ESERCITAZIONI DI *INTRODUZIONE AGLI ALGORITMI*
(A.A. 08/09)

DISPENSA N. 5

Esercizi su alberi e alberi di ricerca

Esercizio 1 (Nodi di altezza k in albero binario.)
Se u è un nodo di un albero binario T (usiamo da ora in poi la notazione $u \in T$ per indicare questo fatto), definiamo *altezza di u in T* (denotata con $alt^T(u)$) come la massima lunghezza di un cammino da u a una foglia di T. Progettare un algoritmo che prenda in input un albero binario T e un intero k e restituisca il numero dei nodi di altezza k in T. Usare la rappresentazione di T con puntatori a figli. Indichiamo con T_{sin} e T_{des} rispettivamente il sottoalbero sinistro e destro dell'albero binario T.

Soluzione. Osserviamo quanto segue. I nodi di un albero binario T sono così ripartiti: (i) nodi di T_{sin}, (ii) nodi di T_{des}, (iii) radice r di T. Se un nodo ha altezza k nel sottoalbero sinistro o destro allora ha altezza k anche in T (l'altezza dipende solo dalla lunghezza del massimo cammino da quel nodo a una foglia, e questa non cambia se vediamo il nodo come nodo di T_{sin} o di T). Dunque tutti i nodi di altezza k in T_{sin} e tutti i nodi di altezza k in T_{des} sono anche nodi di altezza k in T, e vanno contati. Inoltre, dobbiamo contare anche la radice r di T se e solo se ha altezza k. Se indichiamo con $S(T,k)$ il numero dei nodi di altezza k nell'albero T, vale la seguente relazione.

$$S(T,k) = \begin{cases} S(T_{sin},k) + S(T_{des},k) + 1 & \text{se } alt^T(r) = k \\ S(T_{sin},k) + S(T_{des},k) & \text{altrimenti.} \end{cases}$$

In parole povere, il risultato della funzione S per cui vogliamo scrivere un programma, su input (T,k) è uguale al valore della stessa funzione S su (T_{sin},k) più il valore della stessa funzione S su (T_{des},k), $+1$ se la radice è di altezza k. Questa analisi sembra indicare che una visita in *post-ordine* (figlio sinistro, figlio destro, radice) fa al caso nostro. Resta da vedere come raccogliere l'informazione che serve a decidere se dobbiamo aggiungere $+1$ o no al risultato della somma delle due chiamate ricorsive. Per questo ci serve e ci basta poter calcolare l'altezza della radice dell'albero in input. Ma come si calcola l'altezza di un nodo u? Indicando con $figlio.sin(u)$ e $figlio.des(u)$ rispettivamente il figlio sinistro e il figlio destro del nodo u, abbiamo quanto segue.

$$alt(u) = \max\{alt(figlio.sin(u)), alt(figlio.des(u))\} + 1.$$

Dunque anche per calcolare l'altezza di di un nodo ci serve una visita in post-ordine. Possiamo allora usare una unica visita in post-ordine per risolvere il problema. Il

Note preparate da Lorenzo Carlucci, carlucci@di.uniroma1.it.

nostro algoritmo per $S(T,k)$ restituisce in output l'altezza della radice di T e calcola (accumulandola in una variabile esterna *tot*) il numero dei nodi di altezza k in T_{sin} e in T_{des}. Se la radice di T ha altezza k, si conta anche questa in *tot*, altrimenti no.

```
f(nodo u) → intero {
  if (figlio.sin(u)=null) then a = −1
  else a = f(figlio.sin(u))
  if (figlio.des(u)=null) then b = −1
  else b = f(figlio.des(u))
  d = max (a, b) + 1
  if d = altezza then totale = totale + 1
  return d
  }
```

La funzione f qui sopra viene usata come funzione ausiliaria da una funzione che, chiamata su input T, inizializza $altezza = k$ e $totale = 0$, invoca f sulla radice dell'albero T, e restituisce il valore finale accumulato in *totale*.

Esercizio 2 (Distanza minima in albero binario di ricerca)

Per due numeri interi m e n chiamiamo *distanza tra m e n* (che denotiamo con $\delta(m,n)$) il valore assoluto della differenza tra m ed n, ossia poniamo

$$\delta(m,n) = |m-n|.$$

Sia T un albero binario di ricerca. Indichiamo con $\delta_{min}(T)$ la minima distanza di due etichette di due nodi di T. Progettare un algoritmo che, preso in input un albero binario di ricerca T, restituisca il valore di $\delta_{min}(T)$.

Soluzione. A prima vista calcolare la distanza minima richiede molti confronti, ossia i confronti due a due di tutte le etichette dei nodi di T (Esercizio: quanti sono?). Infatti, per definizione,

$$\delta_{\min}(T) = \min\{\delta(m,n) \text{ per } m,n \text{ etichette di } T\}.$$

Sappiamo però che T è un albero di ricerca. Cerchiamo di sfruttare questa proprietà aggiuntiva. Sappiamo che se visitiamo *in ordine* (visita simmetrica) un albero di ricerca, allora incontriamo le sue etichette in ordine non decrescente:

$$e_1 \le e_2 \le \cdots \le e_N,$$

dove N è il numero di nodi di T. Come possiamo sfruttare questo ordinamento per calcolare $\delta_{\min}(T)$ facendo meno confronti di quanto suggerito dalla sua definizione formale? Osserviamo che, se $a \le b \le c$, allora $|a-b| \le |a-c|$ (Esercizio: dimostrare). La distanza tra due elementi non consecutivi è maggiore o uguale della distanza tra due elementi consecutivi. Dunque, se possiamo accedere alle etichette di T in modo non decrescente, possiamo limitarci a confrontare le etichette consecutive due a due scorrendo da sinistra a destra. Abbiamo la scelta se usare una struttura di appoggio (visitare in ordine l'albero salvando le etichette in un vettore, visitare il vettore confrontando gli elementi consecutivi per trovare la distanza minima), oppure no (fare tutto insieme: visita e confronto due a due degli elementi consecutivi). (Domanda: cambia la complessità di tempo dei due approcci? Cambia la complessità di spazio?). Di seguito uno schema della soluzione senza struttura d'appoggio.

```
DistanzaMinima(albero T) → intero {
  DistanzaMinima(T.sin)
  distanza = |precedente − T.info|
  minimo = min (distanza, minimo)
  precedente = T.info
  DistanzaMinima(T.des)
}
```

Qui sopra *minimo* e *precedente* sono variabili globali (Domanda: come inizializzarle?). Occorre aggiungere i dettagli per trattare il caso foglia. Il risultato viene calcolato nella variabile *minimo*.

Morale dell'Esercizio. Sfruttare le proprietà della struttura dati in questione! Se il problema è ristretto ad alberi di ricerca, sfruttare le buone proprietà degli alberi di ricerca (in questo caso la proprietà di ricerca).

Esercizio 3 (Grado medio) Progettare un algoritmo che, preso in input un albero generico T, restituisca in output il valore del *grado medio* di un nodo in T. Usare la rappresentazione di T con puntatore a primo figlio/fratello.

Soluzione. Per cominciare, definiamo il grado medio come la media tra i gradi dei nodi di T e il numero dei nodi di T. Indichiamo con $grado(u)$ il grado del nodo u, i.e., il numero dei figli di u. Formalmente, abbiamo quanto segue.

$$gm(T) = \frac{\sum_{u \in T} grado(u)}{(\text{numero nodi in } T)}.$$

Possiamo allora pensare di scrivere un algoritmo per il calcolo del grado medio come segue: calcoliamo la somma dei gradi dei nodi di T, calcoliamo il numero dei nodi di T, dividiamo le due quantità. Possiamo scorrere l'albero da cima a fondo e calcolare scendendo le due quantità. Possiamo però semplificare l'algoritmo facendo una semplice osservazione, ossia ridescrivendo la somma dei gradi dei nodi in T. Il grado di un nodo u di T è il numero degli archi che escono da u (il numero dei figli di u). Quando contiamo i gradi dei nodi di T stiamo contando il numero di archi di T.

$$\sum_{u \in T} grado(u) = \text{ numero di archi di } T.$$

Ma il numero degli archi di un albero T è il numero dei nodi di T - 1 (Esercizio: dimostrare questa proprietà!). Dunque

$$\sum_{u \in T} grado(u) = \text{ (numero dei nodi di } T) - 1.$$

Allora per risolvere il nostro problema basta scrivere una procedura per contare il numero dei nodi di T, e poi calcolare

$$gm(T) = \frac{\text{numero dei nodi di } T}{(\text{numero dei nodi di } T) - 1}.$$

Morale dell'Esercizio. Sfruttare le proprietà matematiche (non necessariamente algoritmiche) della struttura può semplificare molto l'algoritmo!

Esercizio 4 (Somma dei Cammini) Sia T un albero generico. Chiamiamo *somma dei cammini* di T la somma delle lunghezze dei cammini da un nodo di T alla radice di T. Se u è un nodo di T, indichiamo con $lc^T(u)$ la lunghezza del cammino da u alla radice di T. La somma dei cammini di T, che indichiamo con $sc(T)$ è allora definita come segue.

$$sc(T) = \sum_{u \in T} lc^T(u).$$

Progettare un algoritmo che, preso in input un albero T, restituisca in output il valore di $sc(T)$. Usare la rappresentazione di T con lista figli.

Soluzione. Analizziamo la funzione per cui vogliamo progettare un algoritmo. Supponiamo che l'albero T sia composto da d sottoalberi $T_1, \ldots, T_d$. Cerchiamo di esprimere la relazione tra i valori della funzione sui sottoalberi e il valore della funzione su T. Per ogni sottoalbero immediato T_i di T, per ogni nodo u in T_i, vale quanto segue.

$$lc^T(u) = lc^{T_i}(u) + 1.$$

Ogni nodo u in T_i contribuisce alla somma dei cammini di T_i con un contributo uguale a $lc^{T_i}(u)$ e contribuisce alla somma dei cammini di T con un contributo uguale a $lc^{T_i}(u) + 1$ (il cammino dal nodo u in T_i alla radice di T è lungo quanto il cammino dal nodo u alla radice di T_i più la lunghezza del cammino dalla radice di T_i alla radice di T, che è uguale a 1). Se la somma dei cammini di T_i è

$$sc^{T_i}(u) = \sum_{u \in T_i} lc^{T_i}(u),$$

il contributo di T_i a $lc(T)$ è invece

$$1 + \sum_{u \in T_i} (lc^{T_i}(u) + 1) = 1 + \sum_{u \in T_i} lc^T(u),$$

dove il $+1$ esterno indica il cammino dalla radice di T_i alla radice di T. L'ultima formula ci dà l'idea per un algoritmo: per ogni sottoalbero immediato T_i di T, il contributo di T_i alla somma cammini di T è uguale alla somma delle lunghezze dei cammini dai nodi di T_i alla radice di T, $+1$.

```
LunghezzaCammini(nodo u) {
  SommaDistanze = SommaDistanze + DistanzaCorrente
  DistanzaCorrente = DistanzaCorrente + 1
  Per ogni v in L(u)
    LunghezzaCammini(nodo v)
  DistanzaCorrente = DistanzaCorrente - 1
}

SommaCammini(nodo u) → intero {
  SommaDistanze = 0
  DistanzaCorrente = 0
  LunghezzaCammini(nodo u)
  return SommaDistanze
}
```