Universidad de La Habana

Facultad de Matemática y Computación

Proyecto de Diseño de Análisis y ALgoritmos

Raudel Alejandro Gómez Molina C-411

Proyecto en github

Índice

1. Reducción de Emparejamiento

1.1. Ejercicio

Link del problema en la plataforma **Codeforces**

- límite de tiempo por test: 8 segundos
- límite de memoria por test: 512 megabytes

Te dan un grafo bipartito con n_1 vértices en la primera parte, n_2 vértices en la segunda parte, y m aristas. El emparejamiento máximo en este grafo es el subconjunto máximo posible (en tamaño) de aristas de este grafo tal que ningún vértice está incidente a más de una arista elegida.

Debes procesar dos tipos de consultas en este grafo:

- 1 — elimina el número mínimo posible de vértices de este grafo para que el tamaño del emparejamiento máximo se reduzca exactamente en 1, e imprime los vértices que has eliminado. Luego, encuentra un emparejamiento máximo en este grafo e imprime la suma de los índices de las aristas que pertenecen a este emparejamiento;
- 2 — este tipo de consulta se realizará solo después de una consulta de tipo 1. Como respuesta a esta consulta, debes imprimir las aristas que forman el emparejamiento máximo que has elegido en la consulta anterior. Ten en cuenta que debes resolver el problema en modo en línea. Esto significa que no puedes leer toda la entrada de una vez. Solo puedes leer cada consulta después de escribir la respuesta a la consulta anterior. Usa las funciones `fflush` en C++ y `BufferedWriter.flush` en Java después de cada escritura en tu programa.

Entrada

La primera línea contiene cuatro enteros n_1, n_2, m y q ($1 \le n_1, n_2 \le 2 \cdot 10^5; 1 \le m \le \min(n_1 \cdot n_2, 2 \cdot 10^5); 1 \le q \le 2 \cdot 10^5$).

Luego siguen m líneas. La i-ésima contiene dos enteros x_i y y_i ($1 \le x_i \le n_1; 1 \le y_i \le n_2$), lo que significa que la i-ésima arista conecta el vértice x_i en la primera parte y el vértice y_i en la segunda parte. No hay pares de vértices que estén conectados por más de una arista.

Luego siguen q líneas. La i-ésima contiene un entero, 1 o 2, indicando la consulta i-ésima. Restricciones adicionales sobre las consultas:

- el número de consultas de tipo 1 no superará el tamaño del emparejamiento máximo en el grafo inicial;
- el número de consultas de tipo 2 no superará 3;
- cada consulta de tipo 2 será precedida por una consulta de tipo 1;
- tu solución solo puede leer la i-ésima consulta después de imprimir la respuesta a la consulta $(i-1)$-ésima y vaciar el búfer de salida.

Salida

Para una consulta de tipo 1, imprime la respuesta en tres líneas de la siguiente manera:

- la primera línea debe contener el número de vértices que eliminas;
- la segunda línea debe contener los índices de los vértices que eliminas, de la siguiente manera: si eliminas el vértice x de la parte izquierda, imprime x; si eliminas el vértice y de la parte derecha, imprime $-y$ (índice negativo);
- la tercera línea debe contener la suma de los índices de las aristas en algún emparejamiento máximo en el grafo resultante. Las aristas están numeradas del 1 al m.

Para una consulta de tipo 2, imprime la respuesta en dos líneas de la siguiente manera:

- la primera línea debe contener el tamaño del emparejamiento máximo;
- la segunda línea debe contener los índices de las aristas que pertenecen al emparejamiento máximo. Ten en cuenta que la suma de estos índices debe ser igual al número que imprimiste al final de la consulta de tipo 1 anterior.

Después de imprimir la respuesta a una consulta, no olvides vaciar el búfer de salida.

1.2. Etiquetas

- Flujo
- Emparejamientos
- Cobertura de vértices

1.3. Solución

En el ejercicio se tiene un grafo **bipartito** en el que se debe responder a dos tipos de consultas, en la primera se debe eliminar la mínima cantidad de vértices para que el emparejamiento máximo del grafo se reduzca exactamente en 1 y en la segunda se debe informar las aristas que pertenecen al emparejamiento máximo en el grafo que se tiene actualmente. Una observación importante es que la interactividad del ejercicio no permite responder las consultas de forma **offline** (recibir todas las consultas hacer algún tipo de preprocesamiento conociendo todas las posibles consultas y luego responder cada una de forma individual), por lo que es necesario responder las consultas de forma individual sin conocer las demás e ir manteniendo el estado del grafo.

Definiciones:

- Se define un emparejamiento máximo de un grafo G como $M(G)$.
- Se define un cubrimiento mínimo de un grafo G como $C(G)$.
- Se define el grafo inducido de $G = V, E$, por V' tal que $V' \subset V$ como $I(G, V')$.

1.3.1. Fuerza bruta

El primer acercamiento para resolver este problema es primeramente encontrar un emparejamiento máximo en el grafo de entrada, para esto ya que dicho grafo es **bipartito** se puede usar el algoritmo de **Khun** (basado en el teorema de **Khun**) o **Flujo Máximo**, estos dos algoritmos funcionan en una complejidad de $O(VE)$.

Luego cuando se necesite analizar una consulta del tipo 1, se puede iterar por todos los posibles conjuntos de vértices V' tal que $V' \subset V$ y $|M(I(G, V'))| = |M(G)| - 1$ y elegir el conjunto de mayor cardinalidad, este algoritmo es algo lento ya que su complejidad es de $O(2^{|V|}VE)$.

Para las consultas de tipo 2 solo se debe guardar el emparejamiento encontrado en la consulta de tipo 1 anterior e informar las aristas que pertenecen a dicho emparejamiento, en una complejidad de $O(q)$ donde q es la cardinalidad del emparejamiento máximo del grafo actual.

1.3.2. Teorema Kőnig

El teorema de **Kőnig** plantea que dado un grafo **bipartito** G se cumple que $|C(G)| = |M(G)|$, además plantea una equivalencia entre estos 2 conjuntos es decir como dado un emparejamiento máximo encontrar un cubrimiento mínimo y viceversa.

Demostración:
Sea un grafo **bipartito** $G = V, E$, con A y B los conjuntos de vértices que representan la bipartición, definamos $Q = M(G)$ y $A_q = \{x \in A \mid \exists y \text{ tal que } \langle x, y\rangle \in Q\}$ y $B_q = \{y \in B \mid \exists x \text{ tal que } \langle x, y\rangle \in Q\}$.

En primer lugar si se tiene un cubrimiento de vértices W se cumple que 2 aristas de Q no inciden sobre un mismo vértice de W porque de lo contrario Q no sería un emparejamiento, por otro lado cada arista en Q debe ser cubierta por un vértice de W ya que de lo contrario W no sería un cubrimiento de vértices. Por tanto si $|Q| = |W|$ entonces se puede decir que W es un cubrimiento mínimo.

Sea Z el conjunto de formado por la unión de $A \setminus A_q$ y el conjunto de vértices que a los que se puede llegar usando caminos Q alternantes partiendo de los vértices de $A \setminus A_q$.

Ahora si se define $K = (A_q \setminus Z) \cup (B_q \cap Z)$ se puede demostrar que K es un cubrimiento de vértices. Observe que para cada arista $\langle x, y\rangle \in E$ con $x \in A$ y $y \in B$ se cumple que $x \in A_q \vee y \in B_q$ porque si no Q no sería máximo: si $\langle x, y\rangle$ pertenece a uno de los caminos Q alternantes entonces $y \in B_q \cap Z$ porque todas las aristas que pertenecen a un camino Q alternante tienen un vértice en B_q, de lo contrario tendríamos un camino Q aumentativo; ahora si $\langle x, y\rangle$ no pertenece a ninguno de los caminos Q alternantes entonces $x \in A_q \setminus Z$ porque si $x \notin A_q$ se cumple que $\langle x, y\rangle$ pertenece a un camino Q alternante.

Solo resta demostrar que $|K| = |Q|$. Observe que para toda arista $\langle x, y\rangle \in Q$ con $x \in A$ y $y \in B$ se cumple que: si $\langle x, y\rangle$ pertenece a uno de los caminos Q alternantes entonces $y \in B_q \cap Z \wedge x \notin A_q \setminus Z$ porque $x \in Z$; ahora si $\langle x, y\rangle$ no pertenece a ninguno de los caminos Q alternantes entonces $x \in A_q \setminus Z \wedge y \notin B_q \cap Z$ porque $x \notin Z$. Entonces como $(A_q \setminus Z) \cap (B_q \cap Z) = \emptyset$ por lo enunciado anteriormente se cumple que $|K| = |Q|$.

1.3.3. Obtener el cubrimiento mínimo

Como se demostró anteriormente en el teorema de **Kőnig** existe una equivalencia entre el emparejamiento máximo y el cubrimiento mínimo en un grafo **bipartito**.

Si se toma la demostración anterior para encontrar un cubrimiento mínimo es necesario primeramente encontrar un emparejamiento máximo, para esto se puede usar el algoritmo de **Khun** (basado en el teorema de **Khun**) o **Flujo Máximo**, ambos en una complejidad de $O(VE)$.

Luego se puede encontrar el conjunto de vértices Z mediante un **DFS** restringiendo el uso de caminos que no sean Q alternantes, en una complejidad de $O(V + E)$.

Solo resta formar el conjunto K, lo cual se puede lograr mediante los conjuntos A, B, A_q, B_q y Z, en una complejidad de $O(V)$.

1.3.4. Utilizando Kőnig y el cubrimiento mínimo

Si se emplea el teorema de **Kőnig** demostrado anteriormente se puede pasar a analizar el problema de forma diferente, es decir existe una equivalencia entre $M(G)$ y $C(G)$ donde G es un grafo **bipartito** ahora se puede buscar la menor candad de vértices tal que al eliminarlos la cardinalidad del cubrimiento mínimo disminuya exactamente en 1 y dicho conjunto cumplirá con las restricciones del problema original.

Se puede realizar la siguiente observación si V_c es un cubrimiento de vértices de $G = V, E$ entonces para cualquier vértice $v \in V_c$, sea $V_c' = V_c - v$ y $V' = V - v$, se cumple que V_c' es un cubrimiento de vértices del grafo $I(G, V')$. Para demostrar el anterior enunciado observe que para toda arista $e \in E$ se cumple que e incide sobre alguno de los vértices de V_c, por tanto las únicas aristas de E que no inciden sobre V_c' son las que únicamente incidían sobre v y dichas aristas no pertenecen a $I(G, V')$, por lo que como las aristas de $I(G, V')$ son subconjunto de E entonces se cumple que V_c' cubre todas las aristas de $I(G, V')$.

Ahora dado el razonamiento anterior si V_c es un cubrimiento mínimo de vértices de $G = V, E$ entonces para cualquier vértice $v \in V_c$, sea $V_c' = V_c - v$ y $V' = V - v$, se cumple que V_c' es un cubrimiento mínimo de vértices del grafo $I(G, V')$. Como ya se demostró que V_c' es un cubrimiento de vértices solo falta demostrar que V_c es mínimo. Para ello suponga que existe un cubrimiento menor V_c'' del grafo $I(G, V')$, entonces si se añade v a dicho cubrimiento se obtiene un cubrimiento de G, ya que todas las aristas que estaban en G y no en $I(G, V')$ incidían sobre v lo cual contradice el hecho de que V_c sea mínimo.

Luego si se elimina cualquier vértice v de G que participe en un cubrimiento mínimo obtenemos un grafo que cuya cardinalidad del cubrimiento mínimo disminuye en 1 con respecto a la cardinalidad del cubrimiento mínimo en el grafo original.

Ahora se puede utilizar el resultado demostrado anteriormente para responder eficientemente las consultas de tipo 1. Para ello es necesario primeramente obtener un cubrimiento mínimo en el grafo de entrada (esto se puede lograr mediante el algoritmo descrito anteriormente) y cada vez que se proporcione una consulta de este tipo seleccionar un vértice de dicho conjunto.

Para las consultas de tipo 2, es necesario mantener un emparejamiento que se corresponda con el cubrimiento mínimo de vértices del grafo actual. Para ello se puede emplear el algoritmo descrito anteriormente para encontrar el cubrimiento mínimo en el grafo inicial y cuando se procese una consulta del tipo 1 eliminar la arista del emparejamiento que incide sobre el vértice que se va a eliminar (esto siempre es posible porque para todo vértice del cubrimiento mínimo inicial existe una arista del emparejamiento que incide sobre dicho vértice y para cada arista del emparejamiento inicial exactamente uno de sus 2 vértices pertenece al cubrimiento mínimo), por tanto al eliminar dicha arista se obtiene un emparejamiento con cardinalidad igual al cubrimiento mínimo de vértices y por el teorema de **Kőnig** se puede asegurar que este emparejamiento es máximo.

Entonces para resolver el ejercicio primero es necesario encontrar un cubrimiento mínimo y el emparejamiento máximo asociado a dicho cubrimiento esto tiene una complejidad de $O(VE)$ empleando el algoritmo descrito anteriormente, las consultas de tipo 1 se pueden responder en una complejidad de $O(1)$ y las de tipo 2 en una complejidad de $O(q)$ donde q es la cardinalidad del emparejamiento máximo del grafo actual.

2. Número de subarreglos k-buenos

2.1. Ejercicio

Link del problema en la plataforma **Codeforces**

- límite de tiempo por prueba: 2 segundos
- límite de memoria por prueba: 256 megabytes

Sea bit(x) el número de unos en la representación binaria de un número entero no negativo x.

Un subarreglo de un arreglo se llama k-bueno si consiste solo en números que no tienen más de k unos en su representación binaria. Es decir, un subarreglo (l, r) del arreglo a es bueno si para cualquier i tal que $l \leq i \leq r$ se cumple la condición bit$(a_i) \leq k$.

Se te da un arreglo a de longitud n, que consiste en enteros no negativos consecutivos que comienzan desde 0, es decir, $a_i = i$ para $0 \leq i \leq n - 1$ (en indexación basada en 0). Necesitas contar el número de subarreglos k-buenos en este arreglo.

Dado que la respuesta puede ser muy grande, imprime el resultado módulo $10^9 + 7$.

Entrada

Cada prueba consiste en múltiples casos de prueba. La primera línea contiene un número entero t $(1 \leq t \leq 10^4)$ — el número de casos de prueba. Las siguientes líneas describen los casos de prueba.

La única línea de cada caso de prueba contiene dos enteros n, k ($1 \leq n \leq 10^{18}, 1 \leq k \leq 60$).

Salida

Para cada caso de prueba, imprime un solo número entero: el número de subarreglos k-buenos módulo $10^9 + 7$.

2.2. Etiquetas

- Máscara de bits
- Divide y Conquista
- Programación Dinámica

2.3. Solución

En el ejercicio se tiene un arreglo de los números de 0 a $n-1$ en orden creciente y se debe responder el número de subarreglos k-buenos. La principal dificultad para el ejercicio es contar eficientemente el número de estos subarreglos así que primeramente se puede abordar un algoritmo de fuerza bruta para resolver el ejercicio.

Nota: En la solución del ejercicio se usarán números soportados por la aritmética de la computadora (hasta 64 bits), por lo que las operaciones con los bits de los números se consideran constantes. Para llevar esta solución a otro modelo de cómputo puede considerar las operaciones de bits sobre n en una complejidad de $O(\log(n))$.

2.3.1. Fuerza bruta

Para resolver este ejercicio observe que se pueden probar con todos los posibles subarreglos y determinar por cada subarreglo si este es k-bueno o no. Para ello se puede encontrar todos los subarreglos en una complejidad de $O(n^2)$ y comprobar si un subarreglo es k-bueno tiene una complejidad de $O(n)$, para una complejidad total de $O(n^3)$.

2.3.2. Mejorando la fuerza bruta

Observe que se puede mejorar el algoritmo anterior haciendo la siguiente observación: si se tiene que un subarreglo (l, r) es k-bueno entonces se puede asegurar que para todo x tal que $l < x \leq r$ se cumple que el subarreglo (l, x) es k-bueno, por otro lado si se tiene que un subarreglo (l, r) no es k-bueno entonces se puede asegurar que para todo x tal que $r \leq x \leq n$ se cumple que el subarreglo (l, x) no es k-bueno.

Ahora si para cada índice l se calcula el mayor r tal que (l, r) es k-bueno se puede obtener el número de subarreglos k-buenos que comienzan en l. Para esto se puede iterar por todos los índices x de 1 hasta n en una complejidad de $O(n)$ y para cada índice x, iterar y desde x hasta que se cumpla que $y > n$ o $\text{bit}(a_y) > k$ con una complejidad de $O(n)$, para una complejidad total de $O(n^2)$.

2.3.3. Divide y Conquista

Ahora para continuar mejorando la solución del ejercicio se puede utilizar un enfoque se divide y conquista. Para ello se puede buscar la mayor potencia de 2 que es menor que n o simplemente el último bit activo de n y elegir esta potencia como pivote, observe que en la parte izquierda del pivote solo se deben buscar subarreglos k-buenos de 0 a $2^q - 1$ (q es el último bit activo de n) mientras en la parte derecha del pivote todos los números tienen activado el bit q (q es el último bit activo para cada uno de estos números) por lo que se puede buscar subarreglos de $(k-1)$-buenos de 0 a $n - 2^q - 1$ (aquí el truco radica en restar 2^q a todos los números en la parte izquierda del pivote y trabajar desde 0 a $n - 2^q - 1$).

Entonces resta resolver la parte de la conquista eficientemente, es decir buscar los arreglos k-buenos que tengan el extremo izquierdo de a la izquierda del pivote y el extremo derecho a la derecha del del pivote. Para ello se define la

función $f(n,k) = (c,l,r)$, la función $f(n,k)$ retorna el número de subarreglos k-buenos de 0 a $n-1$, la longitud de del subarreglo k-bueno más grande que comienza en 0 y la longitud de del subarreglo k-bueno más grande que termina en $n-1$, respectivamente, observe que l y r pueden ser igual a 0. Ahora la cantidad de subarreglos pertenecientes a la conquista simplemente es la multiplicación del tamaño del subarreglo más grande que termina en 2^q y el tamaño del subarreglo más grande que comienza en 2^q. El resto de la solución usando este método consiste en el mantenimiento de las invariantes descritas, a continuación se presenta un pseudocódigo:

```
f(n, k){
    q <-- último bit activo de n

    if(k < 0)
        return 0, 0, 0

    if(n == 0)
        return 1, 1, 1

    if(2**q == n)
        q--

    c1, l1, r1 = f(2**q,k)
    c2, l2, r2 = f(n - 2**q,k - 1)

    l = l1 == 2**q ? l1 + l2 : l1
    r = r2 == n - 2**q ? r2 + r1 : r2

    return c1 + c2 + r1 * l2, l, r
}
```

Solo falta analizar la complejidad temporal, en cada llamado recursivo dividimos la entrada en 2 y hacemos 2 llamadas recursivas, la conquista tiene una complejidad de $O(1)$ por lo que el $T(n)$ recursivo quedaría de la siguiente manera: $T(n) = 2T(n/2) + c$, por lo que la complejidad final es $O(n)$.

2.3.4. Programación Dinámica

Si se observa el algoritmo descrito en la sección anterior, cuando se divide el tamaño de la entrada en la parte izquierda del pivote siempre se hace buscan los subarreglos de 0 a $n-1$, donde n siempre es una potencia de 2, por lo que si se memoriza los valores de $f(n,k)$ cuando n es una potencia de 2, la suma de todos los llamados que se realizan en la parte izquierda tiene una complejidad total de $O(\log(n)k)$.

```
f(n, k){
    q <-- último bit activo de n

    if(k < 0)
        return 0, 0, 0

    if(n == 0)
        return 1, 1, 1

    if(dp[q][k] and n == 2**q)
        return dp[q][k]
```

```
    if(2**q == n)
        q--

    c1, l1, r1 = f(2**q,k)
    c2, l2, r2 = f(n - 2**q,k - 1)

    l = l1 == 2**q ? l1 + l2 : l1
    r = r2 == n - 2**q ? r2 + r1 : r2

    if(2**q == n)
        dp[q][k] = c1 + c2 + r1 * l2, l, r

    return c1 + c2 + r1 * l2, l, r
}
```

Entonces con esta modificación el $T(n)$ recursivo quedaría de la siguiente manera: $T(n) = T(n/2) + c$, por lo que la complejidad del $T(n)$ es $O(\log(n))$ y la complejidad final es $O(\log(n)k)$.