Vabank

Juan Miguel Pérez Martínez c-412

September 23, 2024

Problema 3: Vabank

Descripción del Problema:
Gustaw, el jefe de un banco, intenta descubrir el límite M de transacciones permitidas por un sistema antifraude sin ser detectado. Si realiza una transacción mayor a M, se detecta el fraude y puede ser multado. Gustaw comienza con 1 euro en su cuenta y debe descubrir el valor exacto de M en no más de 10^5 operaciones, sin quedarse sin fondos.

Entrada:

- Cada prueba contiene múltiples casos de prueba. La primera línea contiene el número de casos de prueba t ($1 \leq t \leq 1000$).
- Para cada caso de prueba, no hay entrada antes de tu primera consulta, pero puedes estar seguro de que M es un entero y $0 \leq M \leq 10^{14}$.

Salida:

- Para cada caso de prueba, cuando conozcas el valor exacto de M, imprime una línea con el formato "! M". Después de eso, tu programa debe proceder al siguiente caso de prueba o terminar si es el último.

Interacción:

- Cuando quieras hacer una operación, imprime una línea con el formato "? X", indicando que intentas mover X euros ($1 \leq X \leq 10^{14}$).
- Como respuesta, lee una línea que puede ser:
  - "Lucky!" si $X \leq M$. Tu saldo aumenta en X euros.
  - "Fraudster!" si $X > M$. Tu saldo disminuye en X euros.
  - "Fired!" si $X > M$, pero no puedes pagar la multa de X euros. En este caso, tu programa debe terminar inmediatamente.
- Puedes hacer como máximo 10^5 consultas en cada caso de prueba.

Ejemplo:

```
Entrada
1
Lucky!
Lucky!
Lucky!
Lucky!
Lucky!
Fraudster!

Salida
? 1
? 2
? 3
? 4
? 5
? 6
! 5
```

Enlace al Problema:
Problema Vabank en Codeforces

1 Introducción

El problema que estamos resolviendo requiere encontrar el valor límite M en el que las transacciones son detectadas por el sistema anti-fraude. El objetivo es encontrar M de manera eficiente, minimizando la cantidad de consultas necesarias, evitando ser detectado, y sin agotar el balance de Gustaw.

A continuación, discutimos varios enfoques, comenzando por el enfoque más intuitivo (fuerza bruta) y luego explicamos por qué nuestro algoritmo, basado en escalado exponencial y una tabla precomputada, es óptimo para este problema.

2 Enfoque de Fuerza Bruta

Una primera aproximación natural sería simplemente intentar cada valor de transacción X sucesivamente, comenzando desde 1, hasta que se detecte el fraude. Este enfoque sería algo como:

```
X = 1;
while (!fraude_detectado) {
    realizar_transaccion(X);
    X++;
}
```

Aunque este método eventualmente encontrará el valor de M, es extremadamente ineficiente. En el peor de los casos, si M es muy grande (hasta 10^{14}), tomaría una cantidad de tiempo prohibitiva, realizando hasta 10^{14} consultas, lo cual es inviable debido a las restricciones del problema. Claramente, este enfoque no es práctico.

3 Escalado Exponencial

En lugar de probar cada valor uno por uno, podemos mejorar el proceso utilizando una técnica de **escalado exponencial** o **búsqueda exponencial**. En este enfoque, en lugar de incrementar el valor de la transacción de 1 en 1, duplicamos el valor de X en cada paso. Esto nos permite encontrar rápidamente un límite superior para M.

El escalado exponencial funciona de la siguiente manera:

```
X = 1;
while (realizar_transaccion(X)) {
    X = X * 2;
}
```

Este enfoque es mucho más eficiente, ya que el número de consultas necesarias para encontrar un límite superior es aproximadamente $\log_2 M$. Así, en lugar de necesitar hasta 10^{14} consultas, solo necesitamos unas 47 consultas para valores grandes de M. Este es un salto significativo en eficiencia.

4 ¿Por qué no usar Búsqueda Binaria pura después del Escalado Exponencial?

Después de encontrar un límite superior para M mediante el escalado exponencial, una opción podría ser utilizar la **búsqueda binaria** para reducir el intervalo en el que se encuentra M. Sin embargo, esta estrategia tiene algunos problemas en este contexto particular, que se detallan a continuación:

4.1 Riesgo de ser despedido

En una búsqueda binaria tradicional, divides el intervalo en mitades y pruebas el valor intermedio. Sin embargo, en este problema, no puedes simplemente dividir el intervalo sin considerar el **balance disponible**. Si intentas mover una cantidad que excede tu balance, serás despedido o perderás dinero en multas, lo cual podría terminar tu juego prematuramente.

4.2 Desperdicio de consultas

El número de consultas en una búsqueda binaria estándar está limitado por el tamaño del intervalo. Si bien este enfoque es eficiente en algunos problemas, aquí el número de consultas puede ser ineficiente si no se maneja adecuadamente el balance. Algunas consultas pueden no contribuir a mejorar tu situación financiera, limitando las futuras consultas posibles y reduciendo las oportunidades de éxito.

4.3 Falta de optimización del balance

La búsqueda binaria pura no optimiza la cantidad de dinero que puedes acumular. En cambio, al utilizar una **tabla precomputada**, tomas decisiones informadas que maximizan tu balance en cada paso, garantizando que tengas suficiente dinero para pagar posibles multas sin riesgo de quedarte sin fondos. Esto asegura que siempre puedas seguir jugando y buscando M de manera segura.

5 Optimización Mediante Tabla Precomputada

Una vez que hemos encontrado un intervalo en el que sabemos que M se encuentra (entre $X/2$ y X), utilizamos una **tabla precomputada** para optimizar las decisiones posteriores y reducir aún más el número de consultas.

La tabla $f[a, k]$ contiene valores precomputados que guían la elección del siguiente valor de X. Esta tabla se calcula de manera que minimiza el riesgo de caer en una transacción fraudulenta mientras optimiza la cantidad de dinero acumulado.

La tabla se inicializa de la siguiente manera:

```
for (int a = 0; a < N - 1; a++) {
    for (int k = 0; k < N - 1; k++) {
        if (a == 0) {
            f[a, k] = 1;
        } else if (k == 0) {
            f[a, k] = f[a - 1, k + 1];
        } else {
            f[a, k] = f[a - 1, k - 1] + f[a - 1, k + 1];
        }
    }
    if (f[a, 0] >= maxMoney) break;
}
```

La tabla nos permite tomar decisiones más informadas sobre el valor de la próxima transacción, asegurando que nos mantenemos dentro del límite seguro.

6 Construcción de la Tabla Precomputada

La tabla $f[a, k]$ que usamos en nuestro algoritmo nos permite decidir cuál es el siguiente valor óptimo a consultar en función de la diferencia entre el límite superior y el límite inferior en el que sabemos que se encuentra M. Esta tabla se precomputa para evitar cálculos repetitivos durante la ejecución del algoritmo.

6.1 Definición de la Tabla

La tabla $f[a, k]$ está diseñada de tal forma que en la posición (a, k) contiene el valor acumulado que corresponde a la cantidad de dinero que puedes mover (o el rango de valores) sin riesgo de fraude, basado en el número de consultas y el dinero disponible. Formalmente, la tabla se define como:

$$f[a,k] = \begin{cases} 1 & \text{si } a = 0 \\ f[a-1, k+1] & \text{si } k = 0 \\ f[a-1, k-1] + f[a-1, k+1] & \text{si } k > 0 \end{cases}$$

Esta recurrencia nos permite construir la tabla eficientemente de manera iterativa.

6.2 Explicación de la Recurrencia

La idea detrás de esta construcción es que, en cada paso, calculamos el valor que puedes mover de manera segura en función de las decisiones anteriores.

- Para $a = 0$, sabemos que podemos mover una cantidad mínima de 1 euro, ya que es la cantidad inicial con la que comenzamos el juego.
- Si $k = 0$, significa que nos encontramos en un estado donde solo podemos mover lo que la última consulta nos permitió, por lo que tomamos el valor de $f[a-1, k+1]$.
- Si $k > 0$, entonces podemos combinar la cantidad movida en el estado anterior sumando lo que movimos con una consulta anterior más lo que se podría haber movido en el siguiente paso seguro, representado por $f[a-1, k-1] + f[a-1, k+1]$.

De esta manera, la tabla nos permite calcular el valor acumulado de dinero que podemos mover de manera óptima en función de los pasos realizados, manteniéndonos dentro de un margen seguro.

7 Búsqueda Adaptativa Basada en la Tabla Precomputada

Una vez que hemos determinado un intervalo donde se encuentra M, realizamos una búsqueda adaptativa utilizando la tabla precomputada f. Este método no solo nos ayuda a identificar el valor de M de manera más efectiva, sino que también minimiza el riesgo de ser detectado.

La tabla f se construye utilizando programación dinámica (DP), lo que nos permite almacenar resultados intermedios y reutilizarlos en cada paso del algoritmo. Esto es crucial, ya que evita la necesidad de recalcular los valores en cada iteración y permite optimizar las decisiones de transacción.

La búsqueda adaptativa se basa en elegir un punto intermedio óptimo en lugar de simplemente dividir el intervalo a la mitad. Utilizamos los valores en la tabla f para calcular un valor intermedio `bestMid`, que se determina de la siguiente manera:

$$\texttt{bestMid} = \texttt{nowL} + (k \neq 0 ? f[a-1, k-1] : 0)$$

Este enfoque asegura que cada decisión sobre el valor a consultar se fundamenta en la información precomputada, permitiéndonos ajustar el límite inferior y superior de forma más dinámica y eficiente.

8 Búsqueda Adaptativa

Una vez que tenemos el intervalo y la tabla precomputada, nuestro algoritmo realiza una **búsqueda adaptativa**. En lugar de simplemente dividir el intervalo a la mitad (como en una búsqueda binaria), utilizamos la tabla f para determinar un valor óptimo intermedio (`bestMid`) en cada paso.

Este proceso ajusta de manera dinámica el límite inferior y superior, y cada paso garantiza que avanzamos más rápido que con una simple búsqueda binaria.

9 Demostración de Optimalidad

Este enfoque combina dos técnicas que hacen que sea óptimo para este problema:

1. **Escalado exponencial**: Encuentra rápidamente un límite superior con un número logarítmico de consultas. 2. **Tabla precomputada y búsqueda adaptativa**: Reduce aún más el número de consultas necesarias para ajustar el intervalo, tomando decisiones basadas en información precomputada lo que evita que sea decubierto.

En conjunto, este enfoque logra encontrar el valor de M en un número de consultas que es mucho menor que el enfoque de fuerza bruta o incluso que una búsqueda binaria pura.

La cantidad total de consultas es proporcional a $\log M$ para el escalado inicial y a la eficiencia de la búsqueda guiada por la tabla, lo que la convierte en una solución más eficiente que cualquier otro enfoque simple.

10 Código de Solución

Aquí tienes el código en C# para implementar la solución descrita:

```
using System;
using System.Diagnostics;
class Program
{
    const int N = 100;
    const long maxMoney = (long)1e14 + 1;
    static long[,] f = new long[N, N];
    static long money;
    static int rem;

    static bool Ask(long x)
    {
        Console.WriteLine("? " + x);
        Debug.Assert(x < maxMoney && --rem >= 0);

        string response = Console.ReadLine();
```

```
        if (response[0] == 'L')
        {
            money += x;
            return true;
        }
        else
        {
            money -= x;
            Debug.Assert(money >= 0);
            return false;
        }
    }

    static void InitializeTable()
    {

        for (int a = 0; a < N - 1; a++)
        {
            for (int k = 0; k < N - 1; k++)
            {
                if (a == 0)
                {
                    f[a, k] = 1;
                }
                else if (k == 0)
                {
                    f[a, k] = f[a - 1, k + 1];
                }
                else
                {
                    f[a, k] = f[a - 1, k - 1] + f[a - 1, k + 1];
                }
            }
            if (f[a, 0] >= maxMoney) break;
        }
    }


    static void SearchAndSolve()
    {
        long nowL = 1, nowR = maxMoney;
        rem = N;
        money = 1;


        while (nowL < maxMoney && Ask(nowL))
        {
            nowL <<= 1;

        }

        if (money == 0)
        {
            nowR = nowL;
        }

        nowL >>= 1;
```

```
74
75          if (nowL == 0)
76          {
77              Console.WriteLine("! 0");
78              return;
79          }
80
81          int a = 0, k = 0;
82
83          while (f[a, k] < nowR - nowL)
84          {
85              a++;
86          }
87
88          while (nowR - nowL > 1)
89          {
90              Debug.Assert(a >= 0 && k >= 0);
91              Debug.Assert(f[a, k] >= nowR - nowL);
92
93              while (money < (nowR - nowL) + nowL * k)
94              {
95                  Ask(nowL);
96              }
97
98              long bestMid = nowL + (k != 0 ? f[a - 1, k - 1] : 0);
99              bestMid = Math.Min(bestMid, nowR - 1);
100
101             if (Ask(bestMid))
102             {
103                 a--;
104                 k++;
105                 nowL = bestMid;
106             }
107             else
108             {
109                 a--;
110                 k--;
111                 nowR = bestMid;
112             }
113         }
114
115         Console.WriteLine("! " + nowL);
116     }
117
118     static void Main(string[] args)
119     {
120
121         InitializeTable();
122
123
124         int testCount = int.Parse(Console.ReadLine());
125
126         for (int testCase = 0; testCase < testCount; testCase++)
127         {
128
129             SearchAndSolve();
130         }
```

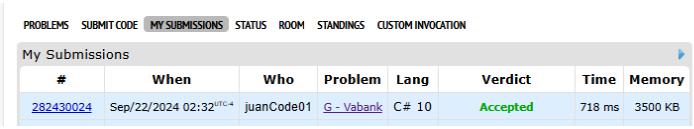
```
    }
}
```

11 Conclusión

En conclusión, el algoritmo presentado combina escalado exponencial, programación dinámica y búsqueda adaptativa basada en una tabla precomputada, lo que permite encontrar el límite M en un número de consultas significativamente menor que el enfoque de fuerza bruta. Este método es óptimo y eficiente, asegurando que siempre se pueda seguir jugando sin arriesgar el saldo de manera innecesaria.

El enfoque demuestra que, al precomputar valores y tomar decisiones informadas, es posible abordar problemas de búsqueda complejos en un espacio de soluciones mucho más grande de manera efectiva.

12 Casos de prueba



PROBLEMS SUBMIT CODE MY SUBMISSIONS STATUS ROOM STANDINGS CUSTOM INVOCATION

My Submissions

#	When	Who	Problem	Lang	Verdict	Time	Memory
282430024	Sep/22/2024 02:32 UTC-4	juanCode01	G - Vabank	C# 10	Accepted	718 ms	3500 KB

Figure 1: Tester pasados en codeforces