

Luiss
Libera Università Internazionale
degli Studi Sociali Guido Carli

Corso di preparazione per la selezione territoriale delle Olimpiadi di Informatica

Lezione 3. Ricorsione, divide et impera & backtracking

Irene Finocchi & Blerina Sinaimeri

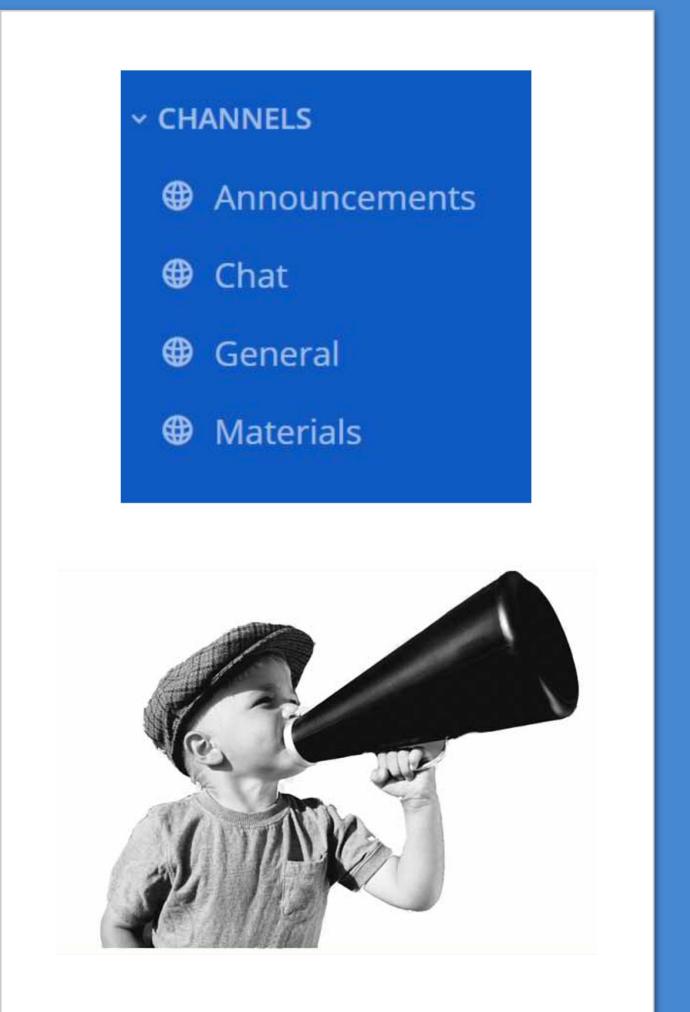
8 aprile 2021

LUISS 



Repetita iuvant (sed stufant!)

- Non avrai altra chat al di fuori di **mattermost**
- <https://coding.luiss.ml/mattermost/> (canale Chat)
 - Usa il canale Chat per le domande
 - Potete rispondere anche voi!
- Controlla i canali: spesso troverai qui le risposte alle tue domande!
- Non inquinare i canali! Contribuisci a mantenerli puliti e non postare contenuti impropri. Grazie!
- Le lezioni si svolgono sempre nell'orario 16:00-19:00





Preferibile se hai problemi di connessione, ma non permette di condividere le tue soluzioni

Lezioni

Puoi accedere alle lezioni tramite YouTube o Webex



Licenza limitata a 1000 utenti; puoi provare l'ebrezza di diventare **panelist** e condividere la tua soluzione ai problemi considerati!



Programma di oggi

1. Soluzione di alcuni esercizi della scorsa settimana
2. Ricorsione
3. Divide et impera (divide & conquer)
4. Un problema di tassellazione: [Prato](#)
5. Un problema dalle Territoriali 2008: [Mappa antica](#)
6. Backtracking
7. Il problema delle [N regine](#)
8. Un problema dalle Territoriali 2011: [Domino massimale](#)
9. Un altro problema di tassellazione: [Piastrelle](#)
10. (Un problema dalle Territoriali 2009: [Treno di container](#))

Soluzioni di alcuni degli esercizi della scorsa settimana



Rope escapes (1/4)

Scrivete in chat per esporre la vostra soluzione:

https://training.olinfo.it/#/task/ois_ropes/statement

Somme costose (2/4)

Scrivete in chat per esporre la vostra soluzione:

https://training.olinfo.it/#/task/gator_somme/statement

Smartphone (3/4)

Scrivete in chat per esporre la vostra soluzione:

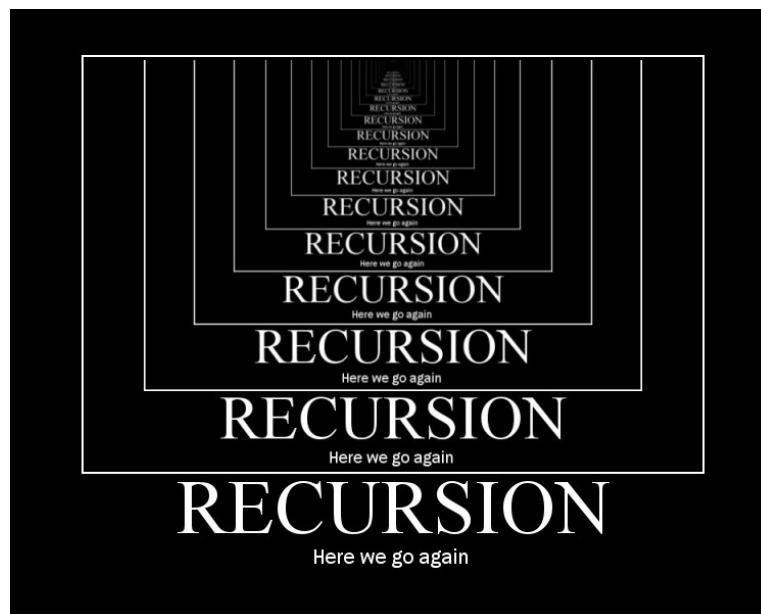
https://training.olinfo.it/#/task/ois_smartphone/statement

Walkway (4/4)

Scrivete in chat per esporre la vostra soluzione:

https://training.olinfo.it/#/task/ois_walkway/statement

Ricorsione



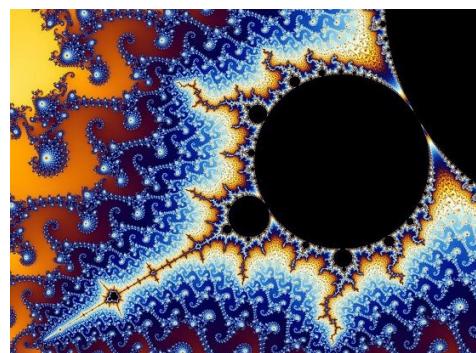
Idea di base

- Concetto semplice, ma molto potente
- Basato sulla nozione di **auto-similarità** e ripetizione

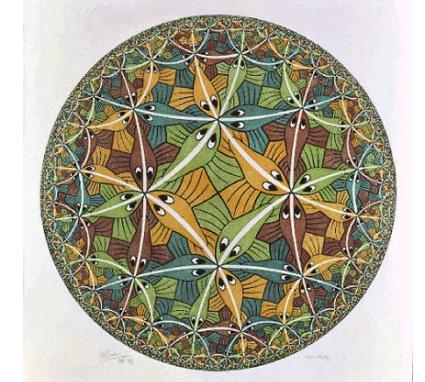
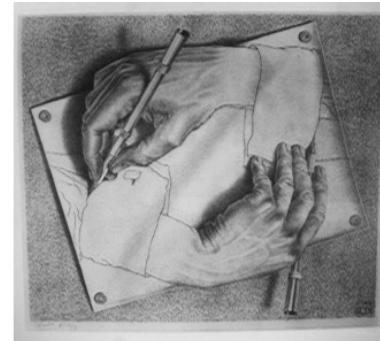
Mettete due specchi in parallelo...



I frattali sono forse la visualizzazione ricorsiva più nota (insieme di Mandelbrot)



Anche Escher usava la ricorsione con maestria!



Ricorsione e problem solving

- Per usare la ricorsione nel progettare la soluzione di un problema, *cercate di risolvere il problema... assumendo che lo abbiate già risolto!*
- Come è possibile?

Definendo la soluzione di un'istanza del problema
in termini della soluzione di istanze più piccole

- La funzione che calcola la soluzione richiamerà se stessa (una o più volte) all'interno del proprio corpo
- Spesso si ottiene codice estremamente sintetico ed elegante

Un esempio: il fattoriale

$$n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \dots \cdot (n-1) \cdot n$$

$$n! = (n-1)! \cdot n$$

$$0! = 1$$

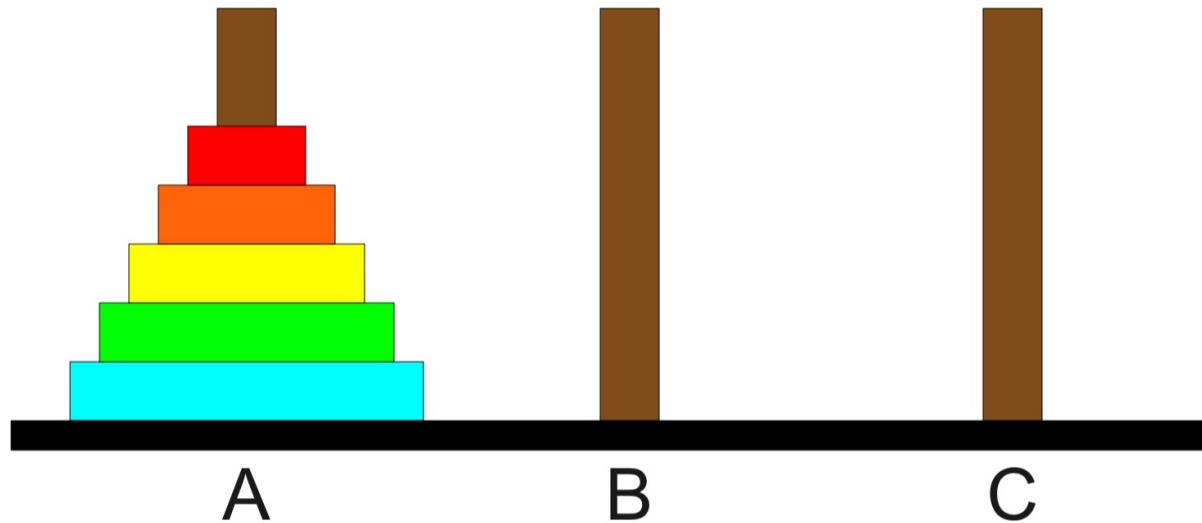
```
int fattoriale(int n) {
    if (n == 0) return 1;
    return n * fattoriale(n-1);
}
```

Gli elementi del procedimento ricorsivo

Per usare un procedimento ricorsivo:

1. Un problema deve essere scomponibile in sottoproblemi **dello stesso tipo**
2. Occorre trovare delle **relazioni** che legano un problema a sottoproblemi simili
3. È necessario conoscere la soluzione di un caso particolare del problema (**caso base** = condizione di terminazione)

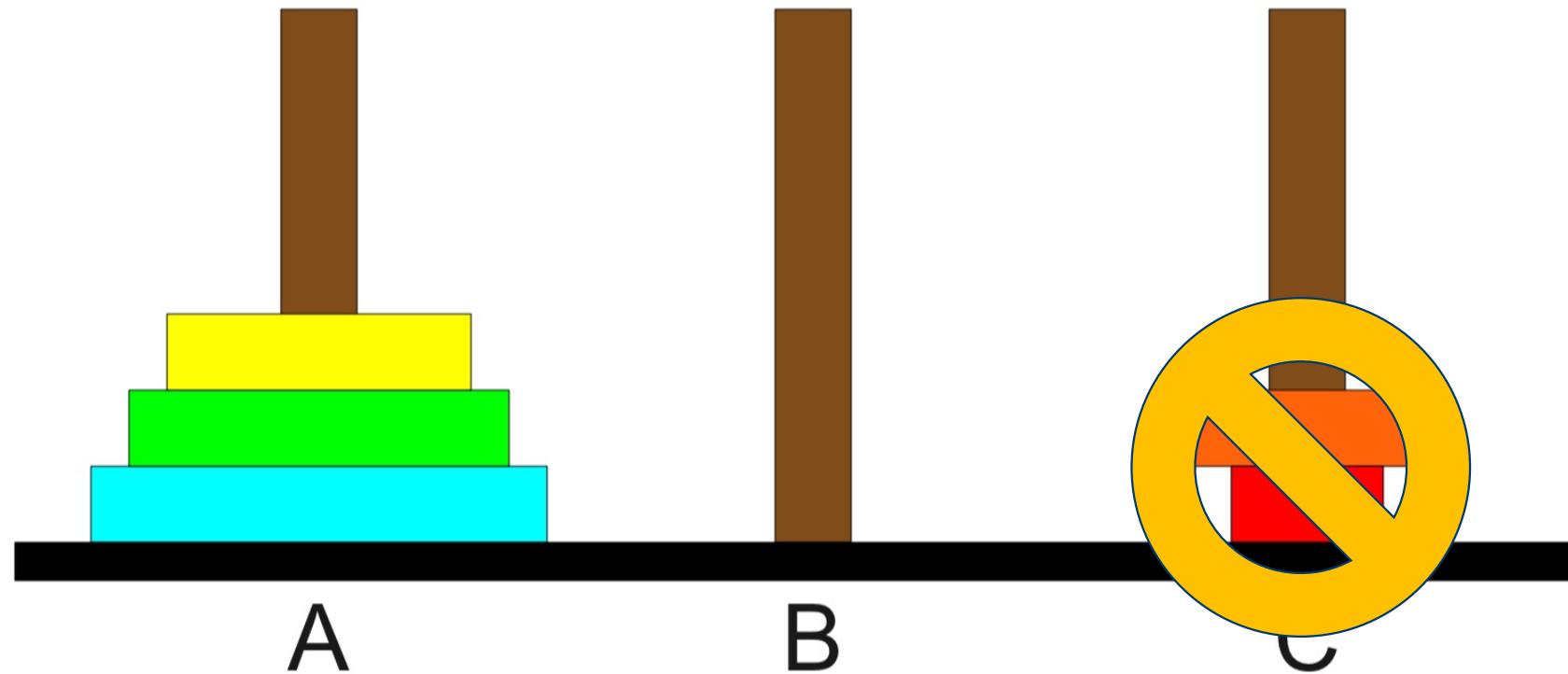
Un altro esempio: le torri di Hanoi



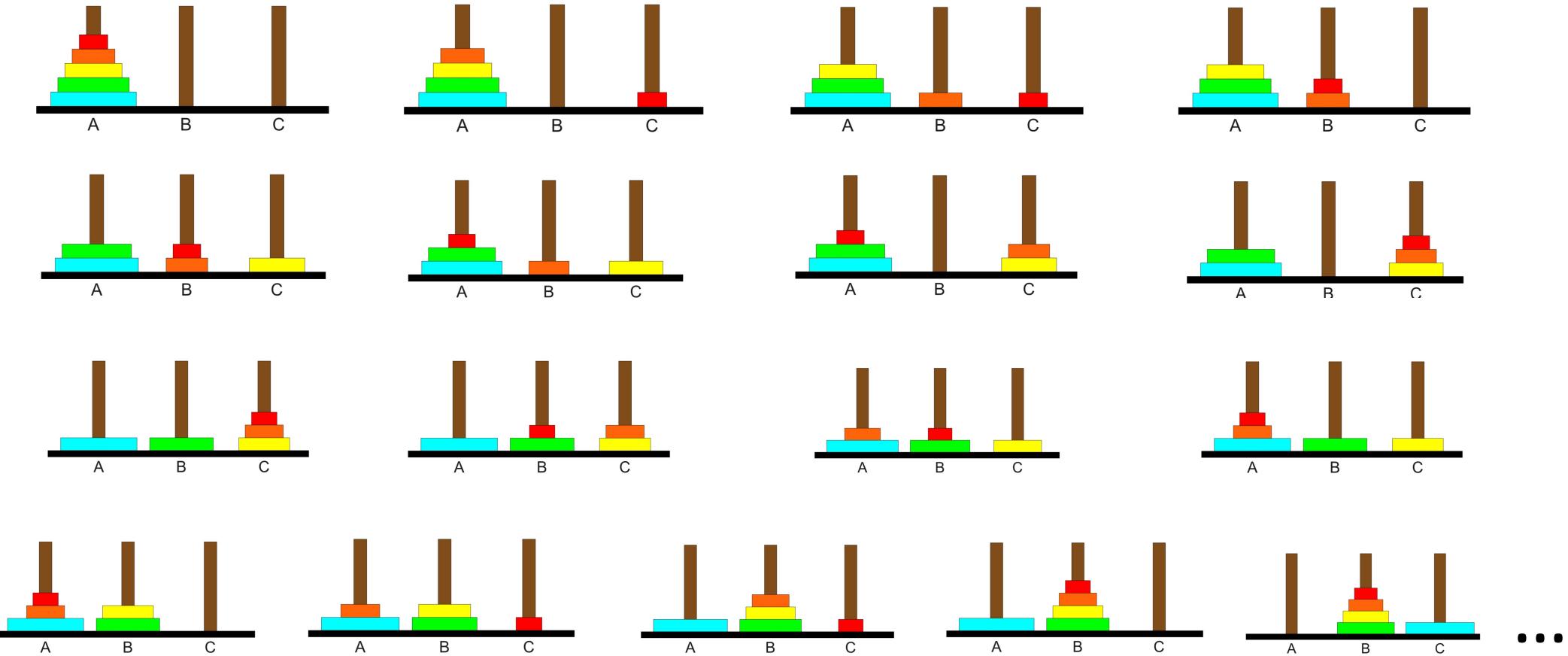
Risolvetelo interattivamente!

<http://dm.compsciclub.ru/app/quiz-hanoi-towers>

Configurazioni proibite



Soluzione step-by-step



Un'implementazione elegante (e sintetica!)

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

Quando usare la ricorsione?

- Problemi di tipo “**divide et impera**” (next): la soluzione si ottiene suddividendo il problema in due o più sottoproblemi simili, che vengono risolti separatamente ricombinando poi le soluzioni per ottenere la soluzione al problema di partenza
- Soluzioni “**a forza bruta**”: per generare tutti i casi possibili tramite **backtracking** (ad esempio, tutti i sottoinsiemi, tutte le permutazioni...) e scegliere il caso ottimo per il problema in questione
- Problemi di **programmazione dinamica** (lezione del 22 aprile)

Problemi con la ricorsione

1. Ricorsione infinita

- Cosa accade se manca il caso base o se la funzione viene richiamata con parametri non corretti?
- Ad esempio, cosa restituisce fattoriale(-1)?

2. Non è immediato calcolare il tempo di esecuzione di funzioni ricorsive

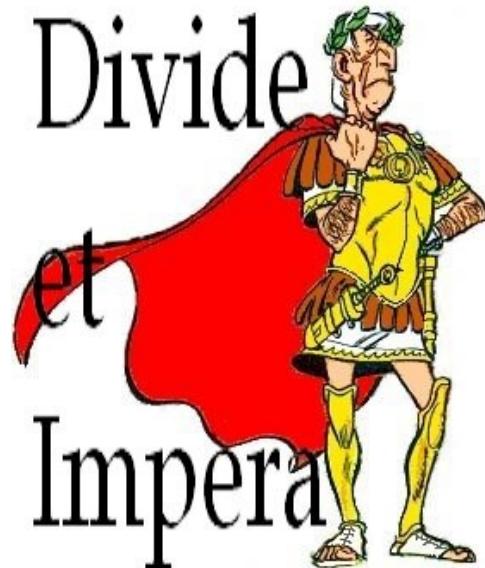
- Si richiedono tecniche matematiche sofisticate (equazioni di ricorrenza)

3. Siate cauti con le chiamate ricorsive: fare più di una chiamata potrebbe far aumentare di molto i tempi di esecuzione (anche in modo esponenziale, e.g., 2^n). Un esempio? Fibonacci!

- Valutate sempre i vincoli del problema (dimensione delle istanze di input)

Divide et impera

In inglese, divide & conquer



Approccio

Struttura ricorsiva:

1. **Dividere** il problema in sottoproblemi simili all'originale, ma più piccoli
2. **Conquistare** i sottoproblemi risolvendoli ricorsivamente. Se sono abbastanza piccoli, risolverli in modo diretto
3. **Combinare** le soluzioni dei sottoproblemi per ottenere la soluzione del problema originale

Un esempio: il mergesort

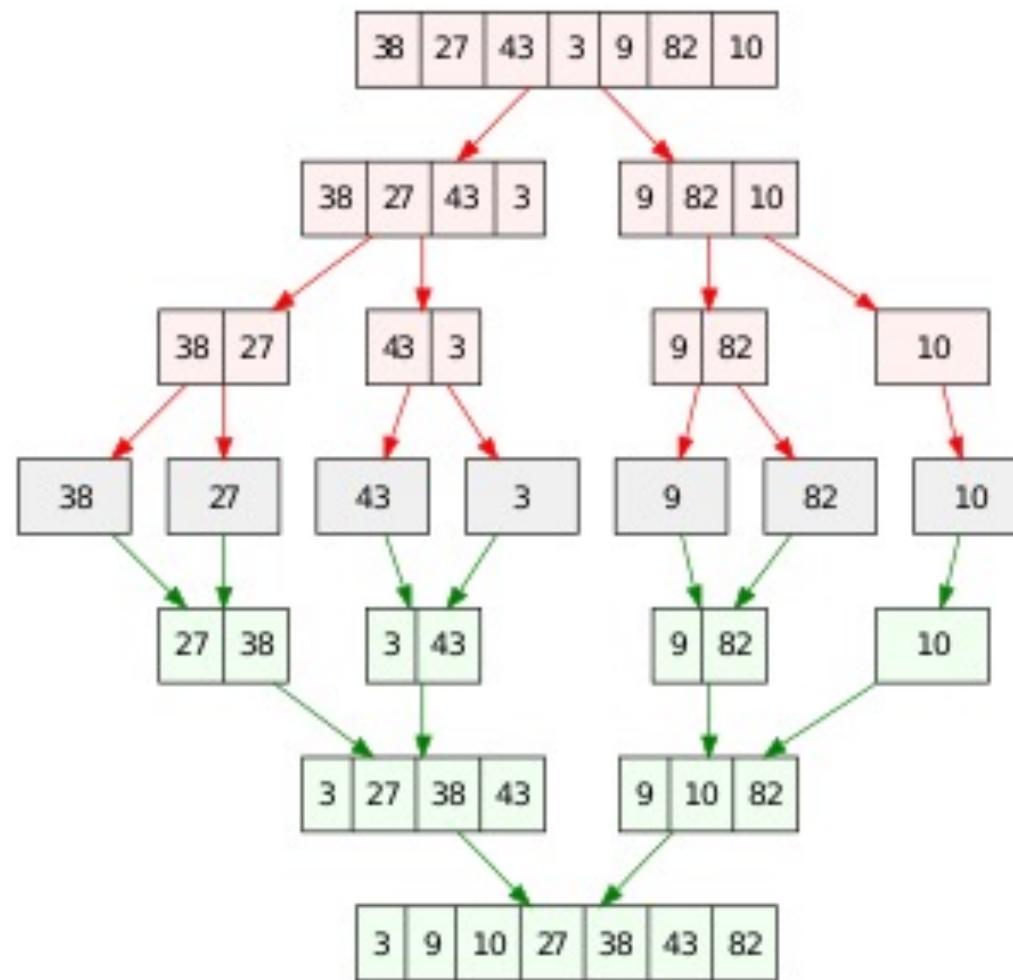
Il problema: ordinare una sequenza di n elementi

L'algoritmo:

1. **Dividere** la sequenza lunga n in due sottosequenze lunghe $\lceil n/2 \rceil$ e $\lfloor n/2 \rfloor$
2. **Ordinare** le due sottosequenze ricorsivamente usando mergesort (a meno che non abbiano un solo elemento)
3. **Combinare** le soluzioni **fondendo** le due sottosequenze ordinate in una sequenza ordinata lunga n

Nota: la fusione di due sequenze ordinate è un problema facile che può essere risolto in tempo lineare (esercizio!)

Graficamente



Implementazione

```
/* 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
    }
}
```

Un problema da un contest Luiss dello scorso anno: Prato

Non disponibile su training/olinfo



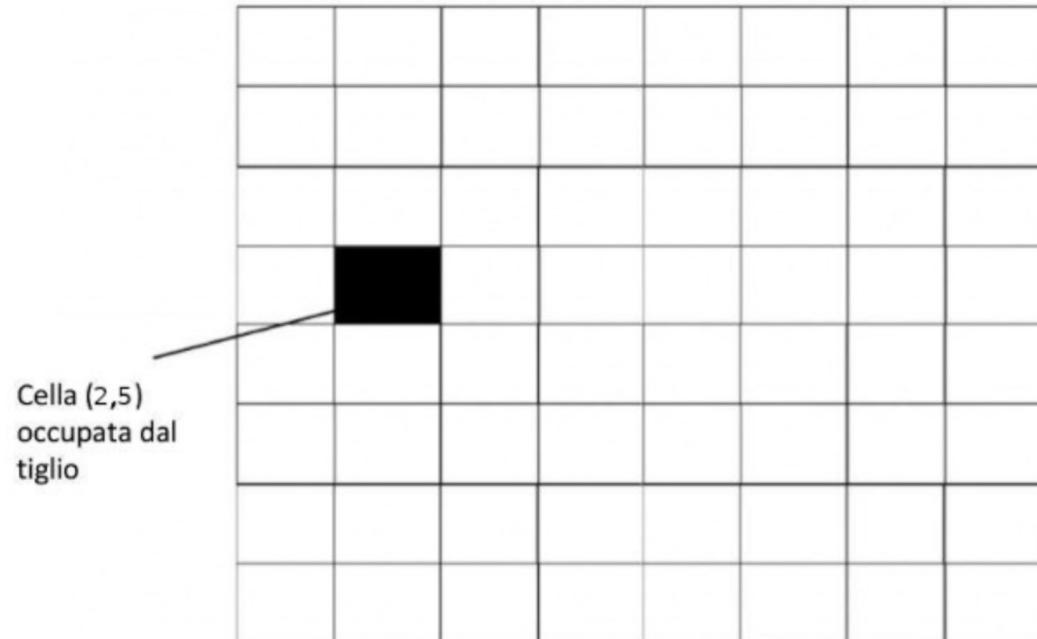
Descrizione del problema

Dopo il lungo periodo di lockdown, Irene è finalmente tornata nella sua casa di campagna nel viterbese. Purtroppo la prolungata mancanza di lavori di giardinaggio ha fatto sì che il giardino quest'anno sia in una condizione pessima e che il pratino, prima verde e folto, non esista quasi più. Poiché ripiantarlo in questa stagione è ormai impossibile, ha deciso di utilizzare un **manto erboso preconfezionato**.

Il giardino è un **quadrato lungo N metri per lato**, dove N è una potenza di 2. Idealmente, si può pensare al giardino come suddiviso in **N^2 punti** che occupano **ciascuno 1 m^2** . Un solo punto, in **posizione (i,j)** , è occupato dal tronco di un grande tiglio e da alcuni cespugli.

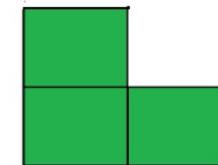
Il giardino

La situazione è illustrata nella seguente figura per N=8:



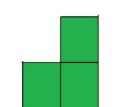
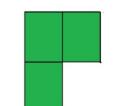
Le piastrelle erbose e le rotazioni

Il manto erboso da utilizzare consiste di rivestimenti, simili a "piastrelle erbose", a forma di L, in cui ciascun lato è lungo 2m



Le piastrelle possono essere ruotate (con rotazioni di 90 gradi) in quattro possibili modi, a seconda della posizione delle due estremità della lettera L: rispetto alla posizione centrale della piastrella, ciascuna delle due estremità può infatti trovarsi in Alto o in Basso, a Sinistra o a Destra.

- La posizione L è indicata dalla configurazione AD (Alto e Destra).
- Ruotandola in senso orario di 90 gradi si ottiene la configurazione BD (Basso e Destra).
- Proseguendo con una ulteriore rotazione di 90 gradi si ottiene la configurazione BS (Basso e Sinistra).
- Infine, ruotando ancora una volta, si ottiene la configurazione AS (Alto e Sinistra).



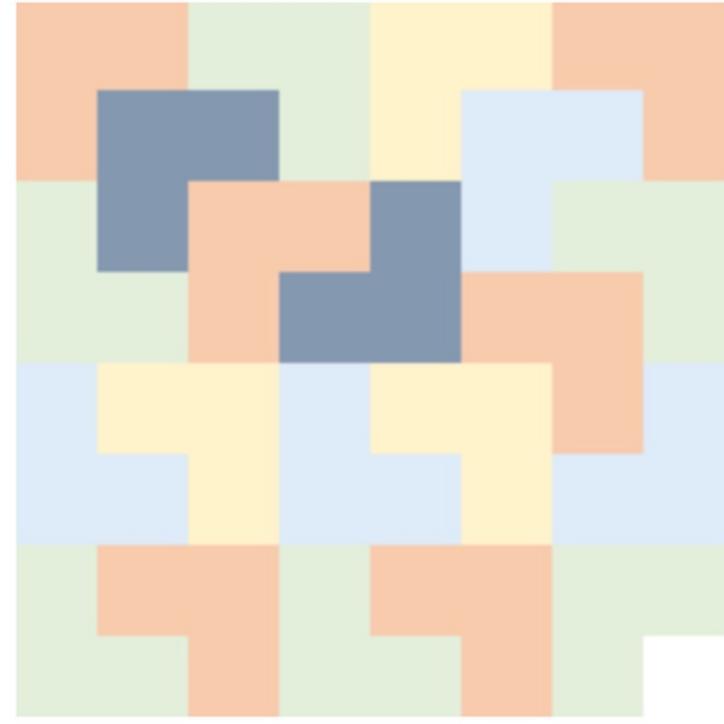
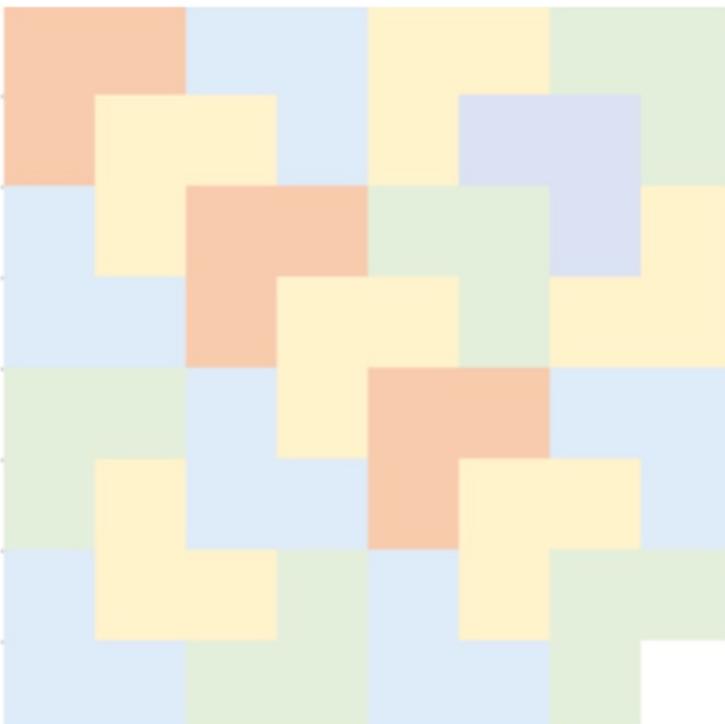
Obiettivo: posiziona le piastrelle erbose

Irene, per contenere i costi, ha acquistato soltanto il numero indispensabile di piastrelle erbose, pari a $(N^2-1)/3$.

Aiutala a posizionare tutte le piastrelle in modo da ricoprire con il manto erboso l'intera area del giardino, ad eccezione del punto (i,j) .

Devi quindi scoprire **in quale posizione va installata ciascuna piastrella erbosa, e come deve essere ruotata**. La posizione della piastrella sarà data dalle coordinate della sua posizione centrale, e la rotazione dalle configurazioni AD, BD, BS o AS.

Soluzione unica? Non è detto!



Input e output

Input

- 3 righe (per ciascun test): la prima contiene un numero intero **N**, potenza di 2, ovvero il lato del giardino. Le due righe successive contengono le coordinate **i** e **j** occupate dal tiglio.

Output

- $(N^2-1)/3$ righe, una per ciascuna piastrella posizionata.
- Per ogni piastrella, deve contenere **le due coordinate della sua posizione centrale (x,y) e la rotazione AD, BD, BS o AS** utilizzata, separate da spazi.
- (Le piastrelle devono essere **restituite in ordine in base alla posizione centrale**: una piastrella in posizione (x,y) precede una piastrella in posizione (a,b) se $x < a$ oppure se $x = a$ e $y < b$)

Esempio e assunzioni lecite

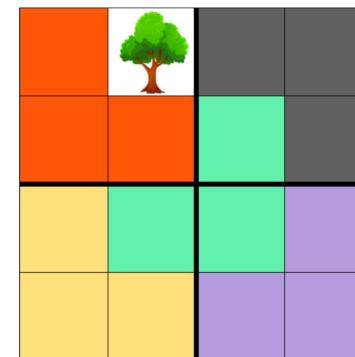
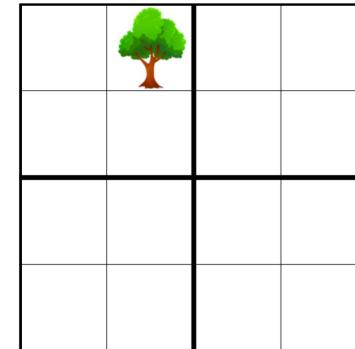
Assunzioni

- $N \geq 2$ è una potenza di 2
- $1 \leq i, j \leq N$
- Esiste almeno una soluzione

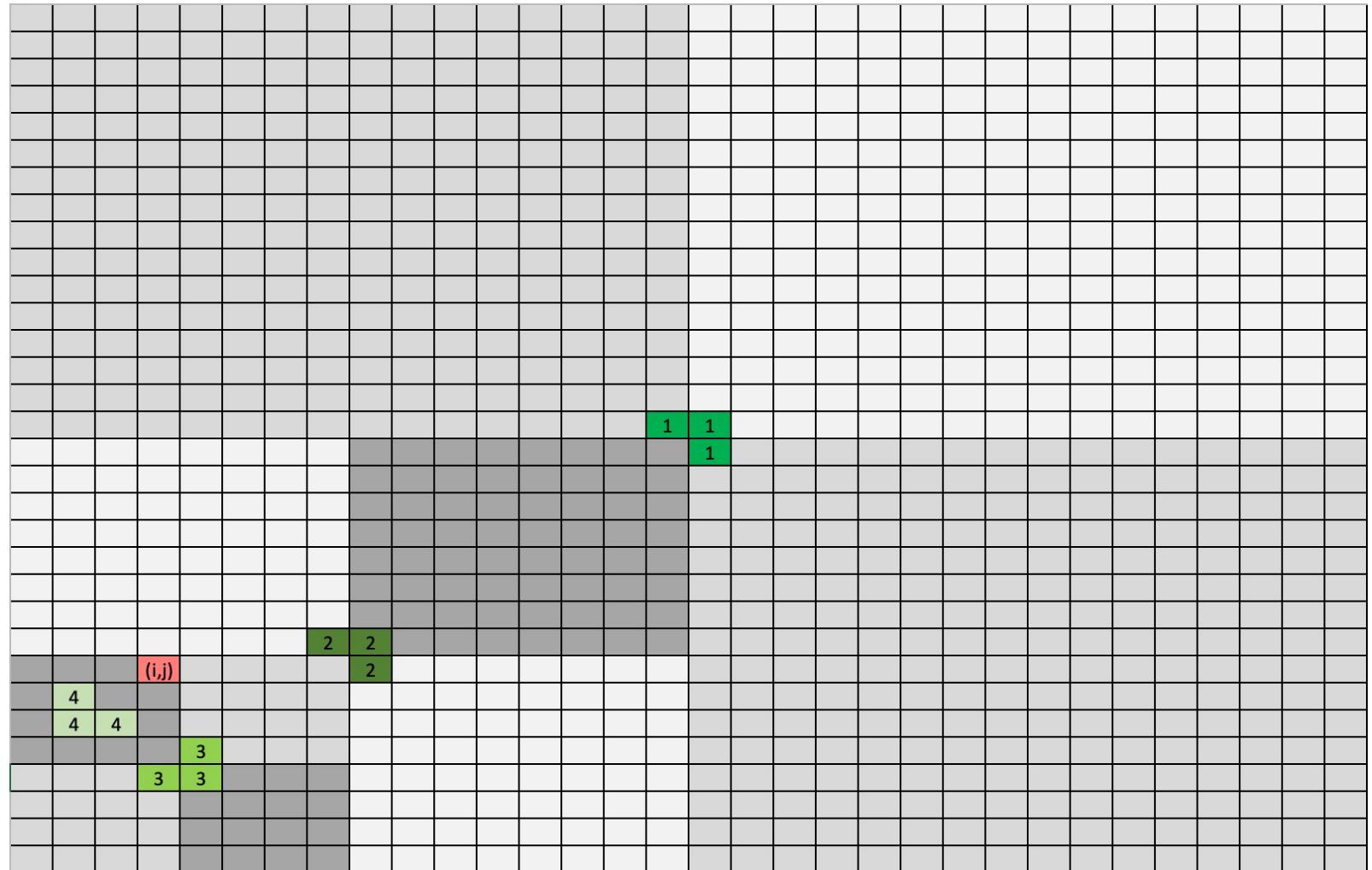
Esempio di input/output	Input:	Output:
	4	1 1 BD
	4	1 4 BS
	2	2 3 BS
		3 1 BD
		4 4 AS

Approccio

1. Passo base: banale per $N = 2$
2. Se $N > 2$, dividi il quadrato $N \times N$ punti in 4 quadrati $N/2 \times N/2$
3. Posiziona una tessera al centro del quadrato $N \times N$, ruotandola in modo che ciascuno dei 3 quadratini nella tessera cada in un quadrato che non contenga la posizione (i, j) .
4. Esegui 4 chiamate ricorsive sui 4 quadrati del tabellone: ognuno di essi ha ora la propria posizione (i, j) occupata (dal tiglio o dalla tessera "centrale")



Approccio



Un problema dalle Territoriali 2008: Mappa antica

<https://training.olinfo.it/#/task/mappa/statement>



Mappa antica (mappa)

Descrizione del problema

Topolino è in missione per accompagnare una spedizione archeologica che segue un'antica mappa acquisita di recente dal museo di Topoinia. Raggiunta la località dove dovrebbe trovarsi un prezioso e raro reperto archeologico, Topolino si imbatte in un labirinto che ha la forma di una gigantesca scacchiera quadrata di NxN lastroni di marmo.

Nella mappa, sia le righe che le colonne del labirinto sono numerate da 1 a N. Il lastrone che si trova nella posizione corrispondente alla riga r e alla colonna c viene identificato mediante la coppia di interi (r, c) . I lastroni segnalati da una crocetta '+' sulla mappa contengono un trabocchetto mortale e sono quindi da evitare, mentre i rimanenti sono innocui e segnalati da un asterisco '*'.

Topolino deve partire dal lastrone in posizione $(1, 1)$ e raggiungere il lastrone in posizione (N, N) , entrambi innocui. Può passare da un lastrone a un altro soltanto se questi condividono un lato o uno spigolo (quindi può procedere in direzione orizzontale, verticale o diagonale ma non saltare) e, ovviamente, questi lastroni devono essere innocui.

Tuttavia, le insidie non sono finite qui: per poter attraversare incolume il labirinto, Topolino deve calpestare il minor numero possibile di lastroni innocui (e ovviamente nessun lastrone con trabocchetto). Aiutate Topolino a calcolare tale numero minimo.

File di input

Il programma deve leggere da un file di nome `input.txt`. La prima riga contiene un intero positivo che rappresenta la dimensione N di un lato del labirinto a scacchiera. Le successive N righe rappresentano il labirinto a scacchiera: la r -esima di tali righe contiene una sequenza di N caratteri '+' oppure '*', dove '+' indica un lastrone con trabocchetto mentre '*' indica un lastrone sicuro. Tale riga rappresenta quindi i lastroni che si trovano sulla r -esima riga della scacchiera: di conseguenza, il c -esimo carattere corrisponde al lastrone in posizione (r, c) . I caratteri NON sono separati da degli spazi.

File di output

Il programma deve scrivere in un file di nome `output.txt`. Deve venire stampato il minimo numero di lastroni innocui (ossia indicati con '*') che Topolino deve attraversare a partire dal lastrone in posizione $(1, 1)$ per arrivare incolume al lastrone in posizione (N, N) . Notare che i lastroni $(1, 1)$ e (N, N) vanno inclusi nel conteggio dei lastroni attraversati.

Assunzioni

- $0 < N \leq 100$
- $0 < r, c \leq N$

Subtask

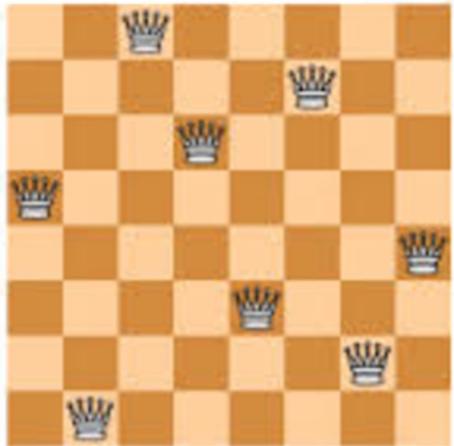
- **Subtask 1 [5 punti]:** caso di esempio.
- **Subtask 2 [30 punti]:** $N \leq 20$.
- **Subtask 3 [30 punti]:** $N \leq 75$.
- **Subtask 4 [35 punti]:** nessuna limitazione specifica.

Esempio di input/output

File input.txt	File output.txt
5 ***** ***** ***** ***** ***** *****	5

Backtracking

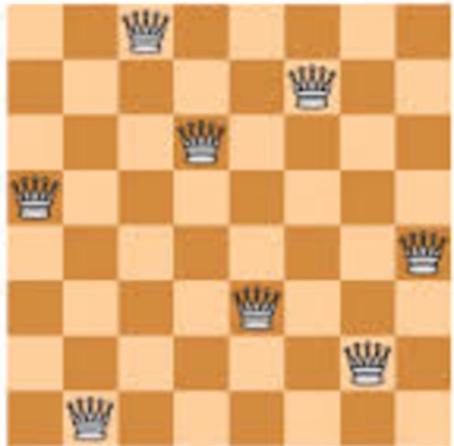
- Backtracking è una tecnica per risolvere i problemi in maniera in modo incrementale **senza esplorare tutte le possibilità.**



Le N-Regine: Posizionare N regine su una scacchiera NxN tali che nessuna di esse possa catturarne un'altra, usando i movimenti standard della regina.

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

Sudoku: di riempire le caselle bianche con numeri da 1 a 9 in modo tale che in ogni riga, in ogni colonna e in ogni regione siano presenti tutte le cifre da 1 a 9, quindi senza ripetizioni.



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

Ci sono 8^2 caselle 8 regine e quindi le possibilità sono :

$$\binom{8^2}{8} = 4,426,165,368$$

Ci sono 9^n possibili modi per riempire n posizioni vuote. Se sono già riempite 20 posizioni, le possibilità per le 61 rimanenti sono =
 16,173,092,699,229,880,893,718,618,465,586,445,357,
 583,280,647,840,659,957,609

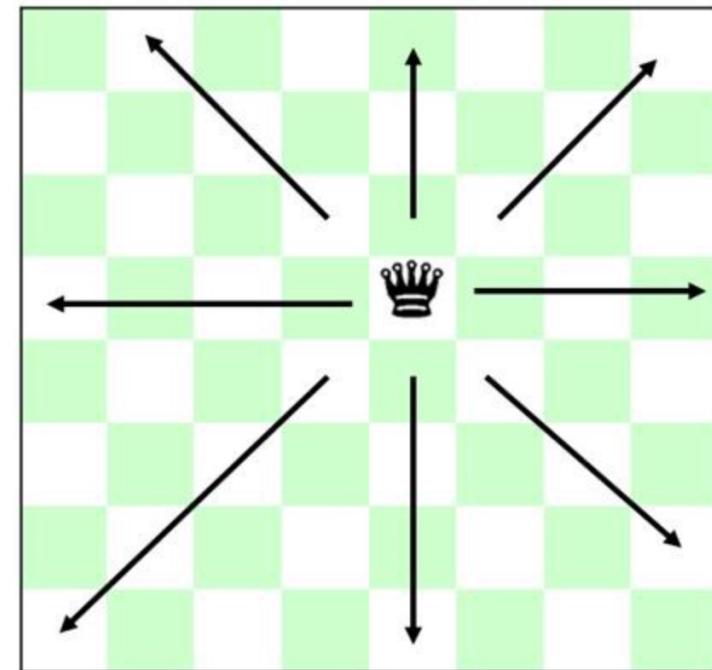
Idea: Cerchiamo di analizzare il problema per trovare dei vincoli

Questi vincoli ridurranno lo spazio delle possibilità e velocizzeranno l'algoritmo.



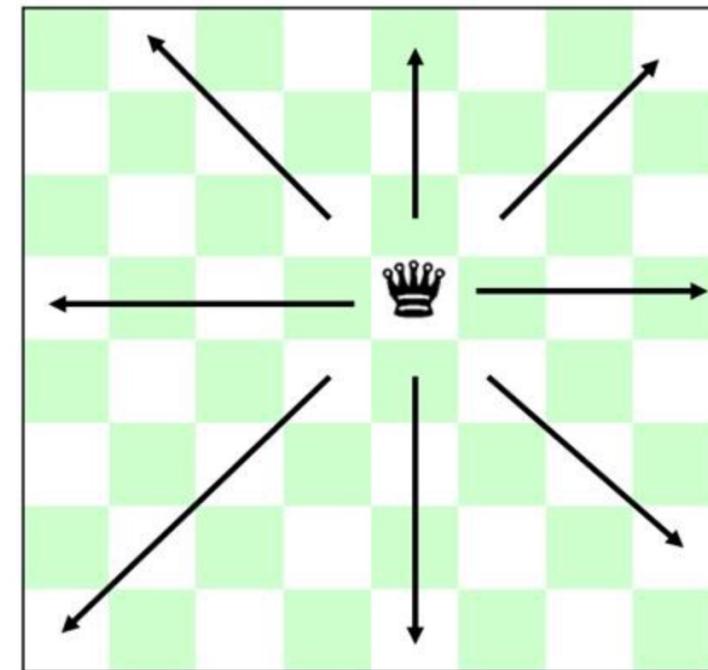
Ridurre lo spazio della ricerca

- Nell'esempio delle regine: Non posso avere 2 regine nella stessa colonna.
- Quante possibilità ho?
 - $8 * 8 * 8 * 8 * 8 * 8 * 8 * 8 = 16,777,216$
 - Una semplice osservazione ci ha permesso di ridurre lo spazio di ricerca di due ordini di grandezza.



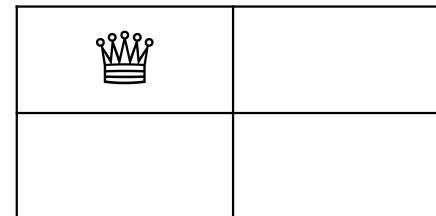
Ridurre lo spazio della ricerca

- Nell'esempio delle regine: Non posso avere 2 regine nella stessa colonna.
 - Decisione 1: Posiziona una regina nella prima colonna
 - Decisione 2: Posiziona una regina nella seconda colonna
 -
 - Decisione n: Posiziona una regina nella n-eseima colonna.



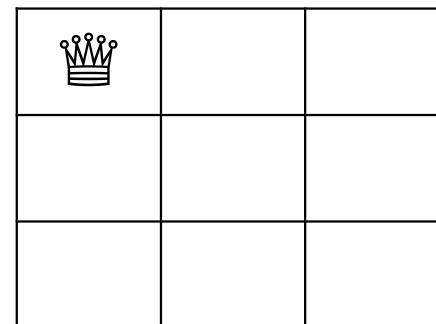
Vediamo dei casi più semplici

- Esiste una soluzione per il scacchiera : 2×2 ?



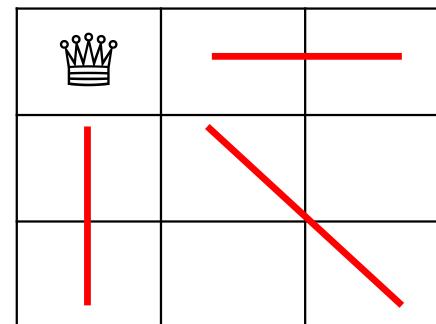
Vediamo dei casi più semplici

- Esiste una soluzione per il scacchiera : 3×3 ?



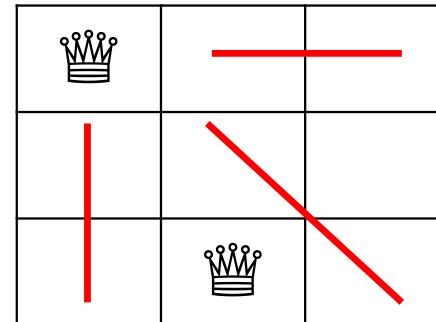
Vediamo dei casi più semplici

- Esiste una soluzione per il scacchiera : 3×3 ?



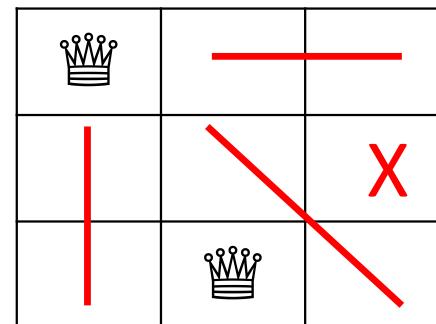
Vediamo dei casi più semplici

- Esiste una soluzione per il scacchiera : 3×3 ?



Vediamo dei casi più semplici

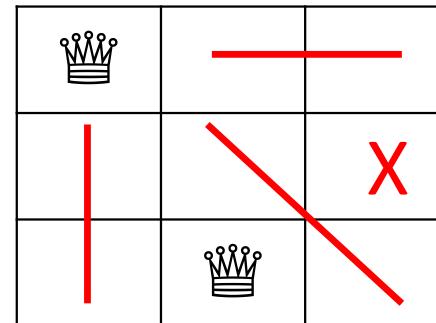
- Esiste una soluzione per il scacchiera : 3×3 ?



Ho finito?

Vediamo dei casi più semplici

- Esiste una soluzione per il scacchiera : 3×3 ?

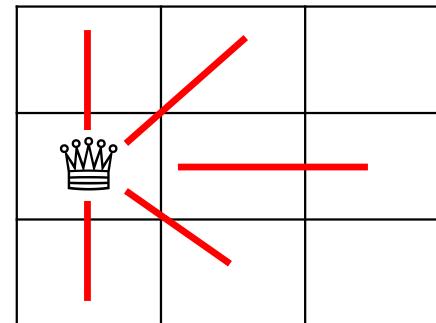


Ho finito?

No! Devo tornare indietro e riconsiderare le scelte fatte. Nell'ultima decisione presa devo decidere diversamente.

Vediamo dei casi più semplici

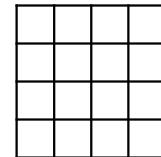
- Esiste una soluzione per il scacchiera : 3×3 ?



Ho finito?

No! Devo tornare indietro e riconsiderare le scelte fatte. Nell'ultima decisione presa devo decidere diversamente.

Esplorare l'albero di decisioni



Scacchiera vuota

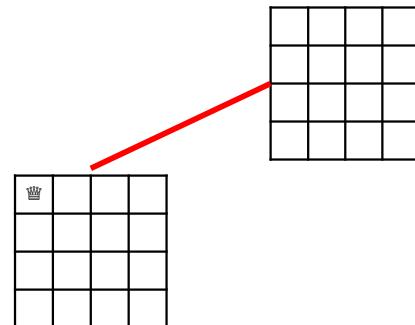
Posizionare una regina: Colonna 1

Posizionare una regina: Colonna 2

Posizionare una regina: Colonna 3

Posizionare una regina: Colonna 4

Esplorare l'albero di decisioni



Scacchiera vuota

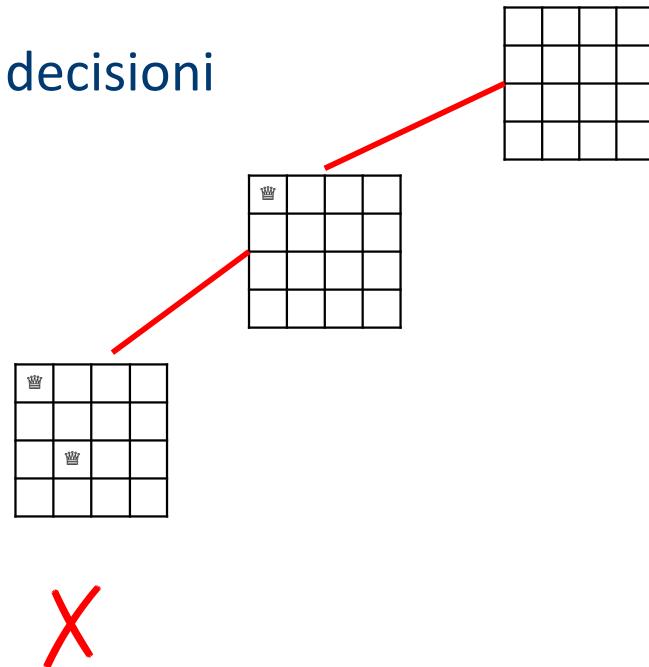
Posizionare una regina: Colonna 1

Posizionare una regina: Colonna 2

Posizionare una regina: Colonna 3

Posizionare una regina: Colonna 4

Esplorare l'albero di decisioni



Scacchiera vuota

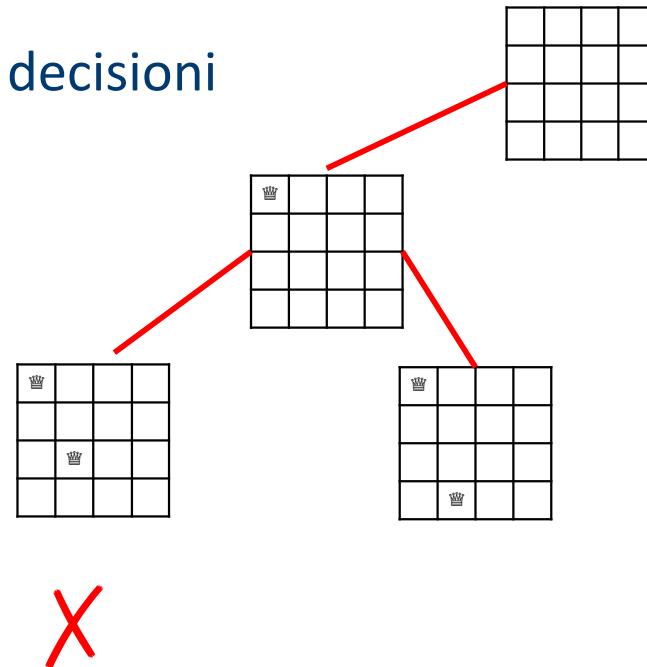
Posizionare una regina: Colonna 1

Posizionare una regina: Colonna 2

Posizionare una regina: Colonna 3

Posizionare una regina: Colonna 4

Esplorare l'albero di decisioni



Scacchiera vuota

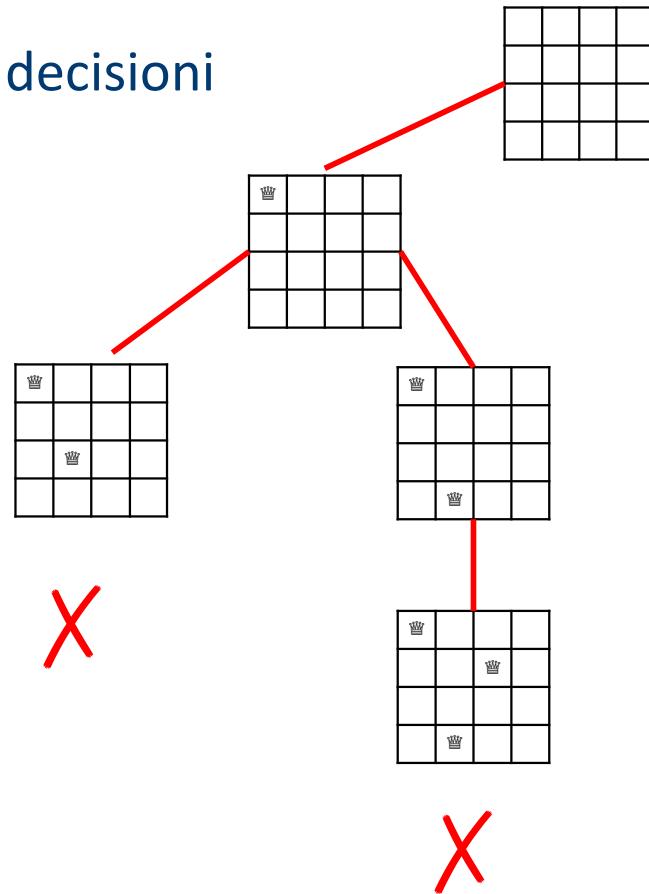
Posizionare una regina: Colonna 1

Posizionare una regina: Colonna 2

Posizionare una regina: Colonna 3

Posizionare una regina: Colonna 4

Esplorare l'albero di decisioni



Scacchiera vuota

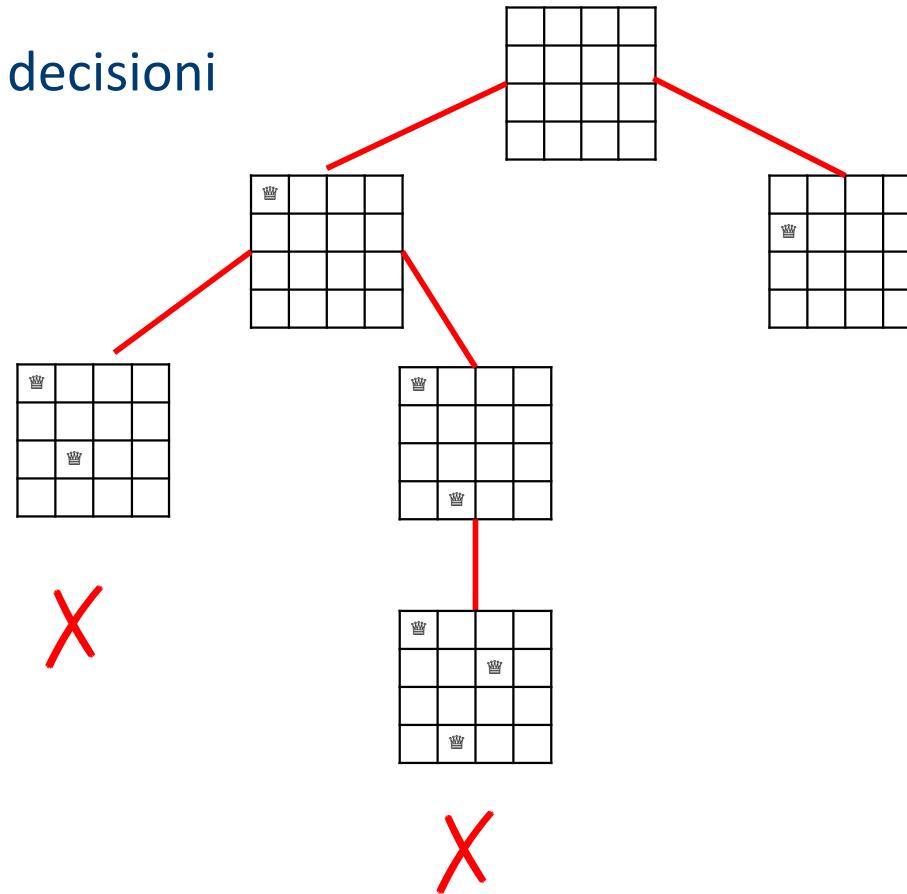
Posizionare una regina: Colonna 1

Posizionare una regina: Colonna 2

Posizionare una regina: Colonna 3

Posizionare una regina: Colonna 4

Esplorare l'albero di decisioni



Scacchiera vuota

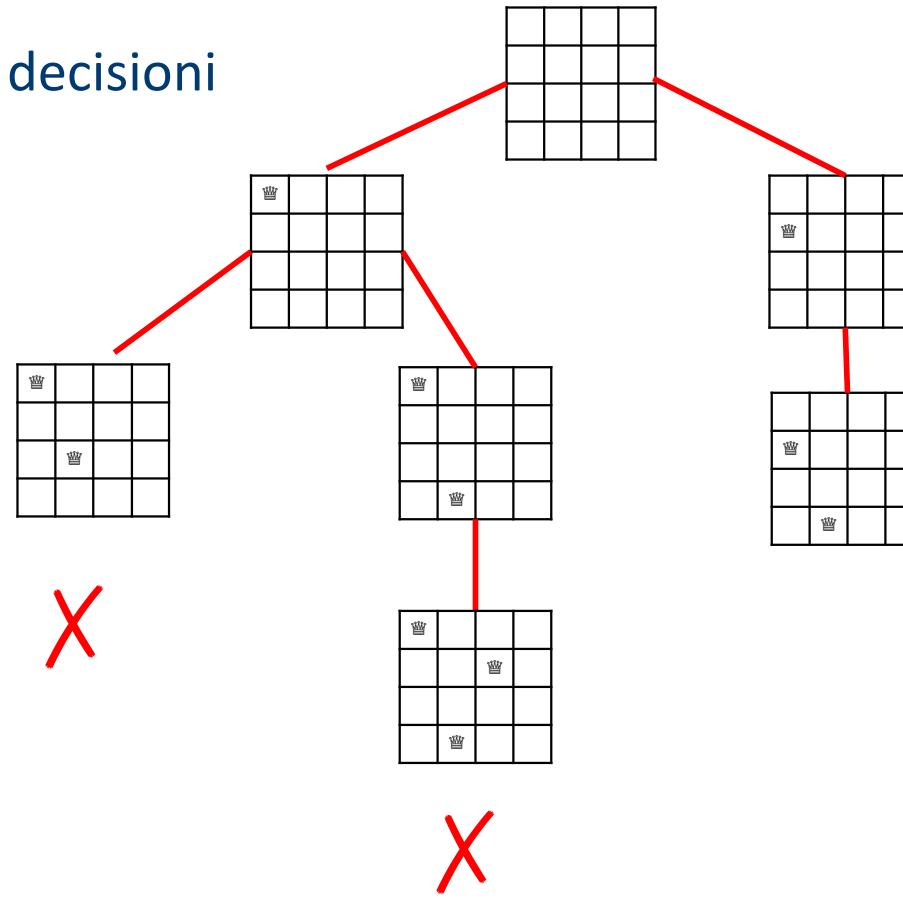
Posizionare una regina: Colonna 1

Posizionare una regina: Colonna 2

Posizionare una regina: Colonna 3

Posizionare una regina: Colonna 4

Esplorare l'albero di decisioni



Scacchiera vuota

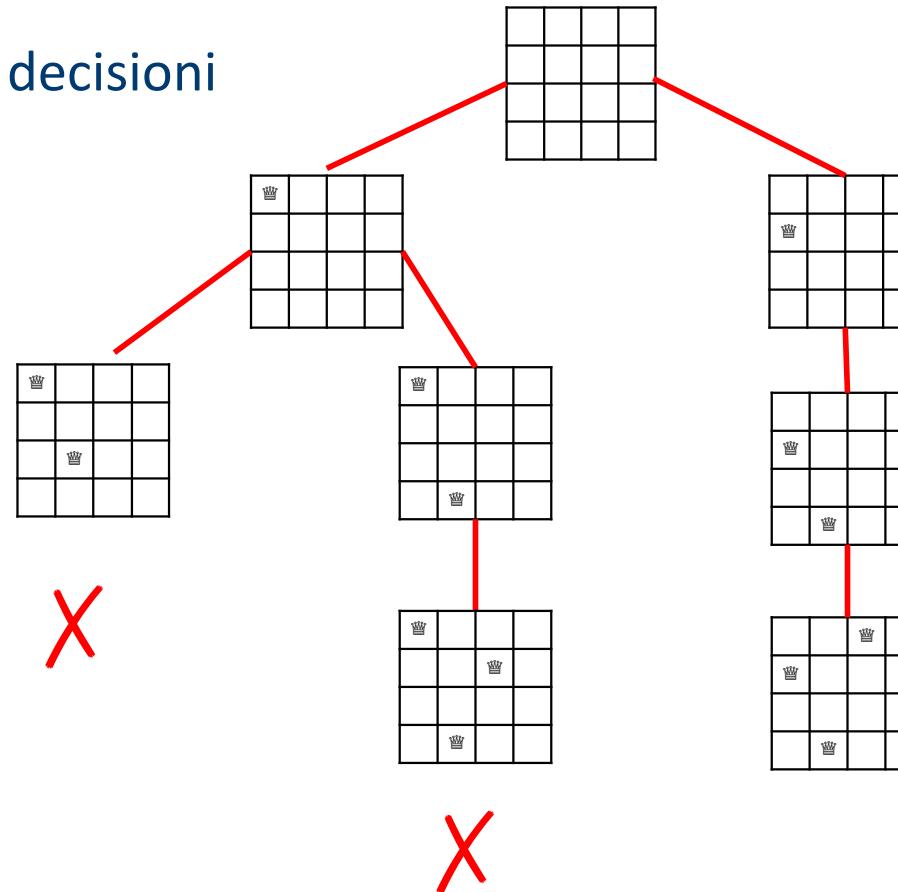
Posizionare una regina: Colonna 1

Posizionare una regina: Colonna 2

Posizionare una regina: Colonna 3

Posizionare una regina: Colonna 4

Esplorare l'albero di decisioni



Scacchiera vuota

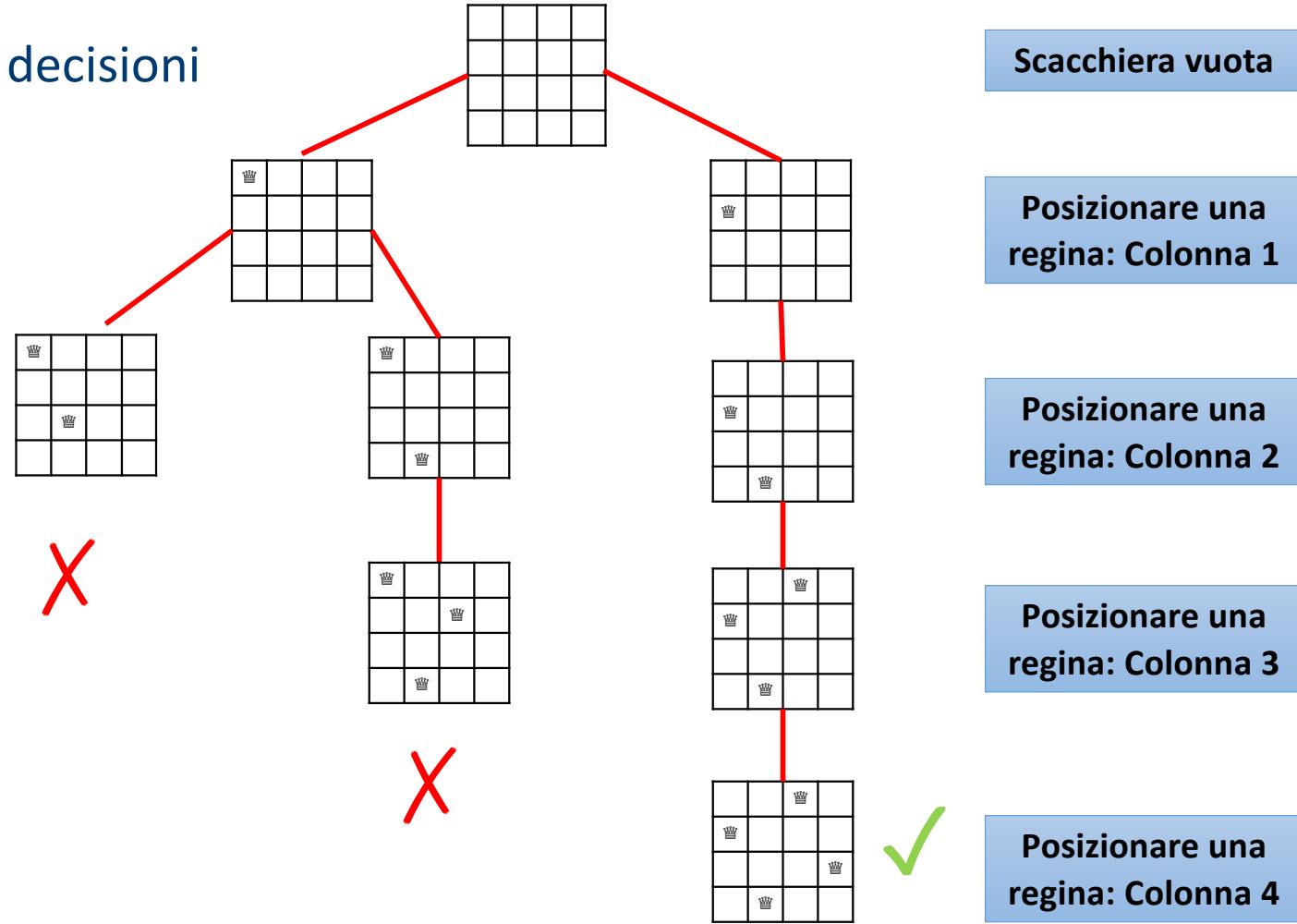
Posizionare una regina: Colonna 1

Posizionare una regina: Colonna 2

Posizionare una regina: Colonna 3

Posizionare una regina: Colonna 4

Esplorare l'albero di decisioni



Idea dell'algoritmo

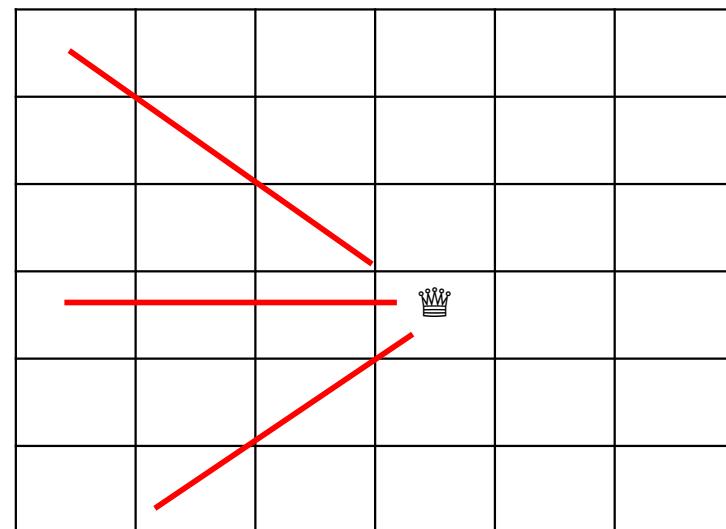
- Comincia dalla prima colonna e considera le colonne una alla volta.
- Finche non trovi una soluzione prova tutte le righe della colonna corrente. Per ogni riga fai la seguente :
 1. Se la regina può essere posizionata in maniera sicura su questa riga allora segna la posizione come parte della soluzione e controlla ricorsivamente se c'è una soluzione per le rimanenti colonne.
 2. Se mettendo la regina in quella posizione trovi la soluzione ritorna soluzione!
 3. Se mettendo la regina in quella posizione non porta ad una soluzione allora « backtrack» al passo (1) e tenta le altre righe.
- Se tutte le righe sono state tentate e niente ha funzionato, ritorna per fare il backtracking.

Soluzione 8-Regine

```
#include <stdbool.h>
#include <stdio.h>
#include <iostream>
using namespace std;
#define N 8
int schacchiera[N][N];// rappresentativo della
schacchiera 8 per 8

/* Stampa la soluzione */
void StampaSoluzione()
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            cout << schacchiera[i][j] << " ";
        cout << endl;
    }
}
```

/* 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 */



Soluzione 8-Regine

```
#include <stdbool.h>
#include <stdio.h>
#include <iostream>
using namespace std;
#define N 8
int schacchiera[N][N];// rappresentativo della
schacchiera 8 per 8

/* Stampa la soluzione */
void StampaSoluzione()
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            cout << schacchiera[i][j] << " ";
        cout<< endl;
    }
}
```

```
/* 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;
```

```

/* Una funzione ricorsiva per risolvere le 8-Reigne */
bool tenta(int col)
{
    /* caso base: se tutte le regine sono state posizionate ritorna true */
    if (col >= N)
        return true;

    /* Considera la colonna col e cerca di posizionare la regina
       tentando le righe una dopo l'altra */
    for (int i = 0; i < N; i++) {
        /* Controlla se la regina si puo posizionare nella posizione
           schacchiera[i][col] */
        if (sicura(i, col)) {
            /* Posiziona la regina nella schacchiera[i][col] */
            schacchiera[i][col] = 1;
            /* chiamata ricorsiva per piazzare le altre regine */
            if (tenta((col + 1)))
                return true;
            /*Se posizionando la regina in schacchiera[i][col] non porta ad
               una soluzione allora rimuovi la regina da schacchiera[i][col] */
            schacchiera[i][col] = 0; // BACKTRACK
        }
    }

    /* Se la regina non puo essere posizionata in nessuna riga nella colona col
       allora ritorna false */
    return false;
}

```

```

int main(){
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            schacchiera[i][j]=0;

    if (tenta(0) == false) {
        printf( »La soluzione non esiste");
    }

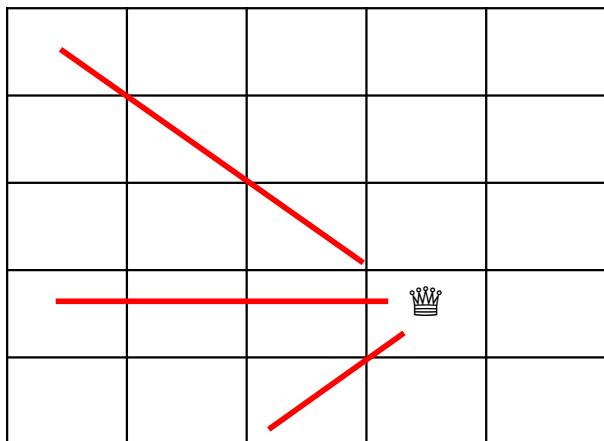
    else    StampaSoluzione();
    return 0;
}

```

Tempo: Esponenziale in N
 ~ 4 a 32 regine
 ~100 volte più veloce della forza brutta

Soluzione 8-Reine: Miglioramenti

- Un modo per rappresentare la soluzione: un vettore $T[N]$ dove $T[i]$ contiene la riga dove è stata posizionata la regina i -esima.



```
/* Funzione che controlla se una posizione schacchiera[i][j] è sicura. */

bool sicura(int riga, int col)
{
    int i, j;
    /* Controlla se ogni regina precedente alla attuale non si trovi sulla stessa riga o su una delle diagonali*/
    for (i = 0; i < col; i++) {
        /* ricava la posizione della regina nella riga iesima*/
        int xriga = T[i];
        /* stessa riga o stessa diagonale */
        if ((xriga==riga || xriga == riga - (col - i) ||
            xriga == riga + (col - i))
            return false;
    }
    return true;
}
```

Soluzione 8-Regine

- Il backtracking permette di produrre **TUTTE** le soluzioni possibili per il problema delle N-Regine.



```
/* Una funzione ricorsiva per risolvere le 8-Regine */
bool tenta(int col)
{
    /* caso base: se tutte le regine sono state posizionate ritorna true */
    if (col >= N){
        stampaSoluzione();
        return true;
    }

    /* Considera la colonna col e cerca di posizionare la regina
     * tentando le righe una dopo l'altra */
    for (int i = 0; i < N; i++) {
        if (sicura(i, col)) {
            schacchiera[i][col] = 1;
            /* chiamata ricorsiva per piazzare le altre regine */
            // if ( tenta((col + 1)) )
            //     return true;
            tenta(col+1);
            schacchiera[i][col] = 0; // BACKTRACK
        }
    }

    /* Se la regina non puo essere posizionata in nessuna riga nella
     * colonna col allora ritorna false */
    return false;
}
```

Esercizi olimpiadi



Piastrelle

<https://training.olinfo.it/#/task/piastrelle/statement>



Piastrellature (piastrelle)

Descrizione del problema

Pippo ha un corridoio di dimensione $1 \times N$ da piastellare. Lui ha a disposizione solo piastrelle di dimensioni 1×1 e 1×2 . Essendo questo il corridoio dell'ingresso di casa sua, lui vorrebbe che fosse il più bello possibile, quindi ha bisogno di conoscere tutte le possibili disposizioni delle piastrelle che potrebbe usare per completare il lavoro.

File di input

Il programma deve leggere da un file di nome `input.txt` nel quale è presente un unico intero N , la dimensione del corridoio.

File di output

Il programma deve scrivere in un file di nome `output.txt`. In ognuna delle K righe stampate deve essere presente una disposizione valida usando il seguente formato:

- Le piastrelle 1×1 vengono rappresentate con `[O]` (lettera ‘o’ maiuscola)
- Le piastrelle 1×2 vengono rappresentate con `[0000]` (quadrupla ‘o’ maiuscola)
- Le piastrelle non vengono separate da alcun carattere
- Le righe devono essere in ordine lessicografico: prima quelle che iniziano con 1×1 poi quelle con 1×2

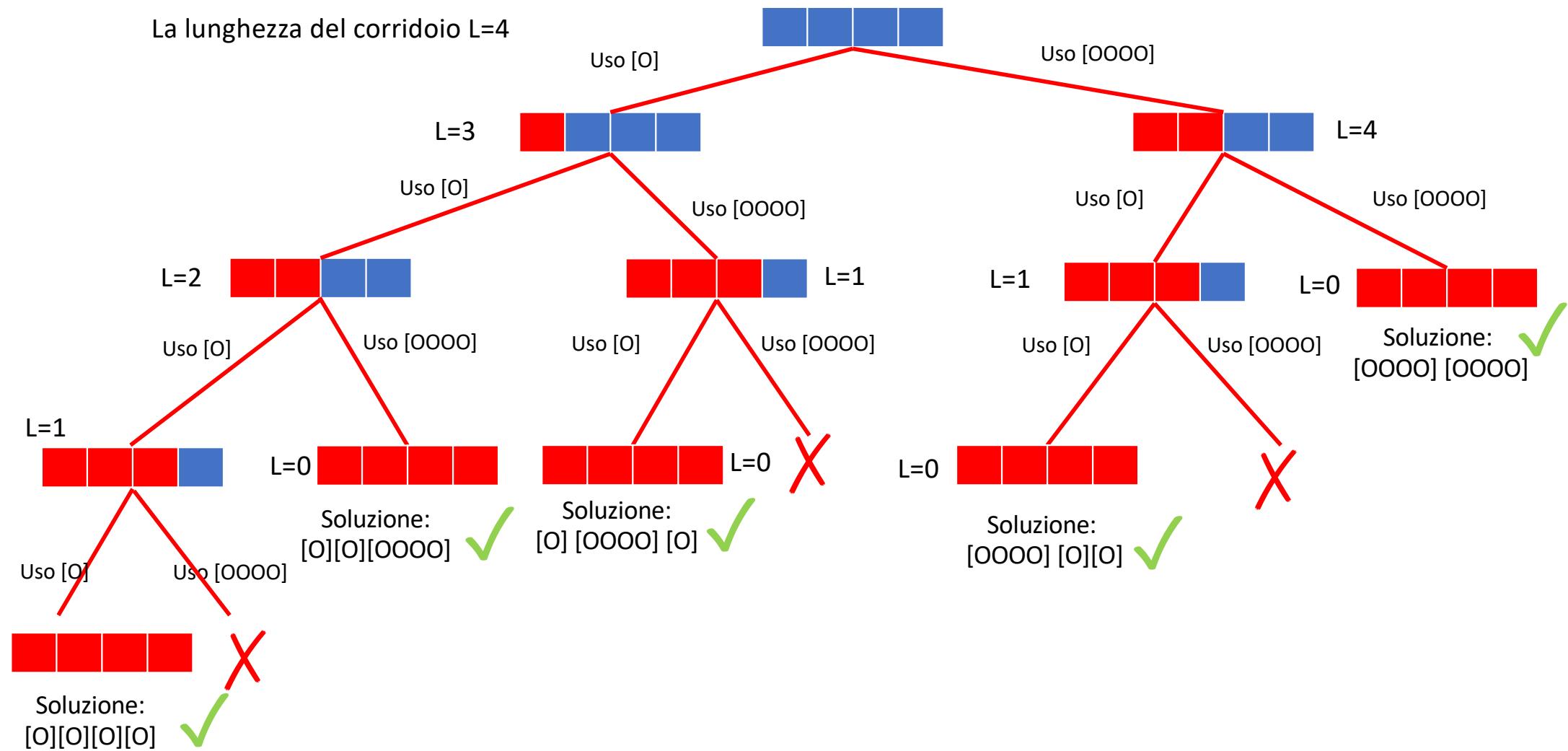
Assunzioni

- $1 \leq N \leq 25$

Esempio di input/output

File input.txt	File output.txt
4	[0] [0] [0] [0] [0] [0] [0000] [0] [0000] [0] [0000] [0] [0] [0000] [0000]

La lunghezza del corridoio L=4



Un problema dalle Territoriali 2011: Domino massimale

<https://training.olinfo.it/#/task/domino/statement>



Selezioni territoriali 2011

Domino massimale (domino)

Difficoltà D = 2.



Descrizione del problema

Sono date N tessere di domino, dove ogni tessera contiene due numeri compresi tra 0 e 6 in corrispondenza delle sue due estremità. Le tessere possono essere ruotate e la regola impone che due tessere possono essere concatenate se le loro estremità in contatto hanno inciso lo stesso numero. Aiuta a trovare il maggior numero di tessere che si possono concatenare a formare un'unica catena: non è detto che si riescano sempre a usare tutte le tessere; inoltre, possono esserci due o più tessere uguali a meno di rotazioni.

Dati di input

Il file `input.txt` è composto da $N + 1$ righe. La prima riga contiene l'intero positivo N , il numero delle tessere a disposizione. Ciascuna delle successive N righe contiene due interi positivi (compresi tra 0 e 6) separati da una spazio, che rappresentano i numeri incisi sulle estremità delle tessere.

Dati di output

Il file `output.txt` è composto da una sola riga contenente il massimo numero di tessere che possono essere concatenate con le regole del domino.

Assunzioni

- $2 \leq N \leq 10$.

Esempi di input/output

File input.txt

```
6  
3 0  
4 0  
2 6  
4 4  
0 1  
1 0
```

File output.txt

```
5
```

Nota/e

- In generale, più configurazioni possono soddisfare i requisti del problema: è sufficiente fornire la lunghezza massima.

Permutazioni ricorsive: intuizione

X_1	X_2	X_3	X_4
X_1	X_2	X_3	X_4
X_1	X_2	X_4	X_3
X_1	X_3	X_2	X_4
X_1	X_3	X_4	X_2
X_1	X_4	X_2	X_3
X_1	X_4	X_3	X_2

x_1	x_2	x_3	x_4
x_1	x_2	x_3	x_4
x_1	x_2	x_4	x_3
x_1	x_3	x_2	x_4
x_1	x_3	x_4	x_2
x_1	x_4	x_2	x_3
x_1	x_4	x_3	x_2
x_2	x_1	x_3	x_4
x_2	x_1	x_4	x_3
x_2	x_3	x_1	x_4
x_2	x_3	x_4	x_1
x_2	x_4	x_1	x_3
x_2	x_4	x_3	x_1
x_3	x_1	x_2	x_4
x_3	x_1	x_4	x_2
x_3	x_2	x_1	x_4
x_3	x_2	x_4	x_1
x_3	x_4	x_1	x_2
x_4	x_1	x_2	x_3
x_4	x_1	x_3	x_2
x_4	x_2	x_1	x_3
x_4	x_2	x_3	x_1
x_4	x_3	x_1	x_2
x_4	x_3	x_2	x_1

X_1	X_2	X_3	X_4
X_1	X_2	X_3	X_4
X_1	X_2	X_4	X_3
X_1	X_3	X_2	X_4
X_1	X_3	X_4	X_2
X_1	X_4	X_2	X_3
X_1	X_4	X_3	X_2
X_3	X_1	X_2	X_4
X_3	X_1	X_4	X_2
X_3	X_2	X_1	X_4
X_3	X_2	X_4	X_1
X_3	X_4	X_1	X_2
X_3	X_4	X_2	X_1
X_4	X_1	X_2	X_3
X_4	X_1	X_3	X_2
X_4	X_2	X_1	X_3
X_4	X_2	X_3	X_1
X_4	X_3	X_1	X_2
X_4	X_3	X_2	X_1

X_1	X_2	X_3	X_4
X_1	X_2	X_3	X_4
X_1	X_2	X_4	X_3
X_1	X_3	X_2	X_4
X_1	X_3	X_4	X_2
X_1	X_4	X_2	X_3
X_1	X_4	X_3	X_2
X_2	X_1	X_3	X_4
X_2	X_1	X_4	X_3
X_2	X_3	X_1	X_4
X_2	X_3	X_4	X_1
X_2	X_4	X_1	X_3
X_2	X_4	X_3	X_1
X_3	X_1	X_2	X_4
X_3	X_1	X_4	X_2
X_3	X_2	X_1	X_4
X_3	X_2	X_4	X_1
X_3	X_4	X_1	X_2
X_4	X_1	X_2	X_3
X_4	X_1	X_3	X_2
X_4	X_2	X_1	X_3
X_4	X_2	X_3	X_1
X_4	X_3	X_1	X_2
X_4	X_3	X_2	X_1

- Nel youtube channel di AlgoRhythmics potete vedere diversi algoritmi descritti tramite le danze. Tra questi avrete anche le 4-Regine. [link video](#)

Un problema dalle Territoriali 2009: Treno di container

<https://training.olinfo.it/#/task/treno/statement>



Treno di container (treno)

Difficoltà D = 2.

Descrizione del problema

Al porto sono arrivati N container della sostanza chimica di tipo A e N container della sostanza chimica di tipo B. I container sono stati caricati, uno dietro l'altro, su di un treno che ne può contenere $2N+2$. Le posizioni dei container sul treno sono numerate da 1 a $2N+2$. Il carico è stato fatto in modo che gli N container di tipo A occupino le posizioni da 1 a N , mentre quelli di tipo B da $N+1$ a $2N$; le rimanenti due posizioni $2N+1$ e $2N+2$ sono vuote.

Per motivi connessi all'utilizzo delle sostanze chimiche nella fabbrica alla quale sono destinate, i container vanno distribuiti sul treno a coppie: ciascun container per la sostanza di tipo A deve essere seguito da uno di tipo B. Occorre quindi che nelle posizioni dispari ($1, 3, 5, \dots, 2N-1$) vadano sistemati esclusivamente i container di tipo A mentre in quelle pari ($2, 4, 6, \dots, 2N$) quelli di tipo B, lasciando libere le ultime due posizioni $2N+1$ e $2N+2$.

A tal fine, viene impiegata una grossa gru, che preleva due container alla volta, in posizioni consecutive $i, i+1$, e li sposta nelle uniche due posizioni consecutive $j, j+1$ libere nel treno (inizialmente, $j = 2N+1$). Tale operazione è univocamente identificata dalla coppia (i, j) , dove entrambe le posizioni i e $i+1$ devono essere occupate da container mentre j e $j+1$ devono essere entrambe vuote.

Per esempio, con $N = 4$, abbiamo inizialmente la configurazione A A A A B B B B * *, dove le due posizioni vuote sono indicate da un asterisco *:

- Il primo spostamento della gru è (4,9) e porta alla configurazione:

A A A * * B B B A B
1 2 3 4 5 6 7 8 9 10

- Il secondo spostamento è (6, 4) e porta alla configurazione:

A A A B B * * B A B
1 2 3 4 5 6 7 8 9 10

- Il terzo spostamento è (2, 6) e porta alla configurazione:

A * * B B A A B A B
1 2 3 4 5 6 7 8 9 10

- Il quarto spostamento è (5,2) e porta alla configurazione:

A B A B * * A B A B
1 2 3 4 5 6 7 8 9 10

- Il quinto e ultimo spostamento è (9,5) e porta alla configurazione desiderata:

A B A B A B A B * *
1 2 3 4 5 6 7 8 9 10

Notare che per $N=4$ è possibile, con cinque spostamenti, sistemare i $2N$ container nell'ordine giusto. Scrivere quindi un programma che determini la successione degli spostamenti eseguiti dalla gru per ottenere un analogo risultato nel caso in cui $3 \leq N \leq 1000$. Si richiede inoltre che il numero K di tali spostamenti non superi il valore $3N$.

Dati di input

Il file `input.txt` è composto da una sola riga, contenente l'intero N che rappresenta il numero di container per ciascuna delle due sostanze.

Dati di output

Il file `output.txt` è composto da $K+1$ righe.

La prima riga contiene due interi positivi separati da uno spazio, rispettivamente il numero K di spostamenti operati dalla gru e il numero N di container per ciascuna delle due sostanze

Le righe successive contengono la sequenza di K spostamenti del tipo (i,j) , tali che partendo dalla sequenza `AAA...ABBB...B**`, si arrivi alla sequenza `ABABAB...AB**` con le regole descritte sopra. Ciascuna delle righe contiene una coppia di interi positivi i e j separati da uno spazio a rappresentare lo spostamento (i,j) .

Assunzioni

- $3 \leq N \leq 1000$,
- $1 \leq i, j \leq 2N+1$,
- $K \leq 3N$.

Esempi di input/output

File input.txt	File output.txt
3	4 3 2 7 6 2 4 6 7 4
4	5 4 4 9 6 4 2 6 5 2 9 5

File input.txt

5

File output.txt

6 5
5 11
2 5
9 2
6 9
3 6
11 3

Esercizi per casa



Esercizi

- Quasi-palindromi (Territoriali 2010)

<https://training.olinfo.it/#/task/quasipal/statement>

- Cerca le somme (GATOR 2015)

<https://training.olinfo.it/#/task/cercalesomme/statement>

- Fractal Painintg (painting) :

https://training.olinfo.it/#/task/ois_painting/statement