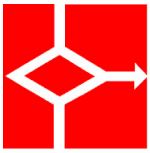

Ministero dell'Istruzione
dell'Università e Ricerca



AICA
Associazione Italiana per l'Informatica
ed il Calcolo Automatico


Olimpiadi Italiane di
INFORMATICA

# Selezioni Territoriali 2014

Testi e soluzioni ufficiali dei problemi

**Problemi a cura di**
Luigi Laura

**Coordinamento**
Monica Gati

**Testi dei problemi**
Luigi Laura, Romeo Rizzi

**Soluzioni dei problemi**
William Di Luigi, Luca Versari

**Supervisione a cura del Comitato per le Olimpiadi di Informatica**

# Indice

# 1 La Congettura di Collatz (`collatz`) [Difficoltà D=1]

## 1.1 Descrizione del problema

Consideriamo il seguente algoritmo, che prende in ingresso un intero positivo $N$:

1. Se $N$ vale $1$, l'algoritmo termina.
2. Se $N$ è pari, dividi $N$ per $2$, altrimenti (se $N$ è dispari) moltiplicalo per $3$ e aggiungi $1$.

Per esempio, applicato al valore $N = 6$, l'algoritmo produce la seguente sequenza (di lunghezza $9$, contando anche il valore iniziale $N = 6$ e il valore finale $1$):

$$6, 3, 10, 5, 16, 8, 4, 2, 1.$$

La congettura di Collatz, chiamata anche congettura $3N + 1$, afferma che l'algoritmo qui sopra termini sempre per qualsiasi valore $N$; in altri termini, se prendo un qualsiasi numero intero maggiore di $1$ applicare la regola numero 2 conduce sempre al numero $1$. È riferendosi a questa celebre congettura che il famoso matematico Erdős ha commentato sul come questioni semplici ma elusive mettono in evidenza quanto poco noi si possa accedere ai misteri del "grande Libro".

Giovanni sta cercando di dimostrare la congettura, ed è interessato alla lunghezza della sequenza. Il vostro compito è quello di aiutare Giovanni scrivendo un programma che, ricevuto in ingresso un numero $N$, calcoli la lunghezza della sequenza che si ottiene a partire da $N$.

### Dati di input

Il file `input.txt` è composto da una riga contenente $N$, un intero positivo.

### Dati di output

Il file `output.txt` è composto da una sola riga contenente un intero positivo $L$: la lunghezza della sequenza a partire da $N$.

### Assunzioni e note

- $2 \leqslant N \leqslant 1\,000$.
- È noto che, per qualsiasi $N$ minore di $1000$, la lunghezza $L$ della sequenza è minore di $200$.

### Esempio di input/output

| **File** `input.txt` | **File** `output.txt` |
|---|---|
| 6 | 9 |

| **File** `input.txt` | **File** `output.txt` |
|---|---|
| 24 | 11 |

## 1.2 Descrizione della soluzione

È sufficiente implementare l'algoritmo descritto nel testo del problema (è richiesto saper distinguere un numero pari da un numero dispari).

## 1.3 Codice della soluzione (C)

```c
#include <stdio.h>

int main() {
    // Legge da input.txt e scrive su output.txt
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    // Legge N
    int N;
    scanf("%d", &N);
    int cnt = 1;
    // Finche' N non e' 1...
    while (N != 1) {
        // Aumenta il numero di passaggi
        cnt++;
        // Calcola l'N successivo
        N = (N%2) ? (3*N+1) : (N/2);
    }
    // Stampa il risultato
    printf("%d\n", cnt);
    return 0;
}
```

## 1.4 Codice della soluzione (C++)

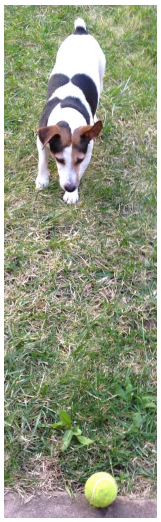
```cpp
#include <cstdio>
#include <iostream>
using namespace std;

int main() {
    // Legge da input.txt e scrive su output.txt
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    // Legge N
    int N;
    cin >> N;
    int cnt = 1;
    // Finche' N non e' 1...
    while (N != 1) {
        // Aumenta il numero di passaggi
        cnt++;
        // Calcola l'N successivo
        N = (N%2) ? (3*N+1) : (N/2);
    }
    // Stampa il risultato
    cout << N << endl;
}
```

### 1.5 Codice della soluzione (Pascal)

```pascal
var N, cnt: longint;
begin
  (* Legge da input.txt e scrive su output.txt *)
  assign(input, 'input.txt');
  assign(output, 'output.txt');
  reset(input);
  rewrite(output);
  (* Legge N *)
  read(N);
  cnt := 1;
  (* Finche' N non e' 1... *)
  while N <> 1 do
  begin
    (* Aumenta il numero di passaggi *)
    inc(cnt);
    (* Calcola l'N successivo *)
    if N mod 2 <> 0 then
      N := 3 * N + 1
    else
      N := N div 2;
  end;
  (* Stampa il numero di passaggi *)
  writeln(cnt);
end.
```

# 2 Giochiamo con Mojito (`mojito`) [Difficoltà D=2]

## 2.1 Descrizione del problema



Mojito, il jackrussell di Monica, è ormai diventato la mascotte dei Probabili Olimpici, i ragazzi che sono candidati a rappresentare l'Italia alle Olimpiadi Internazionali di Informatica 2014 a Taipei, Taiwan. Negli allenamenti a Volterra, Mojito gioca a palla con i ragazzi nel prato: lui porta la pallina al ragazzo più vicino che la calcia via; a quel punto Mojito rincorre la palla, l'acchiappa e la porta di nuovo al ragazzo che ha più vicino... e così via!

Possiamo rappresentare questo gioco con una griglia: supponendo di avere tre ragazzi che giocano con Mojito, rappresentiamo la loro posizione nella griglia, rispettivamente, con R1, R2 e R3. Tutti i ragazzi sono piuttosto metodici, e ogni volta che tirano la palla questa finisce sempre nella stessa posizione (a seconda di chi tira!): sulla griglia indichiamo con P1 il punto in cui finisce la palla tirata da R1, P2 il punto in cui finisce la palla tirata da R2, ecc... La posizione iniziale di Mojito, con la palla, è rappresentata nella griglia da una M. Mojito misura la distanza come il minimo numero di spostamenti orizzontali e/o verticali per andare da una casella a un'altra.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | R1 | | P2 | | | P1 | | |
| 2 | | | | | M | | | |
| 1 | | | | R3 | | P3 | R2 | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Per esempio, consideriamo la griglia qui sopra, di dimensione $8 \times 3$. All'inizio Mojito si trova, insieme con la palla, nella casella $(5, 2)$; il ragazzo più vicino è R3, nella posizione $(4, 1)$, che dista due caselle da lui; il gioco inizia:

- Mojito porta la palla a R3, che la tira nella casella $(6, 1)$;
- a questo punto Mojito, presa la palla, la porta a R2, nella casella $(7, 1)$, che è il più vicino a lui; da qui la palla viene tirata nella casella $(3, 3)$;
- Mojito recupera la palla e la porta a R1, nella casella $(1, 3)$; R1 tira la palla nella casella $(6, 3)$;
- da qui in poi saranno solo R1 e R2 a giocare, visto che quando tira R1 poi Mojito porta la palla a R2 e viceversa.

Notiamo che, nel caso appena descritto, tutti e tre i ragazzi hanno giocato (anche se R3 ha toccato palla solo una volta). Se Mojito ha due o più ragazzi alla stessa distanza, sceglie quello che ha la coordinata $X$ (orizzontale) minore e, se ve ne sono due o più con lo stesso valore, tra questi sceglie quello che ha la coordinata $Y$ (verticale) minore. Mojito è molto concentrato sulla palla, e non riesce a ricordarsi se tutti i ragazzi l'hanno tirata. Il vostro compito è quello di scrivere un programma che calcoli il numero di ragazzi che lanciano la palla almeno una volta!

### Dati di input

Il file `input.txt` è composto da $3+N$ righe. La prima riga contiene due interi positivi $X$ e $Y$: le dimensioni della griglia. La seconda riga contiene una coppia di interi positivi: le coordinate della posizione iniziale di Mojito con la palla. La terza riga contiene $N$, il numero di ragazzi che giocano con Mojito. Ognuna delle successive $N$ righe contiene due coppie di interi: le coordinate dell'$i$-esimo ragazzo (prima coppia di interi) e le coordinate di dove l'$i$-esimo ragazzo tirerà la palla.

### Dati di output

Il file `output.txt` è composto da una sola riga contenente un solo intero non negativo: il numero di ragazzi che giocano con Mojito, ovvero il numero di ragazzi che tirano la palla almeno una volta, a partire dalla posizione iniziale di Mojito.

### Assunzioni

- $1 \leqslant X, Y, N \leqslant 100$
- Le coordinate della griglia vanno da $1$ a $X$ e da $1$ a $Y$ (inclusi).
- Tutte le posizioni nel file di input sono distinte: non ci possono essere due ragazzi nella stessa casella, non ci sono due ragazzi che tirano nella stessa casella, nessun ragazzo tira nella casella dove c'è un altro ragazzo.
- Mojito, inizialmente, è in una casella non occupata da nessun ragazzo e dove nessun ragazzo tira la palla.
- Mojito, piccolo com'è, riesce agevolmente a passare tra le gambe dei ragazzi; non viene quindi ostacolato nel suo movimento da ragazzi presenti in una cella tra lui e la palla.

### Esempio di input/output

| **File** `input.txt` | **File** `output.txt` |
|---|---|
| `5 3`<br>`3 3`<br>`2`<br>`4 3 5 3`<br>`5 1 1 1` | `1` |

| **File** `input.txt` | **File** `output.txt` |
|---|---|
| `8 3`<br>`5 2`<br>`3`<br>`1 3 6 3`<br>`7 1 3 3`<br>`4 1 6 1` | `3` |

## 2.2 Descrizione della soluzione

Ci viene richiesto di implementare un programma che si comporti come Mojito, tuttavia l'insieme dei passi eseguiti dal jackrussell non è finito (non c'è un "passo finale"), quindi replicare esattamente il suo comportamento porterebbe in ciclo infinito (o *in loop*) il programma. Perciò, sfrutteremo il fatto che quando

Mojito passa una seconda volta dallo stesso ragazzo possiamo "bloccarlo" (cosa **difficilissima** da fare nella realtà) perché anche se lo lasciassimo continuare di certo non "visiterebbe" altri ragazzi oltre a quelli che ha già visitato (questo è vero, intuitivamente, perché sappiamo che ogni ragazzo lancia la palla sempre nel punto che ha scelto all'inizio).

Manteniamo quindi un array booleano `visited` nella cui $i$-esima posizione memorizziamo un valore che indica se abbiamo incontrato oppure no il ragazzo $i$-esimo. Il nostro programma sarà quindi composto di un ciclo che ad ogni step raggiungerà un nuovo ragazzo e che verrà eseguito fintantoché il ragazzo raggiunto non è già stato visitato. Asintoticamente, il ciclo viene eseguito $\mathcal{O}(n)$ volte.

Ad ogni step del ciclo cerchiamo il ragazzo più vicino all'attuale posizione di Mojito (risolvendo eventuali "pareggi" nel modo specificato dal testo). Una volta trovato il ragazzo ci basta controllare di averlo già visto (in caso affermativo: usciamo dal ciclo; in caso negativo: segniamo come "visto" il "nuovo" ragazzo), aumentiamo poi di $1$ il valore della soluzione ed aggiorniamo la posizione attuale di Mojito impostandola uguale alla posizione della pallina appena lanciata.

Come si fa però ad identificare "il ragazzo più vicino all'attuale posizione di Mojito"? Il modo più intuitivo sicuramente è quello di scandire uno per uno tutti i ragazzi tenendo traccia di volta in volta del più vicino. Questo ci porta ad una soluzione $\mathcal{O}(n^2)$ che per i limiti su $n$ dati dal testo è più che accettabile. Tuttavia il problema si può risolvere in tempo inferiore, con una soluzione subquadratica: lasciamo come esercizio al lettore la ricerca di un'interessante soluzione $\mathcal{O}(n \log^2 n)$. Per i più audaci, è possibile ottimizzare ulteriormente la soluzione arrivando così a $\mathcal{O}(n \log n)$.

## 2.3 Codice della soluzione (C)

```c
#include <stdio.h>
#include <stdlib.h>
#define MAXX 100
#define MAXY 100
#define MAXN 100

typedef enum{false, true} bool;

int N, X, Y;
int MX, MY;
int RX[MAXN], RY[MAXN], PX[MAXN], PY[MAXN];
bool visited[MAXN];

// Funzione per calcolare la distanza tra due posizioni
int dist(int x1, int y1, int x2, int y2) {
    return abs(x1 - x2) + abs(y1 - y2);
}

// Funzione che, dati due punti, dice se il primo e' "minore" del secondo
int isless(int x1, int y1, int x2, int y2) {
    return (x1 < x2) || (x1 == x2 && y1 < y2);
}

/* Calcola il ragazzo piu' vicino al punto (x, y) e, in caso di parita',
 * usa isless() per decidere.
 */
int nearest(int x, int y) {
    int n = 0;
    int i;
    for(i=1; i<N; i++) {
        int di = dist(x, y, RX[i], RY[i]);
        int dn = dist(x, y, RX[n], RY[n]);
        if((di < dn) || (di == dn && isless(RX[i], RY[i], RX[n], RY[n])))
            n = i;
    }
    return n;
}

int main() {
    // Legge da input.txt e scrive su output.txt
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    int i;
    // Legge le dimensioni della griglia e la posizione di Mojito
    scanf("%d%d%d%d%d", &X, &Y, &MX, &MY, &N);
    // Legge la posizione e "la cella target" di ciascun ragazzo
    for(i=0; i<N; i++)
        scanf("%d%d%d%d", &RX[i], &RY[i], &PX[i], &PY[i]);
    // Inizializza a zero la risposta
    int count = 0;
    // Ripeti all'infinito
    while(true) {
        // Trova il ragazzo piu' vicino
        int nx = nearest(MX, MY);
```

```
55          // Se l'ho gia' visto, mi fermo (altrimenti vado in loop)
56          if(visited[nx])
57              break;
58          // Segno come "visto" questo ragazzo
59          visited[nx] = true;
60          // Aumenta il numero di ragazzi visti
61          count++;
62          // Aggiorna la posizione di Mojito
63          MX = PX[nx];
64          MY = PY[nx];
65      }
66      // Scrivi il risultato
67      printf("%d\n", count);
68      return 0;
69  }
```

## 2.4 Codice della soluzione (C++)

```cpp
#include <iostream>
#include <utility>
#include <cstdio>
#include <cstdlib>
using namespace std;

const int MAXX = 100;
const int MAXY = 100;
const int MAXN = 100;

int N, X, Y;
pair<int, int> posm;
pair<int, int> posr[MAXN], posl[MAXN];
bool visited[MAXN];

// Funzione per calcolare la distanza tra due posizioni
int dist(pair<int, int> p1, pair<int, int> p2) {
    return abs(p1.first - p2.first) + abs(p1.second - p2.second);
}

/* Calcola il ragazzo piu' vicino al punto p e, in caso di parita', scegli il
 * minore (nel senso lessicografico).
 */
int nearest(pair<int, int> p) {
    int n = 0;
    for(int i=1; i<N; i++) {
        int di = dist(p, posr[i]);
        int dn = dist(p, posr[n]);
        if((di < dn) || (di == dn && posr[i] < posr[n]))
            n = i;
    }
    return n;
}

int main() {
    // Legge da input.txt e scrive su output.txt
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    // Legge le dimensioni della griglia e la posizione di Mojito
    cin >> X >> Y >> posm.first >> posm.second >> N;
    // Legge la posizione e "la cella target" di ciascun ragazzo
    for(int i=0; i<N; i++)
        cin >> posr[i].first >> posr[i].second
            >> posl[i].first >> posl[i].second;
    // Inizializza a zero la risposta
    int count = 0;
    // Ripeti all'infinito
    while(true) {
        // Trova il ragazzo piu' vicino
        int nx = nearest(posm);
        // Se l'ho gia' visto, mi fermo (altrimenti vado in loop)
        if(visited[nx])
            break;
        // Segno come "visto" questo ragazzo
```

```
55          visited[nx] = true;
56          // Aumenta il numero di ragazzi visti
57          count++;
58          // Aggiorna la posizione di Mojito
59          posm = posl[nx];
60      }
61      // Scrivi il risultato
62      cout << count << endl;
63  }
```

## 2.5 Codice della soluzione (Pascal)

```
const
  MAXX = 100;
  MAXY = 100;
  MAXN = 100;
var
  N, X, Y: longint;
  MX, MY: longint;
  RX, RY, PX, PY: array[0..MAXN-1] of longint;
  visited: array[0..MAXN-1] of boolean;

(* Funzione che calcola, dati due punti, la loro distanza *)
function dist(x1: longint; y1: longint; x2: longint; y2: longint): longint;
begin
  dist := abs(x1-x2) + abs(y1-y2);
end;

(* Funzione che, dati due punti, dice se il primo e' "minore" del secondo *)
function isless(x1: longint; y1: longint; x2: longint; y2: longint): boolean;
begin
  isless := (x1 < x2) or ((x1 = x2) and (y1 < y2));
end;


(* Calcola il ragazzo piu' vicino al punto (x, y) e, in caso di parita',    *)
(* usa isless() per decidere.                                                *)
function nearest(x: longint; y: longint): longint;
var
  i: longint;
  di, dn: longint;
begin
  nearest := 0;
  for i:=1 to N-1 do
  begin
    di := dist(x, y, RX[i], RY[i]);
    dn := dist(x, y, RX[nearest], RY[nearest]);
    if (di < dn) or ((di = dn) and
                     isless(RX[i], RY[i], RX[nearest], RY[nearest])) then
      nearest := i;
  end;
end;

var
  i: longint;
  count, nx: longint;
begin
  (* Inizializza a False l'array visited *)
  fillchar(visited, sizeof(Visited), False);
  (* Legge da input.txt e scrive su output.txt *)
  assign(input, 'input.txt');
  assign(output, 'output.txt');
  reset(input);
  rewrite(output);
  (* Legge le dimensioni della griglia e la posizione di Mojito *)
  readln(X, Y, MX, MY, N);
```

```
55    (* Legge la posizione e "la cella target" di ciascun ragazzo *)
56    for i:=0 to N-1 do
57      readln(RX[i], RY[i], PX[i], PY[i]);
58    (* Inizializza a zero la risposta *)
59    count := 0;
60    (* Ripeti all'infinito *)
61    while True do
62    begin
63      (* Trova il ragazzo piu' vicino *)
64      nx := nearest(MX, MY);
65      (* Se l'ho gia' visto, mi fermo (altrimenti vado in loop) *)
66      if visited[nx] then
67        break;
68      (* Segno come "visto" questo ragazzo *)
69      visited[nx] := True;
70      (* Aumenta il numero di ragazzi visti *)
71      inc(count);
72      (* Aggiorna la posizione di Mojito *)
73      MX := PX[nx];
74      MY := PY[nx];
75    end;
76    (* Scrivi il risultato *)
77    writeln(count);
78  end.
```

# 3 Corso per Sommelier (`sommelier`) [Difficoltà D=2]

## 3.1 Descrizione del problema

Paolo, per festeggiare il suo quarantesimo compleanno, si è iscritto a un corso per sommelier, dove impara a distinguere ed apprezzare le diverse tipologie di vini. Si è accorto però che, nonostante prenda solo un assaggio di ogni tipo di vino, per lui vale la regola fondamentale delle bevande alcoliche: quando le bevi, mai scendere di gradazione. Infatti, se per esempio Paolo assaggia un vino da 9 gradi e poi uno da 7, il giorno dopo si sveglierà con un grosso mal di testa indipendentemente dalle quantità.

Per fortuna, in ogni serata del corso è disponibile l'elenco dei vini che verranno portati uno dopo l'altro, e di ogni vino viene riportata la gradazione alcolica. Non è ammesso mettere da parte un vino per berlo in seguito: ogni volta che gli viene passato un vino Paolo può decidere se assaggiarlo o meno, versandone un poco nel suo Tastevin.

Inoltre, dal momento che dopo aver assaggiato un vino Paolo deve pulire accuratamente il suo Tastevin con un panno, questa operazione in pratica gli impedisce di assaggiare due vini consecutivi (nell'immagine qui a fianco potete vedere il Tastevin). Paolo desidera assaggiare il maggior numero di vini possibile.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| Cilento | Barolo | Lambrusco | Picolit | Verdicchio | Cannonau | Chianti | Pigato | Donzelle |
| 11 | 13 | 10 | 16 | 12 | 12 | 13 | 11 | 13 |

Ad esempio, se in una serata serviranno i vini mostrati nella tabella qui sopra, nell'ordine in cui compaiono nella tabella, il numero massimo di vini che Paolo può riuscire ad assaggiare, rispettando la regola, è quattro: può iniziare, indifferentemente, con il Cilento o con il Lambrusco, e poi assaggiare Verdicchio, Chianti e Donzelle. In questa maniera, la sequenza delle gradazioni alcoliche non scende mai: 11 (oppure 10), 12, 13, 13. Ovviamente, come si vede nell'esempio, è possibile bere due o più vini con la stessa gradazione alcolica.

### Dati di input

Il file `input.txt` è composto da 2 righe. La prima riga contiene $N$, un intero positivo: il numero di vini che saranno serviti nella serata. La seconda riga contiene $N$ interi positivi: le gradazioni alcoliche dei vini che saranno serviti, nell'ordine in cui saranno serviti.

### Dati di output

Il file `output.txt` è composto da una sola riga contenente un solo intero positivo: il numero massimo di vini che Paolo può assaggiare nella serata, rispettando la regola di non diminuire la gradazione alcolica nella sequenza e, contemporaneamente, il vincolo di dover pulire il Tastevin, che gli impedisce di assaggiare due vini consecutivi.

### Assunzioni

- $2 \leqslant N \leqslant 99$.
- I vini hanno una gradazione alcolica compresa tra 1 e 99.

**Esempio di input/output**

| **File** `input.txt` | **File** `output.txt` |
|---|---|
| 9<br>11 13 10 16 12 12 13 11 13 | 4 |

| **File** `input.txt` | **File** `output.txt` |
|---|---|
| 12<br>11 13 11 10 11 12 16 12 12 11 10 14 | 5 |

## 3.2 Descrizione della soluzione

Utilizziamo la tecnica della *Programmazione dinamica*. L'idea è di mantenere un array (`din`) che contiene nella posizione $i$ il numero massimo di vini che Paolo può bere sapendo che il primo vino bevuto è il numero $i$. L'array viene aggiornato a partire dall'ultimo vino: il valore di una cella è semplicemente il massimo valore delle celle successive (facendo attenzione però ad "ignorare" la cella immediatamente successiva, per via della limitazione di dover ripulire il *tastevin*) che corrispondono ad un vino con una gradazione alcolica non superiore.

Per calcolare il valore di una cella, asintoticamente, impieghiamo tempo $\mathcal{O}(n)$. Dal momento che dobbiamo calcolare il valore per $n$ celle, la soluzione avrà una complessità globale pari a $\mathcal{O}(n^2)$, che per i limiti su $n$ dati dal testo è più che accettabile. Tuttavia il problema si può risolvere in tempo inferiore, con una soluzione subquadratica: lasciamo come esercizio al lettore la ricerca di una soluzione con complessità $\mathcal{O}(n \log n)$.

### 3.3 Codice della soluzione (C)

```c
#include <stdio.h>
#define MAXN 99

int N;
int din[MAXN];
int vini[MAXN];

int main() {
    // Legge da input.txt e scrive su output.txt
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    int i, j;
    // Legge le gradazioni alcoliche dei vini.
    scanf("%d", &N);
    for(i=0; i<N; i++)
        scanf("%d", &vini[i]);
    // A partire dall'ultimo vino...
    for(i=N-1; i>=0; i--) {
        // Calcola il vino migliore per proseguire tra tutti quelli che
        // vanno bene.
        int max = 0;
        for(j=i+2; j<N; j++)
            if(vini[j] >= vini[i] && din[j] > max)
                max = din[j];
        // La piu' lunga sequenza a partire da questo vino e' pari a 1 piu'
        // la piu' lunga sequenza a partire dal miglior vino successivo.
        din[i] = max + 1;
    }
    // Trova il miglior vino da cui iniziare.
    int max = din[0];
    for(i=1; i<N; i++)
        if(din[i] > max)
            max = din[i];
    // Stampa il risultato.
    printf("%d\n", max);
    return 0;
}
```

### 3.4 Codice della soluzione (C++)

```cpp
#include <cstdio>
#include <iostream>
#include <algorithm>
using namespace std;

const int MAXN = 99;

int N;
int din[MAXN];
int vini[MAXN];

int main() {
    // Legge da input.txt e scrive su output.txt
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    // Legge le gradazioni alcoliche dei vini.
    cin >> N;
    for(int i=0; i<N; i++)
        cin >> vini[i];
    // A partire dall'ultimo vino...
    for(int i=N-1; i>=0; i--) {
        // Calcola il vino migliore per proseguire tra tutti quelli che
        // vanno bene.
        int max = 0;
        for(int j=i+2; j<N; j++)
            if(vini[j] >= vini[i] && din[j] > max)
                max = din[j];
        // La piu' lunga sequenza a partire da questo vino e' pari a 1 piu'
        // la piu' lunga sequenza a partire dal miglior vino successivo.
        din[i] = max + 1;
    }
    // Trova il miglior vino da cui iniziare e stampalo
    cout << *max_element(din, din+N) << endl;
}
```

## 3.5 Codice della soluzione (Pascal)

```
const
  MAXN = 99;
var
  N, i, j, max : longint;
  din : array[1..MAXN] of longint;
  vini: array[1..MAXN] of longint;
begin
  (* Legge da input.txt e scrive su output.txt *)
  assign(input, 'input.txt');
  assign(output, 'output.txt');
  reset(input);
  rewrite(output);
  (* Legge le gradazioni alcoliche dei vini. *)
  read(N);
  for i:=1 to N do
  begin
    read(vini[i]);
    din[i] := 0;
  end;
  (* A partire dall'ultimo vino... *)
  for i:=N downto 1 do
  begin
    (* Calcola il vino migliore per proseguire tra tutti quelli che *)
    (* vanno bene.                                                  *)
    max := 0;
    for j:=i+2 to N do
      if (vini[j] >= vini[i]) and (din[j] > max) then
        max := din[j];
    (* La piu' lunga sequenza a partire da questo vino e' pari a 1 piu' *)
    (* la piu' lunga sequenza a partire dal miglior vino successivo.    *)
    din[i] := max + 1;
  end;
  (* Trova il miglior vino da cui iniziare. *)
  max := 0;
  for i:=1 to N do
    if din[i] > max then
      max := din[i];
  (* Stampa il risultato. *)
  writeln(max);
end.
```