

T-shirt

Juan Miguel Pérez Martínez c-412

23 de septiembre de 2024

Problema 2: T-shirt

Descripción del Problema:

Trabajas en Codeforces como pasante en un equipo de n ingenieros, numerados del 1 al n . Quieres regalar una camiseta a cada ingeniero, pero no conoces las tallas exactas que cada ingeniero necesita. Hay m tallas diferentes, numeradas del 1 al m , y cada ingeniero encaja exactamente en una de estas tallas.

Tu amigo Gerald proporciona probabilidades para cada ingeniero i y talla j , indicando la probabilidad de que el ingeniero i encaje en una camiseta de talla j . Debes elegir qué tallas traer para maximizar el número esperado de ingenieros que reciben una camiseta de su talla correcta.

Entrada:

- La primera línea contiene dos enteros n y m ($1 \leq n \leq 3000, 1 \leq m \leq 300$), que denotan el número de ingenieros y el número de tallas de camisetas, respectivamente.
- Luego siguen n líneas, cada línea contiene m enteros separados por espacios. El j -ésimo entero en la i -ésima línea representa la probabilidad de que el ingeniero i encaje en una camiseta de talla j . Cada probabilidad se da como un entero entre 0 y 1000, inclusive. La probabilidad real se calcula como el número dado dividido por 1000.
- Se garantiza que para cualquier ingeniero, la suma de las probabilidades para todas las camisetas m es igual a uno.

Salida:

- Imprime un solo número real que denote el máximo posible del número esperado de ingenieros que recibirán una camiseta.
- El error absoluto o relativo de 10^{-9} es aceptable.

Ejemplo:

Entrada
2 2

```
500 500
500 500
```

```
Salida
1.500000000000
```

```
Entrada
3 3
1000 0 0
1000 0 0
0 1000 0
```

```
Salida
3.000000000000
```

```
Entrada
1 4
100 200 300 400
```

```
Salida
0.400000000000
```

Enlace al Problema:

Problema T-shirt en Codeforces

Descripción General del Problema

El problema abordado en el código consiste en asignar a n ingenieros diferentes m tallas de camisetas, donde cada ingeniero tiene ciertas probabilidades de elegir una talla específica. El objetivo es maximizar la suma de las probabilidades más altas de que cada ingeniero elija una talla específica. La solución se resuelve usando una aproximación de **programación dinámica** con elementos probabilísticos.

Programación Dinámica

La idea principal de la *programación dinámica* es descomponer un problema en subproblemas más pequeños que se superponen y resolver cada subproblema solo una vez, almacenando su resultado para evitar recomputaciones innecesarias.

1. Definición Recursiva del Subproblema

El arreglo de programación dinámica `dp[size, i]` en el código representa la probabilidad de que exactamente i ingenieros no hayan seleccionado la talla

size después de considerar a los primeros i ingenieros. Esta estructura es una tabla DP clásica, donde el subproblema implica calcular el resultado para un subconjunto de ingenieros y luego extender el resultado al conjunto más grande.

2. Recurrencia y Subestructura Óptima

La recurrencia para calcular `dp[size, i]` se basa en una relación de recurrencia que aprovecha los resultados previamente calculados:

$$dp[size, i] = dp[size, i - 1] \times (1 - probabilities[i - 1, size])$$

Esta relación define que si los primeros $i - 1$ ingenieros no han seleccionado una talla particular, entonces para el ingeniero i , la probabilidad se actualiza multiplicando el valor existente por la probabilidad de que el ingeniero i no seleccione esa talla.

La subestructura óptima surge porque resolver el problema para i ingenieros y la talla *size* ayuda a resolverlo para $i + 1$ ingenieros. Esto encaja en el paradigma de la programación dinámica, donde las soluciones óptimas a subproblemas contribuyen a resolver el problema general.

3. Uso de Memoria: Tabla de Abajo Hacia Arriba

Se mantiene una tabla DP bidimensional `dp[size, i]`, donde cada entrada representa un estado en la evaluación recursiva de los subproblemas. Este enfoque es similar a resolver problemas de manera *de abajo hacia arriba*, típico en soluciones de DP).

Análisis Probabilístico

El código utiliza el **análisis probabilístico** para modelar la incertidumbre sobre qué talla de camiseta elegirá un ingeniero. En la fase de inicialización:

- Las preferencias de los ingenieros para cada talla se ingresan como probabilidades.
- Estas probabilidades se dividen entre 1000 para normalizarlas a valores decimales.

Las técnicas de análisis probabilístico, son útiles para analizar el comportamiento de los algoritmos en promedio. En este caso, el análisis probabilístico ayuda a determinar la probabilidad de que un grupo de ingenieros seleccione una talla específica de camiseta, y el **valor esperado** de tomar una decisión se captura mediante la regla de actualización de la programación dinámica.

Ejemplo del Uso de Probabilidades Aleatorias

Las probabilidades son independientes para la elección de cada ingeniero, lo que se alinea con los *ensayos de Bernoulli* y las *distribuciones binomiales*. Para cada ingeniero y cada talla, calculamos la probabilidad acumulada usando la regla:

$$dp[size, i] = dp[size, i - 1] \times (1 - probabilities[i - 1, size])$$

Esto refleja el modelo de distribución binomial para ensayos independientes repetidos.

Selección Greedy de Probabilidades

El uso de una aproximación *greedy* para seleccionar la talla de camiseta más probable para cada ingeniero es evidente al sumar las mayores diferencias de probabilidad.

Este enfoque greedy garantiza que siempre se prioricen las tallas con mayor probabilidad de éxito en cada paso. Aunque no se puede asegurar que el resultado sea siempre globalmente óptimo, esta estrategia es efectiva en este tipo de escenarios, ya que cada ingeniero recibe la camiseta más probable de las que quedan disponibles. Además, después de cada asignación, se actualizan las probabilidades acumuladas para mantener la consistencia del modelo.

Demostración de la Elección Local Óptima

Para un ingeniero i , se selecciona la talla de camiseta t que maximiza la probabilidad de éxito, es decir, $\text{máx}(\text{sumDp}[t])$. Esta decisión local es la mejor en ese momento, ya que maximiza la probabilidad inmediata de que el ingeniero reciba una camiseta adecuada. Matemáticamente, si para un ingeniero i , la probabilidad $p(i, t)$ de que se ajuste a una camiseta de talla t es mayor que la probabilidad para cualquier otra talla t' , entonces:

$$\forall t' \in \{\text{tallas disponibles}\}$$

Esto asegura que en cada paso se está tomando una **decisión local óptima**.

Posible Optimalidad Global

La pregunta clave es si una serie de decisiones locales óptimas puede llevar a una solución globalmente óptima. En algunos problemas, como el de la elección de monedas, el enfoque greedy siempre produce una solución óptima. Sin embargo, en otros casos, esto no siempre es así, ya que las decisiones locales pueden afectar negativamente a las decisiones futuras.

En nuestro caso, el enfoque greedy tiende a funcionar bien debido a las siguientes razones:

- **Independencia de probabilidades:** La probabilidad de que un ingeniero se ajuste a una talla de camiseta es independiente de las probabilidades de ajuste de los otros ingenieros. Por lo tanto, elegir una talla ahora no altera la probabilidad de éxito de los ingenieros futuros.
- **Acumulación de probabilidades:** Después de asignar una talla, actualizamos las probabilidades acumuladas mediante la función `UpdateDp`, lo que asegura que las decisiones futuras se basen en la información más actualizada.
- **Aprovechamiento de las mejores opciones:** Al elegir la mejor opción disponible en cada paso, maximizamos la probabilidad de éxito en el momento actual sin desperdiciar oportunidades importantes para ingenieros posteriores.

Resumen Detallado de los Componentes del Algoritmo

1. **Inicialización:** Se inicializan las probabilidades de que los ingenieros seleccionen las tallas de camisetas.
2. **Cálculo de la Tabla DP:** La tabla `dp` calcula la probabilidad de que los ingenieros no elijan una talla en función del estado anterior del sistema.
3. **Suma de Probabilidades:** Las probabilidades se acumulan usando programación dinámica.
4. **Selección Greedy:** Se seleccionan de forma *greedy* las elecciones más probables y se actualiza la tabla DP.

La **complejidad temporal** general de la solución está dominada por las actualizaciones de la tabla de programación dinámica y la selección greedy de la mejor probabilidad, lo que hace que la complejidad sea aproximadamente $O(n \times m)$.

En conclusión, su solución emplea **programación dinámica** para calcular las probabilidades intermedias y utiliza un algoritmo *greedy* para hacer la selección final de las tallas de camisetas. La combinación de análisis probabilístico con DP y un algoritmo greedy asegura que el problema se resuelva de manera eficiente.

1. Solución en c#

```
using System;
using System.Collections.Generic;

class TShirtProblem
```

```

{
    static void Main()
    {
        // Leer n y m
        string[] input = Console.ReadLine().Split();
        int n = int.Parse(input[0]);
        int m = int.Parse(input[1]);

        // Guardar probabilidades de los ingenieros por cada talla
        double[,] probabilities = new double[n, m];
        for (int i = 0; i < n; i++)
        {
            input = Console.ReadLine().Split();
            for (int j = 0; j < m; j++)
            {
                probabilities[i, j] = int.Parse(input[j]) / 1000.0;
            }
        }

        // Array para almacenar los resultados de DP
        double[,] dp = new double[m, n + 1]; // dp para cada tamaño de camiseta

        // Array para la suma acumulada de probabilidades
        double[] sumDp = new double[m];

        // Inicializamos DP para cada talla
        for (int size = 0; size < m; size++)
        {
            dp[size, 0] = 1.0; // Caso base: nadie tiene esa camiseta

            // Calculamos las probabilidades acumuladas de que los ingenieros usen esa talla
            for (int i = 1; i <= n; i++)
            {
                dp[size, i] = dp[size, i - 1] * (1 - probabilities[i - 1, size]);
            }

            // Sumamos las probabilidades acumuladas
            sumDp[size] = 1.0 - dp[size, n];
        }

        // Lista para priorizar las diferencias de probabilidades entre tallas
        List<double> values = new List<double>();
        for (int size = 0; size < m; size++)
        {
            values.Add(sumDp[size]);
        }
    }
}

```

```

// Sumamos las N diferencias de probabilidades más grandes
double result = 0.0;
for (int i = 0; i < n; i++)
{
    // Encontramos la mayor diferencia y la agregamos al resultado
    int maxIndex = -1;
    double maxValue = -1;
    for (int j = 0; j < m; j++)
    {
        if (sumDp[j] > maxValue)
        {
            maxValue = sumDp[j];
            maxIndex = j;
        }
    }

    // Actualizamos el resultado
    result += maxValue;

    // Actualizamos la DP de la talla que acabamos de usar
    UpdateDp(probabilities, dp, sumDp, maxIndex, n);
}

// Imprimimos el resultado con la precisión requerida
Console.WriteLine($"{result:F12}");
}

// Función para actualizar la DP de una talla después de asignarla a un ingeniero
static void UpdateDp(double[,] probabilities, double[,] dp, double[] sumDp, int size, int n)
{
    // Copiamos los valores de DP anteriores
    double[] tmp = new double[n + 1];

    // Copiamos manualmente la fila 'size' de la matriz dp
    for (int i = 0; i <= n; i++)
    {
        tmp[i] = dp[size, i];
    }

    // Recalculamos la DP para la talla 'size'
    dp[size, 0] = 0;
    for (int i = 1; i <= n; i++)
    {
        dp[size, i] = tmp[i - 1] * probabilities[i - 1, size] + dp[size, i - 1] * (1 - probabilities[i - 1, size]);
    }
}

```

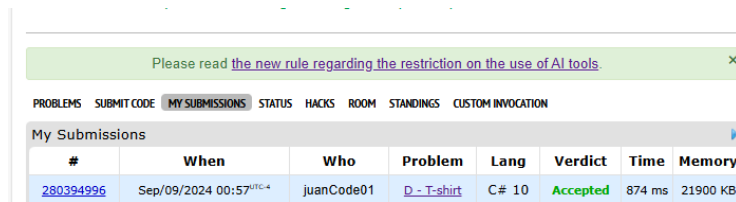
```

// Actualizamos la diferencia de probabilidades
sumDp[size] -= dp[size, n];
}

}

```

2. Casos de prueba



Please read [the new rule regarding the restriction on the use of AI tools](#).

PROBLEMS SUBMIT CODE **MY SUBMISSIONS** STATUS HACKS ROOM STANDINGS CUSTOM INVOCATION

My Submissions

#	When	Who	Problem	Lang	Verdict	Time	Memory
280394996	Sep/09/2024 00:57 ^{UTC+4}	juanCode01	D - T-shirt	C# 10	Accepted	874 ms	21900 KB

Figura 1: Tester pasados en codeforces