Alberi generici

Appello del 3 luglio 2017

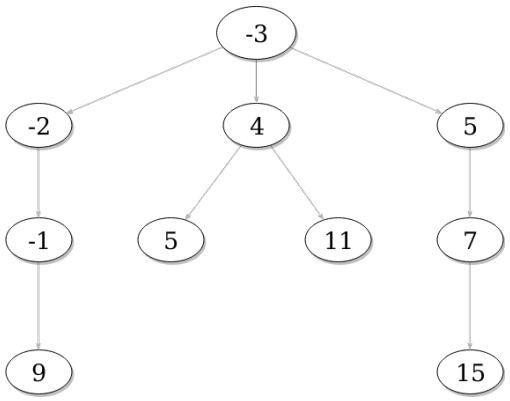
Consegna . Dato un albero generico contenente valori interi (positivi e negativi), si scriva un algoritmo con prestazioni lineari nel numero dei nodi che verifichi se tutti i nodi soddisfano la seguente proprietà: i valori dei figli sono maggiori del valore del padre e per ogni nodo il valore dei fratelli di destra è maggiore del valore del nodo stesso. a) Si descriva brevemente l'algoritmo a parole; b) Si descriva l'algoritmo in linguaggio C++;

Soluzione Parte a) Ho ipotizzato di avere un albero generico (come da consegna) i cui nodi sono dotati di due puntatori (`sotto`, che punta al primo figlio o a `NULL`, e `prox`, che punta al "fratello"). Q è una coda, inizialmente vuota, su cui è definita la funzione length che ne calcola la lunghezza. nodo è stato definito come un nodo dell'albero:

```
-------- -------- -------
| sotto | valore | prox |
-------- -------- -------
```

I nodi vengono memorizzati in una coda `Q` (viene effettuata infatti una visita in ampiezza) e, una volta che inizia la visita ai figli di un nodo, questo viene eliminato dalla coda (`dequeue`).
Per ogni nodo, tranne la radice, vengono verificate le condizioni imposte dalla consegna. In particolare, se un figlio ha un valore minore di quello del padre, l'algoritmo termina indicando che una delle condizioni non è stata rispettata. Si procede analogamente per l'altra condizione.
Non diventa necessario verificare che ogni figlio sia maggiore del padre perché se il primo figlio non è maggiore del padre viene dato errore, se un altro figlio (non il primo) fosse minore del padre necessariamente sarebbe anche minore del fratello di sinistra, infatti se un numero a è minore di un altro numero b sicuramente sarà minore di un numero $b + k$ con $k \in \mathbb{N}$. In quel caso quindi si ricadrebbe in un'altra tipologia di errore. Visto che il problema non richiede di distinguere le tipologie di errore il messaggio di errore è lo stesso e non mi sono preoccupato di memorizzare il valore del padre, operazione che richiederebbe un consumo (anche se non particolarmente elevato) di memoria.

Questo è un esempio di albero che rispetta le condizioni del problema:



Parte b)

```
1  void main()
2  {
3    Q.enqueue(radice);
4
5    while(!Q.empty())
6    {
7      k = Q.dequeue();
8
9      if(k->sotto()->valore <= k->valore)
10     {
11       cout << "La condizione non e' stata rispettata." << endl;
12       return;      // Eventualmente potrei svuotare la coda, ma non e' richiesto
             esplicitamente
13     }
14     else
15     {
16       Q.enqueue(k->sotto());
17     }
18
19     nodo temp = k->sotto();
20     while(temp->prox()!=NULL)
21     {
22       if(nodo->prox()->valore<=nodo->valore)
23       {
24         cout << "La condizione non e' stata rispettata." << endl;
25         return;
26       }
27       else
28       {
29         Q.enqueue(temp->prox());
30         temp = temp->prox();
31       }
32     }
33
34
35   }
36
37   cout << "Tutte le condizioni sono state rispettate." << endl;
38   return;
39 }
```