Pizzas

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.

El problema llegaba cuando tocaba cenar, siempre se proponían varias opciones para cenar, un comedor en el lugar de reuniones, una cafetería un poco al sur del lugar o la pizzería de la esquina, pero este no era el problema, pues los asistentes a la reunión no podían elegir y siempre se iban a la pizzería.

El problema que se encontraban es que nunca sabían si podían comprar pizzas para satisfacer a todos los comensales sin que sobrase.

La pizzería vende M toppings de pizzas diferentes numeradas de 0 hasta $M-1$, cada pizza se divide en **8** porciones y una pizza solo tiene un topping.

En la reunion asisten N personas, y cada persona quiere comer p_i porciones de pizza.

La organización quiere saber si dadas las porciones que quieren comer los asistentes, es posible comprar un número de pizzas que haga que no se malgaste comida.

Input Format

La entrada consiste de varios casos. Cada caso consiste en varias líneas.

Cada caso comienza con dos enteros N, M. El número de asistentes y el número toppings de pizza que nos ofrece la pizzería.

Viene seguido de N líneas, cada línea comienza con un número p_i, el número de porciones que quiere comer la i-ésima persona, seguido de p_i enteros t_{ij} indicando los toppings quiere para sus porciones. Una persona puede querer más de una porción con el mismo topping.

Constraints

$1 \leq N \leq 100$

$1 \leq M \leq 100$

$1 \leq p_i \leq 10^5$

$0 \leq t_{ij} \leq M-1$

Para cada caso la suma de p_i no es mayor que 10^5

Subcasos

Subcaso 1 (15pt):

- $M = 1$
- $N = 1$

Subcaso 2 (20pt):

- $M = 1$

Subcaso 3 (65pt):

- No hay restricciones adicionales

Output Format

Para cada caso debes imprimir $< SI >$, si es posible comprar pizzas sin que sobre comida o $< NO >$, si no es posible. Cada caso debe imprimirse en una línea diferente.

Sample Input 0

```
1 1
8 0 0 0 0 0 0 0 0
5 1
1 0
2 0 0
3 0 0 0
2 0 0
1 0
3 3
3 0 1 2
3 2 2 0
3 1 1 1
```

Sample Output 0

```
SI
NO
NO
```

Explanation 0

En el primer caso (corresponde al primer subcaso), tenemos una pizza y un tipo de topping. El comensal quiere 8 porciones de esta pizza por tanto si compramos una pizza con el topping 0 no sobrará nada.

En el segundo caso y en el tercer caso independientemente de como compremos pizza siempre sobrará.

Solución

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

using namespace std;

int main() {
  int n, m;
  while (cin >> n >> m) {
    vector<int> D(m, 0);
    for (int i = 0; i < n; ++i) {
      int k;
      cin >> k;
      for (int j = 0; j < k; ++j) {
        int x;
        cin >> x;
        ++D[x];
      }
    }
    bool si = true;
    for (int i = 0; i < m; ++i) {
      if (D[i]%8) si = false;
    }
    if (si) cout << "SI" << endl;
    else cout << "NO" << endl;
  }
}
```

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

Gallinas

La granja de Mariano puede ser vista como una recta horizontal, y en ciertas posiciones con coordenadas enteras, hay un corral que inicialmente contiene una única gallina. Las gallinas tienen una capacidad de vuelo limitada, y solo pueden volar de un corral a otro si la diferencia de las coordenadas de los dos corrales es menor que un cierto número k. Diremos que dos gallinas son amigas si una puede volar a donde se encuentra la otra usando tantas granjas como quiera como punto de paso. Por ejemplo, si las gallinas pueden volar a distancia 2 y tenemos 3 gallinas en las posiciones 1, 2 y 4, la gallinas de las posiciones 1 y 4 son amigas, porque la primera gallina puede ir a la posición 4 pasando por la 2. Por supuesto, Mariano quiere que sus gallinas puedan relacionarse entre ellas, pero sin que eso provoque un caos en la granja. Por eso, os pide que dado a, el número de parejas de gallinas que quiere que sean amigas, le digais la distancia mínima que deben ser capaces de volar para que haya almenos a parejas de amigas.

Input Format

La entrada empieza con un entero t, el número de casos. Le siguen t casos, cada uno empieza con dos enteros n y a, el número de corrales (cada uno con una gallina) y el número de parejas de gallinas amigas que quiere Mariano. A continuación hay n enteros, x_i indicando las posiciones de las gallinas. No hay dos corrales en el mismo punto y se cumple $0 \leq x_i \leq 10^9, 1 \leq a \leq n(n-1)/2$.

Constraints

$2 \leq n \leq 10$ (10 puntos)

$2 \leq n \leq 1000$ y $0 \leq x_i \leq 10000$ (13 puntos)

$2 \leq n \leq 1000$ (23 puntos)

$2 \leq n \leq 10^5$ y $a = 1$ (5 puntos)

$2 \leq n \leq 10^5$ y $a = n(n-1)/2$ (8 puntos)

$2 \leq n \leq 10^5$ y no hay restricciones adicionales(41 puntos)

Output Format

Imprimid t lineas, cada una con un único número, la mínima distancia que deben poder volar las gallinas en cada caso.

Sample Input 0

```
2
3 2
1 2 4
3 1
1 2 4
```

Sample Output 0

```
2
1
```

Solución

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

using namespace std;

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

int main() {
  int casos;
  cin >> casos;
  while (casos--) {
   int n;
   ll k;
   cin >> n >> k;
   vi V(n);
   for (int i = 0; i < n; ++i)
     cin >> V[i];

   ll a = 0;
   ll b = 2e9;
   while (a + 1 < b) {
     ll c = (a + b)/2;
     ll res = 0;
     ll cont = 1;
     for (int i = 1; i < n; ++i) {
       if (V[i] - V[i-1] <= c) {
         res += cont;
         ++cont;
       }
       else cont = 1;
     }
     if (res >= k) b = c;
     else a = c;
   }
   cout << b << endl;
  }
}
```

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

Ampliación de palabras

Dada una palabra s queremos aplicarle k veces una transformación. La transformación consiste en cambiar todas las apariciones de la letra a por la palabra p_a, todas las de b por p_b, c por p_c, etc.

Si aplicamos la transformació k veces ¿cual es la letra de la posición i? $i = 1$ representa la primera letra.

Input Format

La entrada consiste en varios casos, cada caso empieza con una la palabra s, seguida de los enteros k y i.

A continuación hay 26 líneas, cada una con una palabra, que continene p_a, p_b, p_c...

Constraints

11 Puntos $1 \leq \mid s \mid, \mid p_{letra} \mid \leq 10, 0 \leq k \leq 1$

22 Puntos $1 \leq \mid s \mid, \mid p_{letra} \mid \leq 2, 0 \leq k \leq 20$

67 Puntos $1 \leq \mid s \mid, \mid p_{letra} \mid \leq 10, 0 \leq k \leq 10000$

$1 \leq i \leq 10^{18}$, además nunca es mayor que la longitud final de la palabra.

Output Format

Una línea para cada caso con la letra en la posición i después de k transformaciones.

Sample Input 0

```
rb 2 16
mqbh
dar
owkk
h
d
qs
dxr
mo
frx
j
bld
efsa
cbyn
c
yg
x
pk
orel
nm
apqf
kho
kmco
h
wnku
w
sqmg
ljj 1 6
vsw
d
q
bx
x
v
rrbl
ptns
fw
qf
```

```
mafa
rr
sof
b
n
vqh
fb
a
x
pqc
c
hch
vfrk
l
oz
kpqp
```

Sample Output 0

```
h
f
```

Solución

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

using namespace std;

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

int main() {
  ll maxim = 1e18;
  string a;
  ll p, k;
  int cas = 0;
  while (cin >> a) {
    cin >> p >> k;
    vvi G(26, vi(p + 1,1));
    vs S(26);
    for (int i = 0; i < 26; ++i)
      cin >> S[i];
    for (int i = 1; i <= p; ++i) {
      for (int j = 0; j < 26; ++j) {
        G[j][i] = 0;
        for (int w = 0; w < (int)S[j].size(); ++w) {
          int jj = S[j][w] - 'a';
          G[j][i] += G[jj][i-1];
          G[j][i] = min(maxim, G[j][i]);
        }
      }
    }
    while (p) {
      for (int i = 0; i < (int)a.size(); ++i) {
```

```
        int j = a[i] -'a';
        if (k <= G[j][p]) {
          a = S[j];
          break;
        }
        k -= G[j][p];
      }
      --p;
    }
    cout << a[k -1] << endl;
  }
}
```

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

Habitaciones

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.

Muchos miembros perdían la llave para entrar a su habitación. Por esto y por otras razones este año la reunión se realizará a distancia.

Aunque no se celebre la reunion, la organización quiere resolver el problema y proponen una solución. Saben que a la reunión acuden N personas, cada persona p_i tiene dos llaves de su habitación, una la llevará encima y la otra la dejará en la habitación de la persona c_i ($c_1 \neq p_i$) en la que confía. Entonces, evidentemente, siempre que la persona c_i pueda acceder a su habitación, podrá entrar y coger la llave de la persona p_i. Se garantiza que existe por lo menos una persona tal que si él puede acceder a su habitación, entonces todas las otras personas podrán acceder a la suya.

A la asociación le interesa saber para cada persona p_i, cuál es el mínimo número de personas que tienen que perder la llave para que no pueda acceder a su habitación.

Input Format

La entrada consiste en varios casos. Cada caso consiste en dos líneas.

La primera línea de cada caso es un entero N, la cantidad de personas que acuden a la reunión. La segunda línea de cada caso son N enteros c_i, la habitación donde la i-ésima persona deja su segunda llave.

Constraints

$2 \leq N \leq 10^5$

$0 \leq c_i < N$

Subcasos

Subcaso 1 (10pt):

- $N \leq 100$

Subcaso 2 (35pt):

- $N \leq 1000$

Subcaso 3 (55pt):

- No hay restricciones adicionales

Output Format

Para cada caso debes imprimir N enteros separados por un espacio y con un salto de línea al final.

El i-ésimo entero es el mínimo número de personas que deben perder la llave para que la persona p_i no pueda acceder a su habitación

Sample Input 0

```
2
1 0
5
1 0 0 0 0
5
1 2 3 2 3
```

Sample Output 0

```
2 2
2 2 3 3 3
4 3 2 2 3
```

Explanation 0

En el primer caso, si cualquiera de los dos puede acceder a su habitación abrirá la habitación del otro, por tanto los dos tienen que perder la llave.

En el segundo caso, para p_0 y p_1 solo si ambos pierden la llave perderán el acceso a su habitación. Para el resto de personas, si p_0 puede acceder a su habitación ellos también. Por tanto hace falta que p_0 y p_1 pierdan la llave para que p_0 no pueda acceder y también que ellos mismos la pierdan.

Solución

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

using namespace std;

typedef vector<int> vi;

vi V;
vi P;
vi D;

int dfs1(int x) {
  if (V[x]) return x;
  V[x] = 1;
  return dfs1(P[x]);
}

int dfs2(int x, int k = 0) {
  if (V[x]) return 0;
  V[x] = 1;
  D[x] = k;
  return 1 + dfs2(P[x], k);
}

int dfs3(int x) {
  if (D[x] != -1) return D[x];
```

```
  return D[x] = 1 + dfs3(P[x]);
}


int main() {
  int n;
  while (cin >> n) {
    P = vi(n);
    for (int i = 0; i < n; ++i)
      cin >> P[i];
    V = vi(n,0);
    int x = dfs1(0);
    V = vi(n,0);
    D = vi(n,-1);
    int k = dfs2(x);
    V = vi(n,0);
    dfs2(x,k);
    for (int i = 0; i < n; ++i) {
      if (i) cout << ' ';
      cout << dfs3(i);
    }
    cout << endl;
  }
}
```

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

Coloreando árboles

Dean ha recibido como regalo un árbol. Un árbol es un grafo conexo sin ciclos (hay exactemente un camino sin repetir vértices entre cada par de vértices). Pero el árbol de Dean tiene una propiedad especial, sus vértices están pintados de 2 colores: blanco y negro. A Dean no le gusta combinar estos dos colores, y por eso decide convertirlo en un árbol monocromático (todos los vértices tienen el mismo color). Por supuesto, quiere hacerlo en el mínimo número de pasos. En cada paso puede hacer lo siguiente:

- Seleccionar un conjunto S de vértices de manera que todos estén conectados y sean del mismo color, y cambiar su color.

Ayudad a Dean a encontrar el mínimo número de pasos que necesita para convertir su árbol.

Input Format

La entrada empieza con un número t, el número de casos. Le siguen los t casos. Cada uno empieza con un entero n, el número de vértices del árbol. Le siguen $n-1$ líneas con 2 enteros a y $b, 0 \leq a, b \leq n-1$ cada una, indicando que los vértices a y b están conectados por una arista. A continuación hay una línea con n enteros indicando el color inicial de cada vertice (0 si es blanco, 1 si es negro).

Constraints

$0 \leq n \leq 10$ (11 puntos)

$0 \leq n \leq 5000$ (27 puntos)

$0 \leq n \leq 100000$ y el árbol es una cadena, es decir, el vertice i está conectado con el $i+1$, para todo i tal que $0 \leq i \leq n-1$ (19 puntos)

$0 \leq n \leq 100000$ y el árbol es una estrella, es decir, el vertice 0 está conectado con el i, para todo i tal que $1 \leq i \leq n$ (8 puntos)

$0 \leq n \leq 100000$ y no hay restricciones adicionales (35 puntos)

Output Format

Imprimid t líneas, cada una con un único entero: el mínimo número de pasos que debe hacer Dean para lograr su objetivo en cada caso.

Sample Input 0

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

Sample Output 0

```
0
1
```

Solución

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

using namespace std;

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

vvi G;
vi P;

int pare(int x) {
  if (P[x] == -1) return x;
  return P[x] = pare(P[x]);
}

int mx;
int dist;

void dfs(int x, int p = -1, int d = 0) {
  if (d > dist) {
    dist = d;
    mx = x;
  }
  for (int i = 0; i < (int)G[x].size(); ++i) {
    int y = G[x][i];
    if (y == p) continue;
    dfs(y,x,d+1);
  }
}


int main() {
  int casos;
  cin >> casos;
  while (casos--) {
    int n;
    cin >> n;
    vpi A(n);
    for (int i = 0; i < n-1; ++i)
      cin >> A[i].first >> A[i].second;
    vi V(n);
    for (int i = 0; i < n; ++i)
      cin >> V[i];
    P = vi(n,-1);
    for (int i = 0; i < n-1; ++i) {
      int x = A[i].first;
      int y = A[i].second;
      if (V[x] == V[y])
```

```
        P[pare(x)] = pare(y);
    }
    G = vvi(n);
    for (int i = 0; i < n-1; ++i) {
      int x = pare(A[i].first);
      int y = pare(A[i].second);
      if (V[x] != V[y]) {
        G[x].push_back(y);
        G[y].push_back(x);
      }
    }
    dist = -1;
    dfs(pare(0));
    dist = -1;
    dfs(mx);
    cout << (dist + 1)/2 << endl;
  }
}
```

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