¿Qué es la programación dinámica?

Problema del profesor borracho 🍺

Hay n $(1 \leq n \leq 10^5)$ cervezas en una fila. La i-ésima de ellas da a_i $(1 \leq a_i \leq 10^9)$ *puntos de ebriedad*.

Un profesor desea beber muchas cervezas de forma que se maximicen sus *puntos de ebriedad*. El problema es que ya está borracho, y está **tan** borracho que al tomar una cerveza bota las dos cervezas adyacentes, rompiéndolas.

¿Cuál es el máximo *puntaje de ebriedad* posible?

Problema del profesor borracho 🍺

Hay n $(1 \leq n \leq 10^5)$ cervezas en una fila. La i-ésima de ellas da a_i $(1 \leq a_i \leq 10^9)$ *puntos de ebriedad*.

Un profesor desea beber muchas cervezas de forma que se maximicen sus *puntos de ebriedad*. El problema es que ya está borracho, y está **tan** borracho que al tomar una cerveza bota las dos cervezas adyacentes, rompiéndolas.

¿Cuál es el máximo *puntaje de ebriedad* posible?

Entrada

```
4
1 2 1 3
```

Salida

```
5
```

Problema del profesor borracho 🍺

Hay n $(1 \leq n \leq 10^5)$ cervezas en una fila. La i-ésima de ellas da a_i $(1 \leq a_i \leq 10^9)$ *puntos de ebriedad*.

Un profesor desea beber muchas cervezas de forma que se maximicen sus *puntos de ebriedad*. El problema es que ya está borracho, y está **tan** borracho que al tomar una cerveza bota las dos cervezas adyacentes, rompiéndolas.

¿Cuál es el máximo *puntaje de ebriedad* posible?

Entrada

```
3
3 5 3
```

Salida

```
6
```

Problema del profesor borracho 🍺

Hay n $(1 \leq n \leq 10^5)$ cervezas en una fila. La i-ésima de ellas da a_i $(1 \leq a_i \leq 10^9)$ *puntos de ebriedad*.

Un profesor desea beber muchas cervezas de forma que se maximicen sus *puntos de ebriedad*. El problema es que ya está borracho, y está **tan** borracho que al tomar una cerveza bota las dos cervezas adyacentes, rompiéndolas.

¿Cuál es el máximo *puntaje de ebriedad* posible?

Entrada

```
6
3 1 1 2 1 2
```

Salida

```
7
```

Memoización:

- Guarda los resultados de llamadas previas a una función para evitar cálculos repetidos.
  - Se usa con recursión
  - Top-down
  - Más directo de implementar

Memoización:

- Guarda los resultados de llamadas previas a una función para evitar cálculos repetidos.
  - Se usa con recursión
  - Top-down
  - Más directo de implementar

Tabulación:

- Calcula y almacena los resultados en orden usando estructuras como vectores o matrices.
  - Enfoque iterativo
  - Bottom-up
  - Más eficiente

Rectangle cutting (CSES 1744)

Dado un rectángulo de $a \times b$, tu misión es cortarlo en cuadrados de lado entero.

¿Cuál es la mínima cantidad de cortes?

Entrada

```
3 5
```

Salida

```
3
```

Rectangle cutting (CSES 1744)

Dado un rectángulo de $a \times b$, tu misión es cortarlo en cuadrados de lado entero.

¿Cuál es la mínima cantidad de cortes?

Entrada

```
1 4
```

Salida

```
3
```

Vocabulario

Componentes de un problema de programación dinámica

Componentes de un problema de programación dinámica

- Ecuación de recurrencia:

$$\mathrm{dp}_i = f(\mathrm{dp}_{i-1}, \mathrm{dp}_{i-2}, ..., \mathrm{dp}_0)$$

Vocabulario

Componentes de un problema de programación dinámica

- Ecuación de recurrencia:

$$\mathrm{dp}_i = f(\mathrm{dp}_{i-1}, \mathrm{dp}_{i-2}, ..., \mathrm{dp}_0)$$

- Estados. En el caso de rectangle cutting son los pares (a, b) que nos importan.

Vocabulario

Componentes de un problema de programación dinámica

- Ecuación de recurrencia:

$$\mathrm{dp}_i = f(\mathrm{dp}_{i-1}, \mathrm{dp}_{i-2}, ..., \mathrm{dp}_0)$$

- Estados. En el caso de rectangle cutting son los pares (a, b) que nos importan.
- Transiciones: Dependencias entre los estados.

Vocabulario

Componentes de un problema de programación dinámica

- Ecuación de recurrencia:

$$\mathrm{dp}_i = f(\mathrm{dp}_{i-1}, \mathrm{dp}_{i-2}, ..., \mathrm{dp}_0)$$

- Estados. En el caso de rectangle cutting son los pares (a, b) que nos importan.
- Transiciones: Dependencias entre los estados.

Si tengo n estados y m transiciones, ¿cuál es mi complejidad?

Vocabulario

Componentes de un problema de programación dinámica

- Ecuación de recurrencia:

$$\mathrm{dp}_i = f(\mathrm{dp}_{i-1}, \mathrm{dp}_{i-2}, ..., \mathrm{dp}_0)$$

- Estados. En el caso de rectangle cutting son los pares (a, b) que nos importan.
- Transiciones: Dependencias entre los estados.

Si tengo n estados y m transiciones, ¿cuál es mi complejidad?

$$O(n \cdot m)$$

(asumiendo que calcular una transición es $O(1)$)

Knapsack (problema de la mochila)

Tenemos n objetos distintos, cada uno con un valor v_i y peso w_i, y una mochila que soporta un peso máximo W.

$1 \leq n \leq 1000, 1 \leq W \leq 10^5, 1 \leq v_i, w_i \leq 1000$

¿Cuál es la suma de valor máximo que podemos llevar en la mochila sin exceder el peso W?

Entrada

```
n W
w1 w2 w3 ... wn
v1 v2 v3 ... vn
```

Salida

```
máx_valor
```

Knapsack (problema de la mochila)

Tenemos n objetos distintos, cada uno con un valor v_i y peso w_i, y una mochila que soporta un peso máximo W.

$$1 \leq n \leq 1000, 1 \leq W \leq 10^5, 1 \leq v_i, w_i \leq 1000$$

¿Cuál es la suma de valor máximo que podemos llevar en la mochila sin exceder el peso W?

Entrada

```
4 10
4 8 5 3
5 12 8 1
```

Salida

```
13
```

Código de barras

Tienes una grilla de pixeles de $n \times m$. Cada pixel puede ser blanco (`.`) o negro (#). Tu tarea es convertirlo a un código de barras cambiando la menor cantidad de pixeles.

En un código de barras válido, todos los pixeles en una columna son del mismo color, y cada barra tiene un ancho de entre x e y pixeles.

$$1 \leq n, m, x, y \leq 1000, x \leq y$$

Código de barras

Entrada

```
n W
w1 w2 w3 ... wn
v1 v2 v3 ... vn
```

Salida

```
máx_valor
```

Código de barras

Entrada

```
6 5 1 2
##.#.
.###.
###..
#...#
.##.#
###..
```

Salida

```
11
```

Nota: una solución posible es

```
.##..
.##..
.##..
.##..
.##..
.##..
```

Dado un arreglo de n enteros, tu tarea es determinar el largo de la subsecuencia (estrictamente) creciente más larga.

$$1 \leq n \leq 2 \cdot 10^5, 1 \leq a_i \leq 10^9$$

Entrada

```
8
7 3 5 3 6 2 9 8
```

Salida

```
4
```