Algoritmica, Corso A e B
Soluzione Compitino del 17/12/2008

Esercizio 1 **[punti 12]**

In un albero binario di ricerca il **vicino** del nodo **x** è il nodo definito nel modo seguente:

a) Se il sottoalbero sinistro e destro sono diversi da **null**, il **vicino** è il nodo scelto tra predecessore e successore con chiave più vicina a quella di **x.**
b) Se uno solo dei sottoalberi di x è diverso da **null**, il **vicino** è il predecessore (o il successore) se il sottoalbero sinistro (oppure destro) è diverso da **null.**
c) Se ambedue i sottoalberi sono **null**, il **vicino** è uguale a **null.**

Scrivere il codice di un algoritmo lineare nel numero di nodi dell'albero, che non usi memoria aggiuntiva, che, per ogni nodo **x** di un albero **t,** stampi il suo **vicino**.
Giustificare brevemente la complessità ottenuta.

La procedura Vicino (u) viene chiamata sulla radice t dell'albero e restituisce due parametri <min, max> cioè il riferimento al minimo e al massimo elemento del sottoalbero di cui u è radice.

```
Vicino(u)
IF(u == Null) return <Null, Null> ;
   ELSE {
   <minsx, maxsx> Vicino(u.sx);
   <mindx, maxdx> Vicino(u.dx);
   IF (u.sx == Null)&& (u.dx == Null){
          Print Null; min = u; max = u;
          }
    ELSE {IF (u.sx =Null) {
              Print mindx; min = u; max = maxdx;
              }
           ELSE IF (u.dx =Null) {
                Print maxsx; min = minsx; max = u;
                 }
             } ELSE {
               IF (u.dato – maxsx) < (mindx – u.dato)
                   Print  maxsx; ELSE Print mindx;}
                   min = minsx; max =maxdx;}
           }
     RETURN < min, max >;
```

La complessità di Vicino è lineare nel numero di nodi dell'albero in quanto esegue una visita in ordine posticipato e, per ogni nodo, esegue operazioni in tempo costante.

Esercizio 2 **[punti 8]**

Si consideri un problema dello Zaino in cui ci sono 3 elementi: articolo1, articolo2 e articolo3, il cui valore è rispettivamente di 60, 100 e 120 euro e il cui peso è di 1, 2 e 3 kg e lo zaino può sopportarne al massimo 5. Mostrare:

1) La tabella di programmazione dinamica che conduce alla soluzione ottima.
2) La soluzione trovata dall'algoritmo.

	0	1	2	3	4	5
0	**0**	**0**	**0**	**0**	**0**	**0**
art1	**0**	**60**	**60**	**60**	**60**	**60**
art2	**0**	**60**	**100**	**160**	**160**	**160**
art3	**0**	**60**	**100**	**160**	**180**	**220**

La soluzione trovata dall'algoritmo corrispondente alla casella M[3] [5] è costituita dagli elementi {art2, art3} e si ricava sull'array partendo da M[3] [5] , passando da M[2] [2] e terminando a M[1] [0].

Esercizio 3 **[punti 12]**

Una tabella hash con liste di trabocco, è organizzata in modo tale che, quando una chiave viene cercata viene spostata alla testa della sua lista di appartenenza. Scrivere il codice dell'algoritmo per la ricerca della chiave k così modificato.
Discutere brevemente quando questa strategia può risultare conveniente.

```
Ricerca(k):
h = Hash (k);
p = Tabella [h];
IF (p = = Null || p.dato == k) RETURN p;
WHILE (p.succ != Null) && (p.succ.dato != k) p = p.succ;
IF (p.succ = = Null) RETURN Null;
temp = Tabella[h];
Tabella[h]= p.succ;
p.succ = p.succ.succ;
Tabella[h].succ = temp;
RETURN Tabella[h];
```

Un'organizzazione di questo tipo è conveniente quando gli elementi della tabella non sono cercati in modo equiprobabile, quindi i più cercati tendono a spostarsi all'inizio delle loro liste, se le ricerche sono abbastanza numerose, cioè quando sono di ordine quadratico rispetto alla lunghezza media delle liste.