Ministero dell'Istruzione
dell'Università e Ricerca

AICA
Associazione Italiana per l'Informatica
ed il Calcolo Automatico

Olimpiadi Italiane di
INFORMATICA

Selezioni Territoriali 2013

Testi e Soluzioni ufficiali dei problemi

Problemi a cura di
Luigi Laura

Supervisione a cura del
Comitato per le Olimpiadi di Informatica:

Coordinamento
Monica Gati

Testi dei problemi
Luigi Laura, Romeo Rizzi

Soluzioni dei problemi
William Di Luigi, Luca Versari

Indice

Capitolo 1

Testi dei problemi

Selezioni Territoriali 2013
[Difficoltà D=1]

Gita a Gardaland (gardaland)

Descrizione del problema

Nel 2012 le Olimpiadi Internazionali di Informatica (IOI) si sono svolte, per la prima volta, in Italia, a Sirmione. Come da tradizione, nella giornata tra le due gare i concorrenti sono andati a divertirsi in un parco giochi, in questo caso, Gardaland. La mattina di quel giorno decine di pullman hanno prelevato i quattro ragazzi che costituiscono la squadra olimpica di ciascuna nazione dal Garda Village, dove erano stati alloggiati, e li hanno portati a Gardaland. Come sempre negli spostamenti, le varie nazioni erano state ripartite a blocco unico tra i pullman, ossia tutti gli atleti di una stessa nazione trovavano posto su uno stesso pullman. Per esempio, sul pullman dell'Italia viaggiavano anche Giappone, Israele e Irlanda. Al ritorno però, come sempre succede alle IOI, dopo una giornata in un parco giochi i ragazzi hanno fatto amicizia tra di loro, e al momento di tornare sui pullman sono saliti alla rinfusa. Grazie al lavoro delle guide, per ogni pullman è stata stilata una lista contenente, per ogni nazione, il numero di ragazzi a bordo. Il vostro compito è quello di aiutare Monica, responsabile dell'organizzazione, a capire se i pullman possono partire, ovvero se tutti i quattro ragazzi di ogni nazione che sono arrivati a Gardaland sono saliti sui pullman. In caso contrario, dovete segnalare a Monica in quanti mancano all'appello, divisi per nazioni.

Dati di input

Il file input.txt è composto da 1+*N*+*L* righe. La prima riga contiene due interi positivi separati da uno spazio: il numero *N* delle nazioni e il numero *L* di righe contenenti informazioni su chi è attualmente già salito sui pullman. (Ciascuna nazione verrà qui rappresentata con un intero compreso tra *0* e *N-1).* Ognuna delle successive *N* righe contiene un intero positivo: nella riga *i+1* (con *i >= 1*) troviamo il numero totale di ragazzi della nazione *i-1*. Ciascuna delle rimanenti *L* righe contiene due interi positivi: un intero compreso tra *0* e *N-1* che rappresenta la nazione, e un intero positivo che specifica quanti ragazzi di quella nazione sono su un certo pullman. Ovviamente una stessa nazione può comparire diverse volte nelle *L* righe, e più precisamente compare su tante righe quanti sono i pullman ospitanti atleti di quella nazione.

Dati di output

Il file output.txt è composto da una sola riga contenente l'intero 0 (zero) se non manca alcun ragazzo. Altrimenti, il file contiene 1+C righe: la prima riga contiene un intero C, ovvero il numero di nazioni che hanno ragazzi ancora a Gardaland. Le restanti C righe contengono due interi: l'identificativo della nazione e il numero di ragazzi di quella nazione che non sono ancora saliti su alcun pullman. E' necessario stampare le nazioni nell'ordine in cui sono state lette, ovvero in ordine crescente in base all'identificativo.

Assunzioni

- $2 \leq N \leq 100$
- $N \leq L \leq 1000$

- Contrariamente alle olimpiadi di informatica reali, dove gareggiano (massimo) 4 ragazzi per ogni nazione, nei casi di input si assume che ogni nazione abbia al massimo 100 ragazzi, e almeno 1 ragazzo. Quindi, indicando con *Ri* il numero di ragazzi della *i*-esima nazione, vale sempre $1 \leq Ri \leq 100$.

Esempi di input/output

File input.txt	**File** output.txt
3 5 4 4 3 0 2 1 3 0 1 2 2 1 1	2 0 1 2 1

File input.txt	**File** output.txt
3 6 4 4 4 0 2 1 3 2 1 0 2 2 3 1 1	0

Nota/e

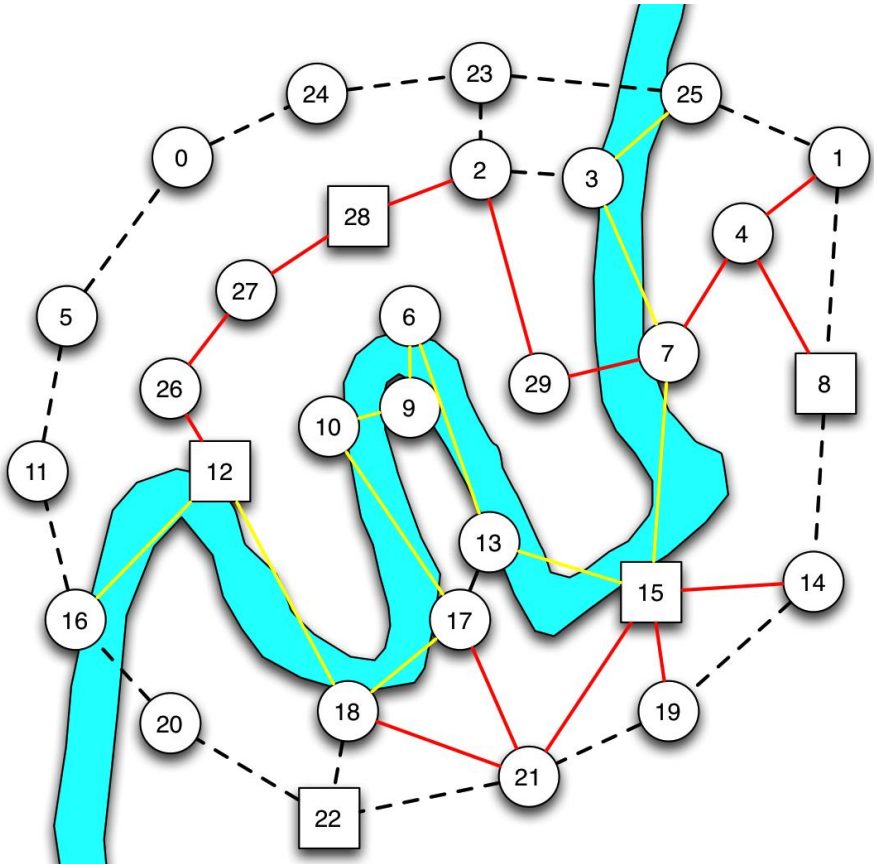
• Un programma che restituisce sempre lo stesso valore, indipendentemente dai dati in input.txt, non totalizza alcun punteggio.

Selezioni Territoriali 2013
[Difficoltà D=2]

A spasso per Brisbane (brisbane)

Descrizione del problema

Nel 2013, le IOI si svolgeranno a Brisbane (in Australia). La rappresentativa italiana ha già iniziato a studiare la città, per capire cosa ci sia di interessante da vedere, e come ci si possa spostare nella giornata libera successiva alla seconda gara delle Olimpiadi. L'offerta di trasporto pubblico a Brisbane è abbastanza variegata: ci sono due linee di bus, di cui una gratuita che gira intorno alla città, e due linee di traghetti che fermano in diversi punti del fiume Brisbane, che taglia la città in due; per quello che riguarda i prezzi, esiste un abbonamento giornaliero a tutti i trasporti pubblici, bus e traghetti insieme, oppure è possibile prendere un più economico abbonamento giornaliero ai soli traghetti, o un ancor più economico abbonamento ai soli bus.
La squadra italiana vorrà visitare il maggior numero di attrazioni possibile e per questo motivo Monica, la responsabile dell'organizzazione, ha deciso di cercare un buon compromesso tra il prezzo dei biglietti e le attrazioni che sarà possibile raggiungere partendo dall'hotel. Data una lista di attrazioni e la mappa dei collegamenti delle diverse linee del trasporto pubblico, il vostro compito è quello di aiutare Monica a capire *quante attrazioni sono raggiungibili* per ogni possibile scelta dei biglietti per i trasporti pubblici.



Per esempio, possiamo fare riferimento alla figura qui sopra, dove ad ogni fermata è associato un *cerchio* (o un *quadrato* nel caso di luogo di attrazione) e i collegamenti sono:

- tratteggiati – collegamenti gratuiti (bus gratuiti e brevi percorsi a piedi);
- rossi – bus a pagamento;
- gialli – traghetto.

Il punto di partenza della rappresentativa italiana è la fermata numero 0; le attrazioni da vedere sono quelle rappresentate con un quadrato, numerate rispettivamente 8, 12, 15, 22 e 28. Come si può vedere, spostandosi con i mezzi gratuiti si raggiungono solo due attrazioni (la numero 8 e la numero 22); comprando il biglietto del bus si raggiungono tutte le attrazioni; comprando il biglietto del traghetto si raggiungono, oltre alla 8 e la 22, anche la 12 e la 15 per un totale di quattro attrazioni. Il biglietto combinato, in questo caso, raggiunge tutte le attrazioni.

Dati di input

Il file input.txt è composto da 1+*A*+*Mg*+*Mb*+*Mt* righe. La prima riga contiene cinque interi positivi separati da uno spazio, che rappresentano il numero *N* delle fermate, il numero *A* di attrazioni, il numero *Mg* dei collegamenti gratuiti, il numero *Mb* dei collegamenti via bus e il numero *Mt* dei collegamenti via traghetto. Ogni fermata è rappresentata da un intero compreso tra 0 e N-1. Le successive *A* righe contengono ognuna una fermata (un intero compreso tra 0 e N-1) corrispondente ad una delle attrazioni che la rappresentativa italiana può visitare. Ognuna delle successive *Mg*+*Mb*+*Mt* righe contiene un collegamento del trasporto pubblico, rappresentato da due interi positivi: le fermate collegate. Le prime *Mg* righe contengono i collegamenti gratuiti (bus gratuiti e brevi percorsi a piedi), poi le successive *Mb* contengono i collegamenti del bus a pagamento e infine le ultime *Mt* righe contengono i collegamenti dei traghetti. Il punto di partenza della rappresentativa italiana è la sempre la fermata numero 0.

Dati di output

Il file output.txt è composto da 4 righe contenenti ognuna un intero non negativo, rispettivamente, il numero di attrazioni raggiungibili:

1. senza comprare biglietti (solo con mezzi gratis);
2. comprando solo il biglietto giornaliero dei bus;
3. comprando solo il biglietto giornaliero dei traghetti;
4. comprando entrambe le tipologie di biglietti.

Assunzioni

• $2 \leq N \leq 1000$

• $N \leq Mg+Mb+Mt \leq 10000$

Esempi di input/output

File input.txt	**File** output.txt
6 2 2 4 2 1 5 0 1 2 5 0 3 1 3 2 4	1 1 2 2

4 5 1 2 3 4	

File input.txt (corrisponde alla figura)	**File** output.txt
30 5 18 14 11 8 12 15 22 28 0 5 0 24 1 8 1 25 2 3 2 23 5 11 8 14 11 16 13 17 14 19 16 20 18 22 19 21 20 22 21 22 23 24 23 25 1 4 2 28 2 29 4 7 4 8 7 29 12 26 14 15 15 19 15 21 17 21 18 21 26 27 27 28 3 7 3 25 6 9 6 13 7 15 9 10 10 17 12 16 12 18 13 15 17 18	2 5 4 5

N**ota/e**

• Il secondo caso di esempio corrisponde alla situazione presentata in figura.
• Un programma che restituisce sempre lo stesso valore, indipendentemente dai dati in input.txt, non totalizza alcun punteggio.

Selezioni Territoriali 2013
[Difficoltà D=2]

Trova la parola (trovaparola)

Descrizione del problema

Visto il successo del gioco Ruzzle, che riprende il noto paroliere, i giochi basati su trovare parole stanno vivendo un periodo molto popolare. Luciano, patito di giochi di tutti i tipi, ha ideato un nuovo gioco, che funziona nel modo seguente: avete una griglia di caratteri e una parola da trovare nella griglia, partendo dalla cella in alto a sinistra. Le uniche mosse consentite sono gli spostamenti a destra o in basso. Ad esempio, considerate la seguente griglia e la parola “olimpiadi”:

O	L	I	V	E	N	T
G	Q	M	P	W	E	R
G	T	R	I	A	Y	E
I	U	I	C	D	P	E
A	F	C	O	I	G	H
J	K	X	C	V	R	S
R	O	M	I	T	A	A
S	T	A	N	L	E	E

In questo caso, la sequenza di spostamenti è “DDBDBDBB”, rappresentando gli spostamenti a destra con il carattere D e quelli in basso con il carattere B. Non esiste alcuna soluzione, invece, se la parola da cercare è “olimpionico”. Il vostro compito consiste nello scrivere un programma che, ricevute in ingresso una parola (da cercare) e una griglia, restituisca la sequenza di spostamenti, qualora esista una soluzione, oppure stampi “ASSENTE”. Se dovessero esistere molteplici sequenze di spostamenti corrette, è sufficiente stamparne una qualunque.

Dati di input

Il file input.txt è composto da 2+*R* righe. La prima riga contiene due interi positivi *R* e *C*: le dimensioni della griglia, ovvero il numero di righe *R* e il numero di colonne *C*. La riga successiva

contiene P, una parola da cercare, rappresentata da una stringa lunga almeno 2 caratteri (alfabetici maiuscoli) e al massimo *R+C-1* caratteri. Le rimanenti *R* righe del file contengono le righe della griglia, rappresentate da stringhe di *C* caratteri alfabetici maiuscoli.

Dati di output

Il file output.txt è composto da una sola riga contenente una stringa di testo: la sequenza di spostamenti necessari per trovare la parola nella griglia, se la parola è presente, oppure la stringa "ASSENTE" (senza le virgolette).

Assunzioni

• $2 \leq R, C \leq 100$

Esempi di input/output

File input.txt	**File** output.txt
8 7 OLIMPIADI OLIVENT GQMPWER GTRIAYE IUICDPE AFCOIGH JKXCVRS ROMITAA STANLEE	DDBDBDBB

File input.txt	**File** output.txt
8 7 OLIMPIONICO OLIVENT GQMPWER GTRIAYE IUICDPE AFCOIGH JKXCVRS ROMITAA STANLEE	ASSENTE

Nota/e

- Un programma che restituisce sempre lo stesso valore, indipendentemente dai dati in input.txt, non totalizza alcun punteggio.

Capitolo 2

Soluzioni dei problemi in Pascal

2.1 Gardaland

```
(* Problema: GARDALAND, selezioni territoriali 2013                    *)
(* Autore: William Di Luigi <williamdiluigi@gmail.com>                 *)
(* L'idea e' di mantenere un array, nella cui posizione i-esima        *)
(* teniamo il numero di atleti di cui la squadra i-esima e' composta. *)
(* Man mano che otteniamo informazioni sugli studenti che sono gia'    *)
(* saliti sui pulman, decrementiamo in modo opportuno e dalla          *)
(* posizione opportuna il numero di studenti che sono saliti.          *)

var
    N, L, i, idSquadra, atletiOk, risposta : longint;
    squadra : array[1..100] of longint;

begin
    (* Redireziona l'input/output da tastiera sui relativi files txt *)
    assign(input, 'input.txt');
    assign(output, 'output.txt');
    reset(input);
    rewrite(output);
    (* Leggi N e L *)
    read(N, L);
    (* Leggi il numero di studenti di ciascuna squadra *)
    for i:=1 to N do
        read(squadra[i]);
    (* Leggi tutte le L informazioni *)
    for i:=1 to L do
    begin
        read(idSquadra, atletiOk);
        (* Aumenta di uno l'id della squadra, per averlo 1-based *)
        inc(idSquadra);
        (* Sottrai il numero di atleti gia' saliti al totale di quella squadra *)
        dec(squadra[idSquadra], atletiOk);
    end;
    (* Contiamo quante sono le squadre che hanno atleti mancanti *)
    risposta := 0;
    for i:=1 to N do
        if squadra[i] > 0 then
            inc(risposta);
    (* Scrivi la risposta *)
    writeln(risposta);
    (* Scrivi quanti atleti mancano ad ogni squadra, in ordine crescente di idSquadra *)
    for i:=1 to N do
        if squadra[i] > 0 then
            writeln(i-1, ' ', squadra[i]); (* Stampa idSquadra 0-based *)
end.
```

2.2 Brisbane

```
(* Problema: BRISBANE, selezioni territoriali 2013                    *)
(* Autore: William Di Luigi <williamdiluigi@gmail.com>                *)
(* Questa e' una traduzione in Pascal della versione di Luca Versari  *)
(* scritta in Linguaggio C.  A differenza di quella,                  *)
(* per semplificare la soluzione, non uso le maschere di bit          *)
(* per selezionare i tipi di archi da prendere, rimpiazzandole con 2  *)
(* parametri booleani (prendi bus, prendi traghetto).                 *)

(* La soluzione essenzialmente consiste di 4 visite del grafo:        *)
(*   (1) usando solo gli archi gratuiti                               *)
(*   (2) usando archi gratuiti e autobus                              *)
(*   (3) usando archi gratuiti e traghetti                            *)
(*   (4) usando archi gratuiti, autobus e traghetti                   *)

var
    N, A, totArchi : longint;
    M : array[1..3] of longint; (* M[1] = Mg, M[2] = Mb, M[3] = Mt *)
    archi : array[1..20000, 1..3] of longint;
    (* 10000 archi + 10000 rovesciati *)
    isAttrazione, visto : array[1..1000] of boolean;

(* DFS = Depth First Search = Visita in profondita'.                   *)
(* Funzione di visita del grafo. Se "usaAutobus" vale True, allora    *)
(* vengono utilizzati gli archi degli autobus, altrimenti vengono     *)
(* ignorati. Analogo comportamento se "usaTraghetto" vale True/False. *)
(* La funzione restituisce il numero di attrazioni raggiungibili.     *)
function dfs(nodo : longint; usaAutobus : boolean; usaTraghetto : boolean)
    : longint;
var
    i : longint;
begin
    (* Segna come "visto" questo nodo *)
    visto[nodo] := True;
    (* Inizializza a zero il risultato,
    ovvero il numero di attrazioni visibili *)
    dfs := 0;
    (* Se il nodo in cui mi trovo e' un'attrazione,
    aumento di 1 il risultato *)
    if isAttrazione[nodo] then
        inc(dfs);
    (* Prova a guardare tutti gli archi *)
    for i:=1 to totArchi do
        (* E' un arco uscente da "nodo"?
        Mi porta ad un nuovo nodo non gia' visto? *)
        if (archi[i][1] = nodo) and not visto[archi[i][2]] then
            (* E' un arco gratis? Se no, ho comprato i biglietti adatti? *)
            if (archi[i][3] = 1) or ((archi[i][3] = 2) and usaAutobus)
            or ((archi[i][3] = 3) and usaTraghetto) then
                (* In tal caso, visita il vicino e
                aumenta le attrazioni viste di conseguenza *)
                inc(dfs, dfs(archi[i][2], usaAutobus, usaTraghetto));
end;

var
    i, j, nodo : longint;
begin
    (* Redireziona l'input/output da tastiera sui relativi files txt *)
    assign(input, 'input.txt');
    assign(output, 'output.txt');
    reset(input);
    rewrite(output);
    (* Leggi N, A, Mg, Mb, Mt *)
    read(N, A, M[1], M[2], M[3]);
```

```
64      (* Inizializza a False l'array isAttrazione *)
65      for i:=1 to N do isAttrazione[i] := False;
66      (* Leggi quali nodi sono attrazioni *)
67      for i:=1 to A do
68      begin
69          read(nodo);
70          isAttrazione[nodo] := True;
71      end;
72      (* Inizializza a zero il totale degli archi *)
73      totArchi := 0;
74      (* Per ogni tipo di arco... *)
75      for j:=1 to 3 do
76          (* ...Leggi gli archi di quel tipo *)
77          for i:=1 to M[j] do
78          begin
79              inc(totArchi);
80              (* Inserisci un nuovo arco nel grafo *)
81              read(archi[totArchi][1], archi[totArchi][2]);
82              archi[totArchi][3] := j;
83              (* j = 1/2/3 = gratis/autobus/traghetto *)
84              inc(totArchi);
85              (* Inserisci lo stesso arco, pero' rovesciato *)
86              archi[totArchi][1] := archi[totArchi-1][2];
87              archi[totArchi][2] := archi[totArchi-1][1];
88              archi[totArchi][3] := archi[totArchi-1][3];
89          end;
90      (* Inizializza a False l'array "visto" e poi chiama la visita *)
91      for i:=1 to N do
92          visto[i] := False;
93      writeln(dfs(0, False, False)); (* Usa solo mezzi gratis *)
94      for i:=1 to N do
95          visto[i] := False;
96      writeln(dfs(0, True, False)); (* Usa anche autobus *)
97      for i:=1 to N do
98          visto[i] := False;
99      writeln(dfs(0, False, True)); (* Usa anche traghetti *)
100     for i:=1 to N do
101         visto[i] := False;
102     writeln(dfs(0, True, True)); (* Usa anche autobus e traghetti *)
103 end.
```

2.3 Trova la parola

```
1   (* Problema: TROVAPAROLA, selezioni territoriali 2013                *)
2   (* Autore: William Di Luigi <williamdiluigi@gmail.com>               *)
3   (* La soluzione implementa una visita in profondita' (DFS) per       *)
4   (* cercare di comporre la parola P partendo dal suo idx-esimo        *)
5   (* carattere, sapendo che ci troviamo in posizione (i, j) nella      *)
6   (* griglia di lettere. Usiamo l'accortezza di evitare di passare due *)
7   (* volte per la stessa cella, dato che la lunghezza di qualsiasi     *)
8   (* percorso che va da (1, 1) a (i, j) e' sempre uguale, e quindi ci  *)
9   (* ritroveremmo ogni volta nella stessa cella (i, j) ma con lo stesso *)
10  (* indice idx da cui "ripartire", e gia' sappiamo che non riusciremo *)
11  (* a comporre la parola (se cosi' non fosse l'avremmo trovata prima). *)
12
13  var
14      R, C : longint;
15      M : array[1..100, 1..100] of char;
16      P : ansistring;
17      visto : array[1..100, 1..100] of boolean;
18
19  (* La funzione restituisce direttamente la stringa richiesta in      *)
20  (* output dall'esercizio. La "stringa soluzione" viene costruita     *)
21  (* concatenando un carattere ('B' o 'D' = Basso o Destra) a sinistra *)
```

```
22  (* della "stringa soluzione" per il sottoproblema che viene generato  *)
23  (* rispettivamente aprendo "in basso" o "a destra". Stiamo attenti    *)
24  (* a restituire 'ASSENTE' quando le due chiamate ricorsive hanno      *)
25  (* entrambe esito 'ASSENTE'.                                          *)
26  function dfs(i : longint; j : longint; idx : longint) : ansistring;
27  var
28      temp : ansistring;
29  begin
30      (* Se sono fuori dalla griglia, o ho gia' visto questa cella,
31      o il carattere non combacia *)
32      if (i > R) or (j > C) or visto[i][j] or (M[i][j] <> P[idx]) then
33      begin
34          dfs := 'ASSENTE';
35          exit;
36      end;
37      (* Se ho fatto combaciare tutti i caratteri *)
38      if idx = length(P) then
39      begin
40          (* Allora restituisco la stringa vuota,
41          in modo che verra' "compilata" dal chiamante *)
42          dfs := '';
43          exit;
44      end;
45      (* Segna che ho visto questa posizione
46      cosi' non ci ripasso due volte *)
47      visto[i][j] := True;
48      (* Prova ad "aprire in basso" *)
49      temp := dfs(i+1, j, idx+1);
50      if temp <> 'ASSENTE' then
51      begin
52          (* Se va bene, restituisci la soluzione *)
53          dfs := 'B' + temp;
54          exit;
55      end;
56      (* Prova ad "aprire a destra" *)
57      temp := dfs(i, j+1, idx+1);
58      if temp <> 'ASSENTE' then
59      begin
60          (* Se va bene, restituisci la soluzione *)
61          dfs := 'D' + temp;
62          exit;
63      end;
64      (* Se arrivo qui, questo ramo ricorsivo non ha soluzione *)
65      dfs := 'ASSENTE';
66  end;
67
68  var
69      i, j : longint;
70  begin
71      (* Redireziona l'input/output da tastiera sui relativi files txt *)
72      assign(input, 'input.txt');
73      assign(output, 'output.txt');
74      reset(input);
75      rewrite(output);
76      (* Leggi R, C *)
77      readln(R, C);
78      (* Leggi la parola P *)
79      readln(P);
80      (* Leggi i singoli caratteri, e intanto
81      inizializza a False l'array "visto" *)
82      for i:=1 to R do
83          for j:=1 to C do
84          begin
85              repeat
86                  read(M[i][j]);
```

```
87              until ('A' <= M[i][j]) and (M[i][j] <= 'Z');
88              visto[i][j] := False;
89          end;
90      (* Cerca partendo da (1, 1) e dall'indice 1 nella stringa P *)
91      writeln(dfs(1, 1, 1));
92  end.
```

Capitolo 3

Soluzioni dei problemi in C++

3.1 Gardaland

```cpp
/*
 * Autore: William Di Luigi <williamdiluigi@gmail.com>
 *
 * L'idea e' di mantenere un array, nella cui posizione i-esima
 * teniamo il numero di atleti di cui la squadra i-esima e' composta.
 * Man mano che otteniamo informazioni sugli studenti che sono gia'
 * saliti sui pulman, decrementiamo in modo opportuno e dalla
 * posizione opportuna il numero di studenti che sono saliti.
 */

#include <fstream>

using namespace std;

ifstream fin("input.txt");
ofstream fout("output.txt");

#define MAXN 110

int N, L, squadra[MAXN], answer;

int main()
{
    fin >> N >> L;
    for (int i=0; i<N; i++) {
        fin >> squadra[i];
    }
    for (int s,x,i=0; i<L; i++) {
        fin >> s >> x;
        squadra[s] -= x;
    }
    for (int i=0; i<N; i++) {
        answer += (squadra[i] > 0);
    }
    fout << answer << "\n";
    for (int i=0; i<N; i++) {
        if (squadra[i] > 0)
            fout << i << " " << squadra[i] << "\n";
    }
    return 0;
}
```

3.2 Brisbane

```cpp
/*
 * Autore: William Di Luigi <williamdiluigi@gmail.com>
 * Esegue una ricerca in profondita' per controllare quante attrazioni
 * sono raggiungibili partendo dal nodo 0.
 * La ricerca viene eseguita quattro volte, rispettivamente:
 *     0) Sul grafo dei mezzi gratis
 *     1) Sul grafo dei mezzi gratis / autobus
 *     2) Sul grafo dei mezzi gratis / traghetti
 *     3) Sul grafo dei mezzi gratis / autobus / traghetti
 * Ogni ricerca ci dice quante attrazioni raggiungo.
 */

#include <fstream>
#include <vector>
#include <algorithm>

using namespace std;

ifstream fin("input.txt");
ofstream fout("output.txt");

#define MAXN 1010

int N, A, M[3];
vector<int> attrazioni;
vector< pair<int,int> > archi[3];

bool visto[MAXN];
vector<int> adj[MAXN];

void dfs(int node) {
    visto[node] = true;
    for (vector<int>::iterator it=adj[node].begin(); it!=adj[node].end(); it++)
        if (!visto[*it])
            dfs(*it);
}

// Prova a raggiungere le attrazioni usando i mezzi descritti in "mask"
int solve(int mask) {
    // "Svuota" il grafo
    for (int i=0; i<MAXN; i++)
        adj[i].clear();
    // Crea il nuovo grafo
    for (int i=0; i<3; i++) if (mask & (1 << i)) {
        // Uso il mezzo i, quindi aggiungo tutti i suoi collegamenti
        for (int j=0; j<M[i]; j++) {
            int u = archi[i][j].first, v = archi[i][j].second;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
    }
    // Conta le attrazioni raggiungibili
    fill(visto, visto+N, false);
    dfs(0);
    int ret = 0;
    for (int i=0; i<A; i++) {
        if (visto[attrazioni[i]]) {
            ret ++;
        }
    }
    return ret;
}

```

```cpp
int main()
{
    fin >> N >> A >> M[0] >> M[1] >> M[2];
    attrazioni.resize(A);
    for (int i=0; i<A; i++) {
        fin >> attrazioni[i];
    }
    for (int i=0; i<3; i++) {
        for (int j=0; j<M[i]; j++) {
            int u, v;
            fin >> u >> v;
            archi[i].push_back(make_pair(u, v));
        }
    }
    // Quante attrazioni raggiungo usando solo mezzi gratis
    cout << solve(1) << "\n";
    // Quante attrazioni raggiungo combinando mezzi gratis e BUS
    cout << solve(2 | 1) << "\n";
    // Quante attrazioni raggiungo combinando mezzi gratis e TRAGHETTO
    cout << solve(4 | 1) << "\n";
    // Quante attrazioni raggiungo combinando mezzi gratis, BUS e TRAGHETTO
    cout << solve(4 | 2 | 1) << "\n";
    return 0;
}
```

3.3 Trova la parola

```cpp
/*
 * Autore: William Di Luigi <williamdiluigi@gmail.com>
 *
 * Programmazione dinamica in R*C:
 * dp[i][j] = prossimo indice (0-based) da matchare per "continuare"
 * (i, j) -> (i+1, j) oppure (i, j)  -> (i, j+1)
 */

#include <fstream>
#include <string>
#include <algorithm>

using namespace std;

ifstream fin("input.txt");
ofstream fout("output.txt");

#define MAXR 110
#define MAXC 110

int R, C;
string P, M[MAXR];

int dp[MAXR][MAXC], path[MAXR][MAXC];

#define BASSO  1
#define DESTRA 2

int main()
{
    fin >> R >> C;
    fin >> P;
    int L = P.length();
    for (int i=0; i<R; i++) {
        fin >> M[i];
    }
    if (M[0][0] != P[0]) {
```

```
        fout << "ASSENTE\n";
        return 0;
    }
    bool trovato = false;
    int ii, jj;
    // Punto di partenza
    dp[0][0] = 1;
    for (int i=0; i<R; i++) {
        for (int j=0; j<C; j++) {
            // Prova a migliorare continuando dall'alto
            if (i > 0 && dp[i-1][j] > 0 && dp[i-1][j] < L && M[i][j] == P[dp[i-1][j]]) {
                if (dp[i][j] < dp[i-1][j] + 1) {
                    dp[i][j] = dp[i-1][j] + 1;
                    path[i][j] = BASSO;
                }
            }
            // Prova a migliorare continuando da sinistra
            if (j > 0 && dp[i][j-1] > 0 && dp[i][j-1] < L && M[i][j] == P[dp[i][j-1]]) {
                if (dp[i][j] < dp[i][j-1] + 1) {
                    dp[i][j] = dp[i][j-1] + 1;
                    path[i][j] = DESTRA;
                }
            }
            // Controlla se ho finito
            if (dp[i][j] == L) {
                trovato = true;
                ii = i;
                jj = j;
            }
        }
    }
    if (!trovato) {
        fout << "ASSENTE\n";
        return 0;
    }
    // Ricostruisci i passi fatti
    string answer = "";
    while (ii != 0 || jj != 0) {
        if (path[ii][jj] == BASSO) {
            answer += "B";
            ii --;
        } else if (path[ii][jj] == DESTRA) {
            answer += "D";
            jj --;
        }
    }
    reverse(answer.begin(), answer.end());
    fout << answer << "\n";
    return 0;
}
```

Capitolo 4

Soluzioni dei problemi in C

4.1 Gardaland

```c
/*
 * Autore: Luca Versari <veluca93@gmail.com>
 * Lidea e di mantenere un array, nella cui posizione i-esima
 * teniamo il numero di atleti di cui la squadra i-esima e composta.
 * Man mano che otteniamo informazioni sugli studenti che sono gia
 * saliti sui pulman, decrementiamo in modo opportuno e dalla
 * posizione opportuna il numero di studenti che sono saliti.
 */
#include <stdio.h>
#define MAXN 110

int npers[MAXN];
int N, L, ans, i, a, b;

int main(){
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    scanf("%d%d", &N, &L);
    ans = N;
    for(i=0; i<N; i++)
        scanf("%d", &npers[i]);
    for(i=0; i<L; i++){
        scanf("%d%d", &a, &b);
        npers[a] -= b;
        if(npers[a] == 0) ans--;
    }
    printf("%d\n", ans);
    for(i=0; i<N; i++)
        if(npers[i])
            printf("%d %d\n", i, npers[i]);
    return 0;
}
```

4.2 Brisbane

```c
/*
 * Autore: Luca Versari <veluca93@gmail.com>
 *
 * Esegue una ricerca in profondita' per controllare quante attrazioni
 * sono raggiungibili partendo dal nodo 0.
 * Viene passata una mask di bit alla funzione dfs per decidere quali archi
 * usare:
```

```
 *   (1) Se il primo bit e' impostato, posso usare gli autobus
 *   (2) Se il secondo bit e' impostato, posso usare i traghetti
 *
 * Quindi ci sono quattro possibilita':
 *   (1) Se m=0, uso solo archi gratuiti
 *   (2) Se m=1, uso archi gratuiti e gli autobus
 *   (3) Se m=2, uso archi gratuiti e i traghetti
 *   (4) Se m=3, uso archi gratuiti, autobus e traghetti
 *
 * Nota: In C++, la struct adj puo' essere sostituita da un vector.
 */
#include <stdio.h>
#include <stdlib.h>
#define MAXN 1010
typedef enum{false, true} bool;

struct adj{
    int size;
    int capacity;
    int* values;
};

void adjInit(struct adj* a){
    a->size = 0;
    a->capacity = 1;
    a->values = malloc(sizeof(int));
}

void adjInsert(struct adj* a, int v){
    if(a->capacity == a->size){
        a->capacity *= 2;
        a->values = realloc(a->values, a->capacity*sizeof(int));
    }
    a->values[a->size++] = v;
}

int N, A, M[3];
struct adj graph[3][MAXN];
bool attr[MAXN];
bool visited[4][MAXN];

int dfs(int n, int m){
    int r, i, j;
    if(visited[m][n]) return 0;
    visited[m][n] = true;
    r = attr[n];
    for(j=0; j<3; j++)
        if((j&m) == j)
            for(i=0; i<graph[j][n].size; i++)
                r += dfs(graph[j][n].values[i], m);
    return r;
}

int main(){
    int i, j, a, b;
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    scanf("%d%d%d%d%d", &N, &A, &M[0], &M[1], &M[2]);
    for(i=0; i<A; i++){
        scanf("%d", &a);
        attr[a] = true;
    }
    for(j=0; j<3; j++){
        for(i=0; i<N; i++) adjInit(&graph[j][i]);
        for(i=0; i<M[j]; i++){
```

```
            scanf("%d%d", &a, &b);
            adjInsert(&graph[j][a], b);
            adjInsert(&graph[j][b], a);
        }
    }
    for(i=0; i<4; i++)
        printf("%d\n", dfs(0, i));
    return 0;
}
```

4.3 Trova la parola

```
/*
 * Autore: Luca Versari <veluca93@gmail.com>
 *
 * La soluzione implementa una visita in profondita' (DFS) per
 * cercare di comporre la parola P partendo dal suo d-esimo
 * carattere, sapendo che ci troviamo in posizione (i, j) nella
 * griglia di lettere. Usiamo l'accortezza di evitare di passare due
 * volte per la stessa cella, dato che la lunghezza di qualsiasi
 * percorso che va da (1, 1) a (i, j) e' sempre uguale, e quindi ci
 * ritroveremmo ogni volta nella stessa cella (i, j) ma con lo stesso
 * indice d da cui "ripartire", e gia' sappiamo che non riusciremo
 * a comporre la parola (se cosi' non fosse l'avremmo trovata prima).
 */
#include <stdio.h>
#define MAXR 100
typedef enum{false, true} bool;

int R, C, i;
char P[2*MAXR];
char grid[MAXR][MAXR+1];
bool vis[MAXR][MAXR];
char ans[2*MAXR];

bool dfs(int r, int c, int d){
    char cur;
    if(r>=R || c>=C) return false;
    if(grid[r][c] != P[d]) return false;
    if(!P[d+1]) return true;
    if(vis[r][c]) return false;
    vis[r][c] = true;
    cur = 0;
    if(dfs(r+1, c, d+1)) cur = 'B';
    if(dfs(r, c+1, d+1)) cur = 'D';
    if(cur) ans[d] = cur;
    return cur;
}

int main(){
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    scanf("%d%d", &R, &C);
    scanf("%s", P);
    for(i=0; i<R; i++) scanf("%s", grid[i]);
    printf("%s\n", dfs(0, 0, 0)?ans:"ASSENTE");
    return 0;
}
```