	http://homes.di.unimi.it/~cordone/courses/2020-algo/2020-algo.html	

Dizionari

Un dizionario T su un insieme universo ordinato U

- rappresenta un sottoinsieme finito di U: $T \subseteq 2^U$
- consente di eseguire operazioni di ricerca, cioè di indicare se un dato elemento di U appartiene a T oppure no

Altre strutture già trattate possono svolgere queste funzioni:

- tabelle
- tabelle ordinate
- liste
- vettori di incidenza

ma presentano forti svantaggi:

- hanno scarsa efficienza temporale per alcune operazioni
  - ricerca in strutture non ordinate
  - inserimenti e cancellazioni in strutture ordinate
- hanno scarsa efficienza spaziale per insiemi universo U molto grandi (eventualmente, infiniti)

Gli alberi binari di ricerca (*ABR*) e le *tabelle hash* cercano di limitarli

Parleremo solo dei primi

Alberi binari di ricerca: operazioni

Sia $\mathcal{T}$ l'insieme di tutti i possibili *ABR* su U

Gli *ABR* ammettono tipicamente le seguenti operazioni

- ricerca: dato un elemento e un *ABR*, indica se l'elemento fa parte dell'*ABR*

$$\mathrm{memberABR} : U \times \mathcal{T} \to \mathbb{B}$$

È l'operazione che dà il nome alla struttura dati

- verifica di vuotezza: dato un *ABR*, indica se è vuoto

$$\mathrm{ABRvuoto} : \mathcal{T} \to \mathbb{B} \qquad \text{(ovvero } \{0,1\}\text{)}$$

- inserimento: dato un elemento e un *ABR*, inserisce l'elemento nell'*ABR*

$$\mathrm{insertABR} : U \times \mathcal{T} \to \mathcal{T}$$

Non c'è controllo sulla posizione dell'elemento

- cancellazione: dato un elemento e un *ABR*, cancella l'elemento dall'*ABR*

$$\mathrm{deleteABR} : U \times \mathcal{T} \to \mathcal{T}$$

Alberi binari di ricerca: operazioni

Sia $\mathcal{T}$ l'insieme di tutti i possibili *ABR* su U

Gli *ABR* ammettono tipicamente le seguenti operazioni

- calcolo del minimo: dato un *ABR*, ne restituisce l'elemento minimo

$$\mathrm{minABR} : \mathcal{T} \to U$$

Se l'ABR è vuoto, restituisce un valore fittizio $+\infty$

- calcolo del massimo: dato un *ABR*, ne restituisce l'elemento massimo

$$\mathrm{maxABR} : \mathcal{T} \to U$$

Se l'ABR è vuoto, restituisce un valore fittizio $-\infty$

Alberi binari di ricerca: operazioni

In matematica basta definire un oggetto per crearlo

Nelle implementazioni concrete, questo in genere non vale
Quindi è opportuno definire

- creazione: crea un albero binario vuoto

$$\mathrm{creaABR} : () \to \mathcal{T}$$

- distruzione: distrugge un albero

$$\mathrm{distruggeABR} : \mathcal{T} \to ()$$

Si noti la scomparsa del concetto di posizione proprio di altre strutture che rappresentano insiemi

- non si accede agli elementi dell'insieme tramite una posizione, ma solo tramite il loro ordine

Albero binario di ricerca

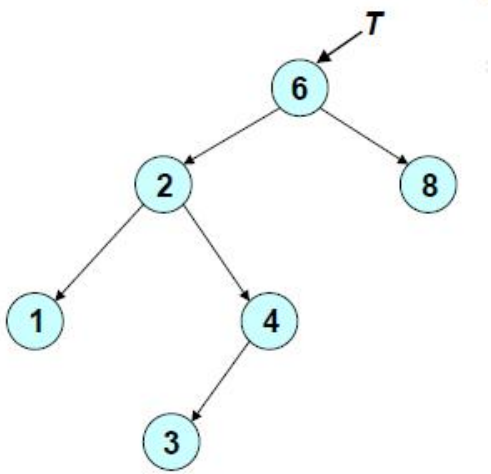
Un albero binario di ricerca (*ABR*) T su un insieme ordinato U è un

- albero binario
- in cui tutti i nodi sono diversi tra loro
- in cui ogni nodo segue tutti quelli del proprio sottoalbero sinistro

$$a_i \succ a_j \quad \text{per ogni } j \in T_s(i) \text{ e per ogni } i \in T$$

- in cui ogni nodo precede tutti quelli del proprio sottoalbero destro

$$a_i \prec a_j \quad \text{per ogni } j \in T_d(i) \text{ e per ogni } i \in T$$



Gli *ABR* gestiscono insiemi non generici, ma totalmente ordinati

ABR: implementazione con puntatori

Gli *ABR* hanno le stesse implementazioni degli alberi binari

Nell'implementazione a puntatori:

- l'albero corrisponde a un puntatore al nodo radice
- ogni elemento dell'albero corrisponde a una struttura con
  - il dato $a \in U$
  - un puntatore alla radice del sottoalbero sinistro (`NULL` se non esiste)
  - un puntatore alla radice del sottoalbero destro (`NULL` se non esiste)
  - un puntatore al nodo padre (`NULL` se non esiste)

```
#define NO_TREE NULL                                   (albero vuoto)

typedef nodo *ABR;                          (l'albero è l'indirizzo della radice)

typedef struct _nodo nodo;
struct _nodo {
   U a;                                        (U è il tipo del nodo generico)
   nodo *Ts;
   nodo *Td;
   nodo *padre;
};
```