Distracciones

En las anteriores reuniones de la Organización Independiente de Epizootias, también conocida como la Organización Mundial de Sanidad Animal, se reunían todos en Barcelona para discutir sobre los temas más relevantes. Pero cada año ocurrían varios problemas.

Los asistentes en la reunion nunca consiguen concentrarse en la reunión, se distraen por cualquier tontería. En la última reunión por ejemplo se entretuvieron con un papel en la pared con las siguientes líneas.

1
11
21
1211
111221
312211

Estuvieron mucho tiempo intentanto descifrar este duro problema hasta que les explicaron la solución. Cada línea simplemente viene generada leyendo en voz alta la línea anterior. Por ejemplo, la segunda línea es "un uno", la tercera es "dos unos" (y no un once) etc.

Para que no vuelva a ocurrir este problema, la organización te pide que hagas un programa que diga el i-ésimo número de la secuencia, $a_0 = 1$.

Input Format

La entrada consiste en un solo caso.

El caso solo contiene un entero N.

Constraints

$1 \leq N \leq 53$

Output Format

La salida consiste una sola línea, el valor N-ésimo de la secuencia.

Sample Input 0

```
1
```

Sample Output 0

```
11
```

Sample Input 1

```
2
```

Sample Output 1

```
21
```

Sample Input 2

```
7
```

Sample Output 2

```
1113213211
```

Solución

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

using namespace std;

typedef vector<int> vi;

string tos(int n) {
  string a;
  while (n) {
    a.push_back(n%10 + '0');
    n /= 10;
  }
  reverse(a.begin(), a.end());
  return a;
}

string pas(string a) {
  int i = 0;
  string b;
  while (i < (int)a.size()) {
    char s = a[i];
    int cont = 0;
    while (i < (int)a.size() and s == a[i]) {
      ++cont;
      ++i;
    }
    b += tos(cont) + s;
  }
  return b;
}


int main() {
  int n;
  cin >> n;
  string a = "1";
  while (n--)
    a = pas(a);
  cout << a << endl;
}
```

// Autor de la solución: Cesc Folch (Miembro del comité organizador)

Múltiplos de 11

Como todo el mundo sabe, el número favorito de Timur es el 11. Por eso, para su cumpleaños pidió como regalo una string con el número 11. Sin embargo, cuando abrió el regalo, encontró una string *s*, que no era la que él quería. Totalmente decepcionado, se puso a llorar y sus padres, queriendo consolarlo, le dijeron que su string tenía muchas substrings que eran múltiplos de 11. Sin embargo, ellos no las saben contar, y por eso, necesitan tu ayuda. Una substring es un grupo de carácteres consecutivos de una string. Por ejemplo, 102 tiene como substrings $1, 0, 2, 10, 02$ y 102, pero no 12.

Notad que las substrings pueden empezar por 0, y por ejemplo la string 02 correspondería al número 2. Sin embargo, se os asegura que el primer dígito de la string *s* que se os da, nunca es cero.

Input Format

La entrada consiste de un entero *t* indicando el número de casos. Le siguen *t* casos, cada uno formado por 2 líneas. En la primera, hay un número *n* que indica la longitud de la string, y en la segunda hay la string *s* que representa el número de Timur.

Constraints

$n \leq 12$ (12 puntos)

$n \leq 300$ (14 puntos)

$n \leq 2000$ (21 puntos)

$n \leq 100000$ y todos los dígitos del número son iguales (20 puntos)

$n \leq 100000$ y no hay restricciones adicionales. (33 puntos)

Output Format

Imprimid *t* líneas, una por cada caso indicando el número de substrings de *s* que son múltiplos de 11.

Sample Input 0

```
2
3
121
4
1000
```

Sample Output 0

```
1
6
```

Explanation 0

En el primer caso, la string completa 121 es la única substring múltiplo de 11. En el segundo, 0 que aparece 3 veces, 00 que aparece 2 y 000 que aparece una única vez son las substrings múltiplo de 11, así que la respuesta es $1 + 2 + 3 = 6$.

Solución

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

using namespace std;

typedef long long ll;
typedef vector<ll> vi;
typedef vector<vi> vvi;

int main() {
  int casos;
  cin >> casos;
  while (casos--) {
    int n;
    cin >> n;
    vi V(n);
    for (int i = 0; i < n; ++i) {
      char a;
      cin >> a;
      V[i] = a - '0';
    }
    vvi D(n + 1, vi(11,0));
    ll res = 0;
    for (int i = 0; i < n; ++i) {
      D[i][0]++;
      res += D[i][V[i]];
      if (i > 0) {
        int dist = (11 + V[i] - V[i-1])%11;
        for (int j = 0; j < 11; ++j) {
          int j1 = (j + dist)%11;
          D[i+1][j1] = D[i-1][j];
        }
        res += D[i+1][0];
      }
    }
    cout << res << endl;
  }
}
```

// Autor de la solución: Cesc Folch (Miembro del comité organizador)

De farolas y ladrones

La noche se acerca y sus peligros acechan. Como es bien sabido, los más ilustres prohombres se convierten en verdaderos truhanes al sonar las doce. Harto de ver como tus vecinos se roban unos a otros por la calle con total impunidad, decides instalar farolas a lo largo de tu calle de manera que estén siempre iluminados y así por lo menos sientan algo de vergüenza al cometer sus fechorías. Sin embargo, no quieres gastar electricidad inútilmente, con lo que decides hacer un programa que, sabiendo donde están tus vecinos en un instante y qué radio ilumina cada farola, calcule el menor número posible de farolas a encender para que estén todos iluminados, o que te informe si no es posible hacerlo. Un vecino se considera iluminado si está dentro de un intervalo cerrado con centro en una farola y con el radio de esta.

Input Format

La entrada consistirá en varios casos. La primera línea contendrá un entero t, el número de casos a resolver.

Para cada caso:

-La primera línea contendrá dos enteros n, m, el número de vecinos y el número de farolas.

-La siguiente línea contiene n enteros x_i con las posiciones de tus vecinos.

-Las últimas m líneas contienen una pareja de números enteros y_i, r_i la posición y el radio de cada farola.

Constraints

$0 \leq x_i, y_i, r_i \leq 10^9, 1 \leq t \leq 100$

- **30 puntos** $1 \leq n \leq 100, 1 \leq m \leq 10$
- **30 puntos** $1 \leq n \leq 1000, 1 \leq m \leq 100$
- **40 puntos** $1 \leq n, m \leq 10^5$

Output Format

Para cado caso escribe en una línea el número mínimo de farolas que hace falta encender para iluminar a todos los vecinos o "imposible" si no se puede resolver.

Sample Input 0

```
3
1 2
5
3 2
7 2
5 5
7 10 3 4 4
3 3
7 1
8 2
3 2
2 4
1 1
0
3 2
```

Sample Output 0

```
1
2
imposible
```

Solución

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

using namespace std;

typedef pair<int,int> pi;
typedef vector<pi> vpi;
typedef vector<int> vi;

int main() {
  int casos;
  cin >> casos;
  while (casos--) {
    int n, m;
    cin >> n >> m;
    vi P(n);
    vpi F(m);
    for (int i = 0; i < n; ++i)
      cin >> P[i];
    sort(P.begin(),P.end());
    for (int i = 0; i < m; ++i) {
      int y,p;
      cin >> y >> p;
      F[i] = pi(y - p, y + p);
    }
    sort(F.begin(), F.end());
    int fin = -1;
    int res = 0;
    int j = 0;
    int i = 0;
    while (i < n) {
      fin = -1;
      while (j < m and F[j].first <= P[i]) {
        fin = max(fin, F[j].second);
        ++j;
      }
      if (fin < P[i]) {
        res = -1;
        break;
      }
      ++res;
      while (i < n and P[i] <= fin) ++i;
    }
    if (res >= 0) cout << res << endl;
    else cout << "imposible" << endl;
  }
}
```

// Autor de la solución: Cesc Folch (Miembro del comité organizador)

Caza de brujas

Corre el año 1692 y la gente de Salem está sedienta de sangre. Las acusaciones de brujería vuelan y el populacho está ansioso para ver las cabezas de las brujas en bandeja. El alcalde, queriendo apaciguar las masas minimizando el baño de sangre, ha decidido consultar las M amistades y enemistades del pueblo de N habitantes: si dos personas son amigas, entonces ambas son brujas o ninguna es bruja, mientras que si son enemigas, exactamente una de ellas es bruja. De todas las posibles combinaciones de brujas, el alcalde elegirá aquella que contenga el menor número de brujas. Es posible que la información del alcalde sea incorrecta y no exista combinación alguna. ¿Puedes calcular cuántas brujas van a ser ejecutadas?

Input Format

La entrada consiste en un entero t, seguido de t casos de prueba. Cada caso empieza con los enteros N, M, el número de habitantes y el número de relaciones entre los habitantes. A continuación, vienen M líneas, cada una con tres enteros r_i, a_i, b_i. Si $r_i = 1$ quiere decir que los habitantes a_i y b_i son amigos, y si $r_i = 2$ entonces son enemigos.

Constraints

Para todos los casos de prueba se cumple $t \leq 50, 1 \leq N, 0 \leq M \leq \frac{N(N-1)}{2}, r_i \in \{1, 2\}$, $0 \leq a_i, b_i < N, a_i \neq b_i$ y para todo par de líneas i, j, se cumple $a_i \neq a_j$ o bien $b_i \neq b_j$.

16 puntos: $N \leq 16, M \leq 30$

17 puntos: $N \leq 3 \cdot 10^4, M \leq 5 \cdot 10^4, r_i = 2$ para todo i

17 puntos: $N \leq 200, M \leq 300$

50 puntos: $N \leq 3 \cdot 10^4, M \leq 5 \cdot 10^4$

Output Format

Una línea por caso con el número mínimo de brujas, o -1 si no existe una configuración válida.

Sample Input 0

```
3
6 5
2 0 1
2 1 2
1 1 3
2 3 4
2 4 5
5 3
1 0 1
2 2 3
2 3 4
3 3
1 0 1
1 1 2
2 0 2
```

Sample Output 0

```
3
1
-1
```

Explanation 0

En el primer caso, las brujas pueden ser 0, 2 y 4 o bien 1, 3 y 5. Por lo tanto, 3 brujas.

En el segundo caso, basta con que 3 sea bruja. Hacer que 3 no sea bruja o que 0 y 1 sean brujas no es óptimo.

En el tercer caso, no se pueden satisfacer las condiciones del problema.

Solución

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

using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;

vvi G[2];
vi D[2];

int a[2];

void dfs(int k, int x) {
  if (D[k][x]) return;
  D[k][x] = 1;
  ++a[k];
  int k1 = 1 - k;
  for (int i = 0; i < (int)G[0][x].size(); ++i) {
    int y = G[0][x][i];
    dfs(k, y);
  }
  for (int i = 0; i < (int)G[1][x].size(); ++i) {
    int y = G[1][x][i];
    dfs(k1, y);
  }
}

int main() {
  int casos;
  cin >> casos;
  while (casos--) {
    int n, m;
    cin >> n >> m;
    G[0] = G[1] = vvi (n);
    D[0] = D[1] = vi (n,0);
    for (int i = 0; i < m; ++i) {
      int k,x,y;
      cin >> k >> x >> y;
      --k;
      G[k][x].push_back(y);
      G[k][y].push_back(x);
    }
    int res = 0;
    for (int i = 0; i < n; ++i) {
      if (D[0][i] or D[1][i]) continue;
      a[0] = a[1] = 0;
      dfs(0, i);
      res += min(a[0], a[1]);
    }
    bool pot = true;
    for (int i = 0; i < n; ++i) {
```

```
      if (D[0][i] and D[1][i])
        pot = false;
    }
    if (pot) cout << res << endl;
    else cout << -1 << endl;
  }
}
```

// Autor de la solución: Cesc Folch (Miembro del comité organizador)

Ninjas

Seguramente has oído hablar de los ninjas japoneses. Lo que probablemente no sabías es que también hay ninjas españoles, pero jamás has oído hablar de ellos porque son mucho mejores que los japoneses. De hecho, en toda la historia de España solo han sido pillados una vez, intentando infiltrar el palacio real. He aquí las crónicas.

El palacio real consiste en n habitaciones numeradas del 0 al $n-1$, conectadas entre sí por $n-1$ pasillos, todos de igual longitud, de tal forma que desde cualquier habitación se puede llegar a cualquier otra sin salir del palacio. También cuenta con múltiples entradas: en cada habitación **que solo tiene un pasillo** hay una puerta que permite entrar y salir del palacio a través de ella.

Un grupo muy numeroso de ninjas está dispuesto a entrar al palacio para plantar una emboscada al rey. Justo cuando están a punto de entrar, un grupo aún más numeroso de guardias reales los divisa. En este momento, los ninjas rápidamente eligen una puerta y entran todos por allí, y los guardias, que han visto por dónde entran los ninjas, pero no tienen tiempo de correr hacia ellos, eligen otra puerta y entran todos por allí.

Una vez dentro, como que los ninjas y los guardias se mueven igual de rápido, cada habitación la ocupa el grupo que haya empezado más cerca de ella, y en caso de empate, se la quedan los guardias (ver los ejemplos). El objetivo de los ninjas es ocupar el máximo número de habitaciones posibles y el objetivo de los guardias es conseguir que los ninjas ocupen el mínimo posible. Sabiendo que ambos eligen su habitación inicial óptimamente, ¿cuántas habitaciones ocupan los ninjas al final?

Input Format

Una línea, con un entero t, seguido de t casos de prueba. Cada caso de prueba consiste en una línea con un entero n, el número de habitaciones en el palacio, seguido de $n-1$ líneas con dos enteros a_i, b_i, que indican que las habitaciones a_i y b_i están conectadas por un pasillo.

Constraints

Para todos los casos de prueba se cumple $t \leq 100, 2 \leq n, 0 \leq a_i, b_i < n$

5 puntos: $n \leq 5$

16 puntos: $n \leq 70$

22 puntos: $n \leq 500$

57 puntos: $n \leq 10000$

Output Format

Una línea para cada caso con el número de habitaciones que ocupan los ninjas.

Sample Input 0

```
3
5
0 1
0 2
0 3
1 4
6
0 1
0 2
0 3
1 4
2 5
7
0 1
0 2
1 3
1 4
2 5
2 6
```

Sample Output 0

```
2
4
1
```

Explanation 0

En el primer caso, los ninjas entran por la puerta de la habitación 4. Vemos que los ninjas están:

- a distancia 0 de la habitación 4.
- a distancia 1 de la habitación 1.
- a distancia 2 de la habitación 0.
- a distancia 3 de las habitaciones 2 y 3.

En este caso, si los guardias entran por la puerta 2 (la 3 daría un resultado similar), están:

- a distancia 0 de la habitación 2.
- a distancia 1 de la habitación 0.
- a distancia 2 de las habitaciones 1 y 3.
- a distancia 3 de la habitación 4.

Por lo tanto los ninjas se quedan con las habitaciones 1 y 4 y los guardias se quedan con las habitaciones 0, 2 y 3. Dejamos como ejercicio al lector que si los ninjas eligen la puerta 2 o la 3, los guardias eligen la otra puerta y los ninjas se quedan con 1 habitación, que no es óptimo. Tened en cuenta que no se puede entrar por la habitación 0 o 1. (no hay puertas ahí, ya que estas habiatciones tienen más de un pasillo)

En el segundo caso, los ninjas entran por la puerta 3. Sin importar por qué puerta entren los guardias, los ninjas ocupan 4 habitaciones. Si los ninjas entraran por otra habitación, ocuparían tan solo 2 habitaciones.

En el tercer caso, sin importar por donde entren los ninjas, los guardias pueden elegir una entrada que deje a los ninjas con 1 habitación ocupada.

Solución

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

using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;
typedef pair<int, int> pi;
typedef vector<pi> vpi;

vpi D[2];
vi P;
vi S;
vi F;
```

```
vvi G;
int n;

void bfs() {
  queue<pi> Q;
  D[0] = D[1] = vpi(n,pi(-1,-1));
  for (int i = 0; i < (int)F.size(); ++i) {
    D[0][F[i]] = pi(0,F[i]);
    Q.push(pi(F[i],F[i]));
  }
  while (!Q.empty()) {
    int x = Q.front().first;
    int from = Q.front().second;
    Q.pop();
    int dist = D[0][x].first;
    if (D[1][x].second == from) dist = D[1][x].first;
    for (int i = 0; i < (int)G[x].size(); ++i) {
      int y = G[x][i];
      if (D[0][y].first == -1) {
        D[0][y].first = dist + 1;
        D[0][y].second = from;
        Q.push(pi(y,from));
      }
      else if (D[1][y].first == -1 and D[0][y].second != from) {
        D[1][y].first = dist + 1;
        D[1][y].second = from;
        Q.push(pi(y,from));
      }
    }
  }
}

int node;

int dfs(int x, int p = -1, int d = 0) {
  int dist = D[0][x].first;
  if (D[0][x].second == node)
    dist = D[1][x].first;
  if (d >= dist) {
    if (P[p] < P[x])
      return n - S[x];
    return S[p];
  }
  int res = n;
  for (int i = 0; i < (int)G[x].size(); ++i) {
    int y = G[x][i];
    if (y == p) continue;
    int k  = dfs(y,x,d+1);
    res = min(res,k);
  }
  return res;
}

int pfs(int x, int p = -1, int d = 0) {
```

```
  P[x] = d;
  S[x] = 1;
  for (int i = 0; i < (int)G[x].size(); ++i) {
    int y = G[x][i];
    if (y == p) continue;
    S[x] += pfs(y,x,d+1);
  }
  return S[x];
}


int main() {
  int casos;
  cin >> casos;
  while (casos--) {
    cin >> n;
    G = vvi(n);
    P = S = vi(n);
    F.clear();
    for (int i = 0; i < n-1; ++i) {
      int x,y;
      cin >> x >> y;
      G[x].push_back(y);
      G[y].push_back(x);
    }
    for (int i = 0; i < n; ++i)
      if (G[i].size() == 1) F.push_back(i);
    pfs(0);
    bfs();
    int res = 0;
    for (int i = 0; i < (int)F.size(); ++i) {
      node = F[i];
      res = max(res, dfs(F[i]));
    }
    cout << res << endl;
  }
}
```

// Autor de la solución: Cesc Folch (Miembro del comité organizador)