Tramos, matrices y recursión

Programación: concepto, ejemplo y problema de juez

Carlos A Delgado S

Pontificia Universidad Javeriana Cali
Sesiones de refuerzo de algoritmia

Los temas

Cómo va cada tema

Tres partes en el mismo orden: el concepto que pide el problema, un ejemplo pequeño con su traza en una tabla, y el problema de UVa, que cada uno resuelve en el computador, en Python o en C++, hasta que pase la muestra.

Tema	Problema
El mejor tramo de un arreglo	UVa 507, *Jill Rides Again*
Rectángulos en una matriz	UVa 108, *Maximum Sum*
Recursión sobre un árbol	UVa 839, *Not so Mobile*

Compilar, correr y comparar

C++

```
g++ -std=c++17 -Wall -Wextra solucion.cpp -o solucion
./solucion < 1.in > salida.out
diff salida.out 1.out
```

Python

```
python3 solucion.py < 1.in > salida.out
diff salida.out 1.out
```

Contenido

1 El mejor tramo de un arreglo

2 Rectángulos en una matriz

3 Recursión sobre un árbol

4 Problemas para practicar

Contenido

1. El mejor tramo de un arreglo
   - Concepto
   - Ejemplo
   - Ejercicio: UVa 507
2. Rectángulos en una matriz
3. Recursión sobre un árbol
4. Problemas para practicar

Tramos y su suma

Cuántos hay

Un tramo de v es un bloque de casillas seguidas, $v[i..j]$ con $i \leq j$. Con n casillas hay $n(n+1)/2$, del orden de n^2: con 4 casillas, diez.

La suma de uno en $O(1)$

Con $acum[0] = 0$ y $acum[k+1] = acum[k] + v[k]$, la suma de $v[i..j]$ es $acum[j+1] - acum[i]$. Los acumulados se preparan con un recorrido, una sola vez.

Tramos y su suma

Con $v = [4, -5, 4, -3]$

k	0	1	2	3	4
$v[k]$	4	-5	4	-3	
$acum[k]$	0	4	-1	3	0

$$v[1..2] = acum[3] - acum[1] = 3 - 4 = -1.$$

V[1..3] acum[4] - acum[1]

0 - 4 = -4

1..1 1..2 1..3 0..1 0..2 0..3 0..0

2..1 2..2 2..3

3..3

El mejor tramo que termina en cada casilla

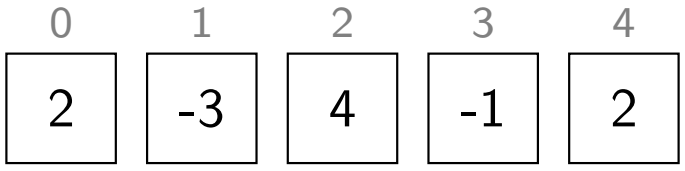
El concepto

En lugar de mirar todos los tramos, se recorre el arreglo una vez y en cada casilla se pregunta por el mejor tramo que termina ahí. Ese tramo es la casilla sola, o el mejor que terminaba en la casilla anterior más esta. Un recorrido, $\Theta(n)$, en vez de $n(n+1)/2$ tramos.

Por qué alcanza

Todo tramo termina en alguna casilla. Si para cada casilla se tiene el mejor que termina ahí, el mejor de todos está entre esos n valores.

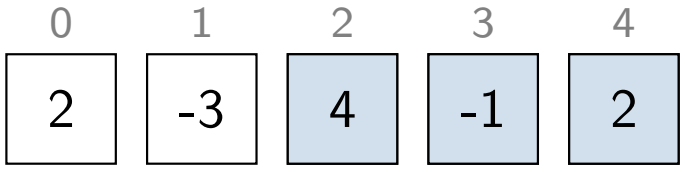
El tramo que más suma



A mano

Busque el tramo de casillas seguidas que más suma. Vale cualquier tramo, desde una casilla sola hasta el arreglo entero.

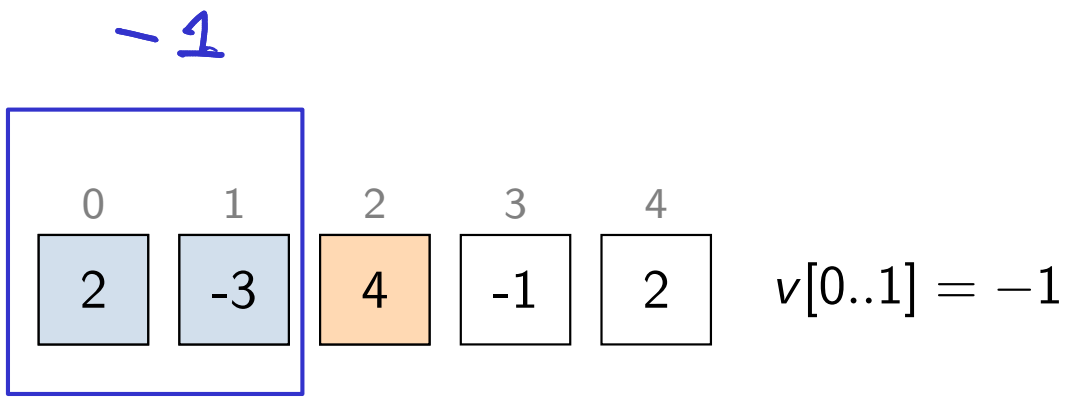
El tramo que más suma



Respuesta: 5

El tramo $v[2..4]$, con $4 - 1 + 2 = 5$. Otros candidatos: $v[0..0]$ suma 2, $v[0..4]$ suma 4, $v[2..2]$ suma 4. El -1 del medio se acepta porque lo que viene después lo paga.

¿Conviene arrastrar lo que ya se lleva?



Pregunta

El tramo que viene arrastrado desde la casilla 0 suma -1. Al llegar a la casilla 2, ¿conviene seguirlo o empezar un tramo nuevo ahí?

¿Conviene arrastrar lo que ya se lleva?

Respuesta

Empezar de nuevo. Un tramo que llega negativo solo resta: $-1 + 4$ da 3 y el que arranca en la casilla 2 da 4. La regla no depende de los números, solo del signo de lo que se lleva. Un recorrido, $\Theta(n)$, con dos variables: el tramo que termina en la casilla actual y el mejor visto.

Python: `mejor_tramo.py`

```
def mejor_tramo(v):
    mejor = v[0]
    actual = v[0]
    for k in range(1, len(v)):
        if actual < 0:
            actual = v[k]
        else:
            actual = actual + v[k]
        if actual > mejor:
            mejor = actual
    return mejor
```

La traza, paso a paso

Estado

`actual` es la mejor suma de un tramo que termina en k, y `mejor` la mejor de todas. Con $v = [2, -3, 4, -1, 2]$ las dos arrancan en $v[0] = 2$.

k	$v[k]$	¿arrastra?	`actual`	`mejor`
inicio			2	2
1	-3	sí, $2 \geq 0$	-1	2
2	4	?	?	?
3	-1	?	?	?
4	2	?	?	?

La traza, paso a paso

k	$v[k]$	¿arrastra?	`actual`	`mejor`
inicio			2	2
1	-3	sí, $2 \geq 0$	-1	2
2	4	no, $-1 < 0$	4	4
3	-1	sí, $4 \geq 0$	3	4
4	2	sí, $3 \geq 0$	5	5

Con todo negativo

En $[-4, -2, -7]$ el recorrido devuelve -2, la casilla menos mala. Arrancar `mejor` en 0 daría 0, que es la respuesta del tramo vacío: solo sirve si el enunciado acepta no tomar nada.

UVa 507: Jill Rides Again

Enunciado, en corto

Una ruta de bus tiene s paradas. Entre la parada i y la $i+1$ hay una calle con un valor n_i, distinto de cero, que dice cuánto le gusta a Jill. Ella baja en una parada, pedalea y vuelve a subir en otra. Hay b rutas, cada una con $2 \leq s \leq 20\,000$ paradas. Se pide el tramo de paradas $i < j$ con la mayor suma $n_i + \cdots + n_{j-1}$.

Desempates y salida

Si varios tramos tienen la suma máxima, gana el más largo (mayor $j - i$); si siguen empatados, el que empieza antes (menor i). Se imprime `The nicest part of route r is between stops i and j`, o `Route r has no nice parts` si la suma máxima no es positiva.

UVa 507: Jill Rides Again

Direcciones

Enunciado: `https://onlinejudge.org/external/5/507.pdf`

Envío: `https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=25&page=submit_problem&problemid=448`

¿Entendimos el problema?

Muestra

```
3
3
-1 6
10
4 -5 4 -3 4
4 -4 4 -5
4
-2 -3 -4
```

En el archivo, un número por línea.

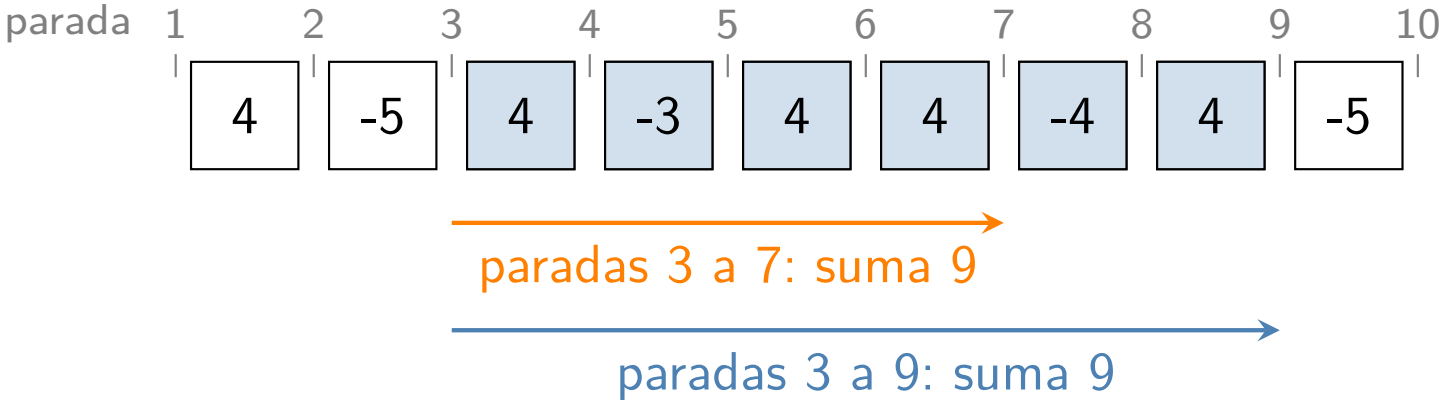
A mano

- Ruta 1: calles -1 y 6. Lo mejor es la segunda calle sola, de la parada 2 a la 3.
- Ruta 3: todas negativas. No hay tramo positivo: `Route 3 has no nice parts`.

Pregunta

En la ruta 2, el tramo de las paradas 3 a 7 suma $4 - 3 + 4 + 4 = 9$. ¿Por qué la respuesta es de la 3 a la 9?

¿Entendimos el problema?



Respuesta

Los dos suman 9 y el de 3 a 9 es más largo: las dos calles que agrega, -4 y 4, suman cero. Es el primer desempate.

Índices

`v[k]` une las paradas $k+1$ y $k+2$, así que el tramo de calles $v[i..j]$ va de la parada $i+1$ a la $j+2$.

A resolver en el computador

El resultado

El programa, en Python o en C++, que pasa la muestra con `diff`.

Contenido

1 El mejor tramo de un arreglo

2 Rectángulos en una matriz
- Concepto
- Ejemplo
- Ejercicio: UVa 108

3 Recursión sobre un árbol

4 Problemas para practicar

Una matriz y lo que cuesta

Cómo se guarda

En Python, una lista de listas: `m[i]` es la fila i y `m[i][j]` la celda. En C++, `vector<vector<int>> m(n, vector<int>(n));` crea n filas de n ceros.

Lo que cuesta

`m[i][j]` es $O(1)$. Recorrer una matriz de $f \times c$ es $\Theta(f \cdot c)$; sumar una fila es $\Theta(c)$ y una columna, $\Theta(f)$.

Cada fila por separado

En Python, `[[0] * n] * n` guarda n veces la misma fila, y escribir en una cambia todas. Las filas se crean una por una, con `append` dentro de un ciclo.

Acumulados por columna

Pregunta

$col[c]$ es la suma de la columna c entre las filas `arriba` y `abajo`. Si `abajo` baja una fila, ¿hay que sumar todo otra vez?

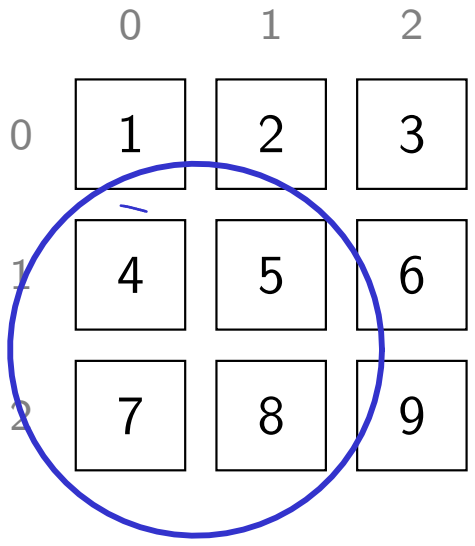
Respuesta

No: a cada columna se le suma la celda nueva, $col[c] \mathrel{+}= m[abajo][c]$. Cuesta $\Theta(n)$ por fila.

Con la matriz de la muestra de UVa 108 y $arriba = 1$

`abajo`	$col[0]$	$col[1]$	$col[2]$	$col[3]$
1	9	2	-6	2
2	5	3	-10	3
3	4	11	-10	1

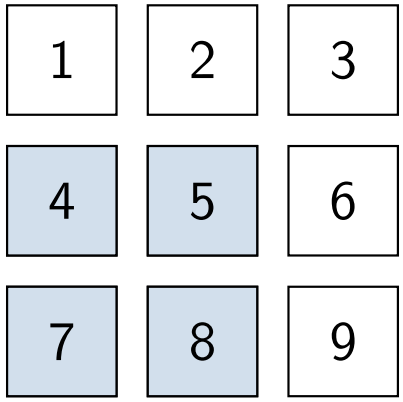
La suma de un rectángulo



A mano

¿Cuánto suma el rectángulo que va de la fila 1 a la 2 y de la columna 0 a la 1?

La suma de un rectángulo



Respuesta: 24

$4 + 5 + 7 + 8$. Recorrer las cuatro celdas es inmediato.

Pregunta

¿Y si hay que responder esa misma cuenta para miles de rectángulos de una matriz de 100×100?

La forma directa

Recorrer cada rectángulo

Dos ciclos anidados sobre las filas y las columnas del rectángulo: hasta $N \cdot M$ celdas por consulta.

Con muchas consultas

En una matriz de 100×100, un rectángulo grande cuesta 10^4 sumas. Mil consultas son 10^7, y cada consulta vuelve a sumar celdas que la anterior ya sumó.

Pregunta

En una dimensión, la suma de un tramo salía en $O(1)$ con los acumulados. ¿Qué sería el acumulado de una matriz?

La tabla de acumulados

Respuesta

Una tabla P con una fila y una columna más que la matriz: $P[i][j]$ acumula todo lo que queda arriba y a la izquierda de la celda (i, j). La fila 0 y la columna 0 quedan en ceros, que es el tramo vacío de una dimensión visto en dos.

Python: `acumulados2d.py`

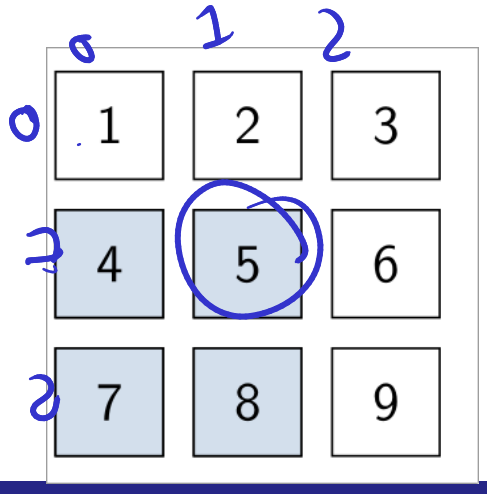
```
def tabla_acumulados(m):
    p = []
    for i in range(len(m) + 1):
        p.append([0] * (len(m[0]) + 1))
    for i in range(len(m)):
        for j in range(len(m[0])):
            p[i + 1][j + 1] = m[i][j] + p[i][j + 1] + p[i + 1][j] - p[i][j]
    return p
```

Llenar la tabla

Estado

Con la matriz 1 a 9, cada celda es $m[i][j]$ más la de arriba y la de la izquierda en P, menos la de la diagonal.

P	0	1	2	3
0	0	0	0	0
1	0	1	3	6
2	0	5	?	?
3	0	?	?	?

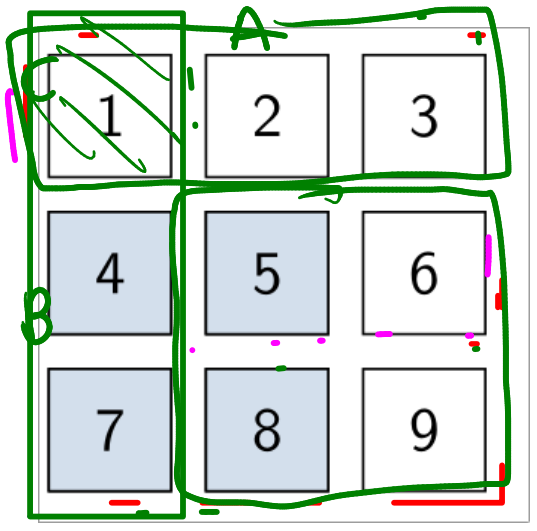


Para la casilla $P[2][2]$

$m[1][1] + P[1][2] + P[2][1] - P[1][1] = 5 + 3 + 5 - 1$. El rectángulo de arriba y el de la izquierda comparten una parte, que entra dos veces y por eso se descuenta.

Llenar la tabla

P	0	1	2	3
0	0	0	0	0
1	0	1	3	6
2	0	5	12	21
3	0	12	27	45



La esquina de abajo a la derecha

$P[3][3] = 45$ es la suma de toda la matriz, de 1 a 9.

Cuatro lecturas por rectángulo

Python: `acumulados2d.py`

```python
def suma_rectangulo(p, f1, c1, f2, c2):
    total = p[f2 + 1][c2 + 1] - p[f1][c2 + 1] - p[f2 + 1][c1] + p[f1][c1]
    return total
```

El rectángulo de hace un momento

Filas 1 a 2, columnas 0 a 1:
$P[3][2] - P[1][2] - P[3][0] + P[1][0] = 27 - 3 - 0 + 0 = 24.$

Las dos trampas

Si se olvida sumar de vuelta la esquina, el resultado se queda corto por la parte que se restó dos veces. Y los índices van corridos en uno: la celda (i, j) de la matriz vive en $P[i + 1][j + 1]$.

UVa 108: Maximum Sum

Enunciado, en corto

Una matriz de $N \times N$ enteros, positivos y negativos. Se pide la mayor suma de un subrectángulo: un bloque de filas y columnas seguidas, de 1×1 en adelante.

Límites

$N \leq 100$; cada valor en $[-127, 127]$. Tras N vienen los N^2 números fila por fila, separados por espacios y saltos de línea en cualquier reparto.

Direcciones

Enunciado: `https://onlinejudge.org/external/1/108.pdf`

Envío: `https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=25&page=submit_problem&problemid=44`

¿Entendimos el problema?

Muestra, tal como viene

```
4
0 -2 -7 0 9 2 -6          2
-4 1 -4 1 -1
8 0 -2
```

0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

A mano

Sale un número: el bloque marcado, filas 1 a 3 y columnas 0 y 1, suma $9 + 2 - 4 + 1 - 1 + 8 = 15$.

Pregunta

La segunda línea trae 8 números y la última 3. ¿Qué dice eso de cómo leer?

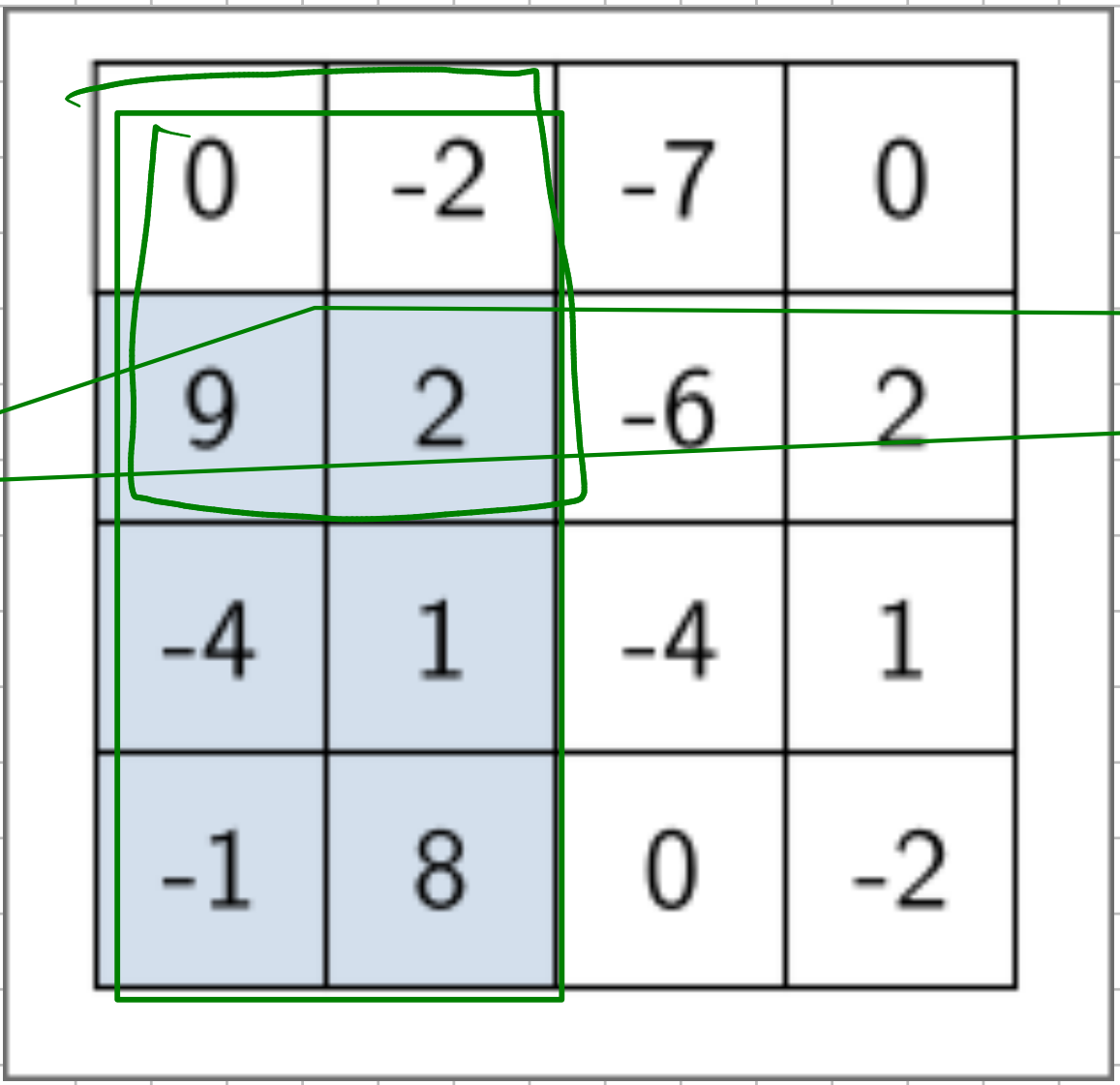
Respuesta

Que se leen N^2 números por palabras; la celda (i, j) es el número $i \cdot N + j$.

A resolver en el computador

El resultado

El programa, en Python o en C++, que pasa la muestra con `diff`.



0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

arr = 1

major = 0

col = (0, -2, -7, 0)

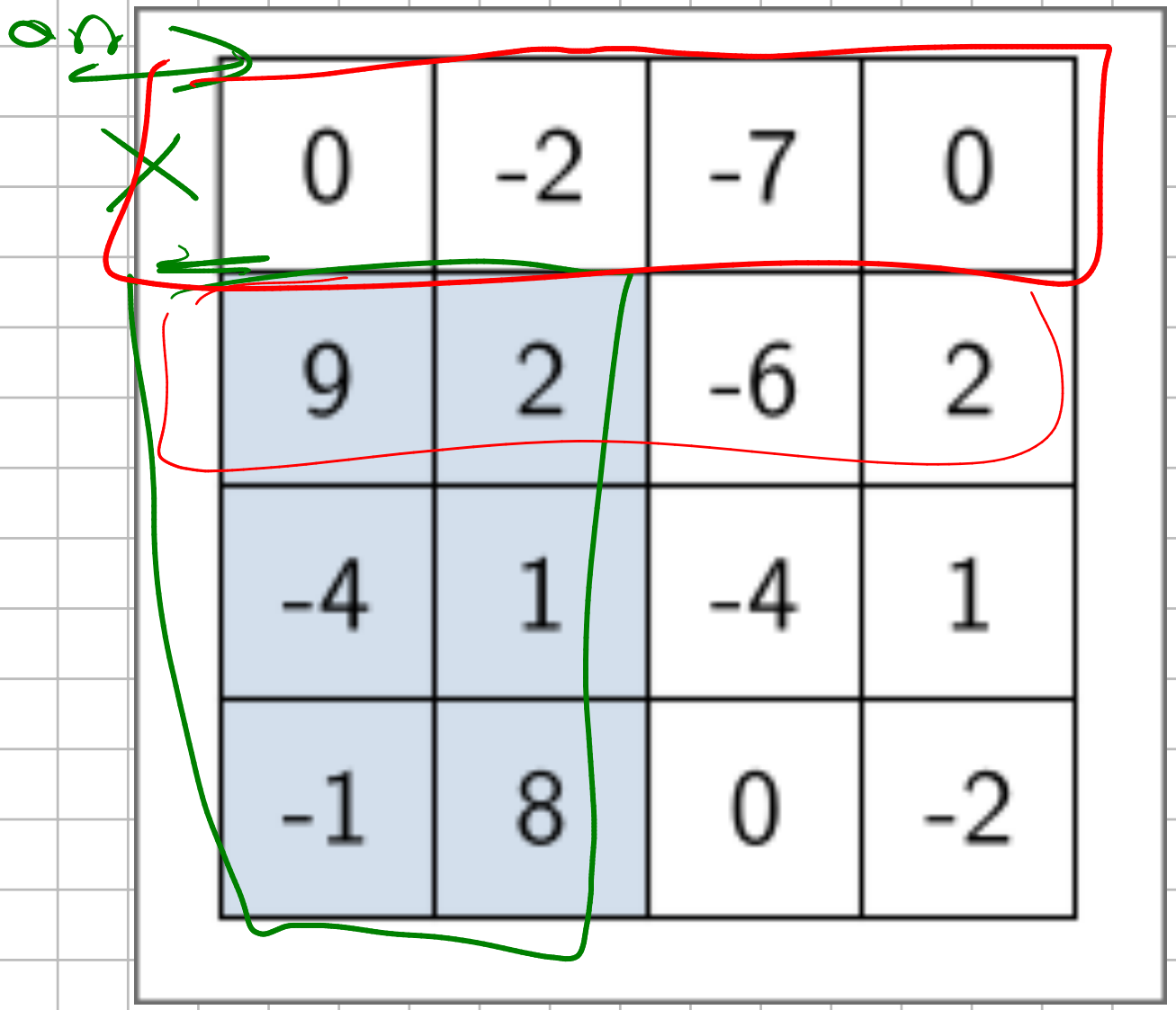
col = (9, 0, -13, 2)

major = 9

col = (5, 1, -17, 3)

col = (4, 9, -17, -1)

major = 13



arr

col = (9, 2, 6, 2)

major = 13

col = (5, 3, -10, 3)

col = (4, 11, -10, -1)

major = 15

Contenido

1 El mejor tramo de un arreglo

2 Rectángulos en una matriz

3 Recursión sobre un árbol
- Concepto
- Ejemplo
- Ejercicio: UVa 839

4 Problemas para practicar

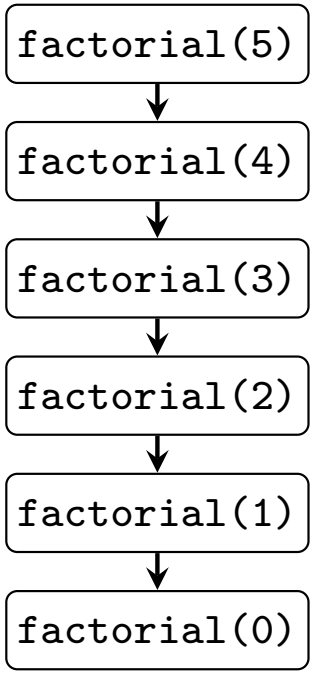
La pila de llamadas, en el depurador

De las estructuras y su costo

Cada llamada empuja un marco con sus parámetros y sus variables locales, y al terminar lo saca. La pila tiene un tamaño fijo, típicamente 8 MB en Linux, y Python corta a los 1000 niveles. En el depurador, `factorial(5)` detenido en el caso base:

```
(gdb) backtrace
#0  factorial (n=0) at factorial.cpp:8
#1  0x0000555555555168 in factorial (n=1) at factorial.cpp:10
#2  0x0000555555555168 in factorial (n=2) at factorial.cpp:10
#3  0x0000555555555168 in factorial (n=3) at factorial.cpp:10
#4  0x0000555555555168 in factorial (n=4) at factorial.cpp:10
#5  0x0000555555555168 in factorial (n=5) at factorial.cpp:10
#6  0x0000555555555193 in main () at factorial.cpp:17
```

Del `backtrace` al árbol de llamadas



El árbol

Cada llamada es un nodo y sus hijos son las llamadas que hace, de izquierda a derecha en el orden en que las hace. `factorial` hace una sola llamada, así que su árbol es una cadena de seis nodos.

La pila es un camino del árbol

En cada momento, la pila guarda el camino desde la raíz hasta la llamada que está corriendo: una llamada pendiente por nivel. El `backtrace` de arriba es ese camino leído de abajo hacia arriba.

Del `backtrace` al árbol de llamadas

Los dos casos

`n == 0` es el caso base: devuelve 1 sin llamar a nadie. El otro es el caso recursivo, `n * factorial(n - 1)`, que llama sobre una entrada más pequeña y así llega al base.

Lo que se lee en el árbol

- El número de nodos es el número de llamadas. Si cada llamada trabaja una cantidad constante fuera de sus llamadas, ese número es el costo.
- La altura del árbol, más uno, es el mayor número de marcos en la pila a la vez: lo que decide si la recursión cabe en 8 MB o en los 1000 niveles de Python.
- El orden en que se visitan los nodos, de arriba hacia abajo y de izquierda a derecha, es el orden en que se hacen las llamadas.

Dos llamadas por nivel

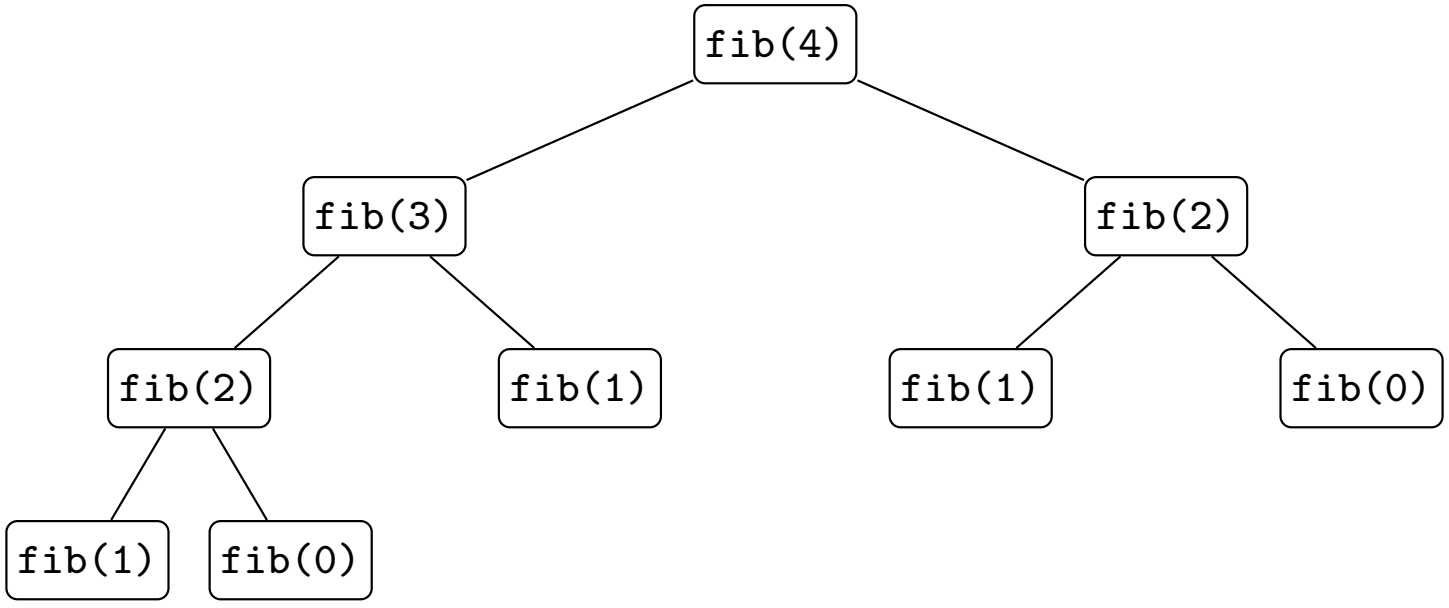
Python: `fib.py`

```python
def fib(n):
    if n <= 1:
        r = n
    else:
        r = fib(n - 1) + fib(n - 2)
    return r
```

Pregunta

Dibuje el árbol de llamadas de `fib(4)`. ¿Cuántas llamadas hay? ¿Cuántos marcos de `fib` hay en la pila como máximo?

El árbol de `fib(4)`



Respuesta

Nueve llamadas, y la función devuelve 3. El camino más largo baja por `fib(4)`, `fib(3)`, `fib(2)` y `fib(1)`: cuatro marcos a la vez. `fib(2)` se calcula dos veces, una en cada rama.

La pila mientras se recorre el árbol

Paso	Pila, del fondo al tope	Devuelve
1	`fib(4)`	
2	`fib(4)`, `fib(3)`	
3	`fib(4)`, `fib(3)`, `fib(2)`	
4	`fib(4)`, `fib(3)`, `fib(2)`, `fib(1)`	1
5	?	?
6	?	?
7	`fib(4)`, `fib(2)`	
8	?	?
9	?	?

Pregunta

En el paso 6, ¿qué llamada entra y qué tiene encima de ella?

La pila mientras se recorre el árbol

Paso	Pila, del fondo al tope	Devuelve
1	`fib(4)`	
2	`fib(4)`, `fib(3)`	
3	`fib(4)`, `fib(3)`, `fib(2)`	
4	`fib(4)`, `fib(3)`, `fib(2)`, `fib(1)`	1
5	`fib(4)`, `fib(3)`, `fib(2)`, `fib(0)`	0
6	`fib(4)`, `fib(3)`, `fib(1)`	1
7	`fib(4)`, `fib(2)`	
8	`fib(4)`, `fib(2)`, `fib(1)`	1
9	`fib(4)`, `fib(2)`, `fib(0)`	0

En el paso 6

`fib(2)` ya devolvió 1 y salió de la pila. Entra `fib(1)`, el segundo hijo de `fib(3)`, y no tiene nada encima: es un caso base.

UVa 839: Not so Mobile

Enunciado, en corto

Un móvil es una barra colgada de un hilo, con un objeto en cada extremo. Está en equilibrio si $W_l \times D_l = W_r \times D_r$, con los pesos W y las distancias D al punto de donde cuelga. En lugar de un objeto, de un extremo puede colgar otro móvil, y su peso es la suma de los pesos de sus objetos. Se pide decidir si todo el móvil está en equilibrio: cada una de sus barras.

Entrada

El número de casos, una línea en blanco, y entre caso y caso otra línea en blanco. Cada barra es una línea `Wl Dl Wr Dr`. Un peso 0 dice que de ese extremo cuelga un submóvil, descrito en las líneas que siguen: primero el izquierdo y después el derecho.

UVa 839: Not so Mobile

Salida

`YES` o `NO` por caso, con una línea en blanco entre dos casos.

Direcciones

Enunciado: `https://onlinejudge.org/external/8/839.pdf`

Envío: `https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=25&page=submit_problem&problemid=780`

¿Entendimos el problema?

Muestra

```
1

0 2 0 4
0 3 0 1
1 1 1 1
2 4 4 2
1 6 3 2
```

Salida

```
YES
```

Pregunta

La primera barra tiene dos ceros. ¿Qué líneas describen el submóvil izquierdo y cuáles el derecho?

Pregunta

¿Cuánto pesa el submóvil izquierdo?

¿Entendimos el problema?

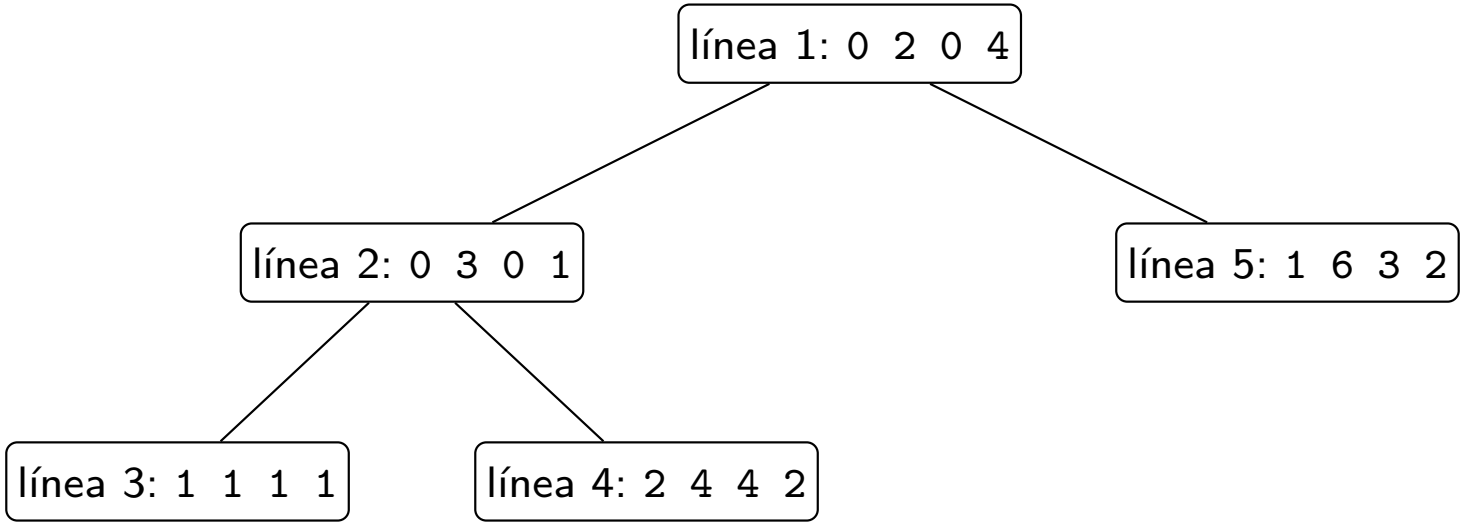
Respuesta

La línea 2, `0 3 0 1`, también tiene dos ceros: su submóvil izquierdo es la línea 3 y su derecho la línea 4. Con eso termina el submóvil izquierdo de la línea 1, y la línea 5 es el derecho. Las líneas no dicen dónde termina cada submóvil: hay que leerlas en orden y saber cuántas consume cada uno.

El peso

La línea 3 pesa $1 + 1 = 2$ y la 4, $2 + 4 = 6$; el submóvil izquierdo pesa 8. Su barra está en equilibrio porque $2 \cdot 3 = 6 \cdot 1$.

El móvil de la muestra



Las líneas y el árbol

Cada barra es un nodo, y cada peso 0 es un hijo. La entrada lista los nodos en el orden de arriba hacia abajo y de izquierda a derecha: la barra, después todo su lado izquierdo y después todo su lado derecho.

A resolver en el computador

El resultado

El programa, en Python o en C++, que pasa la muestra con `diff`.

Contenido

1 El mejor tramo de un arreglo

2 Rectángulos en una matriz

3 Recursión sobre un árbol

4 Problemas para practicar

Cómo se trabajan

Cómo se trabajan

Los tres se resuelven y se envían al juez. Van en orden de dificultad. Cada solución va con su costo: de qué depende el tamaño de la entrada, cuántas veces corre cada ciclo o cuántas llamadas hay, y el total en O frente a los límites del enunciado.

Python en UVa

El juez corre una versión antigua de Python 3: la salida se arma con `print` y con `%`, como en las soluciones de los problemas de clase.

Base. UVa 100: The 3n + 1 problem

Direcciones

Enunciado: `https://onlinejudge.org/external/1/100.pdf`

Envío: `https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=25&page=submit_problem&problemid=36`

Cómo atacarlo

Una función aplica la regla hasta llegar a 1 y cuenta los números que imprime; otra recorre el rango del menor al mayor y se queda con el máximo. La longitud cuenta el 1, así que la de 22 es 16. La entrada se lee hasta fin de archivo.

La trampa

i puede llegar mayor que j: el recorrido va del menor al mayor, pero la salida repite i y j en el orden en que llegaron. Los valores intermedios pasan de 10 000, así que no sirven como posición de un arreglo del tamaño de la entrada.

Intermedio. UVa 10189: Minesweeper

Direcciones

Enunciado: `https://onlinejudge.org/external/101/10189.pdf`

Envío: `https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=25&page=submit_problem&problemid=1130`

Cómo atacarlo

Por cada casilla libre se cuentan las minas de sus ocho vecinas, con los dos desplazamientos recorriendo $-1, 0, 1$ y la comprobación de borde antes del acceso. Las casillas con mina se copian tal cual. El primer número son las filas y el segundo las columnas.

La trampa

Está en la salida: la línea en blanco va entre campos y no después del último. Los bordes y las esquinas tienen menos de ocho vecinas, y un campo de 1×1 no tiene ninguna.

Exigente. UVa 620: Cellular Structure

Direcciones

Enunciado: `https://onlinejudge.org/external/6/620.pdf`

Envío: `https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=25&page=submit_problem&problemid=561`

Cómo atacarlo

Una cadena sana es `A`, una sana seguida de `AB`, o `B` seguida de una sana y de `A`. La función recibe el rango [*ini*, *fin*]: el caso base es una letra, y la primera y la última letra deciden qué caso recursivo se intenta. Cada llamada hace a lo más una llamada, sobre un tramo dos letras más corto, sin copiar la cadena.

Exigente. UVa 620: Cellular Structure

La trampa

Lo de adentro tiene que ser una cadena sana y no vacía: `AB` y `BA` parecen formas válidas y son `MUTANT`. Contar letras no sirve, porque el interior se clasifica completo: `BAABA` es `MUTAGENIC` porque `AAB` es sana, y `BAAB` es `MUTANT` porque `BA` no lo es.

Para cerrar

La altura del móvil

Escriba en Python y en C++ una función recursiva que lea un móvil con el formato de UVa 839 y devuelva cuántos niveles tiene. Pruébela con la muestra, que tiene tres. ¿Cuántas llamadas hace con un móvil de n barras, y cuántos marcos llegan a estar en la pila a la vez?