Soluciones OIFem 2020/21

Final Día 1

Repartiendo galletas

Autora: Blanca Huergo

Teoría

- Bucles
- Módulo / división con enteros

Solución

Sumaremos el número de galletas en cada caja y comprobaremos usando el operador *módulo* si se pueden dividir entre el número de concursantes que hay.

Código

C++

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


int main() {
    int t, n, tot, cur;
    cin >> t;
    while(t--) {
        cin >> n;
        tot = 0;
        for (int i = 0; i < n; i++) {
            cin >> cur;
            tot += cur;
        }
        if (tot % n == 0) cout << tot/n << '\n';
        else cout << "-1\n";
    }
```

```
    return 0;
}
```

Java

```java
import java.util.Scanner;

public class Main {
  public static void main(String args[]) {
    Scanner S = new Scanner(System.in);
    int t = S.nextInt();
    for (int i = 0; i < t; i++) {
      int s = 0;
      int n = S.nextInt();
      for (int j = 0; j < n; j++)
        s += S.nextInt();
      if (s % n != 0) System.out.println(-1);
      else System.out.println(s / n);
    }
    S.close();
  }
}
```

Python

```python
t = int(input())
for i in range(t):
    s = 0
    n = int(input())
    for g in input().split():
        s = s + int(g)
    if s % n != 0:
        print(-1)
    else:
        print(int(s / n))
```

Desenmascarar al ganador

Autor: Jacobo Vilella

Teoría

- Bucles
- Condicionales
- Secuenciación

Solución

Para este problema había que simular la situación descrita en el escenario, teniendo cuidado con el orden de los if, ya que puedes caer en una casilla de Dados y luego en una Oca, por ejemplo.

Código

C++

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int t, a;
    cin >> t;
    while (t--) {
        cin >> a;
        int n = 2 + a; // El numero de jugadores son Max, Izan y los amigos a los que invitan
        vector<int> c(n, 1); // Posiciones en el tablero: Creamos un vector para los jugadores, todos empiezan en la casilla 1
        vector<int> p(n, 0); // Partidas ganadas por cada jugador: Creamos un vector para los jugadores, todos empiezan con 0 partidas ganadas

        int j = 0, d;
        while (true) { // Vamos leyendo mientras haya dados apuntados
            cin >> d;
            if (d == -1)
                break;
            int casilla = c[j] + d;

            if (casilla == 26 || casilla == 53) {
                int d2;
                cin >> d2;
                casilla += d2;
            } // Casillas Dados
            if (casilla > 63)
                casilla = 63 - (casilla - 63); // Se pasa del Jardin de la Oca

            // Ahora ya sabemos la casilla final de este dado, analizamos si es una casilla especial que lleva a otra casilla destino
```

```
            else if (casilla == 42)
                casilla = 30; // Casillas Laberinto
            else if (casilla == 58)
                casilla = 1; // Casillas Calavera
            else if (casilla == 5)
                casilla = 9; // Casillas Oca
            else if (casilla == 9)
                casilla = 14; // Es muy importante usar "else if", si se pone "if" al entrar en una casilla Oca entra en todas las siguientes casillas Oca
            else if (casilla == 14)
                casilla = 18;
            else if (casilla == 18)
                casilla = 23;
            else if (casilla == 23)
                casilla = 27;
            else if (casilla == 27)
                casilla = 32;
            else if (casilla == 32)
                casilla = 36;
            else if (casilla == 36)
                casilla = 41;
            else if (casilla == 41)
                casilla = 45;
            else if (casilla == 45)
                casilla = 50;
            else if (casilla == 50)
                casilla = 54;
            else if (casilla == 54)
                casilla = 59;

            if (casilla == 63) { // Casilla Jardin de la Oca: es el ganador y se comienza una nueva partida
                p[j]++; // Incrementar el contador del ganador
                for (int i = 0; i < (int) c.size(); i++)
                    c[i] = 1; // Todos vuelven a la casilla de salida, Se puede usar tambien fill(c.begin(), c.end(), 1);
            } else { // Guardamos la casilla en la que ha acabado este turno este jugador
                c[j] = casilla;
            }
            j = (j + 1) % n; // Pasamos al siguiente jugador
        }

        // Buscamos quien es el ganador de la noche
        int p_max = 0, j_max = -1; // Inicialmente todos tienen 0 puntos y por lo tanto estan empatados
        for (int i = 0; i < (int) p.size(); i++) {
            if (p[i] > p_max) { // Hemos encontrado un nuevo maximo
                p_max = p[i];
                j_max = i;
            } else if (p[i] == p_max) {
                j_max = -1; // El maximo que hemos encontrado hasta ahora tiene empate
            }
```

```
        }
        cout << (j_max + 1) << '\n';
    }
}
```

Java

```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        int t, a, n;
        Scanner s = new Scanner(System.in);
        t = s.nextInt();
        for (int caso = 0; caso < t; caso++) {
            a = s.nextInt();
            n = 2+a; // El numero de jugadores son Max, Izan y los amigos a los que invitan
            Integer[] c = new Integer[n]; // Posiciones en el tablero: Creamos un array para los jugadores, todos empiezan en la casilla 1
            Arrays.fill(c, new Integer(1));
            Integer[] p = new Integer[n]; // Partidas ganadas por cada jugador: Creamos un array para los jugadores, todos empiezan con 0 partidas ganadas
            Arrays.fill(p, new Integer(0));
            int j = 0, d;
            while(true) {
                // Vamos leyendo mientras haya dados apuntados
                d = s.nextInt();
                if (d == -1)
                    break;
                int casilla = c[j] + d;
                if (casilla == 26 || casilla == 53) {
                    int d2;
                    d2 = s.nextInt();
                    casilla += d2;
                } // Casillas Dados
                if (casilla > 63)
                    casilla = 63 - (casilla - 63); // Se pasa del Jardin de la Oca
                // Ahora ya sabemos la casilla final de este dado, analizamos si es una casilla especial que lleva a otra casilla destino
                else if (casilla == 42)
                    casilla = 30; // Casillas Laberinto
                else if (casilla == 58)
                    casilla = 1; // Casillas Calavera
                else if (casilla == 5)
                    casilla = 9; // Casillas Oca
                else if (casilla == 9)
                    casilla = 14; // Es muy importante usar "else if", si se pone "if" al entrar en una casilla Oca entra en todas las siguientes casillas Oca
                else if (casilla == 14)
                    casilla = 18;
```

```
                else if (casilla == 18)
                    casilla = 23;
                else if (casilla == 23)
                    casilla = 27;
                else if (casilla == 27)
                    casilla = 32;
                else if (casilla == 32)
                    casilla = 36;
                else if (casilla == 36)
                    casilla = 41;
                else if (casilla == 41)
                    casilla = 45;
                else if (casilla == 45)
                    casilla = 50;
                else if (casilla == 50)
                    casilla = 54;
                else if (casilla == 54)
                    casilla = 59;

                if (casilla == 63) { // Casilla Jardin de la Oca: es el ganador y se comienza una nueva partida
                    p[j]++; // Incrementar el contador del ganador
                    for (int i = 0; i < n; i++)
                        c[i] = 1; // Todos vuelven a la casilla de salida
                } else { // Guardamos la casilla en la que ha acabado este turno este jugador
                    c[j] = casilla;
                }
                j = (j + 1) % n; // Pasamos al siguiente jugador
            }

            // Buscamos quien es el ganador de la noche
            int p_max = 0, j_max = -1; // Inicialmente todos tienen 0 puntos y por lo tanto estan empatados
            for (int i = 0; i < n; i++) {
                if (p[i] > p_max) { // Hemos encontrado un nuevo maximo
                    p_max = p[i];
                    j_max = i;
                } else if (p[i] == p_max) {
                    j_max = -1; // El maximo que hemos encontrado hasta ahora tiene empate
                }
            }
            System.out.println(j_max + 1);
        }
        s.close();
    }
}
```

Python

```
t = int(input())
for caso in range(t):
    a = int(input())
    partidas = 0
```

```
    n = 2+a # El numero de jugadores son Max, Izan y los amigos a los que invitan
    c = [1 for i in range(n)] # Posiciones en el tablero: Creamos un array para los jugadores, todos empiezan en la casilla 1
    p = [0 for i in range(n)] # Partidas ganadas por cada jugador: Creamos un array para los jugadores, todos empiezan con 0 partidas ganadas
    j = 0
    while True:
        # Vamos leyendo mientras haya dados apuntados
        d = int(input())
        if (d == -1):
            break
        casilla = c[j] + d
        if (casilla == 26 or casilla == 53):
            d2 = int(input())
            casilla += d2
            # Casillas Dados
        if (casilla > 63):
            casilla = 63 - (casilla - 63) # Se pasa del Jardin de la Oca

        # Ahora ya sabemos la casilla final de este dado, analizamos si es una casilla especial que lleva a otra casilla destino
        elif (casilla == 42):
            casilla = 30; # Casillas Laberinto
        elif (casilla == 58):
            casilla = 1; # Casillas Calavera
        elif (casilla == 5):
            casilla = 9; # Casillas Oca
        elif (casilla == 9):
            casilla = 14; # Es muy importante usar "elif", si se pone "if" al entrar en una casilla Oca entra en todas las siguientes casillas Oca
        elif (casilla == 14):
            casilla = 18;
        elif (casilla == 18):
            casilla = 23;
        elif (casilla == 23):
            casilla = 27;
        elif (casilla == 27):
            casilla = 32
        elif (casilla == 32):
            casilla = 36
        elif (casilla == 36):
            casilla = 41
        elif (casilla == 41):
            casilla = 45;
        elif (casilla == 45):
            casilla = 50;
        elif (casilla == 50):
            casilla = 54;
        elif (casilla == 54):
            casilla = 59;

        if (casilla == 63): # Casilla Jardin de la Oca: es el ganador y se comienza una nueva partida
            p[j] += 1 # Incrementar el contador del ganador
```

```
54                partidas += 1
55                for i in range(len(c)):
56                    c[i] = 1 # Todos vuelven a la casilla de salida
57            else: # Guardamos la casilla en la que ha acabado este turno este jugador
58                c[j] = casilla
59            j = (j + 1) % n # Pasamos al siguiente jugador
60
61            # Buscamos quien es el ganador de la noche
62            p_max = 0
63            j_max = -1 # Inicialmente todos tienen 0 puntos y por lo tanto estan
              ↪ empatados
64            for i in range(len(p)):
65                if (p[i] > p_max): # Hemos encontrado un nuevo maximo
66                    p_max = p[i]
67                    j_max = i
68                elif (p[i] == p_max):
69                    j_max = -1 # El maximo que hemos encontrado hasta ahora tiene
                      ↪ empate
70        print(j_max + 1)
```

E Corp

Autor: David García Quintas

Teoría

- Método de bisección (búsqueda binaria)

Solución

Usamos el método de bisección para encontrar el mayor C para el que esto se cumpla. Si este C es menor al salario más bajo, imprimiremos -1. Para cada número que probemos, podemos simular el proceso del corte para comprobar el resultado.

Código

C++

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

bool esPosible(long long int cDePrueba, long long int objetivo, vector<int> & salarios) {
    long long int tot = 0;
    for (int sal: salarios)
        tot += min((long long int) sal, cDePrueba);
    return tot <= objetivo;
}

long long int maximoC(long long int objetivo, vector<int> & salarios) {
    long long int mn = *min_element(salarios.begin(), salarios.end()), mx = *max_element(salarios.begin(), salarios.end());
    long long int lo = 0, hi = mx, mid, k = -1;
    while(lo <= hi) {
        mid = lo + (hi-lo)/2;
        if (esPosible(mid, objetivo, salarios)) {
            lo = mid+1;
            k = mid;
        } else hi = mid-1;
    }
    if (k < mn)
        return -1;
    return k;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    int t, n;
    long long int T;
```

```
    cin >> t;
    while(t--) {
        cin >> n;
        vector<int> salarios(n);
        for (int i = 0; i < n; i++)
            cin >> salarios[i];
        cin >> T;
        cout << maximoC(T, salarios) << '\n';
    }
    return 0;
}
```

Java

```
import java.io.*;
import java.util.*;

public class Solution {
    static ArrayList<Integer> salarios;
    static boolean esPosible(long cDePrueba, long objetivo) {
        long tot = 0;
        for (int sal: salarios)
            tot += Math.min(sal, cDePrueba);
        return tot <= objetivo;
    }

    static long maximoC(long objetivo) {
        long mn = Collections.min(salarios), mx = Collections.max(salarios);
        long lo = 1, hi = mx, mid, k = -1;
        while(lo <= hi) {
            mid = lo + (hi-lo)/2;
            if (esPosible(mid, objetivo)) {
                lo = mid+1;
                k = mid;
            } else hi = mid-1;
        }
        if (k < mn)
            return -1;
        return k;
    }

    public static void main(String[] args) {
        int t, n;
        long T;
        Scanner s = new Scanner(System.in);
        t = s.nextInt();
        for (int caso = 0; caso < t; caso++) {
            n = s.nextInt();
            salarios = new ArrayList<Integer>();
            for (int i = 0; i < n; i++)
                salarios.add(s.nextInt());
            T = s.nextLong();
            System.out.println(maximoC(T));
        }
        s.close();
```

```
    }
}
```

Python

```python
salarios = []
def esPosible(cDePrueba, objetivo):
    tot = 0
    for sal in salarios:
        tot += min(sal, cDePrueba)
    return tot <= objetivo

def maximoC(objetivo):
    mn = min(salarios)
    mx = max(salarios)
    lo = 0
    hi = mx
    k = -1
    while(lo <= hi):
        mid = lo + (hi-lo)//2
        if esPosible(mid, objetivo):
            lo = mid+1
            k = mid
        else:
            hi = mid-1
    if (k < mn):
        return -1
    return k

t = int(input())
for caso in range(t):
    n = int(input())
    salarios = list(map(int, input().split()))
    T = int(input())
    print(maximoC(T))
```

Swish

Autora: Blanca Huergo

Teoría

- DFS
- Ciclos
- Representación de grafos (matrices y listas de adyacencia)

Solución

La clave para este problema era ver que se podía representar el juego como un grafo. Representamos cada letra del alfabeto con un nodo y, si una carta está formada por una letra c_1 y una letra c_2, añadimos una arista de c_1 a c_2 (pero no al revés). De esta forma, podremos encontrar el mayor Swish buscando el ciclo más grande del grafo. Basta con iniciar una DFS desde cada nodo del grafo y guardar el mayor ciclo que lo incluya. Así, retornamos el mayor de los 26, o -1 si no existe tal ciclo.

Código

C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef long long int ll;
typedef vector <int> vi;
typedef vector <vi> vvi;
typedef pair <int, int> ii;
typedef vector <ii> vii;

vvi G;
vi visited, dp;

const int alfa = 'z' - 'a' + 1;

int cycle(int first, int v, int d){
    visited[v] = 1;

    for (int u: G[v]) {
        if (visited[u]) {
            if (u == first) dp[v] = max(dp[v], 1);
        }
        else dp[v] = max(dp[v], 1 + cycle(first, u, d));
    }

    return dp[v];
}

```

```cpp
int find_longest_cycle(){
    int ans = -1;
    for (int i = 0; i < alfa; ++i){
        visited = vi(alfa, 0);
        dp = vi(alfa, -1e9);
        ans = max(ans, cycle(i, i, 0));

    }

    return ans;
}


int main(){
    ios::sync_with_stdio(false);
    cin.tie(null);

    int t;
    cin >> t;

    for (int w = 0; w < t; ++w){
        vvi M(alfa, vi(alfa, 0));
        int n;
        cin >> n;

        for (int i = 0; i < n; ++i){
            string s;
            cin >> s;
            M[s[0] - 'A'][s[1] - 'A'] = 1;
        }

        G = vvi(alfa);

        for (int i = 0; i < alfa; ++i){
            for (int j = 0; j < alfa; ++j){
                if (M[i][j]) G[i].push_back(j);
            }
        }

        cout << find_longest_cycle() << '\n';
    }
}
```

Java

```java
import java.io.*;
import java.util.*;

public class Solution {
    static ArrayList<ArrayList<Integer>> G;
    static int alfa = 'z' - 'a' + 1;
    static int[] visited = new int[alfa], dp = new int[alfa];
    public static int cycle(int first, int v, int d) {
        int ans = -1;
        visited[v] = 1;
```

```
        for (int u: G.get(v)) {
            if (visited[u] == 1) {
                if (u == first)
                    dp[v] = Math.max(dp[v], 1);
            } else dp[v] = Math.max(dp[v], 1 + cycle(first, u, d));
        }
        return dp[v];
    }

    public static int find_longest_cycle() {
        int ans = -1;
        for (int i = 0; i < alfa; i++) {
            Arrays.fill(visited, 0);
            Arrays.fill(dp, -1000000000);
            ans = Math.max(ans, cycle(i, i, 0));
        }
        return ans;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int t = s.nextInt();
        for (int caso = 0; caso < t; caso++) {
            int[][] M = new int[alfa][alfa];
            for (int i = 0; i < alfa; i++)
                for (int j = 0; j < alfa; j++)
                    M[i][j] = 0;
            int n = s.nextInt();
            for (int i = 0; i < n; i++) {
                String str = s.next();
                M[str.charAt(0) - 'A'][str.charAt(1) - 'A'] = 1;
            }
            G = new ArrayList<ArrayList<Integer>>(alfa);
            for (int i = 0; i < alfa; i++) {
                G.add(new ArrayList<Integer>());
                for (int j = 0; j < alfa; j++) {
                    if (M[i][j] == 1)
                        G.get(i).add(j);
                }
            }
            System.out.println(find_longest_cycle());
        }
        s.close();
    }
}
```

Python

```
G = []
dp = []
visited = []

alfa = ord('z') - ord('a') + 1

def cycle(first, v, d):
```

```
    global visited, G, dp
    ans = -1
    visited[v] = 1

    for u in G[v]:
        if (visited[u]):
            if (u == first):
                dp[v] = max(dp[v], 1)
        else:
            dp[v] = max(dp[v], 1 + cycle(first, u, d))

    return dp[v]

def find_longest_cycle():
    global visited, dp
    ans = -1
    for i in range(alfa):
        visited = [0 for j in range(alfa)]
        dp = [-1e9 for j in range(alfa)]
        ans = max(ans, cycle(i, i, 0))
    return ans

t = int(input())
for w in range(t):
    M = [[0 for i in range(alfa)] for j in range(alfa)]
    n = int(input())
    S = input().split()
    for i in range(n):
        M[ord(S[i][0]) - ord('A')][ord(S[i][1]) - ord('A')] = 1

    G = [[] for i in range(alfa)]

    for i in range(alfa):
        for j in range(alfa):
            if (M[i][j]):
                G[i].append(j)

    print(find_longest_cycle())
```

Alea iacta est

Autor: David García Quintas

Teoría

- Programación dinámica
- Módulos

Solución

Para resolver este problema, había que usar la siguiente fórmula recursiva:

`formas(numDados, caras, objetivo)` = $\sum_{k=1}^{caras}$ `formas(numDados-1, k, objetivo-k)`

Para sacar los 100 puntos, había que guardar resultados anteriores en un vector (programación dinámica) y usarlos en vez de recalcularlos. Además, sabemos que no habrá formas de llegar a s si $cd < s$, por lo que evitamos hacer la DP con s grande, ya que esto nos daría MLE. Otra pequeña optimización es imprimir 0 cuando $s < d$, ya que sumaremos más que s hasta tirando todo 1.

Código

C++

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

int c, mod = 1e9 + 7;

vector<vector<int>> DP;

int formas(int d, int obj) {
    if (DP[d][obj] != -1)
        return DP[d][obj];
    int ans = 0;
    if (d == 0) {
        if (obj == 0) ans = 1;
    } else {
        for (int i = 1; i <= min(c, obj); i++) {
            ans += formas(d-1, obj - i);
            ans %= mod;
        }
    }
    return DP[d][obj] = ans;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
```

```
    int t, d, s;
    cin >> t;
    while(t--) {
        cin >> d >> c >> s;
        if (d*c < s || d > s)
            cout << "0\n";
        else {
            DP.assign(d+1, vector<int>(s+1, -1));
            cout << formas(d, s) << '\n';
        }
    }
    return 0;
}
```

Java

```
import java.io.*;
import java.util.*;

public class Solution {
    static int[][] DP = new int[51][1001];
    static int c, mod = 1000000007;

    static int ways(int d, int obj) {
        if (DP[d][obj] != -1)
            return DP[d][obj];
        int ans = 0;
        if (d == 0) {
            if (obj == 0) ans = 1;
        } else {
            for (int i = 1; i <= Math.min(c, obj); i++) {
                ans += ways(d-1, obj - i);
                ans %= mod;
            }
        }
        return DP[d][obj] = ans;
    }

    public static void main(String[] args) {
        int t, d, obj;
        Scanner s = new Scanner(System.in);
        t = s.nextInt();
        for (int caso = 0; caso < t; caso++) {
            d = s.nextInt();
            c = s.nextInt();
            obj = s.nextInt();
            if (obj < d || c*d < obj)
                System.out.println(0);
            else {
                for (int i = 0; i <= d; i++)
                    for (int j = 0; j <= obj; j++)
                        DP[i][j] = -1;
                System.out.println(ways(d, obj));
            }
        }
```

```
            s.close();
        }
    }
```

Python

```
c = 0
mod = int(1e9 + 7)
DP = []

def formas(d, obj):
    global DP
    if (DP[d][obj] != -1):
        return DP[d][obj]
    ans = 0
    if (d == 0):
        if (obj == 0):
            ans = 1
    else:
        i = 1
        while i <= min(c, obj):
            ans += formas(d-1, obj - i)
            ans %= mod
            i += 1
    DP[d][obj] = ans
    return ans

t = int(input())
for caso in range(t):
    d, c, s = map(int, input().split())
    if (c*d < s):
        print(0)
    else:
        DP = [[-1 for i in range(s+1)] for j in range(d+1)]
        print(formas(d, s))
```

Batalla en el bosque

Autora: Blanca Huergo

Teoría

- Programación dinámica
- DFS
- Máscaras de bits

Solución

Este problema lo resolveremos con programación dinámica y usando máscaras de bits. La solución de 100 puntos se construye de forma más fácil si empezamos con soluciones más lentas y vamos mejorándolas.

Lo primero es ver que el problema es un problema de permutaciones y modelar los árboles como nodos y las lianas como aristas. Es decir, tenemos que encontrar la forma óptima de ordenar los nodos de 0 a $n-1$ tal que sea un orden que se pueda hacer y que dé el mejor resultado en términos de energía posible.

Empezamos quitando todas las permutaciones que no empiezan con 0, ya que el orden final debe de ser sí (es el árbol donde empieza Ragnam). Además, una vez que el comienzo de una permutación ya es incorrecto, no nos merece la pena comprobar posibles finales. Por lo tanto, podemos construir una solución con backtracking que va podando casos ya fallidos. Con esto ya lográbamos 17 puntos. Para pasar de aquí a los 100 puntos, basta con una observación clave: el orden óptimo para visitar un subconjunto de 4 nodos se compone de un orden óptimo de 3 de sus nodos y el nodo restante. Por lo tanto, construiremos las soluciones de subconjuntos cada vez mayores de forma escalonada, aprovechando resultados anteriores. Se pueden sacar puntos parciales intermedios implementando esto de forma ineficiente, pero para llegar a los 100 hay que usar máscaras de bits para representar los subconjuntos dentro de la DP.

Código

C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int n;
vector <vector <int> >g;
vector <long long> e;
vector <int>vist;

void dfs(int u, int x) {
    if (!((1<<u)&x) || vist[u]) return;
    vist[u] = 1;
    for (int i = 0; i < g[u].size(); ++i) {
        dfs(g[u][i], x);
    }
}

bool valid(int x) {
    vist = vector <int>(n, 0);
```

```
    dfs(0, x);
    for (int i = 1; i < n; ++i) {
        if (((1<<i)&x) && !vist[i]) return false;
    }
    return true;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t;
    cin >> t;
    while (t--) {
        cin >> n;
        g = vector <vector <int> >(n);
        e = vector <long long>(n);
        for (int i = 0; i < n; ++i) cin >> e[i];
        int m;
        cin >> m;
        for (int i = 0; i < m; ++i) {
            int u, v;
            cin >> u >> v;
            g[u].push_back(v);
            g[v].push_back(u);
        }
        vector <long long>dp(1<<(n-1), -1);
        dp[0] = e[0];
        for (int i = 1; i < 1<<(n-1); ++i) {
            if (valid(2*i+1)) {
                for (int j = 0; j < n-1; ++j) {
                    if ((1<<j)&i) {
                        if (dp[i^(1<<j)] > e[j+1]) {
                            dp[i] = max(dp[i], 2LL*dp[i^(1<<j)]+e[j+1]);
                        }
                    }
                }
            }
        }
        cout << dp[(1<<(n-1))-1] << endl;
    }
}
```

Java

```
import java.io.*;
import java.util.*;

public class Bosque2 {
  static int n;
  static ArrayList<ArrayList<Integer>> g;
  static long[] e;
  static boolean[] vist;

  static void dfs(int u, int x) {
    if (((1 << u) & x) == 0 || vist[u]) return;
```

```
    vist[u] = true;
    for (int i = 0; i < g.get(u).size(); ++i) {
      dfs(g.get(u).get(i), x);
    }
  }

  static boolean valid(int x) {
    vist = new boolean[n];
    dfs(0, x);
    for (int i = 1; i < n; ++i) {
      if (((1 << i) & x) != 0 && !vist[i]) return false;
    }
    return true;
  }

  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int t = s.nextInt();
    while (t-- > 0) {
      n = s.nextInt();
      g = new ArrayList<ArrayList<Integer>>(n);
      for (int i = 0; i < n; ++i) g.add(new ArrayList<Integer>());
      e = new long[n];
      for (int i = 0; i < n; ++i) e[i] = s.nextInt();
      int m = s.nextInt();
      for (int i = 0; i < m; ++i) {
        int u = s.nextInt();
        int v = s.nextInt();
        g.get(u).add(v);
        g.get(v).add(u);
      }
      long dp[] = new long[1 << (n - 1)];
      Arrays.fill(dp, -1);
      dp[0] = e[0];
      for (int i = 1; i < 1 << (n - 1); ++i) {
        if (valid(2 * i + 1)) {
          for (int j = 0; j < n - 1; ++j) {
            if (((1 << j) & i) != 0) {
              if (dp[i ^ (1 << j)] > e[j + 1]) {
                dp[i] = Math.max(dp[i],
                    ((long) 2) * dp[i ^ (1 << j)] + e[j + 1]);
              }
            }
          }
        }
      }
      System.out.println(dp[(1 << (n - 1)) - 1]);
    }
    s.close();
  }
}
```

Python

```python
n = 0
g = []
e = []
vist = []

def dfs(u, x):
    if ((not((1<<u)&x)) or vist[u]):
        return
    vist[u] = 1
    for node in g[u]:
        dfs(node, x)

def valid(x):
    global vist
    vist = [0 for i in range(n)]
    dfs(0, x)
    for i in range(1, n):
        if (((1<<i)&x) and (not vist[i])):
            return False
    return True

t = int(input())
for caso in range(t):
    n = int(input())
    g = [[] for i in range(n)]
    e = []
    for i in range(n):
        e.append(int(input()))
    m = int(input())
    for i in range(m):
        u, v = map(int, input().split())
        g[u].append(v)
        g[v].append(u)
    dp = [-1 for i in range(1<<(n-1))]
    dp[0] = e[0]
    for i in range(1, 1<<(n-1)):
        if (valid(2*i+1)):
            for j in range(n-1):
                if (1<<j)&i:
                    if dp[i^(1<<j)] > e[j+1]:
                        dp[i] = max(dp[i], 2*dp[i^(1<<j)] + e[j+1])
    print(dp[(1<<(n-1))-1])
```