Ropes Escape (ropes)

Risolto scaricando dagli Allegati il template di programma C++

```
/*
 * This template is valid both in C and in C++,
 * so you can expand it with code from both languages.
 * NOTE: it is recommended to use this even if you don't
 * understand the following code.
 */

#include <stdio.h>
#include <assert.h>

// constraints
#define MAXN 100000

// input data
int N, i;
int P[MAXN], T[MAXN];

int main() {
//  uncomment the following lines if you want to read/write from files
//  freopen("input.txt", "r", stdin);
//  freopen("output.txt", "w", stdout);

    assert(1 == scanf("%d", &N));
    for(i=0; i<N; i++)
        assert(1 == scanf("%d", &P[i]));
    for(i=0; i<N; i++)
        assert(1 == scanf("%d", &T[i]));

    // insert your code here

    printf("%d\n", 42); // print the result
    return 0;
}
```

```
4  // constraints
5  #define MAXN 100000
6
7  // input data
8  int N, i;
9  int P[MAXN], T[MAXN], max[MAXN + 1];
10
11 int main() {
12     freopen("input1.txt", "r", stdin);
13     freopen("output.txt", "w", stdout);
14
15     assert(1 == scanf("%d", &N));
16     for(i=0; i<N; i++)
17         assert(1 == scanf("%d", &P[i]));
18     for(i=0; i<N; i++)
19         assert(1 == scanf("%d", &T[i]));
20
21 int total = 0;
22     for(i = 0; i < N; i++)
23     {
24         if(T[i] > max[P[i]])
25         {
26             total = total + max[P[i]];
27             max[P[i]] = T[i];
28         }
29         else
30         {
31             total = total + T[i];
32         }
33     }
34
35     printf("%d\n", total); // print the result
36     return 0;
37 }
38
```

Tolto il commento alle linee di dichiarazioni dei file (linee 12 e 13), completate le dichiarazioni con l'array ***max*** e la variabile ***total*** , l'alunno ha introdotto le line del suo codice (linee da 21 a 33) e sostituito 42 con ***total*** nel *printf*.

La prof.ssa Finocchi, in alternativa, suggerisce di utilizzare la struttura dati **coda con priorità** (inserendo tutte le corde, ogni volta se ne estrae una, quella di costo minimo, se non ha fratelli, ok, altrimenti la si può riagganciare ad una qualsiasi foglia)

Somme costose (somme)

```
7  void merge(int a[], int p, int q, int r);
8  void mergeSort(int a[], int p, int r);
9
10 int main(){
11     freopen("input.txt", "r", stdin);
12     freopen("output.txt", "w", stdout);
13
14     int N,atc;
15     long long int tot=0;
16     cin>>N;
17     int O[N];
18     for(int i=0;i<N;i++)
19         cin>>O[i];
20     mergeSort(O,0,N-1);
21     for(int i=0;i<N-1;i++){
22         tot+=O[i]+O[i+1];
23         O[i+1]=O[i]+O[i+1];
24         mergeSort(O,i,N-1);
25     }
26     cout<<tot;
27 }
28 void merge(int a[], int p, int q, int r) {
29   int i, j, k=0, b[max];
30   i = p;
31   j = q+1;
```

La soluzione proposta prevede:

- lettura dell'input nell'array O[N]
- iterazione dei passi:
    1. ordinamento di O con ***mergesort*** (funzione spiegata successivamente dalla prof.ssa Finocchi)
    2. sostituzione della somma al posto del 2^ addendo
    3. incremento dell'indice ***i***

la professoressa suggerisce una soluzione **più efficiente** per evitare di eseguire ad ogni passo il *mergesort*:

inserire al posto giusto la somma ottenuta previa **ricerca binaria** della posizione corretta

(la ricerca binaria ha tempo lineare di esecuzione)

in modo da non dover riordinare dopo ogni somma

FUNZIONI RICORSIVE

Un problema ricorsivo si può risolvere iterativamente, ma spesso il codice risulta meno leggibile

esempio funzione RICORSIVA: ***moveTower*** per le **Torri di Hanoi**

Un'implementazione elegante (e sintetica!)

```
void moveTower(int n, char from, char to, char temp) {
    if (n == 1) {
        moveSingleDisk(from, to);
    } else {
     1) moveTower(n – 1, from, temp, to);
     2) moveSingleDisk(from, to);
     3) moveTower(n – 1, temp, to, from);
    }
}
```

from temp to f t to

LUISS Irene Finocchi / Preparazione Olimpiadi Informatica 18

Per 3 dischi (n = 3, n-1 vale 2)

1) moveTower (2, from, temp, to); determina la configurazione 1) con dischi piccolo e medio nella torre **t=temp** utilizzando la torre **to** per appoggio temporaneo del disco piccolo

2) moveSingleDisk (from, to); sposta il disco grande dalla torre **from** alla torre **to**

3) moveTower (2, temp, to, from); sposta i dischi piccolo e medio nella torre **to** utilizzando la **from** per appoggio temporaneo del disco piccolo

funzione moveTower:

- 1^ parametro: numero di dischi da spostare
- 2^ parametro: torre **origine** dello spostamento
- 3^ parametro: torre **destinazione** dello spostamento
- 4^ parametro: torre da usare per appoggio **temporaneo**

problemi *DIVIDE et IMPERA (divide & conquer)*

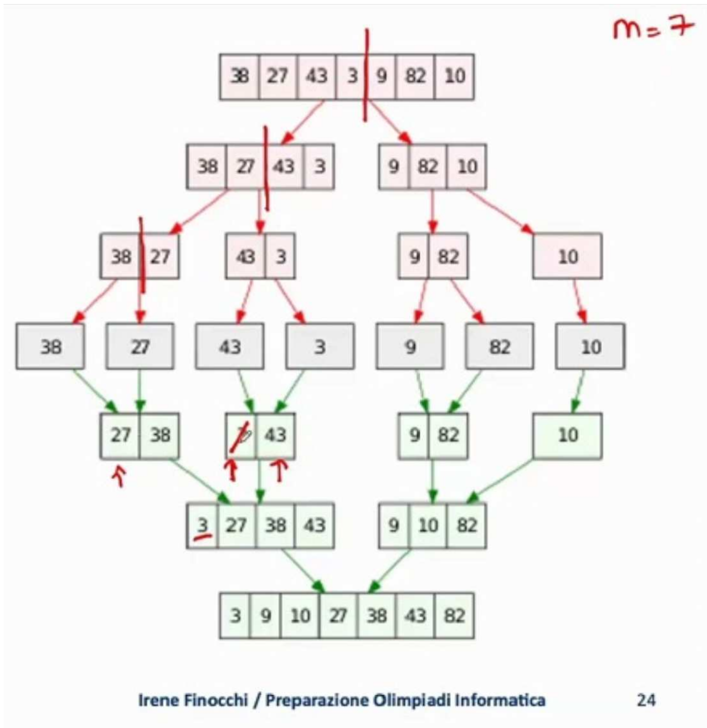
questi problemi vengono risolti con **funzioni ricorsive**

esempio di problema *divide & conquer* risolto con RICORSIVA: **mergesort**

```
/* s e d sono gli indici sinistro e destro del  sottoarray da ordinare */
    void mergeSort (int arr[], int s, int d) {

        if (s < d) {
         int m = (s + d)/2;                 // divide
            mergeSort(arr,   s,   m);       // conquer
            mergeSort(arr, m +1, d);        // conquer
            fondi(arr, s, m, d);            // combine
        }
    }
```

```
/* s e d sono gli indici sinistro e destro del
   sottoarray da ordinare */
void mergeSort(int arr[], int s, int d) {
     if (s<d) {
          int m = (s+d)/2;          // divide
          mergeSort(arr,s,m);       // conquer
          mergeSort(arr,m+1,d);     // conquer
          fondi(arr,s,m,d);         // combine
     }
}
```

Il processo evita una ricorsione infinita perché si arresta quando s raggiunge d (per s = d non esegue azioni e il controllo passa alla funzione chiamante)



La sequenza di 7 numeri viene divisa in 2 sottosequenze, una lunga 4, l'altra 3 (n/2 = 3,5)

In pratica si procede

> nella **suddivisione successiva in 2 sottosequenze** (frecce rosse) fino ad ottenere singoli numeri [38] [27] etc.;

> quindi con il merge di 2 sottosequenze alla volta:

1^ livello (sottosequenze con 1 elemento):

considerando le sottosequenze [38] e [27] basta confrontare i 2 numeri e creare una sequenza di 2^ livello (sottos. da 2) con 27 e poi 38 [27 | 38]

2^ livello (sottosequenze da 2):

- si confrontano i <u>primi numeri</u> (27 e 3) delle sottosequenze da 2 elementi [**27** | 38] e [**3** | 43] e si scrive il più piccolo: **3**
- si confrontano i <u>primi numeri</u> rimasti per ogni sottoseq. (27 nella sottoseq. sinistra e 43 nella sottoseq. destra) e si scrive il più piccolo: **27**
- si confrontano i numeri rimasti (38 e 43) e si scrive il più piccolo: **38**
- resta solo un numero (**43**) nella seconda sottosequenza che va quindi trascritto in coda alla sottosequenza da 4 elementi [3 | 27 | 38 | 43]

Si procede analogamente per le sottosequenze di destra ottenendo una sottosequenza da 3

3^ livello (una sottosequenza da 4 e una da 3):

Si procede in modo analogo per la fusione finale delle sottosequenze di 3^ livello in una lunga 7 [3 | 27 | 38 | 43] e [9 | 10 | 82]

Un esempio della funzione di fusione delle sottostringhe (***fondi***)

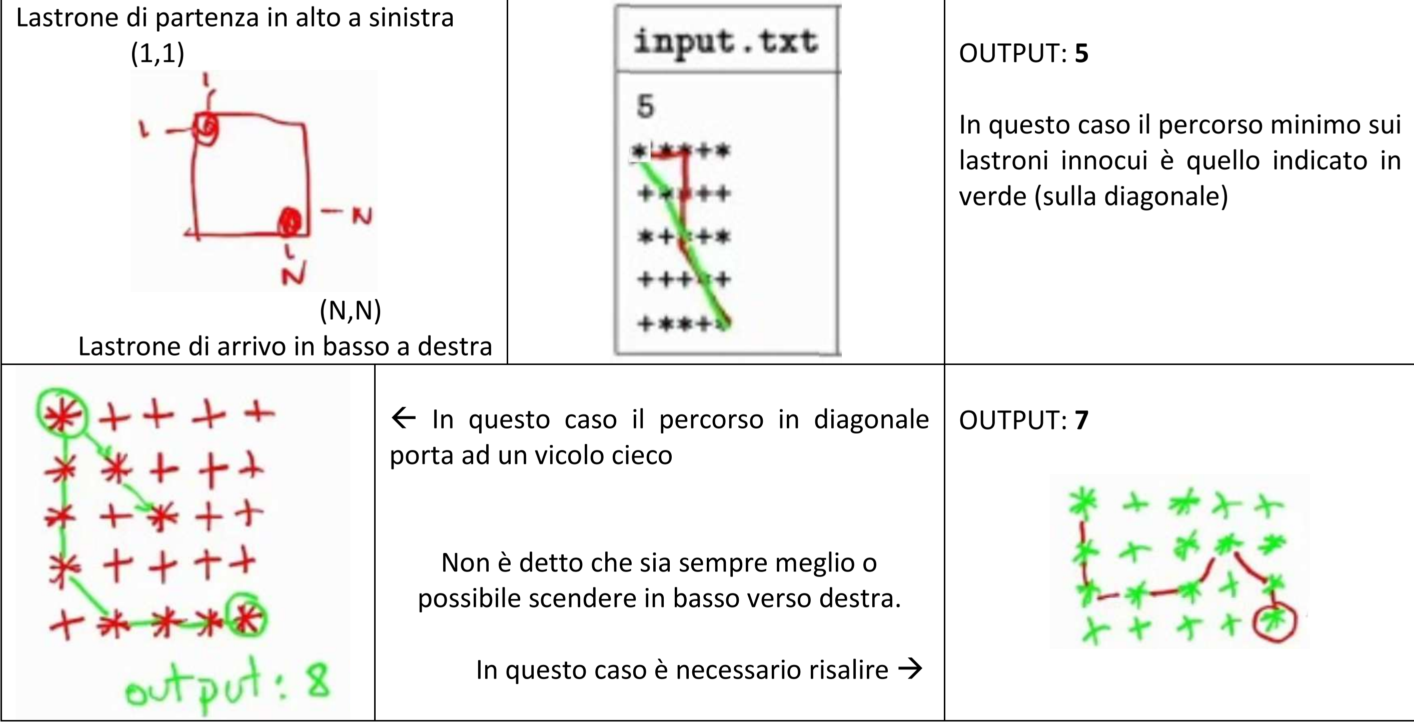
(fonte: https://it.wikibooks.org/wiki/Implementazioni_di_algoritmi/Merge_sort)

```
#include <iostream>
using namespace std;
#define SP ' '
#define N 7

 int aux[N+1]; //[right-left+1];

void fondi(int a[], int left, int center, int right)
{

       int i,j;
       for (i = center+1; i > left; i--)
             aux[i-1] = a[i-1];
       for (j = center; j < right; j++)
             aux[right+center-j] = a[j+1];
       for (int k = left; k <= right; k++)
             if (aux[j] < aux[i])
                   a[k] = aux[j--];
             else
                   a[k] = aux[i++];
}
/* s e d sono gli indici sinistro e destro
    del  sottoarray da ordinare */

void mergeSort (int arr[], int s, int d) {

    if (s < d) {
      int m = (s + d)/2;              // divide
        mergeSort(arr,    s,    m);   // conquer
        mergeSort(arr, m +1, d);      // conquer
        fondi(arr, s, m, d);          // combine
    }
}
```

```
int main(){

int V[N]={38,27,43,3,9, 82,10};

   for (int i=0; i<=6; i++)
        cout << V[i] << SP;
   cout << endl;

   mergeSort(V, 0, N-1);

   for (int i=0; i<N; i++)
        cout << V[i] << SP;
    return 0;
}
```

Esercizio dal Portale: **Mappa antica (mappa) – soluzione RICORSIVA**

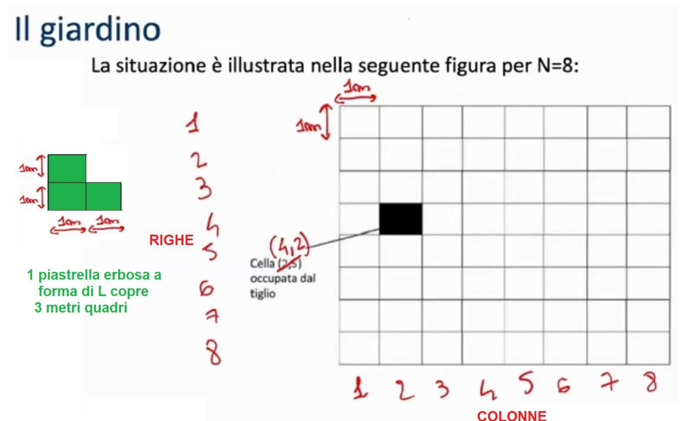
Per le celle interne sono possibili 8 spostamenti (alto, basso, sinistra, destra e sulle 4 diagonali)

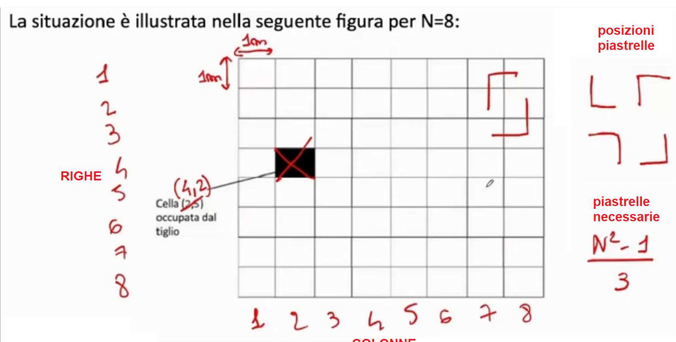


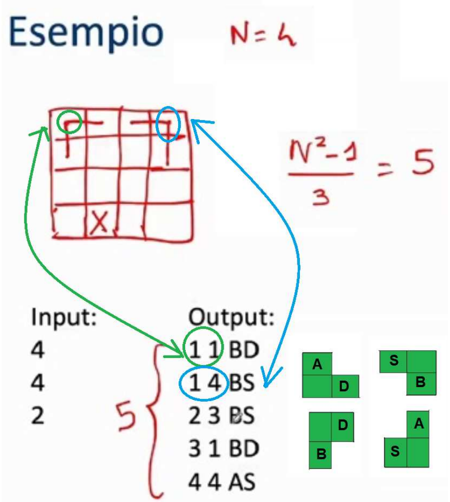
Lastrone di partenza in alto a sinistra (1,1) … (N,N) Lastrone di arrivo in basso a destra	input.txt	OUTPUT: **5** In questo caso il percorso minimo sui lastroni innocui è quello indicato in verde (sulla diagonale)
output: 8	← In questo caso il percorso in diagonale porta ad un vicolo cieco Non è detto che sia sempre meglio o possibile scendere in basso verso destra. In questo caso è necessario risalire →	OUTPUT: **7**

La soluzione proposta da un alunno è praticamente identica a quella descritta nella **Guida Bugatti** pag. 56

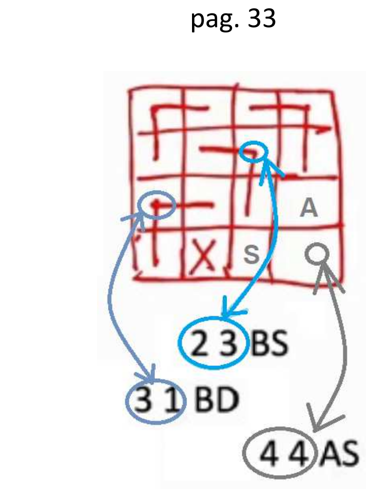
Prato Altro esempio di problema di tipo **divide et impera** che può risolversi con una funzione ricorsiva pag.28 (problema proposto come simulazione di gara nel corso LUISS dello scorso anno)







pag. 33

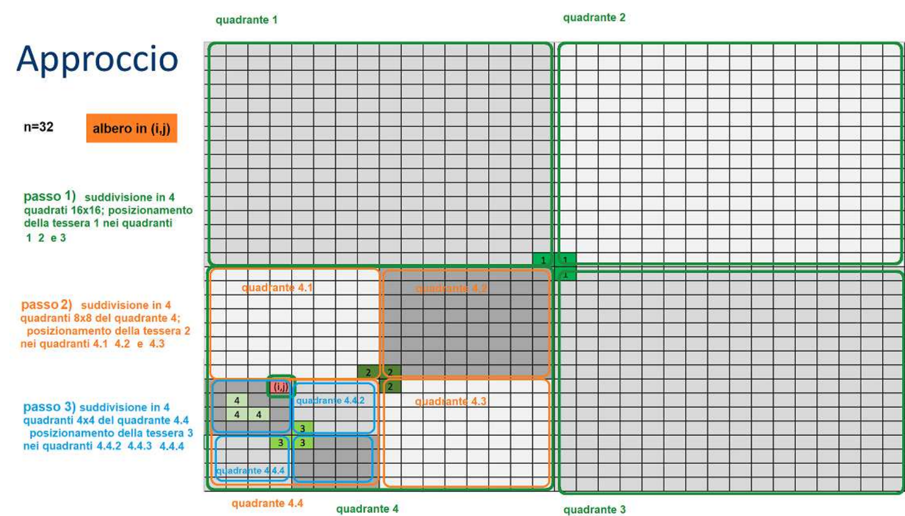


pag. 34

N=4, si divide il giardino in 4 quadrati di lato N/2=2

La prima piastrella si può posizionare in modo da coprire 1 casella in ognuno dei 3 quadranti che non contengono l'albero

pag. 35

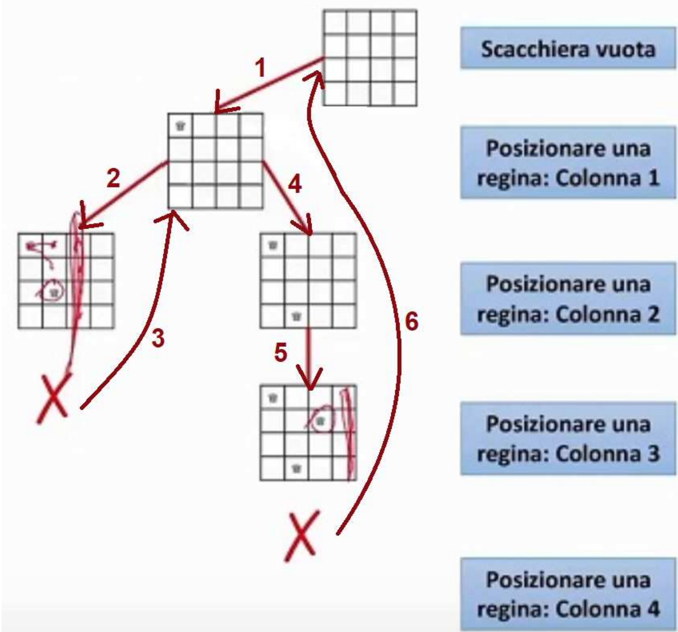


Si procede in modo analogo per N > 4 suddividendo progressivamente il quadrato in quadranti sempre più piccoli e posizionando le piastrelle come indicato per N=4

In questo modo oltre a posizionare una piastrella, si ottengono 4 quadranti ognuno con 1 cella occupata (come la prima che ha un albero da evitare).

Esempio di **Backtracking – le 4 Regine** (pag. 53)

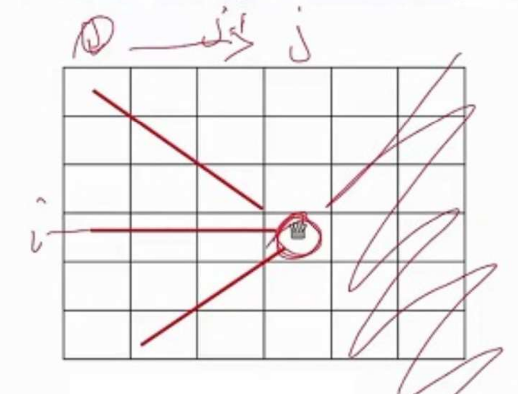
Esplorare l'albero di decisioni



colonna 1:
 1^ tentativo: **1^ regina in (1,1)** **(freccia 1)**
colonna 2:
 1^ tentativo: **2^ regina in (3,2)** **(freccia 2)**
  posizione sicura rispetto alle colonne precedenti
  ma poi nessuna posizione possibile in colonna 3
  quindi BACKTRACK al livello precedente **(frec. 3)**
 2^ tentativo: **2^ regina in (4,2)** **(freccia 4)**
colonna 3:
 1^ tentativo: **3^ regina in (2,3)** **(freccia 5)**
  posizione sicura rispetto alle colonne precedenti
  ma poi nessuna posizione possibile in colonna 4
  quindi BACKTRACK al livello precedente **(frec. 6)**
NOTA: in colonna 3 non ci sono altre posizioni sicure e in colonna 2 non ci sono altre posizioni da esplorare quindi il BACKTRACK **(6)** porta alla scacchiera vuota per realizzare
colonna 1:
 2^ tentativo: **1^ regina in (2,1)** etc etc

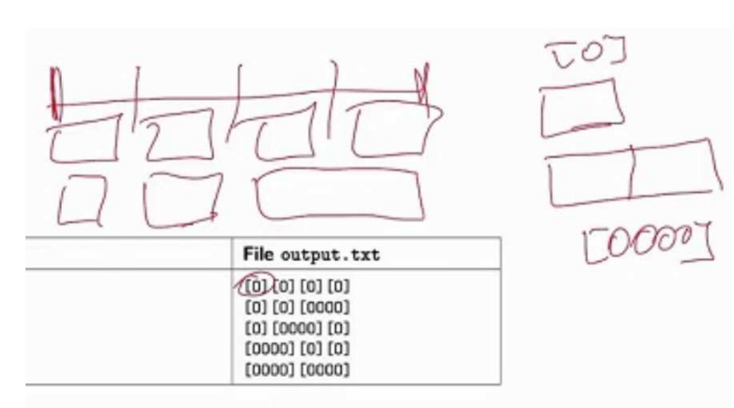
Soluzione 8-Regine

```
/* Funzione che controlla se una posizione
schacchiera[i][j] è sicura. Viene chiamata quando le
regine sono state posizionate nelle prime j-1 colonne.
Controlla se ogni regina precedente alla attuale non si
trovi sulla stessa riga o su una delle diagonali */
```



```
/* Funzione che controlla se una posizione
schacchiera[i][j] è sicura. Viene chiamata quando le
regine sono state posizionate nelle prime j-1 colonne.
Controlla se ogni regina precedente alla attuale non si
trovi sulla stessa riga o su una delle diagonali */

bool sicura(int riga, int col)
{
    int i, j;
    /* Controlla nella stessa riga a sinistra */
    for (i = 0; i < col; i++)
        if (schacchiera[riga][i])
            return false;
    /* Controlla nella diagonale in alto a sinistra */
    for (i = riga, j = col; i >= 0 && j >= 0; i--, j--)
        if (schacchiera[i][j])
            return false;
    /* Controlla nella diagonale in basso a sinistra */
    for (i = riga, j = col; j >= 0 && i < N; i++, j--)
        if (schacchiera[i][j])
            return false;
    return true;
```

Esercizio dal Portale: **Piastrellature (piastrelle)**



soluzione proposta da un alunno→

```
#include <fstream>
#include <iostream>
#include <vector>

using namespace std;
vector<string> res;
void piastr(unsigned int n, string s){
    if(n==0){
        res.push_back(s);
        return;
    }
    else if(n==1){
        res.push_back(s+"[o]");
        return;
    }
    piastr(n-1,s+"[o]");
    piastr(n-2,s+"[oo]");
}
int main(){
ifstream in("input.txt");
ofstream out("output.txt");


  unsigned int n;
    in >> n;
    res.clear();
    piastr(n,"");

    out << res.size()<<' ';
    for(auto el: res) out << el << ' ';

    out << endl;

return 0;
}
```