

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++

```
1  #include <cmath>
2  #include <cstdio>
3  #include <vector>
4  #include <iostream>
5  #include <algorithm>
6  using namespace std;
7
8
9  int main() {
10     int t, n, tot, cur;
11     cin >> t;
12     while(t--) {
13         cin >> n;
14         tot = 0;
15         for (int i = 0; i < n; i++) {
16             cin >> cur;
17             tot += cur;
18         }
19         if (tot % n == 0) cout << tot/n << '\n';
20         else cout << "-1\n";
21     }
```

```
22     return 0;
23 }
```

Java

```
1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String args[]) {
5          Scanner S = new Scanner(System.in);
6          int t = S.nextInt();
7          for (int i = 0; i < t; i++) {
8              int s = 0;
9              int n = S.nextInt();
10             for (int j = 0; j < n; j++)
11                 s += S.nextInt();
12             if (s % n != 0) System.out.println(-1);
13             else System.out.println(s / n);
14         }
15         S.close();
16     }
17 }
```

Python

```
1  t = int(input())
2  for i in range(t):
3      s = 0
4      n = int(input())
5      for g in input().split():
6          s = s + int(g)
7      if s % n != 0:
8          print(-1)
9      else:
10         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++

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int main() {
7      int t, a;
8      cin >> t;
9      while (t--) {
10         cin >> a;
11         int n = 2 + a; // El numero de jugadores son Max, Izan y los amigos a los
            ↳ que invitan
12         vector<int> c(n, 1); // Posiciones en el tablero: Creamos un vector para
            ↳ los jugadores, todos empiezan en la casilla 1
13         vector<int> p(n, 0); // Partidas ganadas por cada jugador: Creamos un
            ↳ vector para los jugadores, todos empiezan con 0 partidas ganadas
14
15         int j = 0, d;
16         while (true) { // Vamos leyendo mientras haya dados apuntados
17             cin >> d;
18             if (d == -1)
19                 break;
20             int casilla = c[j] + d;
21
22             if (casilla == 26 || casilla == 53) {
23                 int d2;
24                 cin >> d2;
25                 casilla += d2;
26             } // Casillas Dados
27             if (casilla > 63)
28                 casilla = 63 - (casilla - 63); // Se pasa del Jardin de la Oca
29
30             // Ahora ya sabemos la casilla final de este dado, analizamos si es
            ↳ una casilla especial que lleva a otra casilla destino
```

```

31     else if (casilla == 42)
32         casilla = 30; // Casillas Laberinto
33     else if (casilla == 58)
34         casilla = 1; // Casillas Calavera
35     else if (casilla == 5)
36         casilla = 9; // Casillas Oca
37     else if (casilla == 9)
38         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
39     else if (casilla == 14)
40         casilla = 18;
41     else if (casilla == 18)
42         casilla = 23;
43     else if (casilla == 23)
44         casilla = 27;
45     else if (casilla == 27)
46         casilla = 32;
47     else if (casilla == 32)
48         casilla = 36;
49     else if (casilla == 36)
50         casilla = 41;
51     else if (casilla == 41)
52         casilla = 45;
53     else if (casilla == 45)
54         casilla = 50;
55     else if (casilla == 50)
56         casilla = 54;
57     else if (casilla == 54)
58         casilla = 59;
59
60     if (casilla == 63) { // Casilla Jardin de la Oca: es el ganador y se
           ↪ comienza una nueva partida
61         p[j]++; // Incrementar el contador del ganador
62         for (int i = 0; i < (int) c.size(); i++)
63             c[i] = 1; // Todos vuelven a la casilla de salida, Se puede
               ↪ usar tambien fill(c.begin(), c.end(), 1);
64     } else { // Guardamos la casilla en la que ha acabado este turno este
           ↪ jugador
65         c[j] = casilla;
66     }
67     j = (j + 1) % n; // Pasamos al siguiente jugador
68 }
69
70 // Buscamos quien es el ganador de la noche
71 int p_max = 0, j_max = -1; // Inicialmente todos tienen 0 puntos y por lo
           ↪ tanto estan empatados
72 for (int i = 0; i < (int) p.size(); i++) {
73     if (p[i] > p_max) { // Hemos encontrado un nuevo maximo
74         p_max = p[i];
75         j_max = i;
76     } else if (p[i] == p_max) {
77         j_max = -1; // El maximo que hemos encontrado hasta ahora tiene
           ↪ empate
78     }

```

```

79     }
80     cout << (j_max + 1) << '\n';
81 }
82 }

```

Java

```

1  import java.io.*;
2  import java.util.*;
3
4  public class Solution {
5
6      public static void main(String[] args) {
7          int t, a, n;
8          Scanner s = new Scanner(System.in);
9          t = s.nextInt();
10         for (int caso = 0; caso < t; caso++) {
11             a = s.nextInt();
12             n = 2+a; // El numero de jugadores son Max, Izan y los amigos a los
13                 ↪ que invitan
14             Integer[] c = new Integer[n]; // Posiciones en el tablero: Creamos un
15                 ↪ array para los jugadores, todos empiezan en la casilla 1
16             Arrays.fill(c, new Integer(1));
17             Integer[] p = new Integer[n]; // Partidas ganadas por cada jugador:
18                 ↪ Creamos un array para los jugadores, todos empiezan con 0 partidas
19                 ↪ ganadas
20             Arrays.fill(p, new Integer(0));
21             int j = 0, d;
22             while(true) {
23                 // Vamos leyendo mientras haya dados apuntados
24                 d = s.nextInt();
25                 if (d == -1)
26                     break;
27                 int casilla = c[j] + d;
28                 if (casilla == 26 || casilla == 53) {
29                     int d2;
30                     d2 = s.nextInt();
31                     casilla += d2;
32                 } // Casillas Dados
33                 if (casilla > 63)
34                     casilla = 63 - (casilla - 63); // Se pasa del Jardin de la Oca
35                 // Ahora ya sabemos la casilla final de este dado, analizamos si
36                 ↪ es una casilla especial que lleva a otra casilla destino
37                 else if (casilla == 42)
38                     casilla = 30; // Casillas Laberinto
39                 else if (casilla == 58)
40                     casilla = 1; // Casillas Calavera
41                 else if (casilla == 5)
42                     casilla = 9; // Casillas Oca
43                 else if (casilla == 9)
44                     casilla = 14; // Es muy importante usar "else if", si se pone
45                 ↪ "if" al entrar en una casilla Oca entra en todas las
46                 ↪ siguientes casillas Oca
47                 else if (casilla == 14)
48                     casilla = 18;

```

```

42     else if (casilla == 18)
43         casilla = 23;
44     else if (casilla == 23)
45         casilla = 27;
46     else if (casilla == 27)
47         casilla = 32;
48     else if (casilla == 32)
49         casilla = 36;
50     else if (casilla == 36)
51         casilla = 41;
52     else if (casilla == 41)
53         casilla = 45;
54     else if (casilla == 45)
55         casilla = 50;
56     else if (casilla == 50)
57         casilla = 54;
58     else if (casilla == 54)
59         casilla = 59;
60
61     if (casilla == 63) { // Casilla Jardin de la Oca: es el ganador y
62         ↪ se comienza una nueva partida
63         p[j]++; // Incrementar el contador del ganador
64         for (int i = 0; i < n; i++)
65             c[i] = 1; // Todos vuelven a la casilla de salida
66     } else { // Guardamos la casilla en la que ha acabado este turno
67         ↪ este jugador
68         c[j] = casilla;
69     }
70     j = (j + 1) % n; // Pasamos al siguiente jugador
71 }
72
73 // Buscamos quien es el ganador de la noche
74 int p_max = 0, j_max = -1; // Inicialmente todos tienen 0 puntos y por
75 ↪ lo tanto estan empatados
76 for (int i = 0; i < n; i++) {
77     if (p[i] > p_max) { // Hemos encontrado un nuevo maximo
78         p_max = p[i];
79         j_max = i;
80     } else if (p[i] == p_max) {
81         j_max = -1; // El maximo que hemos encontrado hasta ahora
82         ↪ tiene empate
83     }
84 }
85 System.out.println(j_max + 1);
86 }
87 s.close();
88 }
89 }

```

Python

```

1 t = int(input())
2 for caso in range(t):
3     a = int(input())
4     partidas = 0

```

```

5 n = 2+a # El numero de jugadores son Max, Izan y los amigos a los que invitan
6 c = [1 for i in range(n)] # Posiciones en el tablero: Creamos un array para
  ↳ los jugadores, todos empiezan en la casilla 1
7 p = [0 for i in range(n)] # Partidas ganadas por cada jugador: Creamos un
  ↳ array para los jugadores, todos empiezan con 0 partidas ganadas
8 j = 0
9 while True:
10     # Vamos leyendo mientras haya dados apuntados
11     d = int(input())
12     if (d == -1):
13         break
14     casilla = c[j] + d
15     if (casilla == 26 or casilla == 53):
16         d2 = int(input())
17         casilla += d2
18         # Casillas Dados
19     if (casilla > 63):
20         casilla = 63 - (casilla - 63) # Se pasa del Jardin de la Oca
21
22     # Ahora ya sabemos la casilla final de este dado, analizamos si es una
23     ↳ casilla especial que lleva a otra casilla destino
24     elif (casilla == 42):
25         casilla = 30; # Casillas Laberinto
26     elif (casilla == 58):
27         casilla = 1; # Casillas Calavera
28     elif (casilla == 5):
29         casilla = 9; # Casillas Oca
30     elif (casilla == 9):
31         casilla = 14; # Es muy importante usar "elif", si se pone "if" al
32         ↳ entrar en una casilla Oca entra en todas las siguientes casillas
33         ↳ Oca
34     elif (casilla == 14):
35         casilla = 18;
36     elif (casilla == 18):
37         casilla = 23;
38     elif (casilla == 23):
39         casilla = 27;
40     elif (casilla == 27):
41         casilla = 32
42     elif (casilla == 32):
43         casilla = 36
44     elif (casilla == 36):
45         casilla = 41
46     elif (casilla == 41):
47         casilla = 45;
48     elif (casilla == 45):
49         casilla = 50;
50     elif (casilla == 50):
51         casilla = 54;
52     elif (casilla == 54):
53         casilla = 59;
54
55     if (casilla == 63): # Casilla Jardin de la Oca: es el ganador y se
56     ↳ comienza una nueva partida
57         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
64     ↪ empatados
65     for i in range(len(p)):
66         if (p[i] > p_max): # Hemos encontrado un nuevo maximo
67             p_max = p[i]
68             j_max = i
69         elif (p[i] == p_max):
70             j_max = -1 # El maximo que hemos encontrado hasta ahora tiene
71             ↪ empate
72     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++

```
1  #include <iostream>
2  #include <vector>
3  #include <climits>
4  #include <algorithm>
5  using namespace std;
6
7  bool esPosible(long long int cDePrueba, long long int objetivo, vector<int> &
   ↪ salarios) {
8      long long int tot = 0;
9      for (int sal: salarios)
10         tot += min((long long int) sal, cDePrueba);
11     return tot <= objetivo;
12 }
13
14 long long int maximoC(long long int objetivo, vector<int> & salarios) {
15     long long int mn = *min_element(salarios.begin(), salarios.end()), mx =
   ↪ *max_element(salarios.begin(), salarios.end());
16     long long int lo = 0, hi = mx, mid, k = -1;
17     while(lo <= hi) {
18         mid = lo + (hi-lo)/2;
19         if (esPosible(mid, objetivo, salarios)) {
20             lo = mid+1;
21             k = mid;
22         } else hi = mid-1;
23     }
24     if (k < mn)
25         return -1;
26     return k;
27 }
28
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(NULL);
32     int t, n;
33     long long int T;
```

```

34     cin >> t;
35     while(t--> 0) {
36         cin >> n;
37         vector<int> salarios(n);
38         for (int i = 0; i < n; i++)
39             cin >> salarios[i];
40         cin >> T;
41         cout << maximoC(T, salarios) << '\n';
42     }
43     return 0;
44 }

```

Java

```

1  import java.io.*;
2  import java.util.*;
3
4  public class Solution {
5      static ArrayList<Integer> salarios;
6      static boolean esPosible(long cDePrueba, long objetivo) {
7          long tot = 0;
8          for (int sal: salarios)
9              tot += Math.min(sal, cDePrueba);
10         return tot <= objetivo;
11     }
12
13     static long maximoC(long objetivo) {
14         long mn = Collections.min(salarios), mx = Collections.max(salarios);
15         long lo = 1, hi = mx, mid, k = -1;
16         while(lo <= hi) {
17             mid = lo + (hi-lo)/2;
18             if (esPosible(mid, objetivo)) {
19                 lo = mid+1;
20                 k = mid;
21             } else hi = mid-1;
22         }
23         if (k < mn)
24             return -1;
25         return k;
26     }
27
28     public static void main(String[] args) {
29         int t, n;
30         long T;
31         Scanner s = new Scanner(System.in);
32         t = s.nextInt();
33         for (int caso = 0; caso < t; caso++) {
34             n = s.nextInt();
35             salarios = new ArrayList<Integer>();
36             for (int i = 0; i < n; i++)
37                 salarios.add(s.nextInt());
38             T = s.nextLong();
39             System.out.println(maximoC(T));
40         }
41         s.close();

```

```
42     }
43 }
```

Python

```
1  salarios = []
2  def esPosible(cDePrueba, objetivo):
3      tot = 0
4      for sal in salarios:
5          tot += min(sal, cDePrueba)
6      return tot <= objetivo
7
8  def maximoC(objetivo):
9      mn = min(salarios)
10     mx = max(salarios)
11     lo = 0
12     hi = mx
13     k = -1
14     while(lo <= hi):
15         mid = lo + (hi-lo)//2
16         if esPosible(mid, objetivo):
17             lo = mid+1
18             k = mid
19         else:
20             hi = mid-1
21     if (k < mn):
22         return -1
23     return k
24
25 t = int(input())
26 for caso in range(t):
27     n = int(input())
28     salarios = list(map(int, input().split()))
29     T = int(input())
30     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++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6
7  typedef long long int ll;
8  typedef vector <int> vi;
9  typedef vector <vi> vvi;
10 typedef pair <int, int> ii;
11 typedef vector <ii> vii;
12
13 vvi G;
14 vi visited, dp;
15
16 const int alfa = 'z' - 'a' + 1;
17
18 int cycle(int first, int v, int d){
19     visited[v] = 1;
20
21     for (int u: G[v]) {
22         if (visited[u]) {
23             if (u == first) dp[v] = max(dp[v], 1);
24         }
25         else dp[v] = max(dp[v], 1 + cycle(first, u, d));
26     }
27
28     return dp[v];
29 }
30
```

```

31 int find_longest_cycle(){
32     int ans = -1;
33     for (int i = 0; i < alfa; ++i){
34         visited = vi(alfa, 0);
35         dp = vi(alfa, -1e9);
36         ans = max(ans, cycle(i, i, 0));
37     }
38 }
39
40 return ans;
41 }
42
43
44 int main(){
45     ios::sync_with_stdio(false);
46     cin.tie(null);
47
48     int t;
49     cin >> t;
50
51     for (int w = 0; w < t; ++w){
52         vvi M(alfa, vi(alfa, 0));
53         int n;
54         cin >> n;
55
56         for (int i = 0; i < n; ++i){
57             string s;
58             cin >> s;
59             M[s[0] - 'A'][s[1] - 'A'] = 1;
60         }
61
62         G = vvi(alfa);
63
64         for (int i = 0; i < alfa; ++i){
65             for (int j = 0; j < alfa; ++j){
66                 if (M[i][j]) G[i].push_back(j);
67             }
68         }
69
70         cout << find_longest_cycle() << '\n';
71     }
72 }

```

Java

```

1  import java.io.*;
2  import java.util.*;
3
4  public class Solution {
5      static ArrayList<ArrayList<Integer>> G;
6      static int alfa = 'z' - 'a' + 1;
7      static int[] visited = new int[alfa], dp = new int[alfa];
8      public static int cycle(int first, int v, int d) {
9          int ans = -1;
10         visited[v] = 1;

```

```

11     for (int u: G.get(v)) {
12         if (visited[u] == 1) {
13             if (u == first)
14                 dp[v] = Math.max(dp[v], 1);
15             } else dp[v] = Math.max(dp[v], 1 + cycle(first, u, d));
16         }
17     }
18     return dp[v];
19 }
20
21 public static int find_longest_cycle() {
22     int ans = -1;
23     for (int i = 0; i < alfa; i++) {
24         Arrays.fill(visited, 0);
25         Arrays.fill(dp, -1000000000);
26         ans = Math.max(ans, cycle(i, i, 0));
27     }
28     return ans;
29 }
30
31 public static void main(String[] args) {
32     Scanner s = new Scanner(System.in);
33     int t = s.nextInt();
34     for (int caso = 0; caso < t; caso++) {
35         int[][] M = new int[alfa][alfa];
36         for (int i = 0; i < alfa; i++)
37             for (int j = 0; j < alfa; j++)
38                 M[i][j] = 0;
39         int n = s.nextInt();
40         for (int i = 0; i < n; i++) {
41             String str = s.next();
42             M[str.charAt(0) - 'A'][str.charAt(1) - 'A'] = 1;
43         }
44         G = new ArrayList<ArrayList<Integer>>(alfa);
45         for (int i = 0; i < alfa; i++) {
46             G.add(new ArrayList<Integer>());
47             for (int j = 0; j < alfa; j++) {
48                 if (M[i][j] == 1)
49                     G.get(i).add(j);
50             }
51         }
52         System.out.println(find_longest_cycle());
53     }
54     s.close();
55 }

```

Python

```

1 G = []
2 dp = []
3 visited = []
4
5 alfa = ord('z') - ord('a') + 1
6
7 def cycle(first, v, d):

```

```

8     global visited, G, dp
9     ans = -1
10    visited[v] = 1
11
12    for u in G[v]:
13        if (visited[u]):
14            if (u == first):
15                dp[v] = max(dp[v], 1)
16            else:
17                dp[v] = max(dp[v], 1 + cycle(first, u, d))
18
19    return dp[v]
20
21 def find_longest_cycle():
22     global visited, dp
23     ans = -1
24     for i in range(alfa):
25         visited = [0 for j in range(alfa)]
26         dp = [-1e9 for j in range(alfa)]
27         ans = max(ans, cycle(i, i, 0))
28     return ans
29
30 t = int(input())
31 for w in range(t):
32     M = [[0 for i in range(alfa)] for j in range(alfa)]
33     n = int(input())
34     S = input().split()
35     for i in range(n):
36         M[ord(S[i][0]) - ord('A')][ord(S[i][1]) - ord('A')] = 1
37
38     G = [[] for i in range(alfa)]
39
40     for i in range(alfa):
41         for j in range(alfa):
42             if (M[i][j]):
43                 G[i].append(j)
44
45     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:

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

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++

```
1  #include <cmath>
2  #include <cstdio>
3  #include <vector>
4  #include <iostream>
5  #include <algorithm>
6  using namespace std;
7
8  int c, mod = 1e9 + 7;
9
10 vector<vector<int>> DP;
11
12 int formas(int d, int obj) {
13     if (DP[d][obj] != -1)
14         return DP[d][obj];
15     int ans = 0;
16     if (d == 0) {
17         if (obj == 0) ans = 1;
18     } else {
19         for (int i = 1; i <= min(c, obj); i++) {
20             ans += formas(d-1, obj - i);
21             ans %= mod;
22         }
23     }
24     return DP[d][obj] = ans;
25 }
26
27 int main() {
28     ios::sync_with_stdio(false);
29     cin.tie(NULL);
```



```

30     int t, d, s;
31     cin >> t;
32     while(t--) {
33         cin >> d >> c >> s;
34         if (d*c < s || d > s)
35             cout << "0\n";
36         else {
37             DP.assign(d+1, vector<int>(s+1, -1));
38             cout << formas(d, s) << '\n';
39         }
40     }
41     return 0;
42 }

```

Java

```

1  import java.io.*;
2  import java.util.*;
3
4  public class Solution {
5      static int[][] DP = new int[51][1001];
6      static int c, mod = 1000000007;
7
8      static int ways(int d, int obj) {
9          if (DP[d][obj] != -1)
10             return DP[d][obj];
11         int ans = 0;
12         if (d == 0) {
13             if (obj == 0) ans = 1;
14         } else {
15             for (int i = 1; i <= Math.min(c, obj); i++) {
16                 ans += ways(d-1, obj - i);
17                 ans %= mod;
18             }
19         }
20         return DP[d][obj] = ans;
21     }
22
23     public static void main(String[] args) {
24         int t, d, obj;
25         Scanner s = new Scanner(System.in);
26         t = s.nextInt();
27         for (int caso = 0; caso < t; caso++) {
28             d = s.nextInt();
29             c = s.nextInt();
30             obj = s.nextInt();
31             if (obj < d || c*d < obj)
32                 System.out.println(0);
33             else {
34                 for (int i = 0; i <= d; i++)
35                     for (int j = 0; j <= obj; j++)
36                         DP[i][j] = -1;
37                 System.out.println(ways(d, obj));
38             }
39         }

```

```
40     s.close();
41 }
42 }
```

Python

```
1  c = 0
2  mod = int(1e9 + 7)
3  DP = []
4
5  def formas(d, obj):
6      global DP
7      if (DP[d][obj] != -1):
8          return DP[d][obj]
9      ans = 0
10     if (d == 0):
11         if (obj == 0):
12             ans = 1
13     else:
14         i = 1
15         while i <= min(c, obj):
16             ans += formas(d-1, obj - i)
17             ans %= mod
18             i += 1
19     DP[d][obj] = ans
20     return ans
21
22 t = int(input())
23 for caso in range(t):
24     d, c, s = map(int, input().split())
25     if (c*d < s):
26         print(0)
27     else:
28         DP = [[-1 for i in range(s+1)] for j in range(d+1)]
29         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++

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  int n;
6  vector <vector <int> >g;
7  vector <long long> e;
8  vector <int>vist;
9
10 void dfs(int u, int x) {
11     if (!(1<<u)&x || vist[u]) return;
12     vist[u] = 1;
13     for (int i = 0; i < g[u].size(); ++i) {
14         dfs(g[u][i], x);
15     }
16 }
17
18 bool valid(int x) {
19     vist = vector <int>(n, 0);
```

```

20     dfs(0, x);
21     for (int i = 1; i < n; ++i) {
22         if (((1<<i)&x) && !vist[i]) return false;
23     }
24     return true;
25 }
26
27 int main() {
28     ios::sync_with_stdio(0);
29     cin.tie(0);
30     int t;
31     cin >> t;
32     while (t--) {
33         cin >> n;
34         g = vector <vector <int> >(n);
35         e = vector <long long>(n);
36         for (int i = 0; i < n; ++i) cin >> e[i];
37         int m;
38         cin >> m;
39         for (int i = 0; i < m; ++i) {
40             int u, v;
41             cin >> u >> v;
42             g[u].push_back(v);
43             g[v].push_back(u);
44         }
45         vector <long long>dp(1<<(n-1), -1);
46         dp[0] = e[0];
47         for (int i = 1; i < 1<<(n-1); ++i) {
48             if (valid(2*i+1)) {
49                 for (int j = 0; j < n-1; ++j) {
50                     if ((1<<j)&i) {
51                         if (dp[i^(1<<j)] > e[j+1]) {
52                             dp[i] = max(dp[i], 2LL*dp[i^(1<<j)]+e[j+1]);
53                         }
54                     }
55                 }
56             }
57         }
58         cout << dp[(1<<(n-1))-1] << endl;
59     }
60 }

```

Java

```

1  import java.io.*;
2  import java.util.*;
3
4  public class Bosque2 {
5      static int n;
6      static ArrayList<ArrayList<Integer>> g;
7      static long[] e;
8      static boolean[] vist;
9
10     static void dfs(int u, int x) {
11         if (((1 << u) & x) == 0 || vist[u]) return;

```

```

12     vist[u] = true;
13     for (int i = 0; i < g.get(u).size(); ++i) {
14         dfs(g.get(u).get(i), x);
15     }
16 }
17
18 static boolean valid(int x) {
19     vist = new boolean[n];
20     dfs(0, x);
21     for (int i = 1; i < n; ++i) {
22         if (((1 << i) & x) != 0 && !vist[i]) return false;
23     }
24     return true;
25 }
26
27 public static void main(String[] args) {
28     Scanner s = new Scanner(System.in);
29     int t = s.nextInt();
30     while (t-- > 0) {
31         n = s.nextInt();
32         g = new ArrayList<ArrayList<Integer>>(n);
33         for (int i = 0; i < n; ++i) g.add(new ArrayList<Integer>());
34         e = new long[n];
35         for (int i = 0; i < n; ++i) e[i] = s.nextInt();
36         int m = s.nextInt();
37         for (int i = 0; i < m; ++i) {
38             int u = s.nextInt();
39             int v = s.nextInt();
40             g.get(u).add(v);
41             g.get(v).add(u);
42         }
43         long dp[] = new long[1 << (n - 1)];
44         Arrays.fill(dp, -1);
45         dp[0] = e[0];
46         for (int i = 1; i < 1 << (n - 1); ++i) {
47             if (valid(2 * i + 1)) {
48                 for (int j = 0; j < n - 1; ++j) {
49                     if (((1 << j) & i) != 0) {
50                         if (dp[i ^ (1 << j)] > e[j + 1]) {
51                             dp[i] = Math.max(dp[i],
52                                 ((long) 2) * dp[i ^ (1 << j)] + e[j + 1]);
53                         }
54                     }
55                 }
56             }
57         }
58         System.out.println(dp[(1 << (n - 1)) - 1]);
59     }
60     s.close();
61 }
62 }

```

Python

```
1  n = 0
2  g = []
3  e = []
4  vist = []
5
6  def dfs(u, x):
7      if ((not((1<<u)&x)) or vist[u]):
8          return
9      vist[u] = 1
10     for node in g[u]:
11         dfs(node, x)
12
13     def valid(x):
14         global vist
15         vist = [0 for i in range(n)]
16         dfs(0, x)
17         for i in range(1, n):
18             if (((1<<i)&x) and (not vist[i])):
19                 return False
20         return True
21
22     t = int(input())
23     for caso in range(t):
24         n = int(input())
25         g = [[] for i in range(n)]
26         e = []
27         for i in range(n):
28             e.append(int(input()))
29         m = int(input())
30         for i in range(m):
31             u, v = map(int, input().split())
32             g[u].append(v)
33             g[v].append(u)
34         dp = [-1 for i in range(1<<(n-1))]
35         dp[0] = e[0]
36         for i in range(1, 1<<(n-1)):
37             if (valid(2*i+1)):
38                 for j in range(n-1):
39                     if (1<<j)&i:
40                         if dp[i^(1<<j)] > e[j+1]:
41                             dp[i] = max(dp[i], 2*dp[i^(1<<j)] + e[j+1])
42         print(dp[(1<<(n-1))-1])
```