FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

13 gennaio 2009 - ANNO ACCADEMICO 2007/08

NOME: **COGNOME:** **MATRICOLA:**

SCRITTO RIDOTTO: **SI** **NO** **(barrare SI se si intende svolgere lo scritto ridotto)**

Esercizio 1

Sia dato il problema di cercare l'elemento minimo di un'insieme di numeri naturali. Indicare la complessità del migliore algoritmo che risolve il problema, al variare della struttura dati utilizzata per memorizzare l'insieme.
Scrivere l'algoritmo per il caso d.

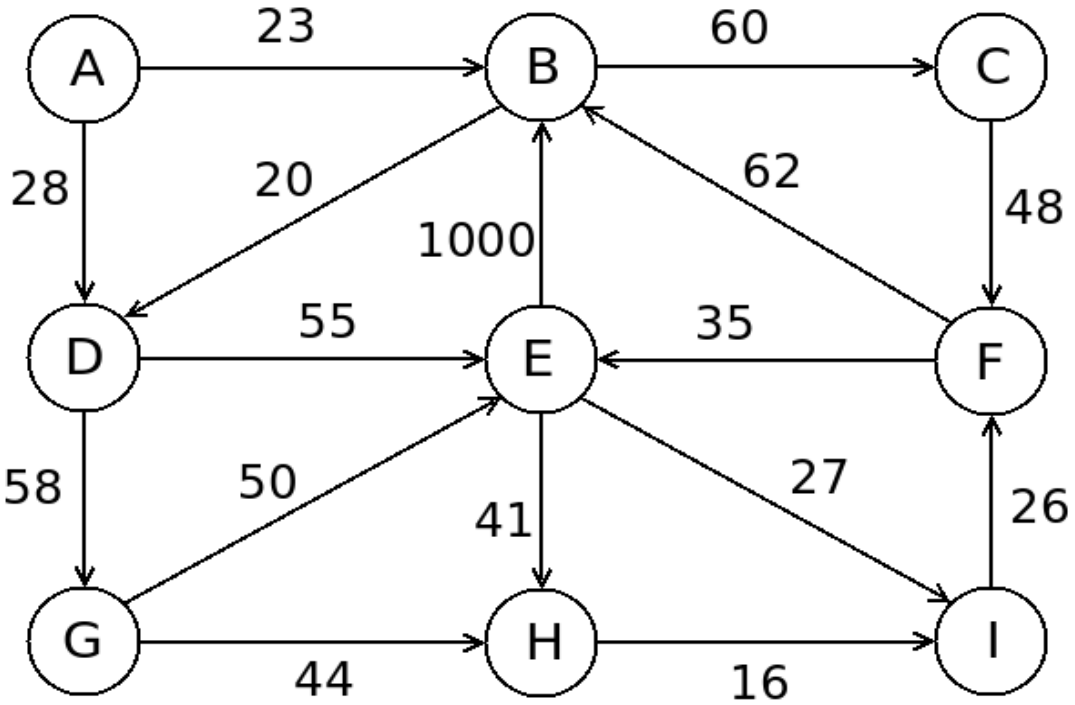
	Struttura dati	**Complessità**
a	Lista non ordinata	O(n)
b	Array non ordinato	O(n)
c	Array ordinato	O(1)
d	Max-heap	O(n)
e	Min-heap	O(1)
f	Albero binario	O(n)
g	Albero binario di ricerca	O(log n)

```
int min(int* max_heap, int n) {
  int minimo = max_heap[n/2];
  for (int i=n/2+1;i<n;i++) {
    if (max_heap[i] < minimo)
      minimo = max_heap[i];
  }
  return minimo;
}
```

Situazione	*Esercizi*	*Tempo*
Senza progetto	**1,2,3,4,5**	80 min
Con progetto	**1,2,3**	40 min

1	*2*	*3*	*4*	*5*

Esercizio 2
Dato il seguente grafo orientato:



a) Supponendo che gli archi siano memorizzati tramite liste di adiacenza (ordinate alfabeticamente), visitare il grafo mediante la visita in profondità e riportare l'ordine in cui vengono visitati i nodi.
b) Supponendo che il grafo sia non orientato, trovare il minimo albero di copertura attraverso l'algoritmo di Kruskal (riportare l'insieme degli archi che compone l'albero e le componenti connesse che si formano durante l'esecuzione dell'algoritmo)

a)

A B C F E H I D G

b)

Componenti Connesse	**G(insieme archi)**
{A}{B}{C}{D}{E}{F}{G}{H}{I}	G0 = {}
{A}{B}{C}{D}{E}{F}{G}{H,I}	G1 = {(H,I)}
{A}{B,D}{C}{E}{F}{G}{H,I}	G2 = G1 U {(B,D)}
{A,B,D}{C}{E}{F}{G}{H,I}	G3 = G2 U {(A,B)}
{A,B,D}{C}{E}{F,H,I}{G}	G4 = G3 U {(I,F)}
{A,B,D}{C}{E,F,H,I}{G}	G5 = G4 U {(E,I)}
{A,B,D}{C}{E,F,G,H,I}	G6 = G5 U {(G,H)}
{A,B,D}{C,E,F,G,H,I}	G7 = G6 U {(C,F)}
{A,B,C,D,E,F,G,H,I}	G8 = G7 U {(D,E)}

Esercizio 3

Sia **l** il puntatore ad una lista singola di interi. Calcolare la complessità dell'istruzione

```
for(int i=0;i<g(f(l)); i++) cout << "!";
```

in funzione della lunghezza **n** di **l** (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) con le funzioni **f** e **g** definite come segue:

```
int f(Elem* h) {
   if (!h) return 1;
   int a=0;
   for(Elem* e=h;e;e=e->next){
      e->inf++;
      a++;
   }
   return g(a*a)+f(h->next);
}
```

```
int g(int x) {
   if (x<=1) return 1;
   int b = 1+g(x/2);
   b+= g(x/2);
   return 2*b;
}
```

Tempo di g

$T_g(x) = cost.$ per $x<=1$

$T_g(x) = cost+2T_g(x/2)$ altrimenti

$T_g(x)$ è $O(x)$

Tempo di f (in funzione della lunghezza n di h)

$T_f(n) = cost.$ per $n=0$

$T_f(n) = cost*n+T_g(n^2)+T_f(n-1) = cost*n^2+T_f(n-1)$ altrimenti

$T_f(n)$ è $O(n^3)$

la variabile a alla fine del for contiene la lunghezza di h quindi n

Risultato di g

$R_g(x) = cost$ per $x <=1$

$R_g(x) = 4+4*R_g(x/2)$ altrimenti

$R_g(x)$ è $O(x^2)$

Risultato di f

$R_f(n) = cost$ per $n<=1$

$R_f(n) = R_g(n^2) + R_f(n-1) = cost*n^4 + R_f(n-1)$ altrimenti

Rf(n) è $O(n^5)$

Calcolo del for

Numero iterazioni: $R_g(R_f(n)) = R_g(n^5) = O(n^{10})$

Complessità singola iterazione = $T_f(n) + T_g(n^5) = O(n^3) + O(n^5) = O(n^5)$

complessità del for: $O(n^{10})*O(n^5)=O(n^{15})$

Esercizio 4
Si scriva una funzione `Node* generaBilanciato(int k)` che restituisce un albero binario bilanciato ad etichette intere di livello k. Ogni nodo dell'albero deve avere come etichetta il proprio livello. Calcolare la complessità della soluzione proposta in funzione di k.

```
Node* generaBilanciato(int k, int n) {
  if (k<0) return NULL;
  Node* result = new Node();
  result -> label = n-k;
  result -> left = generaBilanciato(k-1,n);
  result -> right = generaBilanciato(k-1,n);
  return result;
}

Node* generaBilanciato(int k) {
  return generaBilanciato(k,k);
}
```

complessità(k) = $O(2^k)$

Esercizio 5
Sia dato un albero binario non vuoto contenente n etichette intere. Si supponga che le etichette siano comprese nell'intervallo [0,x-1], dove x è un intero maggiore di 1 noto solamente a tempo di esecuzione. Si scriva una funzione `int moda(Node* t, int x,...)` che calcola la moda delle etichette presenti nell'albero. Si ricorda che la moda di una sequenza di valori è il valore che compare più frequentemente. Si calcoli la complessità della soluzione proposta in funzione di n e x.

```
void visita_moda(Node* t, int* array) {
  if (!t) return;
  array[t->label]++;
  visita_moda(t->left,array);
  visita_moda(t->right,array);
}

int moda(Node* t, int x) {
  int *a = new int[x],i;
  for (i=0;i<x;i++) a[i]=0;
  int max = -1, result;
  visita_moda(t,a);
  for (i=0;i<x;i++) {
    if (a[i] > max) {
      result = i;
      max = a[i];
    }
  }
  delete[] a;
  return result;
}
```

complessità(n,x) = O(x)+O(n)