ESPOSITO CLAUDIA ANTONELLA

PROBLEMA 1

Data un vettore che può contenere numeri interi sia positivi che negativi, trovare il sotto array di numeri contigui che ha la somma più grande, e riportare tale somma.

INPUT

Ogni riga contiene un caso di test, rappresentato dagli elementi del vettore di cui si vuole calcolare la somma del massimo sotto array. I casi di test terminano con una riga END

OUTPUT

Ogni riga contiene il valore della somma richiesto per il corrispondente caso di tesi

CODICE

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream>
#include <vector>
using namespace std;

int maxSum(const std::vector<int>& vettore){
      int max=0;
      int sum = 0;

      for (int num : vettore) {
           sum = std::max(num, sum + num);
           max = std::max(max, sum);
          }
      return max;
}

int main()
{
      ifstream inputFile("\\Users\\espoc\\Documents\\file.txt");

    if (!inputFile.is_open()) {
          cout << "Impossibile aprire il file." << endl;
          return 1;
      }
    string line;
```

```
        while (getline(inputFile, line)) {
            if (line == "END") {
                cout<<"end"<<endl;
                break;
            }
            vector<int> test_case;
                  size_t pos = 0;
            while ((pos = line.find(' ')) != string::npos) {


                                int num = stoi(line.substr(0, pos));
                                test_case.push_back(num);

                        line.erase(0, pos + 1);
                    }

                        try {
                            int num = stoi(line);
                            test_case.push_back(num);
                        } catch (const std::invalid_argument& e) {
                            cout << "Invalid argument: " << e.what() << endl;

                        }


                    int result = maxSum(test_case);
                    cout << result << endl;

        }

        return 0;
}
```

INPUT

-1 -3 4 2

-1 2 -5 7

1 -2 3 5

-2 1 -1 2

-3 4 6 1

7 -1 3 -2

END

OUTPUT

6

7

8

2

11

9

end

ALGORITMO SPIEGAZIONE

COMPLESSITA' O(n) è caratterizzato da un **for** che scorre gli n elementi nel vettore**.**

Nel main apriamo il file di input tramite ifstream (internal stream buffer) che fa operazioni di input e output sui file con cui viene associato

```
ifstream inputFile("\\Users\\espoc\\Documents\\file.txt");
```

in caso di errore di apurtera del file restituiamo un messaggio di errore.

```
    if (!inputFile.is_open()) {
          cout << "Impossibile aprire il file." << endl;
          return 1;
      }
```

La lettura del file si interrompe quando incontra la stringa END

```
  while (getline(inputFile, line)) {
          if (line == "END") {
              break;
          }
```

Il while si ripete fino allo spazio, dato che ogni riga contiene un caso di test

La funzione **Stoi** serve a convertire da stringa a intero e viene successivamente interito nel vettore **test_case**

Una volta terminata l'intera riga, chiamo la funzione sul vettore

```
  vector<int> test_case;
                size_t pos = 0;
          while ((pos = line.find(' ')) != string::npos) {

                                int num = stoi(line.substr(0, pos));
                                test_case.push_back(num);
```

```
            line.erase(0, pos + 1);
        }

        try {
            int num = stoi(line);
            test_case.push_back(num);
        } catch (const std::invalid_argument& e) {
            cout << "Invalid argument: " << e.what() << endl;

        }

    int result = maxSum(test_case);
    cout << result << endl;

}
```

La funzione **maxSum** prende in ingresso il vettore e pongo a zero un valore intero **max** che è la somma massima trovata finora e un valore intero **sum** che contiene la somma corrente del sottoarray in esame .

Per ogni valore letto nel vettore ho sempre che **sum** contiene il massimo tra l'elemento corrente nel vettore e la somma corrente **sum** più l'elemento stesso, ciò consente di decidere se includere l'elemento corrente nel sottoarray o iniziare un nuovo sottoarray. .

Si calcola il massimo tra **sum** e **max**. Questo assicura che **max** contenga sempre il massimo valore tra tutti gli elementi considerati fino a quel momento.

Perciò se il valore corrente aggiungendolo a **sum** ne peggiora la somma corrente, perciò il massimo tra **num** e **sum+num** mi restituisce **num**, inizio un nuovo sottoarray.

Se invece il massimo è **sum+num** , la conta del sottoarray continua perciò **sum** diventa **sum+num e max** viene aggiornato al nuovo valore della somma corrente

esempio

1 -2 3 5

Se il valore corrente è -2 , **sum** deve fare il massimo tra (-2, 1+(-2)) ovvero sum diventa **sum+num** (-1)

Mentre **max** deve fare il massimo tra (1,-1) restituendo perciò 1, in questo modo nel conteggio della somma del sottoarray escludo -2

Il valore successivo è 3(**sum** =-1) . **sum** = max(3,2) =3 , il valore **max** = max(1,3)= 3

Il valore corrente è 5 **sum**= max(3,8) restituisce 8 (la massima somma del sottoarray contiguo)

Max= max(3,8)= 8

Perciò ho trovato il sottoarray di numeri contigui che ha la somma più grande (5,3)-

```
int maxSum (const std::vector<int>& vettore){
	int max = 0;
	int sum = 0;

	for (int num : vettore) {
		 sum = std::max(num, sum + num);
		 max = std::max(max, sum);
		}



	return max;
}
```

PROBLEMA 2

Si supponga di disporre di una scacchiera NxN. Determinare il numero di modi in cui è possibile posizionare N regine in modo tale che nessuna coppia di regine si possa attaccare a vicenda. Quindi, una soluzione richiede che due regine non condividano la stessa riga, colonna o diagonale (si supponga N > 3). Suggerimento: si utilizzi il backtracking.

INPUT L'input è costituito da diversi casi di test. La prima riga contiene il numero di casi di test. Per ogni test case è fornito un numero intero che rappresenta N

OUTPUT

Per ogni test case, il programma riporti in output il numero di modi in cui è possibile posizionare N regine in modo tale che nessuna coppia di regine si possa attaccare a vicenda. Si noti che le soluzioni (ad esempio, 92 per una scacchiera da 8) si riferiscono a soluzioni distinte ma non indipendenti (alcune soluzioni possono essere ottenute da altre tramite operazioni di rotazione e riflessione; nel caso di N=8, le soluzioni indipendenti sono 12).

CODICE

```
#include <iostream>
#include <vector>
#include <bits/stdc++.h>
#include <fstream>
using namespace std;



bool Safe(vector<vector<int>> board, int row, int col){
	int i,j;
```

```
      int N = board.size();
      for(int i = 0; i<col; i++){
            if(board[row][i]){
                  return false;}
      }
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--){
        if (board[i][j]){
            return false;}}

    for (i = row, j = col; j >= 0 && i < N; i++, j--){
        if (board[i][j]){
             return false;}}



      return true;
}
vector<vector<int> > result;
bool backtracking( vector<vector<int>>& board, int col){

      int N = board.size();
      if(col == N){
            vector<int> v;

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

                  for(int j =0; j<N; j++){

                        if(board[i][j]== 1)
                              v.push_back(j+1);
                  }
            }
            result.push_back(v);

            return true;

      }
```

```
	for(int i =0; i<N;i++){

		if(Safe(board,i,col)){

			board[i][col]=1;

			backtracking(board, col+1);

			board[i][col]=0;
		}
	}

	return true;

}

vector<vector<int>> queen(int n){
	result.clear();


	vector<vector<int>> board(n, vector<int>(n,0));

	if(backtracking(board,0)== false){
		return {};
	}

		return result;
	}
int main(){

	ifstream inputFile("\\Users\\espoc\\Documents\\file2.txt");
	if (!inputFile.is_open()) {
		cout << "Impossibile aprire il file." << endl;
		return 1;
		}
```

```
string line;

            getline(inputFile, line);

          int num = stoi(line);


               for(int i =0; i<num;i++){
                     string line;
                     getline(inputFile, line);

                            int num = stoi(line);


                vector<vector<int>> result = queen(num);
                 cout << result.size() << endl;
      }
}
```

INPUT

6
4
5
8
9
6
7

OUTPUT

2
10
92
352
4
40

SPIEGAZIONE CODICE

COMPLESSITA O(n!) in ogni chiamata ricorsiva diminuisco di 1 le posizioni possibili ovvero, per la prima colonna ho n righe tra cui scegliere, per la seconda diventa n-1 ect.. n(n-1)(n-2)…=n!

La lettura del file segue lo stesso procedimento precedente. La prima riga contiene il numero di casi di test, perciò andrò a creare un while che ripete l'estrazione di N per il numero di casi di test necessari.

Ogni riga contiene un numero che indica la dimensione NxN della scacchiera che andrà in ingresso ad una prima funzione che deve restituire un **vector<vector<int>>** che rappresenta la scacchiera.

Richiamo la funzione di backtracking che prende ingresso la scacchiera e la posizione della colonna 0.

```
vector<vector<int>> queen(int n){
	result.clear();

	vector<vector<int>> board(n, vector<int>(n,0));

	if(backtracking(board,0)== false){
		return {};
	}

		return result;
	}
```

La funzione di **backtracking** è una funzione booleana.

```
vector<vector<int> > result;
bool backtracking( vector<vector<int>>& board, int col){

	int N = board.size();
```

Se arrivo alla Colonna N, ovvero avrò percorso tutta la scacchiera, inserisco in un vettore di interi **v** il numero di Regine che ho inserito in questa soluzione.

Result , che è un vettore di vettori di interi, contiene tutti i diversi **v** possibili ovvero l'insieme delle diverse soluzioni trovate. L'output è la dimensione di **Result,** che mi indica quante soluzioni ho trovato.

```
	if(col == N){
		vector<int> v;

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

			for(int j =0; j<N; j++){

				if(board[i][j]== 1)
					v.push_back(j+1);
```

```
            }
        }
        result.push_back(v);

        return true;

    }
```

Data la Colonna **col** per ogni elemento della riga corrispondente controllo se posso piazzare una Regina tramite la funzione booleana **Safe.**

Se è possibile inserirla vado a posizionarla inserendo nella posizione il valore 1 e richiamo la funzione (backtracking) sulla colonna successiva per poi riporre il valore a 0.

```
for(int i =0; i<N;i++){

        if(Safe(board,i,col)){

            board[i][col]=1;

            backtracking(board, col+1);

            board[i][col]=0;
        }
    }

    return true;

}
```

La funzione **Safe** è una funzione booleana che prende in ingresso la scacchiera, la riga corrente e la colonna corrente.

Sia per la Colonna che la riga che la diagonale controllo ogni valore sulla scacchiera, se ci sono altre regine(ovvero se una delle posizioni è uguale a 1) restituisce falso, se non trova niente restituisce vero.

```
bool Safe(vector<vector<int>> board, int row, int col){
    int i,j;
    int N = board.size();

    for(int i = 0; i<col; i++){
        if(board[row][i]){
```

```
                return false;}
      }

    for (i = row, j = col; i >= 0 && j >= 0; i--, j--){
        if (board[i][j]){
            return false;}}

    for (i = row, j = col; j >= 0 && i < N; i++, j--){
        if (board[i][j]){
             return false;}}



      return true;
}
```