Algoritmi e strutture dati

Roberto Cordone

A. A. 2015-16

Capitolo 3

Implementazioni dei dizionari ordinati

Nota: queste dispense sono un rapido riassunto delle lezioni svolte nel dicembre 2015 e gennaio 2016. In buona parte ripetono gli argomenti delle dispense del prof. Goldwurm, con alcune aggiunte, qualche piccola modifica nella presentazione, qualche esempio. Chi notasse errori, incoerenze, oscurità o avesse dubbi è invitato a contattarmi per segnalarli.

3.3 Alberi 2-3

- Il problema fondamentale degli alberi binari di ricerca è la possibilità di incontrare il proprio caso pessimo (dati inseriti in modo ordinato), che rendono le operazioni inefficienti, cioè $O(n)$ anziché $O(\log n)$.
- Per evitare questo problema esistono altre strutture la cui topologia viene dinamicamente aggiornata per conservare il bilanciamento.
- Un albero 2-3 si definisce come un albero con
  - foglie tutte allo stesso livello;
  - nodi interni tutti dotati di 2 o 3 sottoalberi: indicheremo i sottoalberi del nodo r con $T_1(r)$, $T_2(r)$ e $T_3(r)$;
  - etichette sulle foglie associate agli elementi di un dizionario: $E(r) \in A$;
  - tutte le foglie di un sottoalbero hanno etichette inferiori a quelle delle foglie di un sottoalbero successivo: $E(u) < E(v)$ per ogni $u \in T_i(r)$, $v \in T_j(r)$ tale che $i < j$ ($i = 1, 2$ e $j = 2, 3$).
- Negli alberi 2-3, le informazioni stanno sulle foglie, e i nodi interni servono solo a guidare la ricerca. Un albero di altezza h (livelli da 0 a h) ha almeno 2^h foglie e al massimo 3^h foglie. Se indichiamo con n gli elementi del dizionario, l'altezza è compresa fra $\log_3 n$ e $\log_2 n$. Se invece indichiamo con n tutti i nodi, compresi quelli interni, l'espressione è più complicata:

$$\log_3 -1 \leq h \leq \log_2 n$$

- Ogni nodo v interno di un albero 2-3 contiene due informazioni ausiliarie:
  - la chiave massima L_v di tutte le foglie del primo sottoalbero $T_1(v)$
  - la chiave massima M_v di tutte le foglie del secondo sottoalbero $T_2(v)$

Un albero 2-3 radicato in v rappresenta una specie di sequenza ordinata divisa in tre sottosequenze; L_v e M_v sono gli elementi massimi delle prime due sottosequenze, cioè i punti di separazione fra le tre sottosequenze. Si noti che si può usare in alternativa la convenzione che L_v e M_v siano le chiavi degli elementi minimi delle ultime due sottosequenze.

- Esempio: disegnare $(1, 2, 3, 4, 6, 8)$ come albero 2-3 di altezza $h = 2$, con tre coppie o due terne di foglie.
- Un'implementazione diretta è quella con puntatori:

```
struct _elemento {
  A *a;
  int L;
  int M;
  struct _elemento *T1;
  struct _elemento *T2;
  struct _elemento *T3;
  struct _elemento *father;
};

typedef struct _elemento *node;
typedef struct _elemento *tree2_3;

tree2_3 T;
```

dove un albero `T` è visto come un puntatore al nodo radice, e i nodi contengono un puntatore `a` all'informazione di tipo `A` (usato solo nelle foglie), due chiavi intere `L` e `M`, massime dei primi due sottoalberi (nel caso delle foglie, entrambe pari alla chiave dell'elemento), tre sottoalberi e un puntatore `father` verso il nodo padre.

- Al solito, è possibile un'implementazione alternativa, che fa uso di un vettore di strutture, ed è praticamente identica, salvo sostituire i puntatori con indici numerici

```
struct _elemento {
  A *a;
  int L;
  int M;
  int T1;
  int T2;
  int T3;
  int father;
};

typedef int node;
typedef struct _elemento *tree2_3;

tree2_3 T;
```

- La funzione `MIN` è identica a quella degli alberi binari: il primo elemento del dizionario certamente è una foglia e sta nel primo sottoalbero di tutti i suoi antenati, per cui si può trovare scorrendo i primi sottoalberi finché ce ne sono:

```
A *MIN(tree_23 T)
{
  if (T == NULL)
    return NULL;
  else
  {
    r = ROOT(T);
    if (r->T1 == NULL)
      return r->a;
    else
      return MIN(r->T1);
  }

}
```

Ne esiste una semplice versione iterativa:

```
A *MIN(tree T)
{
  if (T != NULL)
    while (T->T1 != NULL)
      T = ROOT(T)->T1;

  return ROOT(T)->a;
}
```

- La funzione `MEMBER` (e la funzione `SEARCH` che fa le stesse cose, ma restituisce un puntatore a un elemento anziché un valore logico) sono molto simili alle corrispondenti funzioni per gli alberi binari, salvo che:

  1. i nodi interni possono avere tre figli, per cui la chiave dell'elemento cercato va confrontata con $L(r)$ e $M(r)$, anziché con una singola chiave;
  2. nei nodi interni non ci sono informazioni, per cui non ci si ferma fino alle foglie.

```
A *SEARCH (A *a, tree2_3 T)
{
  if (T == NULL) return NULL;
  k = KEY(a);
  r = ROOT(T);
  if (r->T1 == NULL) /* se il nodo corrente e' una foglia */
  {
    if (k == r->L)
      return r->a;
    else
      return NULL;
  }
  else if (k <= r->L)
    return SEARCH(a,r->T1);
  else if (k <= r->M)
    return SEARCH(a,r->T2);
  else /* (k > r->M) */
```

```
    return SEARCH(a,r->T3);
}
```

- Esempio: si costruisca l'albero 2-3 con una radice avente due figli, ciascuno con due nipoti, e le seguenti foglie: 2, 5 e 9 appese al primo nipote, 23 e 34 appesi al secondo, 45, 47 e 50 appesi al terzo e 67, 80, 85 appesi al quarto. Si cerchino le seguenti chiavi: 1 47, 23, 85, 90.
- La funzione `INSERT` ha bisogno di conoscere il nodo che farà da padre al nuovo elemento. Tale nodo è un nodo interno al livello più basso, subito sopra le foglie, ed è tale che il nuovo elemento cade entro la sua sottosequenza, o subito prima o subito dopo, cioè cade dopo l'ultimo elemento che la precede e prima del primo elemento che le segue.

Figura 3.1: FIGURA DA FARE: DATO L'ALBERO (1,2) (3,4) (6,8), I NODI 2.5, 3 E 5 HANNO TUTTI COME PADRE IL NODO CENTRALE DEL LIVELLO 1, PERCHE' CADONO FRA 2 E 5.

- La funzione `SEARCH_FATHER` determina questo nodo padre. È identica alla funzione `SEARCH`, salvo che
  1. non si ferma in una foglia, ma al livello immediatamente superiore;
  2. quando il terzo sottoalbero è vuoto, scende nel secondo.

  Questo perché non sta cercando l'elemento, ma il posto in cui metterlo (quindi si ferma nel nodo padre potenziale, ed evita i sottoalberi vuoti, in cui non ci sono padri potenziali).

```
node SEARCH_FATHER (A *a, tree2_3 T)
{
  if (T == NULL) return NULL;
  k = KEY(a);
  p = ROOT(T);
  if (p->T1->T1 == NULL) /* se il primo figlio e' una foglia */
    return p;
  else if (k <= p->L)
    return SEARCH_FATHER(a,p->T1);
  else if ( (k <= p->M) || (p->T3 == NULL) )
    return SEARCH_FATHER(a,p->T2);
  else /* (k > p->M) && (p->T3 != NULL) */
    return SEARCH_FATHER(a,p->T3);
}
```

- Trovato il padre potenziale, il nuovo elemento andrebbe semplicemente appeso ad esso nella posizione giusta in base alla chiave:
  - se il nuovo elemento è uguale a uno dei vecchi, non occorre fare nulla;
  - se `k < p->L` (il nuovo elemento ha chiave minore del primo sottoalbero), bisogna scalare in avanti tutti i sottoalberi;

– altrimenti, se `p->L < k < p->M` (il nuovo elemento ha chiave minore del secondo sottoalbero), bisogna scalare in avanti il secondo e terzo sottoalbero;

– altrimenti, se `k > p->M`, bisogna scalare il terzo;

Il problema è che in questo modo i sottoalberi potrebbero diventare quattro!

Occorre quindi una procedura `REDUCE` che rimetta a posto le cose.

```
tree2_3 INSERT (A *a, tree2_3 T)
{
  if (T == NULL)
    T = CREATE_TREE2_3(a);
  else
  {
    p = SEARCH_FATHER(a,T);
    if ( (p->E != p->T1->E) &&
         (p->E != p->T2->E) &&
         (p->E != p->T2->E) )
    {
      APPEND(a,p,T);
      REDUCE(p,T);
    }
  }
  return T;
}
```

- La funzione `REDUCE(node p, tree2_3 T)` ripara l'albero `T` nel caso in cui il nodo `p` abbia quattro figli. L'idea è di spezzare `p` in due nodi fratelli ciascuno con due figli, e poi risalire l'albero per vedere se la scomposizione crea problemi al padre. In caso positivo, si ripete il procedimento. Nel caso della radice, si spezza la radice in due e si aggiunge una superradice che faccia loro da padre. In dettaglio:

  1. si crea un nuovo albero formato da un solo nodo $\hat{p}$;
  2. si appende a $\hat{p}$ gli ultimi due figli di p, tenendo i primi due in p;
  3. se p era la radice dell'albero, ora abbiamo due radici: si crea una nuova radice $\hat{r}$ con figli p e $\hat{p}$ (e terzo sottoalbero vuoto)
  4. se p non era la radice, determiniamo il nodo pp padre di p e aggiungiamo $\hat{p}$ come figlio di pp subito dopo p;
  5. eseguiamo `REDUCE` anche su pp

- Sistemata la topologia, bisogna correggere le etichette `L` e `M` in tutti i nodi da `p` su fino alla radice. Gli altri nodi dell'albero sono intatti.

- La complessità della funzione `INSERT` è $O(h) = O(\log n)$.

- Esempio: dato un albero la cui radice ha tre figli e le seguenti foglie: 2, 5 e 9 appese al primo figlio, 23 e 34 appese al secondo e 45, 47 e 50 appese al terzo, si aggiunga l'elemento con chiave 6:

  1. si cerca il nodo padre, che è il primo figlio della radice;
  2. si appende 6 come suo terzo figlio, ma in totale sono quattro;

3. la funzione `REDUCE` spacca il primo figlio in due, tenendo 2 e 5 appesi al primo e 6 e 9 alla copia;
4. si calcolano le soglie della copia, che sono $L = 6$ e $M = 9$
5. la copia del primo figlio viene appesa alla radice dopo il primo figlio, ma ora in totale sono quattro;
6. la funzione `REDUCE` spacca la radice in due, tenendo primo figlio e copia appesi alla vecchia radice e secondo e terzo figlio appesi alla copia;
7. si calcolano le soglie della copia, che sono $L = 34$ e $M = 50$;
8. si crea una nuova radice e le si appendono la vecchia radice e la copia;
9. si ricalcolano le soglie a partire dal nodo padre di inserimento, che era il primo figlio della radice:
   - le sue nuove soglie sono: $L = 2$ e $M = 5$;
   - le nuove soglie della vecchia radice sono: $L = 5$ e $M = 9$;
   - le soglie della nuova radice sono: $L = 9$ e $M = 50$.

- Esempio: dato un albero la cui radice ha due figli, ognuno con due nipoti e le seguenti foglie: 2, 5 e 9 appese al primo nipote, 23 e 34 appese al secondo, 45, 47 e 50 al terzo e 67, 80 e 85 al quarto, si aggiunga l'elemento con chiave 44:

  1. si cerca il nodo padre, che è il terzo nipote (primo figlio del secondo figlio della radice);
  2. si appende 44 come suo primo figlio, ma in totale sono quattro;
  3. la funzione `REDUCE` spacca il terzo nipote in due, tenendo 44 e 45 appesi al terzo nipote e 47 e 50 alla copia;
  4. si calcolano le soglie della copia, che sono $L = 47$ e $M = 50$;
  5. la copia del terzo nipote viene appesa dopo il terzo nipote; diventano tre figli del secondo figlio e va bene così;
  6. si ricalcolano le soglie a partire dal nodo padre di inserimento, che era il terzo nipote della radice:
     - le sue nuove soglie sono: $L = 44$ e $M = 45$;
     - le nuove soglie del secondo figlio sono: $L = 45$ e $M = 50$;
     - le soglie della radice restano: $L = 34$ e $M = 85$.

- La funzione `DELETE` applica la funzione `SEARCH` per trovare l'elemento.
  - Se non lo trova, non fa nulla.
  - Se lo trova ed è l'unico nodo, l'albero diventa vuoto.
  - Se lo trova, lo cancella. Questo è semplice (il nodo è una foglia), ma può portare a un nodo padre con un solo figlio.
  - Se il padre rimane con un figlio solo, consideriamo i fratelli del padre:
    * se il fratello precedente o successivo ha tre figli, se ne ruba uno
    * se entrambi hanno due figli, si sposta il figlio unico su un fratello con due figli e si cancella il padre. Può essere che questo lasci il nonno con un solo figlio: nel caso, si ripete.
    * se si arriva a una radice con un solo figlio, si cancella la radice.

- Esempio: dato un albero la cui radice ha due figli, ognuno con due nipoti e le seguenti foglie: 2 e 9 appese al primo nipote, 23 e 34 appese al secondo, 45, 47 e 50 al terzo e 67, 80 e 85 al quarto, si cancelli l'elemento con chiave 50:

1. si cerca il nodo padre, che è il terzo nipote (primo figlio del secondo figlio della radice);
2. si cerca il nodo fra i figli e lo si cancella; restano due figli e tutto va bene;
3. si ricalcolano le soglie a partire dal nodo padre di cancellazione, che era il terzo nipote della radice:
   - le sue soglie non cambiano: $L = 45$ e $M = 47$;
   - le nuove soglie del secondo figlio sono: $L = 47$ e $M = 85$;
   - le soglie della radice restano: $L = 34$ e $M = 85$.

- Esempio: dato un albero la cui radice ha due figli, ognuno con due nipoti e le seguenti foglie: 2 e 9 appese al primo nipote, 23 e 34 appese al secondo, 45 e 47 al terzo e 67, 80 e 85 al quarto, si cancelli l'elemento con chiave 47:

  1. si cerca il nodo padre, che è il terzo nipote (primo figlio del secondo figlio della radice);
  2. si cerca il nodo fra i figli e lo si cancella; resta un solo figlio;
  3. si cerca un fratello del padre (terzo nipote) che abbia tre figli: è il quarto nipote;
  4. si sottrae il primo figlio al fratello del padre e lo si appende al padre;
  5. si aggiornano le soglie del fratello del padre: $L = 80$ e $M = 85$;
  6. si ricalcolano le soglie a partire dal nodo padre di cancellazione, che era il terzo nipote della radice:
     - le sue soglie diventano: $L = 45$ e $M = 67$;
     - le nuove soglie del secondo figlio sono: $L = 67$ e $M = 85$;
     - le soglie della radice restano: $L = 34$ e $M = 85$.

- Esempio: dato un albero la cui radice ha due figli, ognuno con due nipoti e le seguenti foglie: 2 e 9 appese al primo nipote, 23 e 34 appese al secondo, 45 e 67 al terzo e 80 e 85 al quarto, si cancelli l'elemento con chiave 9:

  1. si cerca il nodo padre, che è il primo nipote (primo figlio del primo figlio della radice);
  2. si cerca il nodo fra i figli e lo si cancella; resta un solo figlio;
  3. si cerca un fratello del padre (primo nipote) che abbia tre figli: non ce n'è;
  4. si appende l'unico figlio al fratello successivo (secondo nipote), che ora ha tre figli;
  5. si cancella il primo nipote; ora il primo figlio ha un solo figlio;
  6. si cerca un fratello del padre (primo figlio) che abbia tre figli: non ce n'è;
  7. si appende l'unico figlio al fratello successivo (secondo figlio), che ora ha tre figli;
  8. si cancella il primo figlio; ora la radice ha un solo figlio;
  9. si cancella la radice, e il secondo figlio diventa radice;
  10. si ricalcolano le soglie a partire dal fratello del padre cancellato, che era il secondo nipote della radice:
      - le sue soglie diventano: $L = 9$ e $M = 23$;
      - le nuove soglie del secondo figlio, che ora è la radice, sono: $L = 34$ e $M = 50$.