Algoritmi (modulo di laboratorio)

Corso di Laurea in Matematica

Roberto Cordone
DI - Università degli Studi di Milano

Lezioni:	Martedì 8.30 - 10.30 in aula 8	Mercoledì 10.30 - 13.30 in aula 309
	Giovedì 16.30 - 18.30 in aula 307	Venerdì 10.30 - 12.30 in aula 4
Ricevimento:	su appuntamento (Dipartimento di Informatica)	
E-mail:	roberto.cordone@unimi.it	
Pagina web:	http://homes.di.unimi.it/~cordone/courses/2024-algo/2024-algo.html	
Sito Ariel:	https://mgoldwurmasd.ariel.ctu.unimi.it	

Lezione 13: Dizionari (alberi binari di ricerca) Milano, A.A. 2023/24

Dizionari

Un dizionario T su un insieme universo U totalmente ordinato

- rappresenta un sottoinsieme finito di U: $\mathcal{T} \subseteq 2^U$
- consente di eseguire operazioni di ricerca, cioè di indicare se un dato elemento di U appartiene a T oppure no
- consente di eseguire operazioni di inserimento e cancellazione, quindi è una struttura dinamica

Altre strutture già trattate svolgono queste funzioni con forti svantaggi:

- le tabelle e le liste hanno scarsa efficienza temporale ($\Theta(|T|)$) per la ricerca
- le tabelle ordinate hanno scarsa efficienza temporale ($\Theta(|T|)$) per inserimenti e cancellazioni
- i vettori di incidenza hanno scarsa efficienza spaziale ($\Theta(|U|)$) per insiemi universo U molto grandi (eventualmente, infiniti)

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

Parleremo solo dei primi

Dizionari: operazioni

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

I dizionari ammettono tipicamente le seguenti operazioni

- ricerca: dato un elemento e un dizionario, indica se l'elemento fa parte del dizionario

$$\text{member} : U \times \mathcal{T} \to \mathbb{B}$$

È l'operazione fondamentale di questa struttura dati

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

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

- inserimento: dato un elemento e un dizionario, inserisce l'elemento nel dizionario

$$\text{insert} : U \times \mathcal{T} \to \mathcal{T}$$

Non c'è controllo sulla posizione dell'elemento

- cancellazione: dato un elemento e un dizionario, cancella l'elemento dal dizionario

$$\text{delete} : U \times \mathcal{T} \to \mathcal{T}$$

Dizionari: operazioni

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

I dizionari ammettono tipicamente le seguenti operazioni

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

$$\min : \mathcal{T} \to U$$

Se il dizionario è vuoto, restituisce un valore fittizio $+\infty$

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

$$\max : \mathcal{T} \to U$$

Se il dizionario è vuoto, restituisce un valore fittizio $-\infty$

Dizionari: operazioni

In matematica basta definire un oggetto per crearlo

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

- creazione: crea un dizionario vuoto

$$\mathrm{crea} : () \rightarrow \mathcal{T}$$

- distruzione: distrugge un dizionario

$$\mathrm{distrugge} : \mathcal{T} \rightarrow ()$$

Scompare il concetto di posizione di altre rappresentazioni di insiemi

- agli elementi del dizionario si accede tramite il loro ordine

Albero binario di ricerca

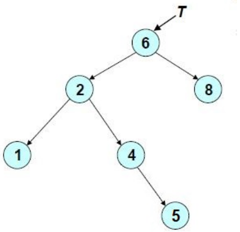
Un albero binario di ricerca (ABR) T su un insieme ordinato U
è un albero binario in cui

1. tutti i nodi sono diversi tra loro
2. ogni nodo segue tutti quelli del proprio sottoalbero sinistro

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

3. 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 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 EMPTY_TREE NULL
```
(*albero vuoto*)

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

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