Proyecto final de la asignatura Diseño y Análisis de Algoritmos

Autor :
Amanda Noris Hernández

4to Año Ciencias de la Computación
Universidad de La Habana

September 22, 2024

1 Bicolorings Problem

Se le proporciona una cuadrícula que consta de 2 filas y n columnas. Cada celda de esta cuadrícula debe estar coloreada en blanco o negro.

Dos celdas se consideran vecinas si tienen un borde común y comparten el mismo color. Dos celdas A y B pertenecen al mismo componente si son vecinas, o si hay un vecino de A que pertenece al mismo componente que B.

Llamemos hermoso a un bicolor si tiene exactamente k componentes.

Cuente el número de hermosos bicolores. El número puede ser lo suficientemente grande, así que imprima el módulo de respuesta 998244353.

Aporte La única línea contiene dos números enteros n y k ($1 \leq n \leq 1000, 1 \leq k \leq 2n$): el número de columnas de una cuadrícula y el número de componentes necesarios.

Producción Imprime un solo número entero: el número de hermosos bicolores módulo 998244353.

2 Solución con fuerza bruta

Para resolver este problema mediante fuerza bruta, se pueden generar todas las posibles configuraciones de colores en la cuadrícula de 2 filas y n columnas. Dado que cada celda puede estar coloreada de blanco o negro, el número total de configuraciones posibles es 2^{2n}, ya que hay dos filas y n columnas, y cada celda tiene 2 opciones de color.

Una vez que se han generado todas las configuraciones posibles, para cada configuración se puede contar el número de componentes conectados que tiene la cuadrícula. Después, se filtran aquellas configuraciones que tienen exactamente k componentes.

Pasos:

1. Generar todas las configuraciones posibles: Cada celda de la cuadrícula puede ser negra o blanca. En total hay 2^{2n} configuraciones posibles.
2. Contar el número de componentes conectados en cada configuración.
3. Filtrar las configuraciones que tengan exactamente k componentes.
4. Imprimir el número total de configuraciones que cumplan la condición, módulo 998244353.

2.1 Complejidad temporal

La complejidad temporal es aproximadamente $O(2^{2n} \times n)$, ya que hay 2^{2n} configuraciones posibles y, para cada configuración, se cuenta el número de componentes en tiempo $O(n)$ usando una búsqueda en profundidad.

3 Solución con programación dinámica

Paso 1: Definición del problema y enfoque

El objetivo es contar cuántas formas hay de colorear una cuadrícula de 2 filas y n columnas de tal manera que haya exactamente k componentes conectados, donde dos celdas pertenecen al mismo componente si:

- Comparten un borde común.
- Están coloreadas con el mismo color.

El problema se presta bien a un enfoque de programación dinámica porque podemos descomponer la cuadrícula en columnas y construir la solución agregando una columna a la vez, recalculando el número de componentes a medida que avanzamos.

Paso 2: Definición del estado dinámico

La programación dinámica se basa en la definición del estado $dp[i][j][\text{mask}]$, donde:

- i es el número de columnas procesadas hasta el momento (de 1 a n).
- j es el número de componentes ya formados hasta la columna i.
- mask es una máscara binaria que describe el color de las dos celdas en la i-ésima columna:
  - El primer bit de la máscara representa el color de la celda inferior.
  - El segundo bit de la máscara representa el color de la celda superior.

Hay 4 posibles valores para la máscara (00, 01, 10, 11), que corresponden a todas las combinaciones posibles de colores (blanco o negro) para las dos celdas de una columna.

Paso 3: Inicialización de la DP

El problema empieza con las columnas vacías, lo que implica que el número de componentes es 0. Para cada posible máscara, se inicializa la tabla dinámica como sigue:

$$dp[0][0][\text{mask}] = 1 \quad \text{para cada mask} = 0, 1, 2, 3$$

Esto representa el hecho de que hay exactamente una manera de tener cero componentes con cero columnas procesadas para cada posible configuración de colores iniciales.

Para cualquier otro estado, la cantidad de maneras es inicialmente 0:

$$dp[i][j][\text{mask}] = 0 \quad \text{para cualquier otro estado.}$$

Paso 4: Transiciones de estado

La clave del enfoque de programación dinámica es calcular cómo una columna afecta el número de componentes. Se define una función $\text{get}(\text{mask}, \text{nmask})$ que devuelve el número de nuevos componentes añadidos al pasar de una máscara mask en la columna i a una nueva máscara nmask en la columna $i + 1$.

Hay varios casos que considerar:

- Si las celdas de una columna actual son del mismo color y las de la siguiente columna también, es posible que no se agreguen nuevos componentes.
- Si hay una diferencia de colores, entonces se pueden agregar nuevos componentes o extender componentes ya existentes.

El estado de transición se actualiza como sigue:

$$dp[i+1][j + \text{get}(\text{mask}, \text{nmask})][\text{nmask}] += dp[i][j][\text{mask}]$$

Esto dice que si hemos procesado i columnas, creado j componentes, y tenemos la máscara mask, entonces al añadir la $i + 1$-ésima columna con máscara nmask, el número de componentes aumentará en $\text{get}(\text{mask}, \text{nmask})$, y sumamos esta nueva configuración a nuestro estado dinámico.

Paso 5: Columna ficticia

Después de procesar las n columnas, el número de componentes puede estar incorrecto debido a cómo se cuentan los componentes parciales. Para corregir esto, se agrega una columna ficticia $n+1$ con una máscara especial, llamada mask0...3. Esta columna asegura que los componentes en la última columna se cuenten correctamente como componentes completos.

Por lo tanto, la respuesta final es la suma de todos los posibles valores de dp que cumplen con el número exacto de componentes k:

$$\text{respuesta} = \sum_{\text{mask}=0}^{3} dp[n][k - \text{get}(\text{mask}, \text{mask0...3})][\text{mask}]$$

Esto suma todas las posibles configuraciones de la última columna con los componentes ajustados correctamente.

Paso 6: Módulo 998244353

Dado que el número de posibles configuraciones puede ser muy grande, todas las operaciones en el algoritmo se realizan módulo 998244353, lo que garantiza que el resultado final sea computacionalmente manejable.

4 Demostración de correctitud de la solución

Definición del estado dinámico

El estado dinámico $dp[i][j][\text{mask}]$ está definido de la siguiente manera:

- i es el número de columnas procesadas hasta ahora (de 1 a n).
- j es el número de componentes conectados formados hasta la columna i.
- mask representa la configuración de colores de las celdas en la columna i. Los valores de la máscara pueden ser:
  - 00: ambas celdas blancas.
  - 01: celda inferior negra y celda superior blanca.
  - 10: celda inferior blanca y celda superior negra.

– 11: ambas celdas negras.

La definición del estado es correcta, ya que captura toda la información necesaria para el problema, es decir, el número de columnas procesadas, el número de componentes formados, y la configuración de colores en la última columna. Estos tres valores son suficientes para determinar las posibles transiciones hacia la siguiente columna.

Condiciones iniciales

Se inicializa $dp[1][0][\text{mask}] = 1$ para cada máscara posible ($\text{mask} = 0, 1, 2, 3$). Esto indica que hay exactamente una forma de comenzar con la primera columna sin componentes formados (es decir, con 0 componentes). Esto es correcto, ya que con una sola columna, independientemente de la configuración de colores, no se puede formar ningún componente.

Transiciones entre los estados

La clave del algoritmo es la función de transición, definida por $get(\text{mask}, \text{nmask})$, que calcula cuántos componentes nuevos se agregan al pasar de la columna i con la máscara mask a la columna $i+1$ con la máscara nmask. La transición se realiza con:

$$dp[i+1][j + get(\text{mask}, \text{nmask})][\text{nmask}] += dp[i][j][\text{mask}]$$

Demostración de correctitud de la función de transición `get(mask, nmask)`

La función `get(mask, nmask)` compara las máscaras `mask` (de la columna i) y `nmask` (de la columna $i+1$) y evalúa cuántos nuevos componentes se forman al agregar la nueva columna. Dependiendo de cómo cambien los colores, los componentes pueden:

- Mantenerse sin cambio (es decir, el número de componentes no aumenta).
- Aumentar en uno o más, dependiendo de si se crean nuevos componentes o si las celdas en la nueva columna separan componentes previos.

Casos a considerar:

Sin cambio de colores (`mask = nmask`):

- Si los colores de la columna i son los mismos que en la columna $i+1$, no se agrega ningún componente nuevo.
- **Ejemplo:** Si `mask = 00` y `nmask = 00`, el número de componentes permanece igual, pues ambas columnas tienen el mismo color.

Cambio en ambas celdas (`cnt = 2`):

- Si las dos celdas cambian de color (por ejemplo, de blanco a negro o viceversa), se pueden formar nuevos componentes, pero depende de la configuración de la columna anterior.
- Si la columna anterior era "completa" (es decir, ambas celdas estaban conectadas en un solo componente), entonces al cambiar los colores se debe agregar un nuevo componente.
- Si la columna anterior no era "completa" (es decir, las dos celdas ya formaban componentes separados), se agregan dos nuevos componentes.

Ejemplos:

- Si `mask = 00` y `nmask = 11`, se añade 1 componente porque las dos celdas cambian de color, pero antes estaban en el mismo componente.
- Si `mask = 01` y `nmask = 10`, se añaden 2 componentes, porque ambas celdas cambian de color y en la columna anterior no formaban un solo componente.

Cambio en una celda (`cnt = 1`):

- Si una sola celda cambia de color (por ejemplo, una es negra y la otra blanca), puede ocurrir una de dos cosas:
  - Se extiende un componente ya existente.
  - Se forma un nuevo componente, dependiendo de si la columna anterior estaba completa o no.

Ejemplo:

- Si `mask = 00` y `nmask = 01`, la celda inferior cambia de color. Si las dos celdas de la columna anterior formaban un componente completo, entonces este cambio extiende el componente sin añadir nuevos. Si no formaban un componente, entonces se puede formar uno nuevo.

Prueba inductiva

Podemos demostrar la correctitud por inducción:

Base de la inducción

Para la primera columna ($i = 1$), el algoritmo inicializa correctamente $dp[1][0][\text{mask}] = 1$ para todas las máscaras posibles, ya que, con una sola columna, hay exactamente una forma de configurar los colores sin formar ningún componente.

Paso inductivo

Supongamos que para una columna i el estado $dp[i][j][\text{mask}]$ es correcto y contiene el número de maneras de colorear las primeras i columnas con exactamente j componentes, dada la configuración mask en la última columna. Al procesar la columna $i + 1$, el número de componentes nuevos se actualiza correctamente usando la función $get(\text{mask}, \text{nmask})$, que asegura que todos los casos posibles están cubiertos. Por lo tanto, si el estado es correcto para i, también lo será para $i + 1$.

Condiciones finales y columna ficticia

Después de procesar todas las columnas, es posible que algunos componentes no estén completamente contados si están "incompletos" (por ejemplo, un componente que cubre solo una parte de la cuadrícula). Para solucionar esto, el algoritmo introduce una columna ficticia $n + 1$ con una máscara especial (maskrevisarqvaahi3, representada en el código como mask^3). Esta columna asegura que los componentes incompletos se cuenten correctamente. La respuesta final es la suma de todas las configuraciones que cumplen con el número exacto de componentes k después de agregar esta columna ficticia:

$$\text{respuesta} = \sum_{\text{mask}=0}^{3} dp[n][k - get(\text{mask}, \text{mask}^3)][\text{mask}]$$

4.1 Complejidad temporal

Complejidad general: O(n2*4m), donde m es el número de filas (2 para este problema).

debe estar mal

5 Implementación

Tanto la implementación del algoritmo de fuerza bruta como el de programación dinamica se pueden entontrar en los archivos adjuntos. Los casos de prueba con los que se comprobó la correctitud de este código son los proporcionados por la plataforma Codeforces.