

Algoritmi ricorsivi su alberi binari e su grafi

Lezione 23 di Programmazione 2

Docente: Giuseppe Scollo

Università di Catania, sede di Comiso (RG)
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Studi in Informatica applicata, AA 2007-8

Indice

1. Algoritmi ricorsivi su alberi binari e su grafi
2. attraversamento ricorsivo di alberi
3. attraversamento preordine non ricorsivo
4. attraversamento per livelli non ricorsivo
5. costruzione di tornei
6. alberi binari di ricerca
7. attraversamento di grafi
8. esercizi

attraversamento ricorsivo di alberi

attraversamento di un albero: visita (l'informazione associata a) tutti i vertici dell'albero

per gli alberi binari, si hanno tre discipline di visita ricorsiva, cioè basate sulla struttura ricorsiva dell'albero:

preordine: visita prima il vertice, poi i sottoalberi di sinistra e destra

inordine: visita il sottoalbero di sinistra, poi il vertice, poi il sottoalbero di destra

l'ordine di visita dei sottoalberi può essere scambiato

postordine: visita prima i sottoalberi di sinistra e destra, poi il vertice

definizione ricorsiva di funzione di attraversamento in preordine (btree* = bmlink):

```
void attraversa(bmlink h, void visita(bmlink))
{ if (h == 0) return;
  visita(h); attraversa(h->l, visita); attraversa(h->r, visita); }
```

per ottenere da questa le analoghe definizioni per le altre due discipline ricorsive di attraversamento basta spostare opportunamente l'invocazione di visita(h)

attraversamento preordine non ricorsivo

le tre discipline di visita ricorsiva di alberi binari danno soluzioni a svariati problemi ad es., fra quelli visti, l'algoritmo ricorsivo per il problema delle torri di Hanoi sfrutta la visita inordine, mentre per il problema del righello le tre discipline di visita corrispondono tutte a soluzioni, diverse per l'ordine di tracciamento

le suddette discipline di visita possono essere realizzate anche con algoritmi non ricorsivi, impiegando una pila

ecco ad esempio un algoritmo non ricorsivo per l'attraversamento preordine
l'ordine delle push dei sottoalberi nella pila è l'inverso di quello della loro visita

```
void attraversa(bmlink h, void visita(bmlink))
{ Stack<bmlink> s(max);
  s.push(h);
  while (!s.empty())
  { visita(h = s.pop());
    if (h->r != 0) s.push(h->r);
    if (h->l != 0) s.push(h->l);
  } }
```

attraversamento per livelli non ricorsivo

nessuna delle tre suddette discipline di visita è adeguata al caso in cui si voglia attraversare un albero binario facendo procedere assieme gli attraversamenti dei suoi sottoalberi

si può adoperare in tal caso l'**attraversamento per livelli**, che consiste nel visitare tutti i vertici di ciascun livello prima di passare a quelli del livello successivo

è notevole il fatto che un algoritmo a tal scopo si può ottenere da quello **non ricorsivo** di attraversamento preordine, semplicemente adoperando una coda in luogo della pila, con le corrispondenti operazioni di inserimento e rimozione, dove l'ordine degli inserimenti coincide stavolta con quello della successiva visita, per via della disciplina FIFO

```
void attraversa(btlink h, void visita(btlink))
{ Queue<btlink> q(max);
  q.put(h);
  while (!q.empty())
  { visita(h = q.get());
    if (h->l != 0) s.put(h->l);
    if (h->r != 0) s.put(h->r);
  }
}
```

costruzione di tornei

è frequente la costruzione esplicita di alberi binari in algoritmi ricorsivi *divide et impera* riconsideriamo ad es. il problema della ricerca del massimo fra gli elementi di un array di interi, proposto nella lezione 20: vediamone ora una soluzione basata sulla costruzione di un **torneo**:

albero binario completo, su un insieme ordinato, in cui l'elemento contenuto in ciascun vertice interno è uguale al maggiore degli elementi contenuti nei suoi figli
gli elementi dell'insieme sono inoltre contenuti nelle foglie dell'albero

l'algoritmo è simile a quello già visto nella lezione 20, eccetto che la soluzione è qui prodotta dalla funzione ricorsiva di costruzione del torneo su (una parte de)gli elementi dell'array, che restituisce un puntatore alla radice del torneo costruito

```
struct btree { Elem elem; btree *l, *r;
              btree(Elem x) { elem = x; l = 0; r = 0; } };
typedef btree* btlink;
btlink max(Elem a[], int l, int r)
{ int m = (l+r)/2; btlink x = new btree(a[m]); if (l == r) return x;
  x->l = max(a, l, m); x->r = max(a, m+1, r);
  Elem u = x->l->elem, v = x->r->elem;
  if (u > v) x->elem = u; else x->elem = v;
  return x;
}
```

alberi binari di ricerca

un torneo è un caso particolare di una struttura dati molto utile per la realizzazione di efficienti algoritmi di ricerca su insiemi ordinati:

un **albero binario di ricerca**, ingl. Binary Search Tree (**BST**), è un albero binario ordinato, su un insieme ordinato di elementi, in cui l'elemento contenuto in ciascun vertice interno è maggiore o uguale a tutti gli elementi contenuti nei vertici del suo sottoalbero di sinistra e minore o uguale a tutti gli elementi contenuti nei vertici del suo sottoalbero di destra

è facile intuire come, ad es., l'algoritmo di ricerca binaria di un elemento in un array ordinato possa tradursi in un algoritmo di ricerca in un **BST**

l'efficienza della ricerca in un **BST** è tanto migliore quanto più il **BST** è prossimo alla condizione di bilanciamento

si può ottimizzare il bilanciamento di un **BST** scegliendo opportunamente la radice nel corrispondente albero libero un problema che spesso si presenta è quello di mantenere un bilanciamento ottimale, o almeno accettabile, di un **BST** a seguito di numerose operazioni di inserimento o cancellazione di vertici

attraversamento di grafi

concludiamo con dei cenni sulla generalizzazione degli algoritmi (ricorsivi e non) visti per l'attraversamento di alberi binari all'attraversamento di grafi, ad es. rappresentati da liste di adiacenza

la differenza essenziale sta nel fatto che nell'attraversamento di alberi **non può accadere di rivisitare lo stesso vertice**, grazie alla proprietà caratteristica degli alberi, mentre nell'attraversamento di un grafo occorre tener traccia delle visite effettuate, per evitare di ripeterle (e dunque per garantire la terminazione dell'algoritmo)

si possono distinguere due discipline di attraversamento:

in profondità, ingl. depth-first, che generalizza l'attraversamento ricorsivo di alberi

in ampiezza, ingl. breadth-first, che generalizza l'attraversamento per livelli di alberi (non ricorsivo)

analogamente al caso degli alberi, per i grafi:

l'attraversamento in profondità può realizzarsi con un algoritmo ricorsivo, oppure con uno non ricorsivo basato sull'uso di una pila

nel secondo caso, preferibilmente con una disciplina di gestione dei duplicati, che ne eviti l'inserimento nella pila, per mantenere la dimensione della stessa proporzionale al numero di vertici piuttosto che di archi

l'attraversamento in ampiezza può realizzarsi con un algoritmo non ricorsivo, basato sull'uso di una coda

esercizi

1. definire una funzione non ricorsiva per l'attraversamento postordine di un albero binario, adoperando una pila
2. definire una funzione non ricorsiva per l'attraversamento inordine di un albero binario, adoperando una pila
3. esercizio 5.81 del testo (Sedgewick, 2003), p.250: mostrare che, se una foresta è rappresentata da un albero binario secondo la corrispondenza biunivoca indicata nella lezione precedente, l'attraversamento preordine della foresta si realizza con l'attraversamento preordine dell'albero binario, mentre quello postordine della foresta si realizza con quello inordine dell'albero binario
4. esercizi 5.88 e 5.89 del testo (Sedgewick, 2003), p. 256: definire una funzione ricorsiva che calcoli la lunghezza del cammino interno di un albero binario, e determinare per induzione il numero di invocazioni ricorsive effettuate a seguito della sua prima invocazione (inclusa nel numero)
5. definire una funzione ricorsiva che rimuova da un torneo tutti i vertici che contengono un elemento di valore inferiore a un valore dato, con rilascio della memoria ad essi allocata
6. esercizio 5.98 del testo (Sedgewick, 2003), p. 262: definire una funzione non ricorsiva, basata sull'uso di una pila, per l'attraversamento in profondità di un grafo rappresentato da liste di adiacenza