Soluciones entrenos OIE

Girona temps de flors

Desde hace más de 60 años la ciudad de Girona celebra la exposición de flores "Girona temps de flors", en la que se decora cada rincón y espacio emblemático del barrio antiguo con flores. El resultado es una ciudad llena de flores y, como no, turistas curiosos de todo el mundo que llenan las calles del barrio antiguo de Girona para disfrutar de la maravillosa exposición.

Pero como todo, tiene una parte oscura. Los pobres ciudadanos que viven tranquilamente durante todo el año en el barrio antiguo, ven sus calles infestadas de turistas que parece que no saben distinguir entre una carretera y una acera, o entre una flor y un coche que quiere pasar.

Después de años viviendo en el barrio antiguo de Girona, este año Cesc va a cumplir el sueño secreto de todos los residentes de su barrio: ¡empujar al máximo número de turistas en el camino a casa!

Este año la exposición consta de n puntos decorados con flores, en el punto i hay t_i turistas haciendo fotos. El barrio antiguo tiene una gran cantidad de callejones y entre cada dos puntos de exposición de flores existe un callejón que los une. El callejón que va del punto i al j tiene longitud a_{ij}, tened en cuenta que los callejones son de un solo sentido.

Cesc empieza en el punto 0 y quiere ir a su casa que está en el punto 1 empujando el máximo número de turistas posible. Pero tiene un poco de prisa y no quiere tardar más de S segundos, si cada segundo recorre una unidad de distancia y puede empujar a los turistas sin perder tiempo, ¿cuántos turistas puede empujar de camino a casa como máximo?

Importante: Para ir del punto i al punto j no hace falta usar el callejón directo, se puede usar cualquier combinación de callejones.

Solo se puede empujar una vez a cada turista.

Se garantiza que siempre tiene tiempo de ir del punto 0 al punto 1.

Input Format

La entrada consiste en varios casos.

Cada caso empieza con los enteros n, S.

La siguiente línea contiene los n enteros t_i.

Cada una de las siguientes n líneas contiene los enteros a_{ij}. La línea i contiene los enteros $a_{i0}, a_{i1}, \ldots, a_{in}$.

Constraints

$1 \leq a_{it}, t_i \leq 10^3$

$a_{ii} = 0$

$1 \leq S \leq 2 \cdot 10^4$

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

- 11 Puntos: $2 \leq n \leq 8$ y $a_{ij} = 1$ para todo i y j distintos.
- 12 Puntos: $2 \leq n \leq 8$ y a_{ij} es el mínimo para ir de i a j.
- 13 Puntos: $2 \leq n \leq 8$
- 11 Puntos: $2 \leq n \leq 15$ y $a_{ij} = 1$ para todo i y j distintos.
- 12 Puntos: $2 \leq n \leq 15$ y a_{ij} es el mínimo para ir de i a j.
- 13 Puntos: $2 \leq n \leq 15$
- 28 Puntos: $2 \leq n \leq 18$

Output format

Una línea para cada caso con la respuesta.

Solución

Primero observamos que, como n es pequeña, podemos hacer un Floyd-Warshall para calcular la mínima distancia entre todos los pares de puntos.

Consideramos el grafo formado por los nodos: $(\{p_0, p_1, \ldots, p_{n-1}\}, k)$, donde p_i solo puede tomar valores 0 o 1 y $0 \leq k \leq n-1$. Conectamos un nodo $(\{p_0, p_1, \ldots, p_{n-1}\}, u)$ a un nodo $(\{q_0, q_1, \ldots, q_{n-1}\}, v)$ solo si en todos las posiciones donde u tiene un 1, v también tiene un 1 y $q_v = 1$ y $p_v = 0$. Además a esta arista le asignamos como peso la mínima distancia entre u y v. ¿Cuál es la idea detrás de esta construcción? La idea es que cada nodo representa una situación en la que nos podemos encontrar: En un momento dado, solo nos interesa saber que puntos hemos visitado previamente y donde estamos actualmente, de modo que cada nodo representa justamente esto: $p_i = 1$ si ya hemos visitado el nodo i y es 0 si aún no lo hemos visitado, además k representa el nodo en el que nos encontramos actualmente.

Ahora en este grafo tenemos que encontrar en que estados del tipo $(\{1, 1, p_2, p_3, \ldots, p_{n-1}\}, 1)$ podemos llegar en tiempo $\leq S$ ($p_0 = 1$ porque empezamos en 0 y lo habremos visitado seguro y el último nodo a visitar es el 1). El número de turistas que habremos empujado si estamos en el nodo $(\{p_o, p_1, p_2, \ldots, p_{n-1}\}, k)$ será $p_0 \cdot t_0 + p_1 \cdot t_1 + \cdots + p_{n-1} \cdot t_{n-1}$, es decir, la suma de los turistas en los sitios por los que hayamos pasado.

Cuando tenemos secuencias binarias (es decir, de 0 y 1), por comodidad las solemos representar en base 2. En particular podemos representar $\{p_0, p_1, \ldots, p_{n-1}\}$ como $p_0 + 2 \cdot p_1 + 2^2 \cdot p_2 + 2^3 \cdot p_3 + \cdots + 2^{n-1} p_n - 1$, es decir, la podemos representar con un número. Nota: en c++ podemos escribir 2^n como $1 << n$.

Finalmente, un par de detalles: en primer lugar no hace falta que construyamos de manera explícita el grafo. En segundo lugar, aprovechando que tenemos para cada par de nodos la distancia mínima, no es necesario hacer un Dijkstra para asegurar que estamos yendo a cada punto en el menor tiempo posible, simplemente podemos hacer una modificación de un BFS.

Código

C++

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

using namespace std;
typedef long long int ll;
typedef pair<int, int> ii;
typedef vector <int> vi;
typedef vector <vi> vvi;

int main() {
    int n, s;
    while (cin >> n >> s){
        vi turistas(n);
        for (int i = 0; i < n; ++i) cin >> turistas[i];

        vvi A(n, vi(n));
        for (int i = 0; i < n; ++i){
            for (int j = 0; j < n; ++j) cin >> A[i][j];
        }

        //Floyd-Warshall: ahora A[i][j] será la mínima distancia de i a j
        for (int k = 0; k < n; ++k){
            for (int i = 0; i < n; ++i){
                for (int j = 0; j < n; ++j){
                    if (A[i][k] + A[k][j] < A[i][j]){
                        A[i][j] = A[i][k] + A[k][j];
                    }
                }
            }
        }

        vvi best(1 << n, vi(n, s+1)); //aquí guardamos la menor distancia con la que hemos podido llegar al nodo (p,k). Donde p en binario representa los p0,p1,p2,... que hemos comentado en la solución
        best[1][0] = 0; //podemos llegar al 0 en tiempo 0

        queue <ii> Q; //vamos añadiendo los nodos a los que podemos llegar en tiempo <= S
        Q.push({1, 0}); //añadimos el nodo: "hemos visitado 0 y estamos en 0"

        while (not Q.empty()){
            ii aux = Q.front(); Q.pop();

            int bit = aux.first, punto = aux.second;
            int time = best[bit][punto]; //podemos llegar al nodo {bit, punto} con este tiempo mínimo

            for (int i = 0; i < n; ++i){
                if (not (bit & (1 << i))){ //si en la representación binaria de bit, en la posición i no hay un 0 (es decir, no hemos visitado el punto i):
                    int nextbit = bit | (1 << i); //nextbit es bit pero con un 1 en la posición i.
```

```
48                      int nexttime = time + A[punto][i]; //el tiempo para llegar al
                        ↪  nodo{nextbit,i} será el que llevamos hasta ahora más la
                        ↪  distancia entre el punto actual y el i.
49
50                      if (best[nextbit][i] > nexttime and nexttime <= s){
51                          if (best[nextbit][i] == s+1) Q.push({nextbit, i}); //si no
                            ↪  hemos visitado el nodo {nextbit, i} lo añadimos para
                            ↪  visitar
52                          best[nextbit][i] = nexttime; //guardamos el tiempo mínimo
                            ↪  para llegar al nodo {nexbit, i}
53                      }
54                  }
55              }
56          }
57
58          //Ahora ya hemos visitado todos los nodos que hemos podido en tiempo <= S.
            ↪  Sabemos que hemos podido visitar {p,k} si best[p][k] <= s. De entre todos
            ↪  los {p,k} que hayamos visitado nos interesan aquellos que terminen en el
            ↪  punto 1, es decir los que tengan k = 1
59          int solucion = 0;
60          for (int i = 0; i < best.size(); ++i){
61              if (best[i][1] <= s) { //hemos visitado {i,1}?
62                  int value = 0; //guardamos el número de turistas que hemos empujado
                    ↪  al llegar a este nodo
63
64                  for (int j = 0; j < n; ++j){
65                      if ((1 << j) & i) value += turistas[j]; //si en la posición j
                        ↪  tenemos un 1, hemos visitado el punto j y, por tanto,
                        ↪  empujado a esos turistas
66                  }
67
68                  solucion = max(solucion, value);
69              }
70          }
71
72          cout << solucion << endl;
73      }
74  }
```

Python

No hay solución con Python. Python es mucho más lento que c++ y, en este caso, al hacerlo con Python nos daría Time limit exceeded.