UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 12 Dicembre 2003

TRACCIA A

PARTE 1 - TUTTI (Tempo disponibile 2 Ore)

Esercizio 1.

Si consideri un torneo di Tennis, e si consideri la coppia <Nome1, Nome2> indicante il nome di due tennisti che hanno disputato un match. Si consideri un albero binario di tali coppie in cui la radice specifica i tennisti arrivati al match finale, mentre i nodi foglia specificano le coppie di tennisti nelle partite iniziali del torneo. Infine, ciascun nodo intermedio contiene il nome del tennista che ha vinto nella partita individuata dal suo figlio sinistro ed il nome del tennista che ha vinto nella partita individuata dal suo figlio destro. Si definisce *Avversario Iniziale* di un tennista x, l'avversario che x ha incontrato nella sua prima partita del torneo.

- Scrivere la funzione
  void TennistiSfortunati(AlberoB<coppia> torneo, char& N1, char*& N2)*
  che, ricevuto in input un albero rappresentante un torneo di tennis, restituisca opportunamente i nomi degli *Avversari Iniziali* dei due tennisti arrivati in finale.
- Scrivere una funzione *int partecipanti(AlberoB<coppia> torneo)* che restituisca il numero di tennisti che ha partecipato al torneo.

N.B. E' lecito supporre che l'albero è pieno.

Esercizio 2.

Determinare la complessità della funzione *esistePuntoDiSella*. Considerare sia il caso migliore che il caso peggiore

```
bool esistePuntoDiSella(int M[][N], int N)
{       bool trovato=false;
        for (int i=0; (i<N) && (!trovato); i++)
                for (int j=0; (j<N) && (!trovato); j++)
                        if (puntoDiSella(M,i,j,N))
                                trovato=true;
        return trovato;
}
```

```
bool puntoDiSella(int M[][N], int r, int c, int N)
{       int min, max;
        min=M[r][0];
        max=M[0][c];
        for (int i=1; i<N; i++)
                if (M[r][i]<min)
                        min=M[r][i];
        for (i=1; i<N; i++)
                if (M[i][c]>max)
                        max=M[i][c];
        if ((M[r][c]==min) && (M[r][c]==max))
                return true;
        else
                return false;
}
```

Esercizio 3.

Si consideri il problema di determinare il Massimo in un vettore di interi non ordinato. Indicare e motivare se è possibile risolvere tale problema con la tecnica **Divide et Impera**. Se si individua che ciò è possibile, indicare a grandi linee il metodo con cui il problema potrebbe essere risolto e specificare se la tecnica **Divide et Impera** produce, da un punto di vista della complessità, un possibile miglioramento rispetto alla soluzione iterativa dello stesso problema.

PARTE 2 – SOLO CHI NON HA CONSEGNATO IL PROGETTO (Tempo 2 Ore)

Si deve progettare un piccolo sistema per la gestione di uno Zoo. Lo Zoo deve contenere un elenco di Animali ed un elenco di Alimenti.
Definire le classi *Animale* ed *Alimento*.
Ciascun Alimento è caratterizzato da un *nome* e dall'altezza (in cm) a cui si trova nello zoo (ad esempio se si tratta di foglie, queste saranno su un albero. Alcune foglie saranno a 200 cm, altre a 50 cm, e così via. Se si tratta di latte, questo sarà servito dagli inservienti, quindi la sua altezza sarà 0 cm). Due alimenti con lo stesso nome, ma ad altezze diverse sono descritti da due oggetti differenti di tipo Alimento.
Ciascun *Animale* è caratterizzato da un nome, da un'altezza (in cm) e da un peso (in Kg). Definire *almeno* le classi *Giraffa*, *Pappagallo* e *Tartaruga* derivate opportunamente dalla classe *Animale*. Ciascuna di queste classi deve fornire una funzione *altezza()* che restituisca l'altezza dell'animale in considerazione, la funzione *bool vola()* che indica se l'animale in questione può volare e la funzione *bool mangia(Alimento& a)* che restituisca *true* se l'animale mangia l'alimento *a* in base solo al nome dell'alimento. (Per gli scopi dell'esame, supporre che la giraffa mangia foglie, il pappagallo mangia mangime e foglie, la tartaruga mangia lattuga e foglie.
Lo Zoo, quindi deve contenere una lista di Alimenti ed una lista di Animali.
Nella classe Zoo, scrivere la funzione LeggiAnimali che riempie opportunamente la lista di Animali (l'utente prima indica il tipo di animale che sta inserendo e poi i suoi dati) e la funzione LeggiAlimenti.
Scrivere anche, nella classe Zoo, la funzione *bool sopravvivenzaGarantita()* che verifica se all'interno dello Zoo ciascun animale riesce a mangiare almeno un alimento. Un animale *A* riesce a mangiare un alimento *B* se *A mangia B* e se l'altezza di B è minore dell'altezza di A oppure A riesce a volare.
Infine, scrivere una funzione *stampaSituazione* che stampa, ciascun animale nello zoo e, per ciascun animale, l'elenco degli alimenti che riesce a mangiare.

N.B. Si può supporre di avere a disposizione tutte le classi sviluppate nel materiale didattico.
N.B. 2 E' assolutamente richiesto l'uso della versione con i template delle liste
N.B. 3 Fare particolarmente attenzione alla tipologia delle classi ed alla collocazione delle varie funzioni richieste.

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 12 Dicembre 2003

TRACCIA B

PARTE 1 - TUTTI (Tempo disponibile 2 Ore)

Esercizio 1.

- Scrivere una funzione *livMax* che ricevuto un Albero binario di interi determini il massimo livello di un nodo all'interno dell'albero (si ricorda che il nodo radice ha livello 1)
- Scrivere una funzione *penultimi* che ricevuto un Albero Binario di interi A, calcoli il livello massimo *l* dei nodi in A e, quindi, restituisca la lista degli elementi posti al livello *l-2* nell'albero A.

Esercizio 2.

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione.

```
bool elabora(int M[][N], int V[N])
{for (int k=0; i<N; i++)
        if (trovato(M[k][k],V))
           for (int i=0; i<N;i++)
                for (int j=i; j<N; j++)
                        if (M[i][j]==M[k][k])
                                M[i][j]=0;

        else
          for (int x=0; x<N; x++)
                M[x][k]=V[x];
}
```

```
bool trovato(int val, intV[N])
{   bool b=false;
    int i=0;
    while (i<N) && (!b)
    {       if (val==V[i]) b=true;
            else i++;
    }
  return b;
}
```

Esercizio 3.

Si consideri il problema di determinare il Massimo in un vettore di interi non ordinato. Indicare e motivare se è possibile risolvere tale problema con la tecnica **Golosa**. Se si individua che ciò è possibile, indicare a grandi linee il metodo con cui il problema potrebbe essere risolto e specificare se la tecnica **Golosa** produce, da un punto di vista della complessità, un possibile miglioramento rispetto alla soluzione classica dello stesso problema.

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 5 Luglio 2004

Tempo: 3 ore

Esercizio 1.

Si supponga di avere tre classi A, B e C. A contiene un intero *pippo* ed è la classe base; B e C sono classi derivate da A. B contiene (oltre all'intero *pippo*) un carattere *pluto*, mentre C contiene (sempre oltre all'intero pippo) una stringa *paperino*.

- Scrivere opportunamente le definizioni di A, B e C
- Predisporre opportunamente nella definizione delle classi un meccanismo che consenta a ciascun oggetto di poter dire di che tipo è (se è di tipo A, B o C) utilizzando il polimorfismo.
- Se si vuole memorizzare un insieme di oggetti di tipo A, B e C in uno stesso vettore V, cosa si deve indicare al posto di *TipoElem* nella seguente definizione?

  Vettore<*TipoElem*> V

Si supponga quindi di avere a disposizione un oggetto V di tipo Vettore di *TipoElem* che memorizzi oggetti sia di tipo A, B o C (si supponga che tale array sia stato già in qualche modo letto). Si vuole suddividere gli oggetti in V in tre liste distinte LA, LB ed LC in modo tale che LA contenga solo oggetti di tipo A, LB contenga solo oggetti di tipo B ed LC contenga solo oggetti di tipo C.

- Scrivere una funzione *void separa(Vettore<TipoElem> V, Lista<A> & LA, Lista<B> & LB, Lista<C> & LC)* che ricevuto in input il vettore V separi gli oggetti di tipo A, B e C generando le tre liste LA, LB ed LC (sostituire nell'intestazione sopra a *TipoElem* il tipo giusto di elementi che deve possedere V secondo la risposta data al punto precedente).

NB. E' possibile utilizzare tutte le funzioni delle classi Vettore, Lista e, se necessario, Iteratore dosponibili sulle dispense per risolvere il problema.

Esercizio 2.

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione.

```
int f(int V1[N], int V2[N])
{bool b=false;
 int i=1;
  while ((!b) && (i<N)){
          b=(g(V1[i], V2) && g(V2[i], V1));
          i=i*2;
  }
 return i;
}
```

```
bool g(int val, int &V[N])
{   bool b=false;
    for (int i=0; (i<N)&&(!b); i++)
          if (val==V[i]) b=true;
    }
   return b;
}
```

Esercizio 3.

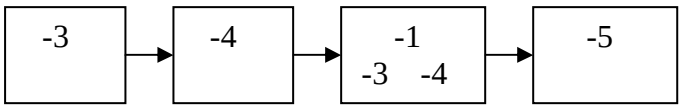
- Scrivere la funzione <u>membro</u> della classe AlberoB: *int* AlberoB::*GradoDiUscitaPositivo()* che restituisce il numero di nodi figli della radice che hanno un contenuto maggiore di 0 (chiaramente se il nodo non ha figli la funzione restituisce 0)
- Scrivere quindi una funzione <u>esterna</u> alla classe AlberoB di nome *ripulisci* che riceve in input un albero binario e si comporta cosi':
  - Visita l'albero in profondità, esaminando prima i nodi piu' profondi
  - Quando un nodo è esaminato, la funzione considera il suo contenuto ed il suo "grado di uscita positivo": se il nodo è un nodo foglia, la funzione non fa nulla; se il nodo

non è foglia, il suo contenuto è maggiore di 0, ed il "grado di uscita positivo" è pari a 0, il contenuto del nodo viene modificato in -1, altrimenti il nodo viene lasciato intatto.

Esempio: L'albero

diventa :

- Scrivere infine una funzione <u>esterna</u> alla classe AlberoB di nome *isola* che riceve un albero modificato tramite la funzione *ripulisci*, visita l'albero secondo la visita posticipata e restituisce la lista dei sottoalberi contenenti solo elementi minori di 0.
  Esempio: per l'esempio di prima, la lista dei sottoalberi contiene:



- Si può scrivere la funzione in modo che restituisca solo i sottoalberi più grandi? (nell'esempio precedente –3 e –4 sono contenuti nel sotto albero visualizzato nel 3 nodo) Accennare, senza riscrivere la funzione, al modo in cui cio' potrebbe essere fatto.

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 10 Settembre 2004

Tempo: 3 ore

Esercizio 1.
Si rammenti che in un albero binario di ricerca, nuovi dati sono inseriti nell'albero in modo che, in qualsiasi momento, viene mantenuto complessivamente il seguente ordinamento: sia N un nodo dell'albero, ciascun nodo Ns a sinistra di N è tale che Ns<N (chiaramente si intende il contenuto dei nodi N ed Ns), mentre ciascun nodo Nd a destra di N è tale che Nd>= N; di conseguenza, la funzione *ins* della classe AlberoBR inserisce i nuovi elementi nell'albero in modo da assicurare sempre tale ordinamento.
Sia dato un array V di D elementi ordinati in senso crescente. Scrivere una funzione che, a partire da V, costruisca opportunamente un albero binario di ricerca contenente tutti i dati di V in modo tale che l'albero alla fine degli inserimenti risulti BILANCIATO.
N.B. E' necessario inserire i dati dell'array secondo un opportuno ordinamento per garantire che l'albero alla fine sia bilanciato.

Suggerimento: Una possibile intestazione di tale funzione potrebbe essere:
const int D=....

```
template <class T>
void costruisci(T V[D], AlberoBR<T> & A, int inf, int sup)
```

in cui la prima chiamata della funzione va fatta con un albero A vuoto e con inf=0, sup=D-1

Esercizio 2.
Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione.

```
int pippo(int V1[N], int V2[N])
{bool pluto=false;
 int i=1;
 while ((!pluto) && (i<N)){
   pluto=((paperino(V1[i], V2[N-i]) &&
          (paperino(V1[N-i], V2[i]));
   i=i*2;
  }
 return i;
}
```

```
bool paperino(int val1, int val2)
{
   return (val1!=val2);
}
```

Esercizio 3.
Si implementi, utilizzando l'ereditarietà ed il polimorfismo, una gerarchia di classi che consenta di gestire la tariffazione degli utenti di una compagnia telefonica. A tal proposito, si definisca la classe base Conto. Ogni conto ha un intestatario, un codice, e deve consentire di determinare con una funzione il traffico (in minuti) effettuato da un utente. L'aggiornamento del traffico effettuato, viene realizzato tramite una funzione *NuovoTraffico* che riceve la durata dell'ultima chiamata.
Si specializzi la classe Conto definendo opportune sottoclassi in modo da offrire agli utenti due profili di tariffazione: il ContoPrepagato e il ContoAbbonamento (ciascun utente può avere una tariffa per minuto diversa, ma si trascurino differenze di tariffe nell'arco della giornata).

Entrambe le classi devono fornire una funzione virtuale “SituazioneCorrente” che restituisca il credito residuo per un utente “prepagato” e la spesa effettuata fino a quel momento da un “abbonato”. (NB. per un conto prepagato ogni chiamata fa scalare il proprio credito, mentre per un conto di un abbonato la spesa aumenta ad ogni chiamata). Infine, le classi devono fornire una funzione virtuale “Aggiorna” che consente di aggiornare la situazione dell’utente dopo un intervento dell’utente stesso (Ricarica del conto per ContoPrepagato o pagamento della bolletta per ContoAbbonamento).
Riassumendo, si implementino i costruttori, gli eventuali distruttori e tutte le funzioni necessarie all’efficacia delle classi (nell’ottica dell’ereditarietà e del polimorfismo) per :

- aggiornare la situazione di traffico dei vari clienti (prepagati o abbonati) dopo ogni telefonata
- calcolare la “SituazioneCorrente” di ciascun utente (sia prepagato, sia abbonato) in base alla sua personale tariffazione al minuto
- consentire di aggiornare la situazione dell’utente dopo un intervento dell’utente stesso (Ricarica del conto per ContoPrepagato o pagamento della bolletta per ContoAbbonamento).

Se si vuole memorizzare un insieme di oggetti di tipo ContoPrepagato e ContoAbbonamento in uno stesso vettore V, cosa si deve indicare al posto di *TipoElem* nella seguente definizione?

Vettore<*TipoElem*> V

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 7 Gennaio 2005

TRACCIA A

Tempo: 3 ore

Esercizio 1

Si definisca una gerarchia di classi tramite l'ereditarietà ed il polimorfismo in grado di gestire numeri generici. In particolare, si definisca una classe astratta *Numero* e due classi derivate *NumeroComplesso* (un numero complesso è descritto da una parte reale *re* e da una parte immaginaria *imm*) e *NumeroRazionale* (un numero razionale è costituito da un numeratore *num* ed un denominatore *den*) e definire di queste almeno i costruttori.
Si gestisca infine opportunamente tramite le funzioni virtuali ed il polimorfismo la stampa di un numero generico tramite una funzione *stampa()*. In particolare, un numero complesso va stampato nel seguente formato: *re* +/- j *imm* , dove *re* e *imm* sono i valori delle corrispondenti variabili, j è un carattere fisso ed il segno + o - va fatto stampare congruentemente con il valore di *imm* (NB se imm è negativo il - viene stampato automaticamente; al contrario, se *imm* è positivo il + non viene stampato). Un numero razionale, invece, va stampato nel formato *num* / *den*.

Esercizio 2.

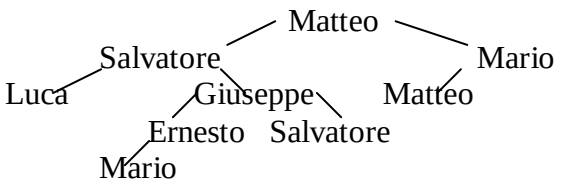
Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione analizzando tutti i possibili casi.

```
void elabora(int M1[][N], int M2[][N])
{   if (!triangolare(M1) && triangolare(M2))
        for (int i=0; i<N; i++)
            for (int j=0; j<N; j++)
                M1[i][j]=M2[i][j]);
    else
         cout << "triangolare";
}
```

```
bool triangolare(int M[][N])
{   for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
            if (M[i][j]!=M[j][i])
                return false;
    return true;
}
```

Esercizio 3

Si consideri un albero binario che rappresenta un albero genealogico. In tale albero, ciascun nodo memorizza (utilizzando il tipo di dati *char**) il nome di un individuo *P* ed i suoi due nodi figli contengono i nomi dei primi due figli maschi avuti da *P*. Si noti che i figli dei figli di *P* sono i nipoti di *P* (nel caso dell'albero in questione un individuo può avere al più 4 nipoti memorizzati nell'albero)
Scrivere una funzione *famigliaTradizionalista* che riceva in input un albero binario di *char** e restituisca *true* se per ciascun individuo che abbia almeno un nipote, esiste almeno uno dei suoi nipoti cui sia stato dato il suo nome; *false* altrimenti
Ad esempio, la famiglia rappresentata dall'albero di sinistra



<u>è tradizionalista</u> in quanto: il nome *Matteo* è tramandato tramite il figlio di *Mario*, il nome *Salvatore* è tramandato tramite il figlio di *Giuseppe* ed il nome *Giuseppe* è tramandato tramite il figlio di *Ernesto*. Tutti gli altri elementi dell'albero non hanno nipoti o figli, quindi la famiglia è tradizionalista.
Invece, la famiglia rappresentata dall'albero di destra <u>non è tradizionalista</u> in quanto il nome *Giuseppe* non è tramandato al suo (unico) nipote che, invece, è stato chiamato *Mario* (si noti che anche se il nome Mario appare già nell'albero, questo non è il nonno del Mario più "giovane", quindi non va considerato).
NB. Gli alberi da considerare potrebbero essere molto più profondi di quelli visti in questi esempi, quindi va verificata la condizione per tutti i nodi dell'albero a qualsiasi livello; ovviamente, gli individui che non hanno dei nipoti o dei figli non influiscono negativamente nella verifica della condizione.

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 7 Gennaio 2005

TRACCIA B

Tempo: 3 ore

Esercizio 1

Si definisca una gerarchia di classi tramite l'ereditarietà ed il polimorfismo in grado di gestire forme generighe. In particolare, si definisca una classe astratta *Forma* e due classi derivate *Segmento* (caratterizzato dalla sua lunghezza) e *Rettangolo* (caratterizzato dalla dimensione dei lati) e definire di queste almeno i costruttori.
Si gestisca infine opportunamente tramite le funzioni virtuali ed il polimorfismo la stampa a video di un segmento o di un rettangolo tramite una funzione *stampa()*. In particolare, un segmento di lunghezza *l* va disegnato con *l* occorrenze di caratteri '-'. Un rettangolo di lati *l* e *m*, invece, va disegnato tramite una serie di caratteri '*' distribuiti su *l* righe e *m* colonne.
Esempio: Un segmento di lunghezza 5 va stampato così: -----, mentre un rettangolo di lati 2 e 5 va stampato come:
*****
*****

Esercizio 2.

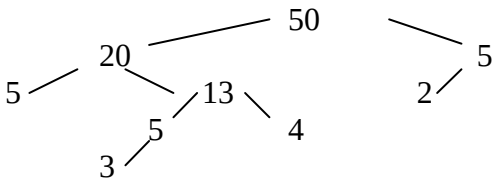
Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione analizzando tutti i possibili casi.

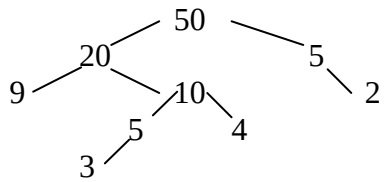
```
bool elabora(int M[][N], int V[N])
{   bool almenoUna=false;
    int i,j,k;
    for (i=0; i<N; i++)
        if (tuttiZeri(M, i))
        {   almenoUna=true;
            for (j=0; j<N; j++)
                for (k=0; k<N; k++)
                    M[i][j]+=V[k];
        }
        else
            for (j=0; j<N; j++)
                M[i][j]+=V[j];
    return almenoUna;
}
```

```
bool tuttiZeri(int M[][N], int r)
{   for (int i=0; i<N; i++)
        if (M[r][i]!=0)
            return false;
    return true;
}
```

Esercizio 3

Un albero binario di interi, rappresentante i salari dei dipendenti di un'azienda, è detto *popolare* se, per ogni nodo N dell'albero, la somma dei salari dei nodi figli di N più la somma dei salari dei figli dei figli di N è inferiore al salario di N (si noti che, essendo l'albero binario, per ciascun N si devono considerare al piu' 6 nodi). Scrivere una funzione *popolare* che riceve in input un albero binario di interi e determina se l'albero è popolare o meno.
Ad esempio, l'albero di sinistra





<u>è popolare</u> in quanto soddisfa la condizione, mentre l'albero di destra non è popolare in quanto, ad esempio, il dipendente con salario 10 ha che i nodi figli e figli dei figli hanno una somma di salari pari a 12, maggiore del suo. Analogamente, il dipendente con salario 20 ha che i nodi figli e figli dei figli hanno una somma di salari pari a 28.
NB. Gli alberi da considerare potrebbero essere molto più profondi di quelli visti in questi esempi, quindi va verificata la condizione per tutti i nodi dell'albero a qualsiasi livello; ovviamente, basta che la condizione non sia verificata per un nodo affinchè l'albero non sia popolare.

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

PROVA DI LABORATORIO

Data: 7 Gennaio 2005

Tempo: 30 minuti

Data la classe "Quadrato" - scaricabile all'indirizzo:
http://www.mat.unical.it/terracina/asd/Quadrato.h

che fornisce il costruttore *Quadrato(int l)* per creare un quadrato di lato l ed una funzione *int Area()* che restituisce l'area del quadrato, implementare un main che:

- definisca opportunamente una "Lista di quadrati";
- legga da standard input 10 quadrati e li inserisca nella lista;
- calcoli e stampi su standard output la somma delle aree dei quadrati inclusi nella lista.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
http://www.mat.unical.it/terracina/asd

UNIVERSITÀDELLACALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 6 Luglio 2005

Tempo: 3 ore

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione.

```
bool pariConcentrica ( int [] [n] M, int n ) {
    int numeroCornici = n/2;
    for ( int i = 0; i < numeroCornici; i++ ) {
        if ( ! pari( M, n, i ) )
            return false;
    }
    if ( n % 2 == 0 )
        return true;
    else
        return ( M[numeroCornici][numeroCornici] % 2 == 0 );
}
```

```
bool pari ( int [] [n] M, int n, int k ) {
    int somma = 0;
    for ( int i = 0; i < n; i++ )
        for ( int j = 0; j < n; j++ )
            if ( i == k || i == n - k - 1 || j == k || j == n - k - 1 )
                somma += M[i][j];
    return ( somma % 2 == 0 );
}
```

Esercizio 2

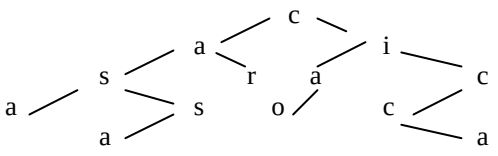
Si definisca, utilizzando l'ereditarietà ed il polimorfismo, una gerarchia di classi per la gestione di località turistiche. In particolare, si definisca una classe astratta *LocalitàTuristica* e due classi derivate *LocalitàMarina* e *LocalitàMontana* .
Ciascuna località turistica è caratterizzata da un nome e dalla provincia di appartenenza (stringhe di al più 20 caratteri). Una località marina viene inoltre caratterizzata dal numero di lidi presenti sul suo territorio, mentre una località montana dalla tipologia di piste da sci presenti. Esistono tre possibili tipologie di piste da sci: BLU, ROSSA e NERA. Una località montana può contenere varie piste, ciascuna delle quali caratterizzata da una tipologia indipendente dalla tipologie delle altre. Nel presente esercizio, interessa riuscire ad esprimere solo se una data tipologia è presente o meno, senza contare il numero di piste. (Per esempio, una località contiene piste rosse e nere ma non blu, mentre un'altra ne contiene blu e nere ma non rosse, e così via)
Per ognuna delle tre classi definirne opportunamente i costruttori.
Si definiscano inoltre in maniera opportuna le seguenti funzioni:

- *leggiLocalità* che consente di leggere da tastiera le caratteristiche di una località
- *stampaInfo* che consente di visualizzare in output il nome e la provincia di una località
- *benAttrezzata* che, nel caso di una località marina restituisce *true* se il numero di lidi è maggiore di 5 e *false* altrimenti, mentre, nel caso di una località montana restituisce *true* se la località possiede almeno due tipologie di piste da sci, *false* altrimenti.

Scrivere quindi un main che definisca e legga opportunamente un array di 10 località turistiche, chiedendo preventivamente per ciascuna località se essa è marina o montana (n.b., non si sa a priori quante località marine verranno lette e quante località montane; si sa solo che in tutto sono 10). Alla fine della lettura dell'array, per ciascuna località nell'array, stampi il nome, la provincia e se la località risulta ben attrezzata o meno.
Se si ritiene necessario, si può utilizzare la classe Vettore generalizzato definita durante il corso.

Esercizio 3

Si supponga di avere un albero binario in cui ciascun nodo contiene un carattere. In particolar modo, l'albero è riempito in modo tale da memorizzare parole: una parola è costituita dalla giustapposizione dei caratteri che vanno dalla radice fino ad una delle foglie. Per esempio, l'albero:



Contiene le parole:
casa, cassa, car, ciao e cicca
ma anche le parole:
c, ca, cas, cass, ci, cia, cic, cicc (cioè le parole che non fanno raggiungere una foglia)

Scrivere la funzione *bool esisteNellAlbero(AlberoB<char> A, char * parola)*
che determina se la stringa *parola* ricevuta in input è presente nell'albero (nel qual caso restituisca *true*) o meno (*false*).
N.B. è importante considerare tutti i casi "strani" che si possono verificare. Per esempio, se la parola è troppo lunga, o ne è presente solo un pezzo, etc.
Scrivere inoltre la funzione *bool esisteAll(AlberoB<char> A, char * parola)* che verifica se *parola* è presente nell'albero non solo a partire dalla radice, ma anche a partire da uno qualsiasi dei suoi nodi interni. (Nell'esempio di sopra, l'albero in questo caso contiene <u>anche</u> le parole: a, sa, ssa, asa, assa, r, ar, o, ao, iao, ca, cca, icca

UNIVERSITÀDELLACALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

PROVA DI LABORATORIO

Data: 6 Luglio 2005

Tempo: 30 minuti

Data la classe "Quadrato" - scaricabile all'indirizzo:
http://www.mat.unical.it/terracina/asd/Quadrato.h

che fornisce il costruttore *Quadrato(int l)* per creare un quadrato di lato l ed una funzione *int Perimetro()* che restituisce il perimetro del quadrato, implementare un main che:

- definisca opportunamente una "Coda di quadrati"; (l'implementazione della classe coda è reperibile all'indirizzo http://www.mat.unical.it/terracina/asd/Coda.zip)
- legga da standard input prima un intero *n* e poi *n* quadrati inserendoli opportunamente nella coda;
- Alla fine della lettura stampi su standard output i perimetri dei quadrati nello stesso ordine in cui sono stati letti.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
http://www.mat.unical.it/terracina/asd

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 14 Settembre 2005

Tempo: 3 ore

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione.

```
int elabora(int M1[][N], int M2[][N])
{    bool b=false;
     int i=1;
     if (!simmetrica(M1))
         while ((!b) && (i<N)){
             b=(M1[i][i]<M2[i][i]);
             i++;
         }
     return i;
}
```

```
bool simmetrica(int M[][N])
{   for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
            if (M[i][j]!=M[j][i])
                return false;
    return true;
}
```

Esercizio 2

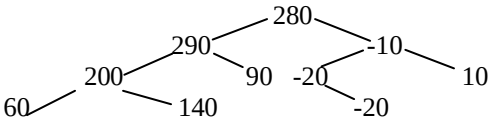
Si definisca, utilizzando l'ereditarietà ed il polimorfismo, una gerarchia di classi per la gestione di pubblicazioni scientifiche. In particolare, si definisca una classe astratta *PubblicazioneScientifica* e due classi derivate *Articolo* e *Applicazione*.
Ciascuna pubblicazione scientifica è caratterizzata da un titolo e dall'autore (stringhe di al più 20 caratteri). Un articolo viene inoltre caratterizzato dal numero "*nbib*" di altre pubblicazioni che lo includono nella bibliografia. Un'applicazione viene invece caratterizzata dall'esistenza o meno di un prototipo funzionante che implementa l'applicazione proposta.
Per ognuna delle tre classi definirne opportunamente i costruttori.
Si definiscano e si implementino inoltre in maniera opportuna le seguenti funzioni:

- *leggiPubblicazione* che consente di leggere da tastiera le caratteristiche di una pubblicazione
- *stampaInfo* che consente di visualizzare in output il titolo e l'autore di una pubblicazione
- *daSelezionare* che, nel caso di un articolo restituisce *true* se il numero di riferimenti bibliografici (*nbib)* è maggiore o uguale a 3 e *false* altrimenti, mentre, nel caso di un'applicazione restituisce *true* se esiste un prototipo funzionante, *false* altrimenti.

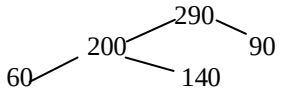
Scrivere quindi un main che definisca e legga opportunamente un array di 15 pubblicazioni, chiedendo preventivamente per ciascuna pubblicazione se trattasi di articolo o applicazione (N.B., non si sa a priori quanti articoli e quante applicazioni verranno lette, si sa solo che in tutto sono 15 pubblicazioni scientifiche). Alla fine della lettura dell'array, per ciascuna pubblicazione nell'array, il main deve stamparne il titolo, l'autore e se la pubblicazione risulta da selezionare o meno.

Esercizio 3

Si supponga di avere un albero binario di numeri interi, in cui ciascun nodo non foglia contiene un valore uguale alla somma dei valori presenti nei due nodi figli (uno dei quali potrebbe anche essere nullo). Un esempio è il seguente albero:



Scrivere la funzione *bool trova(AlberoB<int> A, int num, AlberoB<int> & B)* che determina se l'intero *num* ricevuto in input è presente o meno nell'albero *A* ed, eventualmente, restituisce il relativo sottoalbero. In particolare, se *num* non è presente in *A* la funzione deve restituire *false* e *B* deve essere l'albero nullo. Se *num* è presente in *A* la funzione deve restituire *true* e l'albero *B* deve essere il sottoalbero di *A* avente come radice il valore cercato. Nell'esempio precedente, per *num*=290 la funzione deve restituire *true* e *B* deve essere l'albero:



Scrivere inoltre la funzione *bool estraiAddendi(AlberoB<char> A, int num, Coda<int>& addendi)* che, utilizzando la funzione precedente (supporla comunque validamente implementata), innanzitutto verifica la presenza di *num* nell'albero e, in caso affermativo, inserisce nella coda *addendi* i valori (cioe' le foglie dell'albero sotteso a *num*) che devono essere sommati per ottenere come risultato *num*. Nell'esempio precedente, per *num*=290 la funzione deve restituire *true* e deve inserire nella coda: "60 140 90" (NB. non è rilevante l'ordine con cui i valori sono inseriti nella coda).

UNIVERSITÀDELLACALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

PROVA DI LABORATORIO

Data: 14 Settembre 2005

Tempo: 30 minuti

Data la classe "Dipendente" - scaricabile all'indirizzo:
http://www.mat.unical.it/terracina/asd/Dipendente.h

che fornisce il costruttore *Dipendente(int c, double pm)* per creare un dipendente con codice *c* e paga mensile *pm* ed una funzione *double pagaAnnua()* che restituisce lo stipendio annuo del dipendente, implementare un main che:

- definisca opportunamente una "Lista di dipendenti";
- legga da standard input 10 dipendenti e li inserisca nella lista;
- calcoli e stampi su standard output la somma delle paghe annue dei dipendenti inclusi nella lista.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
http://www.mat.unical.it/terracina/asd

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 21 Dicembre 2005

Tempo: 3 ore

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione.

```
bool elabora(int M[][N], int V[N])
{   bool b=false;
    int i=1;
    while (i<=N && !b) {
        if (somma(M) || V[i-1] == 0)
            b=true;
        i = i+2;
    }
    return b;
}
```

```
bool somma(int M[][N])
{   int s=0;
    for (int i=0; i<N; i++)
        if (M[i][i]!=0)
            for (int j=0; j<N; j++)
                s += M[i][j];
    if (s > 0)
        return true;
    else
        return false;
}
```

Esercizio 2

Si supponga di avere un oggetto *elencoProvince* di tipo Vettore< char *> contenente l'elenco delle province italiane. Si supponga inoltre di avere un oggetto *elencoAziende* di tipo Lista<Azienda> contenente le informazioni su alcune aziende italiane (la definizione della classe Azienda è data di seguito).

NB. Supporre che il vettore e la lista siano gia' riempiti

```
Class Azienda{
public: char * nome;
        char * provincia;  //la provincia in cui risiede l'azienda
        int nDip;  // indica il numero dei dipendenti
}
```

Scrivere una funzione *AziendeMigliori* che ricevuti in input gli oggetti *elencoProvince* e *elencoAziende* restituisca in output (non su video!) un Vettore< char *> che abbia la stessa dimensione di *elencoProvince* e contenga, per ciascuna provincia, il <u>nome</u> dell'Azienda che ha piu' dipendenti <u>nella sua provincia</u>.

NB. E' obbligatorio usare gli iteratori per scorrere gli elementi della lista *elencoAziende*.

Esercizio 3

Si supponga di avere un albero di natale *A*, rappresentato da un oggetto di tipo Albero<int> in cui ciascun nodo contiene il peso di una delle palline appese all'albero. Scrivere una funzione che riceva in input l'albero *A* e restituisca *true* se l'albero è "pendente", *false* altrimenti.

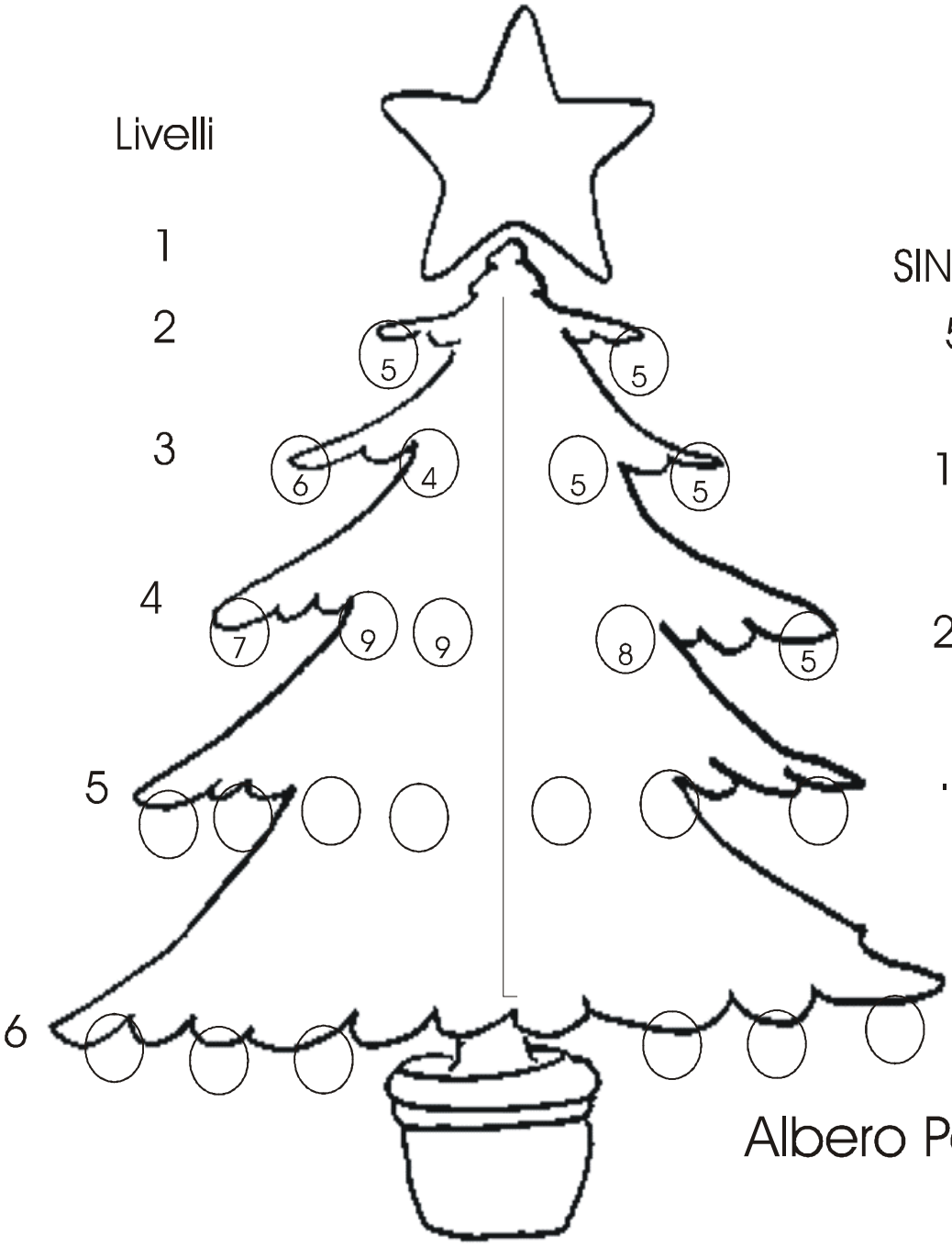
Un albero è detto "pendente" se esiste <u>almeno uno</u> dei suoi livelli *l* tale che la somma dei pesi delle palline a *sinistra* del tronco sul livello *l* è diversa dalla somma dei pesi delle palline a *destra* del tronco sullo stesso livello *l*. Una pallina è a sinistra del tronco se fa parte del sotto-albero sinistro del nodo radice di *A*, mentre è a destra del tronco se fa parte del sotto-albero destro del nodo radice di *A*.

NB. In questo caso, la radice di *A* rappresenta la stella in cima all'albero di natale e quindi non deve essere considerata nel calcolo della funzione "pendente" in quanto è posta sul tronco.

Inoltre, si noti che l'albero non è necessariamente pieno.

Vedere Esempio sul retro del foglio.

Suggerimento: suddividere l'albero *A* in due sotto-alberi: uno contenente le palline a sinistra del tronco e l'altro contenente le palline a destra del tronco e successivamente effettuare le opportune elaborazioni distintamente sui due alberi.



Pesi

SIN	DES	
5	5	OK
10	10	OK
25	13	NO!
....	....	

Albero Pendente

BUON NATALE!

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

PROVA DI LABORATORIO

Data: 21 Dicembre 2005

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di numeri interi terminata da -1;
- stampi su standard output la stessa sequenza letta in input ma in ordine inverso (utilizzando la struttura dati 'stack').

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
`http://www.mat.unical.it/terracina/asd`
ed in particolare la classe *Stack* disponibile all'indirizzo
`http://www.mat.unical.it/terracina/asd/Stack.zip`

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

Data: 4 Luglio 2006

Tempo: 3 ore

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione,

```
void elabora(int M[][N]) {
   bool b=false;
   int i = 0, k = 0, j = N;
    while (i<N/2 && j>N/2 && !b) {
        if ( f1(M, k++) )
          b = true;
        if ( f2(M) ) {
          i = max( 0, i-1 );
          j = max( N/2, j-1 );
        }
        else {
          i = min( i+1, N/2 );
          j = min( j+1, N );
        }
    }
}
```

sapendo che la funzione **f1(M,k)** ha la seguente complessità:

per k < N → O(N)
e può terminare restituendo indifferentementeTrue o False

per k >= N → O(1)
e termina restituendo sempre True

mentre la funzione **f2(M)** ha la seguente complessità:

$O(N^2)$
e può terminare restituendo indifferentementeTrue o False

N.B. le funzioni min(...) e max(...) si possono considerare con complessità costante O(1).

Esercizio 2

Si supponga di avere un oggetto *elencoCategorie* di tipo Vettore< char *> contenente un elenco di categorie merceologiche. Si supponga inoltre di avere un oggetto *elencoCommercianti* di tipo Lista<Commmerciante> contenente le informazioni su alcuni commercianti (la definizione della classe Commerciante è data di seguito) ed una matrice quadrata M di dimensione NxN contenente le informazioni sulle quantità di merci scambiate tra i commercianti. In particolare il valore M[x,y] conterrà il numero di prodotti che x ha venduto ad y, dove x ed y sono rispettivamente il codice del commerciante venditore ed il codice del commerciante acquirente. Si noti che M[x,y] è in genere diverso da M[y,x].

NB. Supporre che il vettore, la lista e la matrice siano già riempiti.

```
Class Commerciante{
public:    int codice;  // avrà un valore compreso tra 1 ed N
           char * nome;
           char * categoria;  //la categoria merceologica trattata
}
```

Scrivere una funzione *maxMerceScambiata* che ricevuti in input gli oggetti *elencoCategorie*, *elencoCommercianti* e la matrice M, stampi a video, per ciascuna categoria, i <u>nomi</u> della coppia di commercianti che <u>trattano entrambi quella categoria</u> e che si sono scambiati la <u>maggiore quantità di merce</u> (quanto x ha venduto ad y + quanto y ha venduto ad x).

NB. E' obbligatorio usare gli iteratori per scorrere gli elementi della lista *elencoCommercianti* ed è vietato trasporre gli elementi della lista in un vettore.

Esercizio 3

Dopo il primo turno di qualificazione a gironi, il campionato mondiale di calcio prevede la prosecuzione ad *eliminazione diretta* tra coppie di squadre qualificate nei vari gironi con la qualificazione al turno successivo della squadra vincente e l'esclusione della squadra perdente. Si supponga che ogni squadra sia caratterizzata da un livello di merito:

B indica una squadra 'brava'
M indica una squadra 'media'
S indica una squadra 'scarsa'

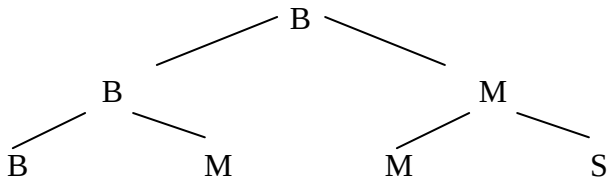
Si vuole decidere se il torneo è stato ben strutturato, intendendo con ciò che non ci sono stati nel primo turno ad eliminazione diretta scontri tra due squadre entrambe brave o scontri tra due squadre entrambe scarse.

Si supponga quindi di avere un albero binario *A* di tipo alberoB<char> che rappresenta gli scontri effettuati nella fase ad *eliminazione diretta* del campionato mondiale di calcio. Per semplicità, i nodi di *A* hanno come contenuto informativo solamente il livello di merito della squadra (il carattere B/M/S).

Scrivere una funzione che ricevuto in input l'albero *A* restituisca True se il torneo è stato ben strutturato, False altrimenti.

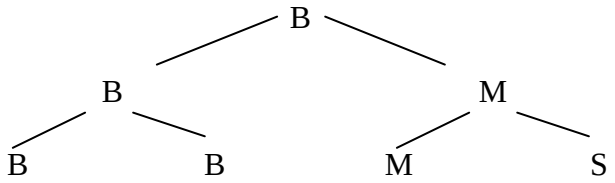
N.B. Supporre che l'albero sia completo e sia già stato valorizzato. Non si può invece fare nessuna assunzione sulla profondità dell'albero (cioè sul numero di squadre coinvolte nella fase ad eliminazione diretta).

Esempio: l'albero

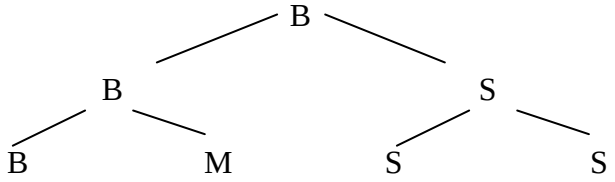


rappresenta un torneo <u>ben strutturato</u> in quanto i primi scontri (quelli sul livello piu' basso) sono stati tra B-M e M-S (quindi non c'è stata nessuna partita tra B-B o S-S)

mentre l'albero



e l'albero



rappresentano tornei <u>non ben strutturati</u> infatti il primo ha uno scontro iniziale B-B ed il secondo uno scontro iniziale S-S.

UNIVERSITÀDELLACALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e strutture dati

PROVA DI LABORATORIO

Data: 4 Luglio 2006

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di numeri interi terminata da -1;
- inserisca tali valori in una struttura dati 'coda';
- estragga i vari elementi presenti nella coda stampando a video il valore massimo.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
http://www.mat.unical.it/terracina/asd
ed in particolare la classe *Coda* disponibile all'indirizzo
http://www.mat.unical.it/terracina/asd/Coda.zip

UNIVERSITÀDELLACALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture Dati

Data: 6 Settembre 2006

Tempo: 3 ore

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione,

```
void elabora(int M[][N]) {
   bool b=false;
   int j, i = 0;  char c;
   while (i<N && !b) {
      cin >> c;
      switch (c) {
        case 'a':
          if ( f(M, i) )
             for ( j=0; j<N; j++ )
                cout << M[i, j];
          else
             cout << 0;
          break;
        case 'b':
          if ( f(M, i*i) )
             b = true;
          break;
        default:
          for ( j=0; j<N; j++ )
             cout << M[i, j];
      } //fine switch
      i++;
   } //fine while
}
```

sapendo che la funzione **f(M,i)** ha la seguente complessità:

per i < N → O(N)
e può terminare restituendo indifferentementeTrue o False

per i >= N → O(1)
e termina restituendo sempre True

Esercizio 2

Si supponga di avere un oggetto *elencoVendite* di tipo Lista<Vendita> contenente le informazioni su alcune vendite (la definizione della classe Vendita è data di seguito).

```
Class Vendita{
public:   int codVenditore;    // il codice che identifica un venditore
          int codAcquirente;   // il codice che identifica un acquirente
          int importo;         // l'importo pagato dall'acquirente al venditore
}
```

Si suppone che la lista sia già riempita. Inoltre, si suppone che i possibili valori per 'codVenditore' varino da 1 a 10 ed i possibili valori per 'codAcquirente' varino da 1 a 100. Si noti che non è detto che tutti i valori in questo intervallo siano presenti nella lista; inoltre, uno stesso venditore può comparire piu' volte nella lista (insieme ad acquirenti eventualmente diversi), così come uno stesso acquirente può comparire piu' volte nella lista (insieme a venditori eventualmente diversi).

Scrivere una funzione *maxVendite* che ricevuta in input la lista *elencoVendite* stampi a video:

- il codice del venditore che ha realizzato l'incasso massimo (per incasso realizzato da un venditore si intende la somma degli importi relativi alle vendite effettuate da quel venditore);
- il codice del venditore che ha realizzato il secondo incasso massimo;
- il codice dell'acquirente che ha acquistato solo dal venditore che ha realizzato il secondo incasso massimo (se non esiste un acquirente con tale caratteristica stampare un messaggio informativo, se ne esiste più di uno stampare solo il primo trovato).

N.B. E' obbligatorio usare gli iteratori per scorrere gli elementi della lista *elencoVendite* ed è vietato trasporre gli elementi della lista in un vettore.

Esercizio 3

Dato un nodo *n*, in seguito indichiamo con *An* il sottoalbero di cui *n* è radice.

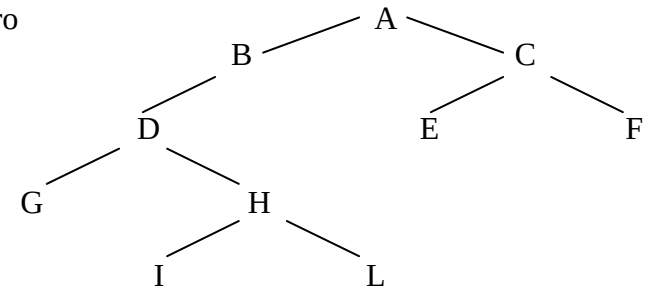
Un albero binario A si definisce *equilibrato* se, <u>per ogni nodo</u> *n* di A si verifica la seguente condizione:

- il numero di nodi in *An* aventi esattamente un figlio, è strettamente minore della metà della profondità di *An*.

Un albero nullo è per definizione equilibrato.

Scrivere una funzione che, ricevuto in input un albero binario *A* di tipo alberoB<char>, restituisca True se l'albero è *equilibrato*, False altrimenti.

Esempio: l'albero

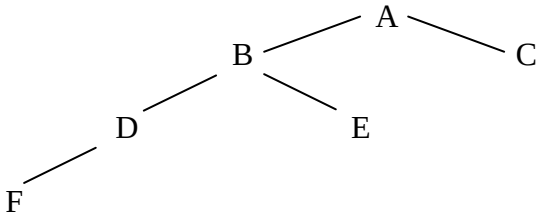


<u>è equilibrato</u> infatti per esempio:

per il nodo A	1 (num. nodi con 1 figlio)	<	5/2 (metà della profondità)
per il nodo B	1 (num. nodi con 1 figlio)	<	4/2 (metà della profondità)
per il nodo C	0 (num. nodi con 1 figlio)	<	2/2 (metà della profondità)
per il nodo D	0 (num. nodi con 1 figlio)	<	3/2 (metà della profondità)
per il nodo E	0 (num. nodi con 1 figlio)	<	1/2 (metà della profondità)

ecc.

mentre l'albero



<u>non è equilibrato</u> infatti:

per il nodo D	1 (num. nodi con 1 figlio)	=	1 (metà della profondità)

anche se per gli altri nodi la condizione sarebbe verificata.

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture dati

PROVA DI LABORATORIO

Data: 6 Settembre 2006

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di caratteri terminata da '$';
- inserisca tali valori in una struttura dati 'stack';
- estragga i vari elementi presenti nello stack stampando a video quei caratteri ripetuti 2 volte di seguito. Per esempio, nel caso della sequenza dfggrttafr$ dovranno essere stampati i caratteri 'g' e 't'.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
http://www.mat.unical.it/terracina/asd
ed in particolare la classe *Stack* disponibile all'indirizzo
http://www.mat.unical.it/terracina/asd/Stack.zip

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture Dati

Data: 8 Gennaio 2007

Tempo: 3 ore e 15 minuti

Esercizio "apri-busta"

(è necessaria la corretta risoluzione di questo esercizio per procedere alla valutazione del compito)

Dato un vettore di interi V di dimensione N+1, determinare l'indice i $\in$[0, N/2] tale che la somma V[i] + V[N-i] abbia valore massimo.

Esempio: dato il vettore V=[3,-2,6,2,15,-4] con N=5 il valore da restituire è 1 che corrisponde al valore massimo V[1] + V[4] = -2 +15 = +13

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione, illustrando chiaramente il ragionamento seguito.

```
void elabora( int M[][N], int V[N] ) {
    bool b=false;
    int j, i , k = N;

    for (  j=0; j<N && !b; j++  ) {
            i = 0;
            while ( i<N && k>0 ) {
                    if ( f(M, k) ) {
                            for ( int x=0; x<N; x++ )
                                    if ( M[i][j] == V[x] )
                                            k = 0;
                    }
                    else
                            if ( f(M, i) )
                                    b = true;
                    i++;
            } //fine while
            for ( int x=0; x<N; x++ )
                    if ( M[x][j] != V[x] )
                            b = true;
    } //fine for
}
```

sapendo che la funzione **f(M,i)** ha la seguente complessità:

per i >= N → O(N)
e può terminare restituendo indifferentemente ***true*** **o** ***false***

per i < N → O(1)
e termina restituendo sempre ***false***

Esercizio 2

Si supponga di avere già a disposizione la seguente funzione: *bool confronta(AlberoB<int> A)* che, ricevuto in input un albero binario di interi *A*, restituisce:

true se tutti i nodi del sottoalbero di sinistra hanno valore <u>minore</u> della radice di *A* e tutti i nodi del sottoalbero di destra hanno valore <u>maggiore o uguale</u> alla radice di *A*;

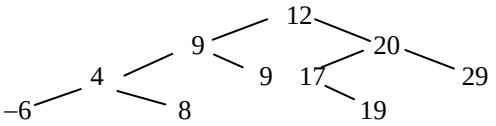
false altrimenti.

Utilizzando la funzione *confronta* e tutte le funzioni di libreria eventualmente utili, definire la funzione *bool ordinato(AlberoB<int> A)* che, ricevuto in input un albero binario di interi *A*, restituisca *true* se *A* ha le caratteristiche di un albero binario di ricerca, *false* altrimenti.

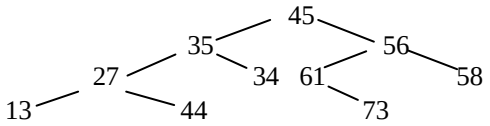
Si rammenta che, in un albero binario di ricerca, **per ogni nodo *v* dell'albero** è sempre verificata la seguente proprietà: tutti i nodi del sottoalbero sinistro di *v* hanno valore minore di *v* e tutti i nodi del sottoalbero destro hanno valore maggiore o uguale a *v*.

Esempio: per l'albero A_1 la funzione *ordinato* deve restituire *true* in quanto tutti i nodi dell'albero soddisfano la proprietà dell'albero binario di ricerca, mentre per l'albero A_2 deve restituire *false* in quanto vi è almeno un nodo (per es. il 44, il 34, il 61, il 73...) che non soddisfano la condizione richiesta.

A_1



A_2



Esercizio 3

Partizione Equilibrata - Dato un insieme A di n interi $x_1, x_2, \ldots x_n$, determinare una partizione di A in due sottoinsiemi S_1 ed S_2 tali che:

- $|S_1| + |S_2| = n$ (per $|S_i|$ si intende la cardinalità di S_i)
- e la somma degli elementi in S_1 sia quanto più possibile vicina alla somma degli elementi in S_2 (formalmente: tale che $| \Sigma_{xi \in S1} \; x_i \; - \; \Sigma_{xj \in S2} \; x_j |$ sia minima).

Descrivere un algoritmo (per grandi linee, o in pseudo-codice) che risolva tale problema in O(n log n).

Si suggerisce di considerare l'utilizzo di una delle seguenti tecniche: Divide et Impera, Golosa, Programmazione Dinamica

<u>N.B.</u> Sono ammesse anche soluzioni approssimate del problema. Chiaramente le soluzioni ottime saranno preferite.

Esempio: dato l'insieme A={23,-15, 6,-5,33,7, -2,8,54} una partizione in due sottoinsiemi S_1 ed S_2 avente la proprietà richiesta è data da S_1={-15,7,8,54} ed S_2={23,6,-5,33,-2}. Infatti, la somma degli elementi in S1 è pari a 54, mentre la somma degli elementi in S2 è pari a 55.

UNIVERSITÀDELLACALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture dati

PROVA DI LABORATORIO

Data: 8 Gennaio 2007

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input un intero *x*;
- legga da standard input una sequenza di interi terminata da -1;
- inserisca tali valori (escluso *x*) in una struttura dati 'lista';
- utilizzando gli iteratori scorra opportunamente la lista e determini se *x* è presente più di una volta nella lista;
- in caso affermativo stampi a video la posizione della prima occorrenza di *x* nella lista.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
`http://www.mat.unical.it/terracina/asd`
ed in particolare le classi *ListaInt, NodoInt ed Iteratore* disponibili all'indirizzo
`http://www.mat.unical.it/terracina/asd/ListaIntIt.zip`

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture Dati

Data: 3 Luglio 2007

Tempo: 3 ore e 15 minuti

Esercizio "apri-busta"

(è necessaria la corretta risoluzione di questo esercizio per procedere alla valutazione del compito)

Scrivere una funzione *verificaVettore* che prende in input due vettori di interi **V** e **W** (di dimensioni rispettivamente N ed M) ed un intero **k** e restituisce un valore booleano. In particolare, la funzione deve restituire **true** se per ogni elemento V[i] appartenente a V, esiste in W un elemento W[j] tale che V[i] + W[j] = k, **false** altrimenti.

Esempio: dati i vettori V=[1, 5, 2], W=[5, 10, 9, 8, 3] e k=10 la funzione deve restituire true.
dati i vettori V=[1, 5, 2], W=[5, 10, 9, 8, 3] e k=11 la funzione deve restituire false.

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione, illustrando chiaramente il ragionamento seguito.

```
void elabora( int M[][N], int V[N] ) {
    bool b=false;
    int j, i;
    char c;

    while (i<N && !b) {
        cin >> c;
        switch (c) {
            case 'a':
                if ( f2(V) )
                    for ( j=0; j<N; j++ )
                        cout << M[i, j];
                break;
            case 'b':
                if ( f1(M) && f2(V) )
                    b = true;
                else
                    cout << V[i];
                break;
            default:
                cout << V[i,];
        } //fine switch

        for ( int x=0; x<N; x++ )
            if ( M[x][i] != V[x] )
                b = true;

        i++;
    } //fine while

}
```

sapendo che la funzione **f1(M)** ha la seguente complessità:

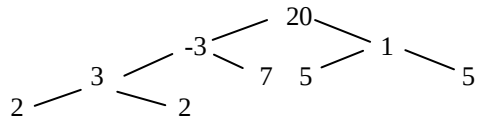
O(N)
e può terminare restituendo indifferentemente ***true* o *false***

mentre la funzione **f2(V)** ha la seguente complessità:
O(N^2)
e può terminare restituendo indifferentemente ***true* o *false***

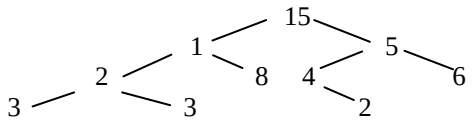
Esercizio 2

Dato un albero binario di numeri interi, scrivere una funzione che restituisca true se l'albero è *equi-totale* cioè, **per ogni nodo**, la somma dei valori presenti nel sottoalbero sinistro è esattamente uguale alla somma dei valori presenti nel sottoalbero destro. Per esempio, l'albero A_1 è equi-totale mentre l'albero A_2 non lo è (il nodo 4 ha come totale 0 per il sottoalbero sinistro e 2 per il sottoalbero destro)

A_1



A_2



Esercizio 3

Si suppone di avere a disposizione due archivi contenenti informazioni relative alle strade ed alle attività commerciali di una certa provincia italiana. In particolare, tali archivi sono così strutturati:

Archivio indirizzi: Via, Città, IdIndirizzo

Archivio attività: Nome, Tipologia, IdIndirizzo

Il campo 'IdIndirizzo' è un codice che identifica univocamente una strada ed è utilizzato nel secondo archivio per indicare l'indirizzo in cui l'attività commerciale è situata.

Si vuole realizzare un accesso efficiente ai suddetti archivi, al fine di rispondere, nel più breve tempo possibile, alle seguenti interrogazioni:

1. **Elencare i nomi di tutte le attività commerciali di una data tipologia;**
2. **Elencare i nomi di tutte le attività commerciali ubicate in un dato indirizzo (Via e Città).**

Descrivere una o più **strutture dati** idonee a realizzare tale accesso efficiente, tenendo presente che tali strutture non potranno sostituire gli archivi descritti precedentemente (già definiti e non modificabili) ma dovranno essere utilizzate esclusivamente quale supporto per l'accesso efficiente agli archivi.

Valutare inoltre la **complessità**, in termini di tempo di esecuzione, delle due interrogazioni suddette, considerando l'utilizzo delle strutture dati proposte.

N.B. si può supporre di non avere alcuna limitazione riguardo lo spazio di memorizzazione, in quanto ciò che si intende ottimizzare è unicamente il tempo di esecuzione.

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture dati

PROVA DI LABORATORIO

Data: 3 Luglio 2007

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di interi terminata da -1;
- inserisca tali valori in una struttura dati 'stack';
- legga da standard input un intero *x*;
- estragga i vari elementi presenti nello stack stampando a video quegli elementi multipli di *x*.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
`http://www.mat.unical.it/terracina/asd`
ed in particolare la classe *Stack* disponibile all'indirizzo
`http://www.mat.unical.it/terracina/asd/Stack.zip`

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture Dati

Data: 4 Settembre 2007

Tempo: 3 ore e 15 minuti

Esercizio "apri-busta"

(è necessaria la corretta risoluzione di questo esercizio per procedere alla valutazione del compito)

Scrivere una funzione *distinti* che prende in input un vettore di interi **V** e restituisce il numero di **elementi distinti** di **V**, cioè quelli che compaiono una sola volta in **V**.

Esempi: dato il vettore V=[15, 12, 3, 15, 7, 12] la funzione deve restituire 2
(infatti 3 e 7 sono gli unici 2 valori che compaiono una sola volta in V)
dato il vettore V=[15, 12, 15, 12] la funzione deve restituire 0.

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione, illustrando chiaramente il ragionamento seguito.

```
void elabora( int M[][N], int V[N] ) {
   bool b=false;
   int x, h = 0;

   for (int i=0; i<N && !b; i++) {

      cin >> x;

      if (x>0) {
         for ( int k=0; k<N; k++ )
            if ( M[k][i] != V[k] )
               b = true;
         if ( f1(M) && f2(V) )
            cout << V[i];
      }
      else {
         if ( f1(M) )
            for ( int j=0; j<N; j++ )
               cout << M[i, j];
      }

      while( h<N && !b ) {
         if ( M[i][h] < 0 )
            b = true;
         h++;
      } // fine while

   } //fine for

} // fine elabora
```

sapendo che la funzione **f1(M)** ha la seguente complessità:

O(N)
e può terminare restituendo indifferentemente ***true*** **o** ***false***

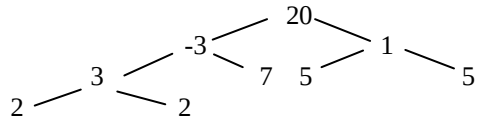
mentre la funzione **f2(V)** ha la seguente complessità:
O(N^2)
e può terminare restituendo indifferentemente ***true*** **o** ***false***

Esercizio 2

Dato un albero binario di numeri interi che rappresentano dei pesi, chiamiamo *peso del percorso nell'albero* la somma dei pesi dei nodi visitati nel percorrere l'albero dalla radice fino ad una foglia. Chiaramente si avranno diversi *pesi di percorsi* a seconda dei rami che si sceglie di percorrere. **Adottando la tecnica golosa**, scrivere una funzione che, dato un albero binario di pesi, restituisca il massimo (o una sua approssimazione visto che la tecnica golosa non garantisce il raggiungimento dell'ottimo) tra i possibili pesi dei percorsi nell'albero.

Esempio: dato l'albero A_1, il massimo peso del percorso in A_1 è 26;

A_1



Esercizio 3

Sia dato un insieme di **N documenti**, ognuno di essi caratterizzato da al più **M parole chiave**. Si vuole realizzare un motore di ricerca per l'individuazione di documenti tramite la specifica di query di ricerca. In particolare deve essere possibile fare due tipi di interrogazioni:

- ritrovare i (riferimenti ai) documenti che contengono **tutte** le parole chiave specificate;
- ritrovare i (riferimenti ai) documenti che contengono **almeno una** tra le parole chiave specificate.

Per ogni interrogazione si suppone di poter specificare al massimo tre parole chiave.

Chiaramente, un sistema di questo tipo necessita di una progettazione tale da minimizzare il più possibile i tempi di risposta.

Descrivere una o più **strutture dati** idonee a memorizzare i dati per un siffatto motore di ricerca (documenti e parole chiave associate) in modo da garantire i tempi di risposta più efficienti.

Valutare inoltre la **complessità**, in termini di tempo di esecuzione, dei due tipi di interrogazione suddetti, considerando l'utilizzo delle strutture dati proposte.

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture dati

PROVA DI LABORATORIO

Data: 4 Settembre 2007

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di caratteri terminata da '$';
- inserisca tali valori in una struttura dati 'stack';
- legga da standard input un intero *x* compreso tra 1 ed il numero di elementi inseriti nello stack;
- estragga i vari elementi presenti nello stack stampando a video solo i primi *x* elementi estratti.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
`http://www.mat.unical.it/terracina/asd`
ed in particolare la classe *Stack* disponibile all'indirizzo
`http://www.mat.unical.it/terracina/asd/Stack.zip`

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture Dati

Data: 19 Dicembre 2007

Tempo: 3 ore e 15 minuti

Esercizio "apri-busta"

(è necessaria la corretta risoluzione di questo esercizio per procedere alla valutazione del compito)

Scrivere una funzione che, ricevuti in input due vettori di interi V1 e V2, restituisca '*true*' se **ogni** elemento di V1 è presente anche in V2, '*false*' altrimenti.

Esempi: dati i vettori V1=[3,-2,3] e V2=[5,6,3,8,6,-2] il valore da restituire è '*true*';
dati i vettori V1=[4,-2,6] e V2=[5,6,3,6,-2] il valore da restituire è '*false*';

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione, illustrando chiaramente il ragionamento seguito.

```
void elabora( int M[][N], int V[N] ) {
   bool b=false;
   int i = 0;
   int k = N;  // N è una costante > 0

    while ( k > 0 ) {
            i = 0;
            for ( int x=0; x<N; x++ )
                    if ( M[x][i] == V[x] )
                            k = 0;
            //fine 1°for
            for ( int j=0; j<N && !b; j++  ){
                    if ( f(M, k) ) {
                            if ( f(M, i) )
                                    b = true;
                    }
                    else
                            for ( int x=0; x<N; x++ )
                                    if ( M[i][j] == V[x] )
                                            cout << x;
                    i++;
            } //fine 2°for
            k--;
    } //fine while
}
```

sapendo che la funzione **f(M,i)** ha la seguente complessità:

per i >= N → O(1)
e può terminare restituendo indifferentemente ***true* o *false***

per i < N → O(N)
e può terminare restituendo indifferentemente ***true* o *false***

Esercizio 2

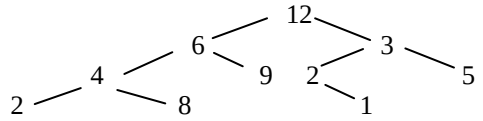
Definire la funzione *bool equiNodi(AlberoB<int> A)* che, ricevuto in input un albero binario di interi *A*, restituisca *true* se e solo se **per ogni nodo** N di *A* è verificata la seguente proprietà:

- la differenza (in valore assoluto) tra il **numero** di nodi dei due sottoalberi (figlio sinistro e figlio destro di N) è minore o uguale al **contenuto informativo** del nodo N.

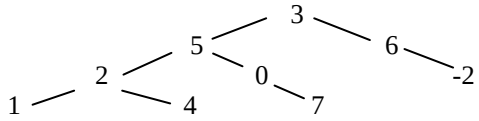
Se anche un solo nodo dell'albero non verifica la suddetta proprietà, la funzione deve restituire *false*.

Esempio: per l'albero A_1 la funzione *equiNodi* deve restituire *true*, mentre per l'albero A_2 deve restituire *false* in quanto vi è almeno un nodo (in particolare: il nodo 3, il nodo 0 ed il nodo -2) che non soddisfano la condizione richiesta.

A_1



A_2



Esercizio 3

Dato un grafo G(V,E), verificare se G è la chiusura transitiva di se stesso, e cioè: per ogni coppia di nodi i e j $\in$ V, se esiste un cammino tra i e j deve esistere anche l'arco (i,j) $\in$ E.

<u>N.B.</u> Si possono considerare come già definite e quindi richiamabili, tutte le funzioni sui grafi studiate durante il corso (compresa la funzione per la ricerca di un cammino tra due nodi: *bool trovaCammino(const Grafo& G, int start, int end, Vettore<bool> & visitato, Lista<int> cammino)*).

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture dati

PROVA DI LABORATORIO

Data: 19 Dicembre 2007

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di interi terminata da -1;
- inserisca tali valori in una struttura dati 'lista';
- legga da standard input un intero *x*;
- utilizzando gli iteratori, scorra opportunamente la lista e calcoli la somma di tutti i valori in lista che sono minori o uguali ad *x*;
- stampi a video il totale calcolato.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
`http://www.mat.unical.it/terracina/asd`
ed in particolare le classi *ListaInt, NodoInt ed Iteratore* disponibili all'indirizzo
`http://www.mat.unical.it/terracina/asd/ListaIntIt.zip`

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture Dati

Data: 8 Aprile 2008 (Appello Straordinario)

Tempo: 3 ore

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione, illustrando chiaramente il ragionamento seguito.

```
void elabora( int M[][N], int V[N] ) {
    bool b=false;
    int k=0, h=0;

    for ( int i=0; i<N; i++ ) {
        while( h<N && !b ) {
                if ( M[i][h] < 0 )
                        b = true;
                h++;
        } // fine primo while
        while (  k<N && !b ) {
                if ( !f1(M) )
                        for ( int y=0; y<N && !b; y++ )
                                for ( int j=0; j<N; j++ )
                                        M[i][j] = M[j][i] )
                else
                        for ( int x=0; x<N; x++ )
                                M[k][x] = V[x] )

                for ( int j=0; j<N; j++ )
                        if ( f2(M,V) )
                                b = true;
                k++;
        } // fine secondo while
    } // fine for esterno
}
```

sapendo che:

la funzione **f1(M)** ha complessità **O(N)** e può terminare restituendo indifferentemente ***true*** **o** ***false***

la funzione **f2(M,V)** ha complessità **O(N^2)** e può terminare restituendo indifferentemente ***true*** **o** ***false***

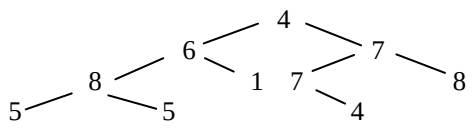
Esercizio 2

Definire la funzione *bool listaFoglie(Lista<int> L, AlberoB<int> A)* che, data una lista *L* di numeri interi ed un albero binario *A*, anche questo di numeri interi, restituisca *true* se la lista *L* contiene esattamente gli stessi valori dei nodi *foglia* dell'albero *A* presi nello stesso ordine in cui essi appaiono da sinistra verso destra, *false* altrimenti.

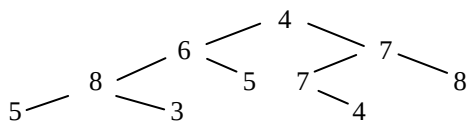
N.B.: non è lecito supporre che l'albero sia pieno!

Esempio: data la lista L_1 = [5, 5, 1, 4, 8] e l'albero A_1 la funzione *listaFoglie* deve restituire *true*. Data la lista L_1 e l'albero A_2 la funzione deve restituire *false* in quanto l'elenco degli elementi della lista L1 è diverso da quello delle foglie di A2. Data la lista L_2 = [5, 3, 5, 4, 8] e l'albero A_2 la funzione deve restituire *true*.

A_1



A_2



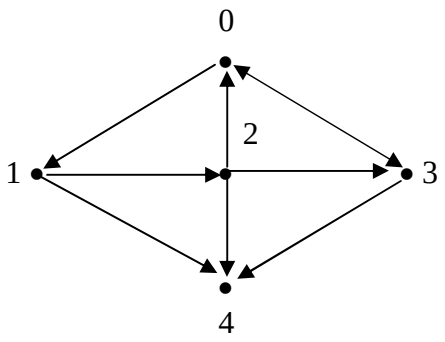
Esercizio 3

Se, in un grafo orientato *G(V,E)*, il numero di coppie di nodi $<v_i, v_j>$ (con $v_i, v_j \in V$ e $v_i \neq v_j$) che hanno almeno un nodo adiacente in comune, è maggiore o uguale ad un dato valore costante *k* (intero positivo), diciamo che *G* verifica la proprietà Ad_k.

Definire una funzione che, ricevuti in input un grafo *G(V,E)* ed un valore intero *k*, restituisca *true* se *G* verifica la proprietà Ad_k, *false* altrimenti.

Esempio: il grafo sotto raffigurato verifica la proprietà Ad_k, per valori di $k <= 4$ mentre non la verifica per valori di $k >= 5$. Infatti le coppie di nodi che hanno almeno un nodo adiacente in comune sono:

- <0, 2> (il nodo 3)
- <1, 2> (il nodo 4)
- <1, 3> (il nodo 4)
- <2, 3> (i nodi 0 e 4)



UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture dati

PROVA DI LABORATORIO

Data: 8 Aprile 2008

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di interi terminata da -1;
- inserisca tali valori in una struttura dati 'stack';
- estragga i vari elementi presenti nello stack stampando a video il valore massimo.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
`http://www.mat.unical.it/terracina/asd`
ed in particolare la classe *Stack* disponibile all'indirizzo
`http://www.mat.unical.it/terracina/asd/Stack.zip`

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture Dati

Data: 1 Luglio 2008

Tempo: 3 ore

Esercizio 1

Studiare la complessità, nel caso migliore e nel caso peggiore, della funzione *inserisci*. Si considerino a complessità costante tutti i metodi della classe Iteratore (cioè: VaiInTesta(), isNull(), getCurrentValue(), MoveNext()). Riguardo il metodo della classe ListaInt: addIncoda(), considerare sia il caso in cui la classe memorizza anche il puntatore alla coda della lista, sia il caso in cui la classe memorizza esclusivamente il puntatore alla testa della lista.

Per una valutazione positiva dell'esercizio, dovrà essere illustrato chiaramente il ragionamento seguito. Non saranno ritenute valide le soluzioni riportanti semplicemente il risultato finale di complessità (anche se corretto).

```
void inserisci (int V[N], ListaInt& L)
{
   for (int i=0; i<N; i++)
      if (!trovato(V[i], L))
            L.addInCoda(V[i]);
      else
            cout << "Il numero " << V[i]
            << " è già in lista"<< endl;
}
```

```
bool trovato(int i, ListaInt& L)
{
   Iteratore it(L);
   it.VaiInTesta();
   while (!it.isNull())
   {
      if (it.getCurrentValue() == i)
            return true;
      it.MoveNext();
   }
   return false;
}
```

Esercizio 2

Sia dato un albero binario rappresentante un torneo di calcio ad eliminazione diretta, in cui ogni squadra è rappresentata da un carattere diverso (si supponga che 'I' individui l'Italia). Il figlio sinistro ed il figlio destro di un certo nodo *n*, rappresentano le due squadre che disputano una partita ed il nodo *n* la squadra risultata vincitrice. Chiaramente, la radice dell'albero rappresenta la vincitrice dell'intero torneo.

Scrivere delle opportune funzioni per determinare:

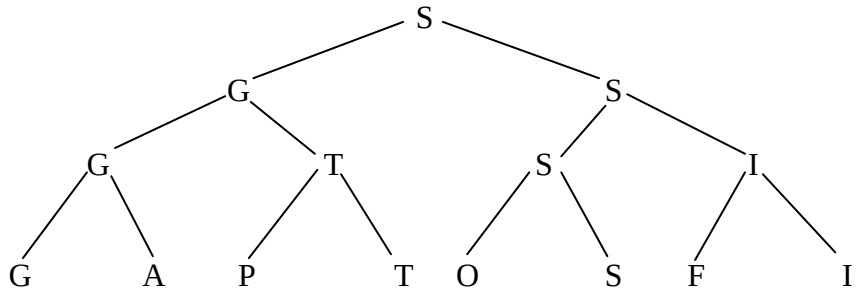
1. quante partite ha disputato l'Italia;
2. quante ne avrebbe disputate se fosse arrivata in finale.

Inoltre, scrivere una funzione che, date due squadre S1 ed S2, sapendo che S1 ha superato più scontri rispetto ad S2, restituisca il *distacco* tra le due squadre, cioè quanti rami del sottoalbero con radice S1, bisogna attraversare per raggiungere S2. La funzione dovrà restituire -1 nel caso in cui S2 non sia contenuto nel sottoalbero suddetto.

Esempio: Dato l'albero A raffigurato nella pagina seguente:

1. l'Italia (I) ha giocato 2 partite;
2. se fosse arrivata in finale ne avrebbe disputate 3.

Considerando S1='G' ed S2='P' il distacco è 2. Mentre considerando S1='G'ed S2='F' il distacco è -1.



Esercizio 3

Si supponga di aver definito la classe ‘studente’, caratterizzata da una matricola ed un insieme di esami. Per semplicità, supponiamo che gli esami siano identificati da un intero compreso tra 0 ed una costante positiva ‘N’. Gli esami superati da ogni studente sono quindi memorizzati in un vettore di booleani ‘E’. Se per un certo studente risulta E[i] = true significa che quello studente ha superato l’esame i.

Dato un insieme di studenti **S**, si vuole determinare se esiste un sottoinsieme **R** $\subseteq$ **S** tale che entrambe le seguenti condizioni siano soddisfatte:

1. l’unione degli esami superati dagli studenti in **R** comprenda TUTTI gli N esami previsti;
2. l’intersezione degli esami superati da ogni coppia di studenti in **R** sia vuota.

Esempio:

Se consideriamo l’insieme:

S = { (matr=111; E=[0,0,1,0,1,0]), (matr=112; E=[1,1,1,0,1,1]), (matr=113; E=[1,0,0,0,0,0]),
(matr=114; E=[0,1,1,0,1,0]), (matr=115; E=[0,1,0,1,0,0]), (matr=116; E=[0,0,0,0,0,1]) }

la risposta al problema è positiva, in quanto il seguente sottoinsieme soddisfa le due condizioni suddette:

R = { (matr=111; E=[0,0,1,0,1,0]), (matr=113; E=[1,0,0,0,0,0]), (matr=115; E=[0,1,0,1,0,0]),
(matr=116; E=[0,0,0,0,0,1]) }

Risolvere il problema utilizzando la tecnica di *backtracking* e l’algoritmo generico studiato durante il corso. In particolare, indicare lo spazio delle soluzioni candidate, descrivere la rappresentazione adottata per la soluzione, e definire l’implementazione (in pseudo codice) solo per le funzioni: ‘canAdd(x,sol)’ e ‘isComplete(sol)’.

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture dati

PROVA DI LABORATORIO

Data: 1 Luglio 2008

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di caratteri terminata da '&';
- inserisca tali valori in una struttura dati 'coda';
- estragga i vari elementi presenti nella coda stampando a video quei caratteri ripetuti 2 volte di seguito. Per esempio, nel caso della sequenza paautts$ dovranno essere stampati i caratteri 'a' e 't'.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
http://www.mat.unical.it/terracina/asd
ed in particolare la classe *Coda* disponibile all'indirizzo
http://www.mat.unical.it/terracina/asd/Coda.zip

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture Dati

Data: 2 Settembre 2008

Tempo: 3 ore

Esercizio 1

Studiare la complessità, nel caso migliore e nel caso peggiore, della funzione *impostaMat* in relazione ai valori N (dimensione delle matrici quadrate) ed M (dimensione del vettore).
Per una valutazione positiva dell'esercizio, dovrà essere illustrato chiaramente il ragionamento seguito. Non saranno ritenute valide le soluzioni riportanti semplicemente il risultato finale di complessità (anche se corretto).

```
void impostaMat (int D[][N], int V[M],
                 int & S[][N])
{
   for (int i=0; i<N; i++)
      for (int j=0; j<N; j++)
         S[i,j] = sommaV(V, D[i,j])
}
```

```
int sommaV(int V[M], int lim)
{
   assert(lim >=0 && lim <= M);
   int somma = 0;
   for (int i=0; i<lim; i++)
      somma += V[i];
   return somma;
}
```

Esercizio 2

Si scriva una funzione che, dato in input un vettore di caratteri, di dimensione $n=2^k$ con $k>0$, costruisca un albero binario di stringhe (completo) con le seguenti caratteristiche:

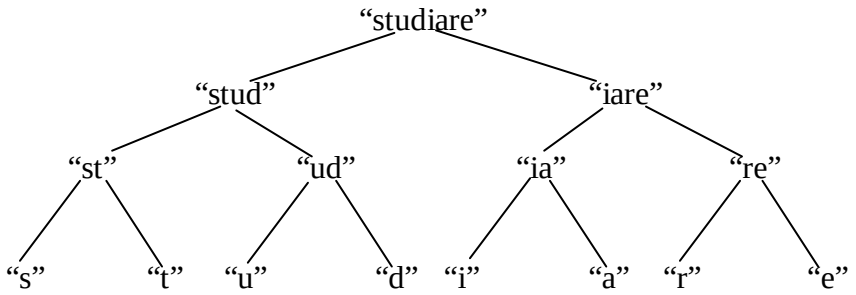
- ogni foglia ha come contenuto informativo la stringa "*c*" dove *c* è un carattere presente nel vettore; l'ordine in cui sono presenti i caratteri nelle foglie dell'albero è lo stesso in cui essi compaiono nel vettore (cioè la prima foglia a partire da sinistra ha come contenuto il carattere in prima posizione, la seconda foglia il carattere in seconda posizione e così via…);
- il contenuto informativo dei nodi ad un generico livello i (diverso dal primo) è ottenuto concatenando le stringhe presenti nei suoi due nodi figli;
- la radice contiene esattamente la stringa costituita dalla concatenazione di tutti i caratteri presenti nel vettore di input.

Si ricorda che la funzione di concatenazione di stringhe è *strcat(s1,s2,s_out)*

Esempio: Dato il seguente vettore di caratteri di dimensione $2^3=8$:

s	t	u	d	i	a	r	e

l'albero binario di stringhe da costruire è il seguente:



Esercizio 3

Si vogliono collocare un insieme di studenti in una data aula, cercando di minimizzare le possibilità di collaborazioni, in particolare, evitando di sistemare in posti adiacenti studenti tra loro amici.
Si supponga di poter identificare gli *n* studenti tramite un intero progressivo (da *1* ad *n*) e di ricevere in input una matrice quadrata *A* di dimensione *n* x *n* tale che *A[i,j] = true* se lo studente *i* è amico dello studente *j*. La matrice è simmetrica, infatti si suppone che se *i* è amico di *j* anche *j* è amico di *i*, ed ha tutti i valori nella diagonale principale uguali a *true* (ogni studente è amico di se stesso).
Gli *n* studenti devono essere collocati in un'aula che ha esattamente *n* posti sistemati su $\sqrt{n}$ file, ognuna delle quali ha $\sqrt{n}$ posti a sedere. Le assegnazioni dei posti partono dal primo posto della fila 1 fino al completamento della fila, procedendo poi analogamente per le file successive. Un determinato posto può essere assegnato ad uno studente solo se nessuno degli studenti già assegnati ai posti adiacenti al suo, sono suoi amici. Il generico caso del posto in fila *h* e posizione *k* nella fila, può essere assegnato solo a studenti che non siano amici degli studenti già sistemati nei seguenti posti:

- fila: *h* posizione: *k-1* (stessa fila, posizione precedente)
- fila: *h-1* posizione: *k-1* (fila precedente, posizione precedente)
- fila: *h-1* posizione: *k* (fila precedente, stessa posizione)
- fila: *h-1* posizione: *k+1* (fila precedente, posizione successiva)

Chiaramente, i posti più esterni (prima ed ultima fila, prima ed ultima posizione) hanno uno o più posti adiacenti in meno da controllare.

Esempio:
Data in input la matrice:

	1	2	3	4	5	6	7	8	9
1	t	t	f	t	f	t	f	t	f
2	t	t	f	t	f	f	f	f	f
3	f	f	t	f	f	t	f	t	f
4	t	t	f	t	f	f	f	f	t
5	f	f	f	f	t	f	f	t	f
6	t	f	t	f	f	t	f	f	f
7	f	f	f	f	f	f	t	f	f
8	t	f	t	f	t	f	f	t	f
9	f	f	f	t	f	f	f	f	t

una possibile sistemazione in un'aula con 3 file di 3 posti è:

	1	2	3
1	1	3	2
2	5	7	9
3	4	6	8

Risolvere il problema utilizzando la tecnica di *backtracking* e l'algoritmo generico studiato durante il corso. In particolare, indicare lo spazio delle soluzioni candidate, descrivere senza ambiguità le strutture dati utili a rappresentare la soluzione, e definire l'implementazione delle funzioni: 'canAdd(x,sol)' e 'isComplete(sol)' in base alle strutture dati scelte.

UNIVERSITÀDELLACALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture dati

PROVA DI LABORATORIO

Data: 2 Settembre 2008

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di numeri interi terminata da '-1';
- inserisca tali valori in una struttura dati 'stack';
- estragga i vari elementi presenti nello stack stampandoli a video;
- stampi, infine, il massimo tra tutti i valori in input.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
http://www.mat.unical.it/terracina/asd
ed in particolare la classe *Stack* disponibile all'indirizzo
http://www.mat.unical.it/terracina/asd/Stack.zip

UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture Dati

Data: 15 Aprile 2009 (Appello Straordinario)

Tempo: 3 ore

Esercizio 1

Studiare la complessità nel caso migliore e nel caso peggiore della seguente funzione, illustrando chiaramente il ragionamento seguito.

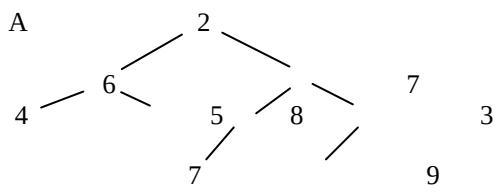
```
void elabora (int M[][N], int V[N]) {

        bool b=false;
        int i = 0, k = 0;
        while (i<N && !b) {
                if ( fun(M, V, k++) )
                        b = true;
                if ( V[i] > 0 )
                        i = max( 0, i-1 );
        else
                        i = min( i+1, N );
        }
}

N.B. le funzioni min(...) e max(...) si possono
considerare con complessità costante O(1).
```

```
bool fun ( int M[][N], int V[N], int t ) {

        for ( int k=0; k<N; k++ ) {
                if ( t<N ) {
                        if ( V[k]%2 == 0 )
                                if (M[k][k] >
V[k])
                                        return
false;
                                else {
                                        for ( int
j=0; j<N; j++ )

        if ( M[k][j] > V[j] )

        return false;
                                }
```

Esercizio 2

Dato un albero binario *A* di numeri interi positivi, scrivere un programma C++ che determini e stampi quale tra le tre visite: anticipata, posticipata ed infissa consentirebbe di stampare il maggior numero di nodi secondo la seguente regola:

stampare un valore solo se è il primo o se è maggiore dell'ultimo stampato precedentemente.



Esempio: dato l'albero A sopra raffigurato, le stampe consentite dalle tre visite sarebbero:

anticipata: 2, 6, 7, 8, 9
posticipata: 4, 5, 6, 7, 8, 9
infissa: 4, 6, 7, 8, 9

quindi la funzione deve restituire: posticipata

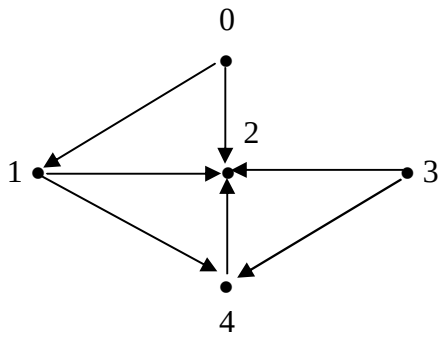
N.B. l'output della funzione deve essere solo il risultato relativo alla visita, non la stampa dei nodi.

Esercizio 3

Siano dati in input un grafo orientato *G(V,E)* ed un vettore di interi positivi *P* di dimensione pari al numero di nodi di *G*. Supponendo che ad ogni nodo $v \in V$ sia associato un peso dato da *P[v]*, scrivere una funzione che verifichi se il grafo *G* soddisfa o meno la seguente proprietà:

per ogni nodo $v \in V$ il peso di *v* moltiplicato per il numero di nodi adiacenti a *v* è maggiore della somma dei pesi dei nodi adiacenti a *v*.

Esempio: nel caso del grafo sotto raffigurato, se il vettore di pesi in input è P = [12, 9, 2, 5, 6] la funzione deve restituire *true* in quanto la proprietà suddetta è verificata da ogni nodo del grafo. Considerando invece il vettore di pesi P = [12, 9, 5, 5, 6] la funzione deve restituire *false*. Infatti il nodo 3 non soddisfa la proprietà, in quanto esso ha due nodi adiacenti: 2 e 4 e si ha P[3] * 2 = 10 che non è maggiore di P[2] + P[4] = 11.



UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Esame di Algoritmi e Strutture dati

PROVA DI LABORATORIO

Data: 15 Aprile 2009

Tempo: 30 minuti

Implementare una funzione main che:

- legga da standard input una sequenza di interi terminata da -1;
- inserisca tali valori in una struttura dati 'stack';
- estragga i vari elementi presenti nello stack stampando a video solo i valori che sono in sequenza crescente.

N.B. E' possibile usare tutte le classi di libreria definite durante il corso e disponibili sul sito:
http://www.mat.unical.it/terracina/asd
ed in particolare la classe *Stack* disponibile all'indirizzo
http://www.mat.unical.it/terracina/asd/Stack.zip