Ejercicio 1: Juego para dos

Tuki dibujó un grafo G y le propuso a Kitu el siguiente juego. Inicialmente, Tuki elegirá dos nodos distintos v y w del grafo, y luego Kitu eliminará a lo sumo un eje de G. Si al final del proceso sigue habiendo un camino entre v y w, entonces Tuki gana, mientras que caso contrario gana Kitu.

Como Tuki no sabe cuál es la mejor estrategia decidió elegir v y w al azar, de forma equiprobable, y ahora quiere saber cuál es la probabilidad de **perder** siguiendo esta estrategia, asumiendo que Kitu juega de forma perfecta.

Descripción de una instancia

Hay un único caso de test.

La primera línea contiene dos enteros $2 \leq N \leq 10^5$ y $1 \leq M \leq 5 \times 10^5$. Identificaremos los nodos del grafo con números entre 1 y n.

Luego siguen m líneas, cada una con un par de enteros v y w denotando que vw es un eje de G.

Descripción de la salida

Un único número redondeado a 5 posiciones decimales que indique la probabilidad de que Tuki **pierda** el juego.

Ejemplo de entrada y salida

Se presenta un ejemplo de entrada y su correspondiente salida:

Entrada de ejemplo	Salida esperada de ejemplo
4 4 1 2 1 3 2 4 4 1	0.50000

Se pide:

1. Diseñar un algoritmo eficiente para resolver este problema. Este algoritmo debe resolver los casos de prueba del juez en el límite de tiempo estipulado.

Mini apunte de probabilidad: la probabilidad de perder se puede definir como

$$P(perder) = \frac{\#formas\_de\_perder}{\#formas\_de\_jugar}$$

donde $\#formas\_de\_perder$ es la cantidad de jugadas (v, w) de Tuki que permiten a Kitu ganar. Por propiedades de la probabilidad vale que

$$\frac{\#formas\_de\_perder}{\#formas\_de\_jugar} = 1 - \frac{\#formas\_de\_ganar}{\#formas\_de\_jugar}$$

Mini apunte de redondeo: En C++ una forma sencilla de redondear un `double` al momento de escribirlo en consola es usando el método `setprecision`[2].

Ejercicio 2: Dominós

Tuki preparó sobre una mesa n piezas de dominó con el objetivo de generar el mayor efecto cascada posible. Cada pieza d fue colocada de tal forma que solo puede caerse en una dirección, y, por lo tanto, Tuki ya conoce qué piezas caerían por contacto directo si d cayera. Si la pieza i al caer tira la pieza j por contacto directo entonces diremos que (i, j) es un *par de caída*.

Como Tuki dispuso las piezas de una forma muy complicada, no está seguro de qué piezas deben tirarse para asegurar que eventualmente todas las piezas caigan. Naturalmente, quiere encontrar un subconjunto de piezas de tamaño mínimo que asegure esto mismo. A estos conjuntos los llamaremos *óptimos*.

Debemos ayudar a Tuki a encontrar, dada la distribución de las piezas sobre la mesa, un cierto subconjunto óptimo.

Descripción de una instancia

Hay un único caso de test.

Su primera línea contiene con dos enteros $1 \leq n \leq 10^5$ y $1 \leq m \leq 10^7$, donde n indica la cantidad de piezas que hay sobre la mesa, y m la cantidad de *pares de caída*. Identificaremos cada pieza de dominó con un número entre 1 y n.

Luego siguen m líneas, cada una con dos enteros i y j representando el par de caída (i, j).

Descripción de la salida

Se deben imprimir dos líneas.

La primera debe contener un único entero k, indicando el tamaño de algún conjunto *óptimo*.

En la siguiente línea se debe imprimir el *menor*[3] conjunto óptimo.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
7 6 1 2 2 3 3 1 4 5 5 6 4 6	3 1 4 7

Notemos que en el ejemplo el conjunto {2, 4, 7} también es óptimo, pero {1, 4, 7} es otro conjunto óptimo menor.

[2] https://cplusplus.com/reference/iomanip/setprecision/

[3] Diremos que un conjunto óptimo B_1 es menor a otro B_2 si B_1 visto como secuencia ordenada es menor lexicográficamente que B_2 visto como secuencia ordenada

Se pide:

1. Diseñar un algoritmo eficiente para resolver este problema. Este algoritmo debe resolver los casos de prueba del juez en el límite de tiempo estipulado.

Ejercicio 3: Modems

Se quiere proveer de internet a N oficinas del subsuelo del pabellón $0 - \infty$. Para ello, se compraron W modems, los cuales pueden instalarse en cualquier oficina, brindándole acceso a internet.

Como $W < N$ se deberán comprar también cables para conectar algunas oficinas entre si. Se pueden comprar cables UTP o de fibra óptica, los cuales tienen un costo por metro de U y V, respectivamente. Los cables UTP tienen la condición particular de que solo pueden usarse entre dos oficinas si estas están a menos de R centímetros.

Dadas las posiciones de las oficinas en un eje cartesiano (expresadas en centímetros), la cantidad de modems adquiridos. el precio por metro de los cables UTP y de fibra óptica, y la distancia R, debemos encontrar cuál es la mínima cantidad de dinero que debe pagarse en cables para conectar todas las oficinas. Puntualmente, queremos saber cuánto debe gastarse en cables UTP y cuánto en cables de fibra óptica.

Descripción de una instancia

La primera línea indica el número c de casos de test.

Cada uno comienza con una línea con 5 enteros N, R, W, U y V, donde $1 \leq N \leq 1000$ denota la cantidad de oficinas del pabellón, $1 \leq R \leq 10000$ indica la distancia a partir de la cual no se pueden usar cables UTP, $W < N$ la cantidad de modems adquiridos y $1 \leq U \leq V \leq 10$ el precio por centímetro de los cables UTP y de fibra óptica, respectivamente.

La siguientes N lineas contienen, cada una, 2 enteros x e y. Los enteros (x, y) de la i-ésima línea indican la posición de la i-ésima oficina dentro de un eje cartesiano, expresada en centímetros.

Descripción de la salida

Para el caso de test i se debe imprimir una línea con el mensaje Caso $\#i$: A B donde A es el costo que debe pagarse en cables UTP, y B el correspondiente en cables de fibra óptica. Ambos números deben redondearse a 3 posiciones decimales.

Observación: Si el costo por centímetro de ambos cables es el mismo, entonces se prefiere usar UTP por sobre fibra óptica (siempre y cuando la restricción de longitud R lo permita).

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	**Salida esperada de ejemplo**
2 3 1 1 1 1 0 0 0 1 1 0 6 1 3 2 3 0 0 0 2 2 0 3 2 2 3 3 3	Caso #1: 2.000 0.000 Caso #2: 4.000 6.000

Se pide:

1. Diseñar un algoritmo eficiente basado en el algoritmo de Kruskal para resolver el problema. El mismo debe resolver los casos de prueba del juez en el límite de tiempo estipulado.

2. Presentar un informe de hasta **5 páginas** conteniendo:

   - Una presentación y descripción del problema.
   - Una explicación del algoritmo desarrollado.
   - Una justificación de su correctitud.[4]
   - Un estudio y una comparación empírica de la performance del algoritmo para distintas implementaciones del algoritmo de Kruskal. Puntualmente, se debe:
     - ☐ Indicar y justificar cuál es teóricamente la mejor implementación de Kruskal para este caso de uso.
     - ☐ Verificar empíricamente si en la práctica se ve alguna diferencia entre las distintas implementaciones.
     - ☐ Estudiar si se ve alguna diferencia en los tiempos de ejecución cuando se usa la estructura de DSU sin alguna de sus optimizaciones (por ejemplo, sin *Union by size / rank* o sin *path compression*).

[4] Recordar el invariante del algoritmode Kruskal puede ser útil.