Trabajo práctico 1

Fecha límite de entrega: Viernes 9 de septiembre, hasta las 17:00 hs.
Fecha estimada de devolución: Tres semanas después de la fecha en la que es entregado el TP.
Primer fecha de reentrega: Dos semanas después de devueltas las correcciones, hasta las 17:00 (viernes) o hasta las 22 (martes).
Segunda fecha de reentrega: Viernes 9 de diciembre, hasta las 17:00 hs.

Este trabajo práctico consta de 4 problemas. Para aprobar el mismo se requiere aprobar todos los problemas.
Los requisitos de aprobación del mismo así como los criterios de corrección están detallados en las pautas de aprobación que pueden encontrar en el repositorio de la materia.

En este trabajo práctico se evaluarán contenidos de técnicas algoritmicas. Las técnicas que serán evaluadas en el mismo serán:

- Backtracking
- Programación dinámica
- Algoritmos Golosos
- Divide and Conquer

Problema 1: La tienda de Apu

A Homero le dio hambre y quiere ir a la tienda de Apu a comprar donas. Como a Homero le gusta mucho comer, quiere comer la mayor cantidad de donas posible. Homero cuenta con un billete de P pesos.

Al llegar al Kwik-e-mart, Homero descubre que Apu sólo tiene en stock N packs de donas, y las cantidades de donas de cada pack son $D_1, D_2, \ldots, D_N$. Un pack con D_i donas cuesta D_i pesos, por lo que Homero va a poder comprar cuantas donas quiera siempre y cuando no compre más de P (si no, no va a poder pagarlas). No puede comprar fracciones de packs: o los compra enteros o no compra ninguna dona del pack, pero puede elegir qué packs comprar.

Su tarea es ayudar a Homero, y decirle cuántas donas puede comprar como máximo, sabiendo que no puede comprar más de P donas.

El algoritmo debe tener una complejidad temporal $\mathbf{O}(N \times 2^{\frac{N}{2}})$, siendo N la cantidad de packs de donas.

Entrada

La primera línea consta de un valor entero positivo P, que indica el presupuesto de Homero, y un entero positivo N, que indica la cantidad de packs que tiene Apu en su tienda.

A esta línea le sigue una línea con N enteros positivos, indicando los tamaños D_i de cada pack.

La entrada contará con el siguiente formato:

```
P N
D1 D2 ... DN
```

Salida

La salida debe constar de una línea que indique la máxima cantidad de donas que le puede comprar Homero a Apu, con el siguiente formato:

```
C
```

siendo C esta cantidad.

Entrada de ejemplo 1	**Salida para la entrada de ejemplo 1**
10 4 3 3 3 3	9

Entrada de ejemplo 2	**Salida para la entrada de ejemplo 2**
12 5 1 2 3 4 5	12

Explicación del ejemplo 1:
Homero tiene a su disposición 4 packs de 3 donas cada uno. Si compra 3 de los 4 packs se lleva 9 donas, y no los puede comprar todos porque no le alcanza su presupuesto para comprar 12 donas.

Explicación del ejemplo 2:
En este caso a Homero le conviene comprar todos los packs menos el pack de 3 donas, y de esa manera llevarse 12 donas, o comprar los packs de 3, 4 y 5 donas, y también se lleva de esa manera 12 donas.

Pesos mínimo y máximo: 8 y 10.

Cotas recomendadas para testear: Se recomienda testear el problema con valores de $P \leq 10^9$, $N \leq 40$, $D_i \leq 10^9$

Pista: No se conocen algoritmos polinomiales para resolver este problema.

Problema 2: El cumpleaños de Lisa

Se acerca el cumpleaños de Lisa y ella lo quiere festejar. Como Lisa siempre quiso ser muy popular, quiere que sus fiestas sean lo más divertidas posible.

Lisa tiene N amigas que quiere invitar a su cumpleaños, pero no todas se llevan bien entre sí, y entre las que se llevan bien hay algunas que se llevan mejor que otras. Lisa sabe qué tan bien se llevan cada una de sus amigas con todas las demás, y en función de esto quiere organizar varias fiestas. Lisa puede organizar cualquier cantidad de fiestas entre 1 y N.

Para cada una de las amigas de Lisa, que llamaremos $A_1, \ldots, A_N$, sabemos qué tan bien se lleva con cada una de sus demás amigas, y sabemos también que la diversión en una fiesta es proporcional a qué tan bien se llevan los invitados. Las amigas A_i y A_j aportan $D_{i,j}$ diversión si coinciden en una misma fiesta. La presencia de Lisa no modifica la diversión que hay en una fiesta y Lisa siempre está presente en todas sus fiestas. Si $D_{i,j} < 0$ entonces la diversión de una fiesta donde coinciden las amigas A_i y A_j disminuye.

Si Lisa quiere que cada una de sus amigas vaya a exactamente una de sus fiestas de cumpleaños, ¿Cuál es la máxima diversión que pueden sumar las fiestas de Lisa?

Su tarea es ayudar a Lisa, y calcular esta cantidad máxima de diversión que puede tener sumando todas sus fiestas de cumpleaños.

El algoritmo debe tener una complejidad temporal $\mathbf{O}(3^N)$, siendo N la cantidad de amigas de Lisa.

Entrada

La primera línea consta de un valor entero positivo N, que indica la cantidad de amigas que Lisa quiere invitar a su cumpleaños.

A esta línea le siguen N líneas con N enteros cada una, indicando los números $D_{i,j}$. Se asegura que $D_{i,i} = 0$ para todo $1 \leq i \leq N$ y que $D_{i,j} = D_{j,i}$ para todos $1 \leq i, j \leq N$.

La entrada contará con el siguiente formato:

```
N
D11 D12 ... D1N
D21 D22 ... D2N
...
DN1 DN2 ... DNN
```

Salida

La salida debe constar de una línea que indique la máxima diversión que puede obtener Lisa en su cumpleaños, con el siguiente formato:

```
F
```

siendo F esta cantidad de diversión.

Entrada de ejemplo 1	**Salida para la entrada de ejemplo 1**
3 0 -1 1 -1 0 -1 1 -1 0	1

Entrada de ejemplo 2	**Salida para la entrada de ejemplo 2**
4 0 2 -1 0 2 0 0 -1 -1 0 0 2 0 -1 2 0	4

Entrada de ejemplo 3	Salida para la entrada de ejemplo 3
4 0 2 2 -1 2 0 0 2 2 0 0 2 -1 2 2 0	7

Explicación del ejemplo 1:
La amiga A_2 se lleva mal con todas las demás y su presencia en frente a otras amigas siempre disminuye la diversión de la fiesta, por lo que Lisa la piensa invitar a una fiesta exclusiva para ella donde no haya otras amigas presentes. Al ser sólo una amiga, la diversión de esa fiesta es 0. A A_1 y A_3 conviene invitarlas juntas para que aporten diversión y vayan a una fiesta con diversión 1. La máxima diversión posible es entonces $0 + 1 = 1$.

Explicación del ejemplo 2:
Las amigas A_1 y A_2 se llevan muy bien juntas y las queremos invitar a la misma fiesta, lo mismo pasa con las amigas A_3 y A_4, luego podemos tener una diversión de 4. Además, cualquier otro par de amigas que juntemos en una misma fiesta no van a aportar diversión.

Explicación del ejemplo 3:
Si juntamos a las 4 amigas en una fiesta la diversión de la misma es 7. Si quisieramos separar a dos amigas en fiestas distintas, sería a las que restan diversión si van juntas (A_1 y A_4, ya que cualquier otro par de amigas que separemos resta diversión separarlas). Al separar a estas amigas, la máxima diversión posible sería 8, pero como la amiga A_2 no puede estar en las dos fiestas, hay 2 de diversión (los 2 que aporta si está con A_1 o los que aporta si está con A_4) que no los va a poder aportar, por lo que la diversión de 8 nunca se alcanza y luego 7 es el máximo.

Pesos mínimo y máximo: 8 y 10.

Cotas recomendadas para testear: Se recomienda testear el problema con valores de $N \leq 18$, $|D_{i,j}| \leq 10^7$.

Pista: A la hora de calcular complejidades, pueden usar que $2^N P(N) \subseteq \mathbf{O}(3^N)$ siendo $P(N)$ un polinomio en la variable N sin necesidad de demostrarlo.

Problema 3: Experimentos nucleares

El señor Burns está haciendo experimentos en su planta nuclear de Springfield. Para poder generar energía de la manera más económica posible, y maximizar así su ganancia, ha contratado al profesor Frink para que le resuelva un problema que le está generando un gasto innecesario de energía.

El profesor, gracias a sus conocimientos de ciencia, ha reducido el problema de la planta nuclear a un sencillo problema de sumas y máximos, pero que por falta de tiempo no puede resolver. El problema que tiene ahora el señor Burns es el siguiente: dado un arreglo de números enteros, debe hallar el subarreglo de suma máxima, es decir, dados N enteros $A_1, ..., A_N$, encontrar el i y el j $(i \leq j)$ tales que $A_i + A_{i+1} + ... + A_{j-1}$ sea lo más grande posible.

Su tarea es ayudar al señor Burns, encontrando el valor de la suma que nuestro millonario amigo quiere maximizar.

El algoritmo debe tener una complejidad temporal $\mathbf{O}(N)$, siendo N la cantidad de números del arreglo.

Entrada

La primera línea consta de un valor entero positivo N, que indica la cantidad de números del arreglo.

A esta línea le sigue una línea con N enteros cada una, indicando los números A_i del arreglo.

La entrada contará con el siguiente formato:

```
N
A1 A2 ... AN
```

Salida

La salida debe constar de una línea que indique la máxima suma posible de un subarreglo, con el siguiente formato:

```
S
```

siendo S la máxima suma posible.

Entrada de ejemplo 1	**Salida para la entrada de ejemplo 1**
3 2 -1 2	3

Entrada de ejemplo 2	**Salida para la entrada de ejemplo 2**
4 -1 -1 -1 -1	0

Entrada de ejemplo 3	**Salida para la entrada de ejemplo 3**
14 1 2 3 4 5 -1 -2 -3 -4 -5 1 2 3 4	15

Explicación del ejemplo 1:
Cualquier suma que no incluya al primer y al tercer elemento del arreglo será menor a 3, luego conviene tomar el arreglo entero y sumar 3 (suma que no puede ser superada por la suma de otro subarreglo).
Explicación del ejemplo 2:
En este caso conviene tomar el subarreglo vacío.
Explicación del ejemplo 3:
En este caso conviene tomar los primeros 5 números.

Pesos mínimo y máximo: 7 y 9.

Cotas recomendadas para testear: Se recomienda testear el problema con valores de $N \leq 10^5$.

Problema 4: El error de Smithers

Por un error de comunicación, Smithers entendió mal el resultado de la investigación científica del profesor Frink. El problema era un poco más complejo de lo que parecía, y no era necesario maximizar una suma en un arreglo sino hacer algo totalmente distinto. Cuando el profesor Frink le habló de matrices de 3×3 Smithers anotó números enteros, y cuando dijo "encontrar si el producto de un subarreglo de longitud exactamente L es igual a una matriz M dada"Smithers anotó "maximizar".

La conclusión que podemos sacar de esto es que Smithers hizo cualquier cosa menos lo que tenía que hacer. Ahora que nos avisó de esto, queremos resolver el problema que nos planteó el profesor pero sin dejar de resolver el caso anterior (porque nunca se sabe si en realidad estaba bien la versión anterior y lo que nos plantea ahora Smithers es en realidad lo que entendió mal).

Su tarea es ayudar a Smithers a arreglar su error, dándole un algoritmo que resuelva el siguiente problema: dada una matriz $M \in \mathbb{Z}_{10007}^{3\times3}$ y un arreglo de N matrices en $\mathbb{Z}_{10007}^{3\times3}$, decidir si existe un subarreglo de longitud L cuyo producto sea igual a M.

$\mathbb{Z}_{10007}$ son los enteros módulo 10007 (en el que cada operación se calcula módulo 10007) y $X^{3\times3}$ son las matrices de 3 filas y 3 columnas con coeficientes en X, en este caso, con $X = \mathbb{Z}_{10007}$

El algoritmo debe tener una complejidad temporal $\mathbf{O}(N \log N)$, siendo N la cantidad de matrices del arreglo.

Como este TP no se trata sobre matrices, van a tener disponible un archivo matriz.h con una implementación de matrices de 3×3 con todas las operaciones necesarias para resolver este problema. Además, pueden considerar que, al ser matrices de tamaño fijo, todas estas operaciones son $\mathbf{O}(1)$. Si utilizan C++ para este problema pueden incluir este archivo en su código, y si utilizan otro lenguaje pueden usar el mismo para entender cómo hacer cada operación y poder programarlas de la misma manera en el lenguaje que elijan.

Entrada

La primera línea consta de un valor entero positivo N, que indica la cantidad de matrices del arreglo, y un entero $1 \leq L \leq N$, que representa la longitud del subarreglo buscado de la manera que se describe en el enunciado.

A esta línea le sigue una línea con los 9 coeficientes de la matriz M (en el orden $M_{1,1}, M_{1,2}, M_{1,3}, M_{2,1}, M_{2,2}, M_{2,3}, M_{3,1}, M_{3,2}, M_{3,3}$), y N líneas con los 9 coeficientes de cada una de las matrices del arreglo. Todos los coeficientes de la entrada serán enteros entre 0 y 10006 inclusive.

La entrada contará con el siguiente formato:

```
N L
M11 M12 M13 M21 M22 M23 M31 M32 M33
A11 A12 A13 A21 A22 A23 A31 A32 A33
...
Z11 Z12 Z13 Z21 Z22 Z23 Z31 Z32 Z33
```

siendo A la letra que usamos para simplificar el formato de entrada, y que representa la matriz 1 del arreglo y Z lo mismo con la matriz N, pero esto no quiere decir $N = 26$, es sólo notación que se usa para explicar el formato de entrada.

Salida

La salida debe constar de una línea que indique la máxima suma posible de un subarreglo, con el siguiente formato:

```
R
```

siendo R la respuesta, que puede ser el string "`SI`", o el string "`NO`"(sin las comillas).

Entrada de ejemplo 1	Salida para la entrada de ejemplo 1
3 2 1 0 0 0 1 0 0 0 1 2 0 0 0 1 0 0 0 3 1 1 1 2 2 2 3 3 3 0 1 0 1 0 0 0 0 1	NO

Entrada de ejemplo 2	Salida para la entrada de ejemplo 2
4 2 30 24 18 84 69 54 138 114 90 1 0 0 0 1 0 0 0 1 1 2 3 4 5 6 7 8 9 9 8 7 6 5 4 3 2 1 2 0 1 1 2 0 0 1 2	SI

Entrada de ejemplo 3	Salida para la entrada de ejemplo 3
4 3 1 0 0 0 1 0 0 0 1 5 0 0 0 5 0 0 0 5 1 0 0 0 1 0 0 0 1 4003 0 0 0 4003 0 0 0 4003 1 0 0 0 1 0 0 0 1	SI

Entrada de ejemplo 4	Salida para la entrada de ejemplo 4
4 2 30 24 18 84 69 54 138 114 90 1 2 3 4 5 6 7 8 9 1 0 0 0 1 0 0 0 1 9 8 7 6 5 4 3 2 1 2 0 1 1 2 0 0 1 2	NO

Explicación del ejemplo 1:
Como necesitamos usar dos matrices, y tenemos 3, la del medio (la segunda) la vamos a tener que usar obligatoriamente. Como esta matriz no es inversible no podemos usarla para formar la identidad utilizando sólo productos.

Explicación del ejemplo 2:
Las matrices 2 y 3 tienen como producto la matriz M:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 30 & 24 & 18 \\ 84 & 69 & 54 \\ 138 & 114 & 90 \end{pmatrix}$$

Explicación del ejemplo 3:
Recordemos que los coeficientes de la matriz son números en $\mathbb{Z}_{10007}$, no es necesario que se preocupen por esto ya que la multiplicación y el inverso módulo 10007 están ya implementados en matriz.h (aunque si les interesa pueden mirar esta implementación para ver cómo se hacen las cuentas).

Explicación del ejemplo 4:
Este ejemplo es igual que el 2, pero ahora las matrices que hay que elegir son la primera y la tercera, y al no ser consecutivas no podemos elegirlas, ya que sólo podemos elegir dos matrices consecutivas.

Pesos mínimo y máximo: 7 y 11.

Cotas recomendadas para testear: Se recomienda testear el problema con valores de $N \leq 10^4$.

Pista 1: Si usan las matrices implementadas en matriz.h pueden abstraerse de como se multiplican matrices y solo usar el producto de matrices.

Pista 2: Si intentan dividir matrices no siempre lo van a poder hacer, ya que no necesariamente las matrices son inversibles.