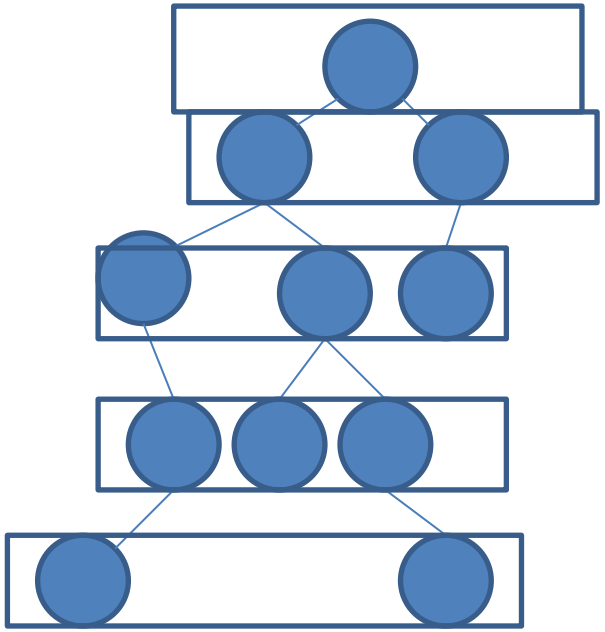
esercizio 1

esercizio assegnato venerdì 29/5 e poi discusso in classe

vogliamo scrivere un programma capace di eseguire una ricerca di un nodo con campo info=y in un albero binario percorrendo i nodi dell'albero in larghezza (breadth-first) o a livelli. Cosa significhi è spiegato nella seguente figura:



prima la radice, poi i suoi figli, poi i figli dei figli e così via per ogni livello i nodi sono considerati da sinistra a destra

nel seguito facciamo riferimento alla figura della slide precedente:

per percorrere i nodi in larghezza, costruiremo una lista concatenata in cui inizialne metteremo la radice, poi, toglieremo la radice e metteremo i suoi figli, poi toglieremo il figlio sinistro (che si trova all'inizio) e metteremo in fondo alla lista i suoi figli, all'inizio della lista ora c'è il figlio destro della radice, lo togliamo e mettiamo in fondo alla lista i suoi figli. A questo punto sulla lista ci sono i 3 nipoti della radice, insomma tutti i nodi a livello 2. Dovrebbe essere chiaro che continuando così percorreremo tutti i nodi per livelli.

il nome FIFO sta per First In First Out, cioè “chi arriva prima viene servito prima”, insomma si tratta di una coda di attesa normale
la coda FIFO è costituita da valori di tipo punt che puntano ai nodi di un albero binario di tipo nodo

diremo che un valore x di tipo FIFO è corretto se i suoi due campi puntato al primo e all’ultimo nodo di una lista di tipo punt corretta.
Con lista(x) indicheremo questa lista che può anche essere vuota (nel qual caso primo e fine sarebbero 0).

per fare quanto appena descritto conviene avere una lista concatenata in cui sia “facile” togliere il prio nodo e aggiungere dei nodi alla fine.
Un modo di rendere facili queste 2 operazioni è di avere una lista concatenata in cui oltre all’inizio si mantenga anche un puntatore all’ultimo nodo della lista.
Per questo definiamo una struttura come segue:
struct FIFO{punt*primo,*fine;};
dove punt è la seguente struttura:
struct punt{nodo* p; punt* next;};
e finalmenente
struct nodo{int info; nodo*left,*right;}
assumeremo che tutti questi tipi abbiano costruttori utili

servono 2 funzioni che realizzino le 2 operazioni che ci servono sulla coda FIFO: togli_prima e metti_fondo.
Esse devono rispettare i seguenti prototipi:

PRE=(x è corretto e lista(x) non è vuota)
FIFO togli_primo(FIFO x)
POST=(restituisce un nuovo valore v di tipo FIFO corretto e tale che lista(v) = L', dove lista(x)=a::L' e il nodo a è stato deallocato)

PRE=(x è corretto, a punta ad un nodo)
FIFO metti_fondo(FIFO x, nodo* a)
POST=(restituisce v di tipo FIFO corretto con lista(v) = lista(x)@punt(a,0))

Una volta realizzate queste 2 funzioni la ricerca in larghezza su un qualsiasi albero binario diventa facile. Bastano 2 funzioni:

1) una funzione che costruisce un valore FIFO x tale che lista(x) è costituita da un solo nodo punt che punta alla radice dell'albero
2) una funzione capace di ripetere ricorsivamente queste operazioni: prendere un valore FIFO x, togliere il primo nodo n da lista(x), e inserire in fondo a lista(x) dei valori punt che puntano ai nodi dell'albero che sono i figli del nodo puntato da n. Ogni volta che questa funzione toglie un nodo dall'inizio di lista(x), controlla se il nodo appena considerato ha il campo info=y . In questo caso la ricerca termina.

la funzione (1) la chiamiamo f1 e la (2) f2, esse obbediscono a queste specifiche:
PRE=(albero(r) è corretto, y è definito)
FIFO f1(nodo*r, int y)
POST=(restituisce il valore FIFO x che è corretto e tale che lista(x) è costituita da un solo nodo che punta a r)

PRE=(x è corretto, y definito)
nodo*f2(FIFO x, int y)
POST=(stampa i campi info di tutti i nodi degli alberi con radice in lista(x) attraversati per livelli fino a trovarne uno con campo info=y o fino ad esaurire tutti i nodi. Nel primo la lista corrente va deallocata)
la POST di f2 non è precisa, ma almeno rende l'idea

per capire esattamente l’output richiesto, si guardi ai test automatici