2.7

```
//*****************
// IMPORTANTE
//
// Ana Álava Papí
// E01
// E01
//
//**************************************************

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <cmath>
#include <algorithm>

template <class T>

using tMatriz = std::vector<std::vector<T>>;


//*******************************
// Aqui explicacion del algoritmo de vuelta atras.
// Como es el árbol de exploración y como se van calculando las soluciones
/*
Recorro la matriz de afinidades, en cada iteración incremento en uno la posicion de las parejas en el vector sol,
de manera que se identifique que esas personas ya tienen pareja. Incermento el contador de afinidades y un contador de parejas_finales.

En la función esValida compruebo si la i==k para no emparejar a una persona consigo misma, compruebo el vector sol, si alguna de las posiciones indicadas son
mayor que 1 esa(s) persona(s) ya tiene(n) pareja, por último compruebo que la afinidad que tienen el uno por el otro es mayor que 0

Si la pareja es válida compruebo si he llegado al final (k = n-1) y si eso se cumple, compruebo también si el número de parejas es el correcto, es decir,
que nadie se haya quedado sin pareja. Si no se ha llegado al final llamo a la función incrementando la función en 1.

*/
//
//*******************************


bool esValida(int const& i, int const& k, std::vector<int>const& sol, tMatriz<int> const& afinidades) {

	if (i == k) return false;//No se puede emparejar con uno mismo
	if (sol[i] > 1 || sol[k] > 1)return false;//Las parejas deben de ser únicas (solo se puede tener una pareja)
	if (afinidades[i][k] == 0 || afinidades[k][i] == 0) return false;//Afinidad > 0

	return true;
```

No se puede preguntar en el esValido si ya se emparejó antes, lo que hay que hacer es seguir

emparejando a los siguientes

```
}

// Aqui funcion que resuelve el problema

void resolver(int const& n, tMatriz<int> const& afinidades, std::vector<int>& sol, int& max_act,
int& sol_mejor, int& parejas_finales, int k) {
    for (int i = 0; i < n; ++i) {
        //marco
        sol[i]++;
        sol[k]++;
        parejas_finales++;
        max_act += afinidades[i][k] + afinidades[k][i];

        if (esValida(i,k,sol,afinidades)) {
            if (k = n - 1 && parejas_finales == n/2) {//FIN

                if (max_act > sol_mejor) sol_mejor = max_act;
            }
            else resolver(n, afinidades, sol, max_act, sol_mejor, parejas_finales, k +
1);
        }

        //desmarco
        sol[i]--;
        sol[k]--;
        parejas_finales--;
        max_act -= afinidades[i][k] + afinidades[k][i];
    }
}


// Pare lectura de datos y mostrar los resultados
void resuelveCaso() {
    // Lectura de datos
    int n, max_act, sol_mejor, parejas_finales;
    std::cin >> n;

    tMatriz<int> afinidades;
    afinidades.resize(n, std::vector<int>(n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            std::cin >> afinidades[i][j];
        }
    }
    sol_mejor = 0;
    max_act = 0;
    parejas_finales = 0;
    std::vector<int> sol(n, 0);
    //LLamada a la funcion de vuelta atras
    resolver(n, afinidades, sol, max_act, sol_mejor, parejas_finales, 0);
```

if (sol[k]>1) { resolver (. . . k+1); - 0.3.

no hace falta y solo lia el codigo

optimizacion -0.5

```
	// Escribir el resultado
	std::cout << sol_mejor << "\n";

}

int main() {
	// Para redireccionar la entrada a un fichero
#ifndef DOMJUDGE
	std::ifstream in("E3.txt");
	auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif

	int numCasos; std::cin >> numCasos; std::cin.ignore();
	for (int i = 0; i < numCasos; ++i)
		resuelveCaso();

#ifndef DOMJUDGE // para dejar todo como estaba al principio
	std::cin.rdbuf(cinbuf);
	system("PAUSE");
#endif

	return 0;
}
```