Taller 1: Técnicas Algorítmicas

Fecha del taller: 12 de abril de 2023

Este taller consta de 6 ejercicios tomados de UVa online judge. Estos pueden resolverse empleando las herramientas que vimos relacionadas a *backtracking*, programación dinámica y algoritmos *greedy*. Los objetivos puntuales del taller son:

- Implementar las distintas ideas que fueron desarrolladas durante las prácticas.
- Validar la correctitud y performance de las soluciones propuestas mediante un juez en instancias reales, y no solo teóricamente.
- Despejar cualquier duda que surja con los docentes.
- Seguir practicando las distintas técnicas aprendidas.

Si bien durante el taller el juez es el método de validación de los algoritmos propuestos, recomendamos fuertemente intentar comprender y justificar la correctitud del código desde un punto de vista más formal. Específicamente, se recomienda:

- ☐ Diseñar y pensar los algoritmos antes de empezar a programar.
- ☐ Estimar la complejidad computacional de los algoritmos propuestos.
- ☐ Escribir apropiadamente en notación matemática las funciones recursivas que se implementan.
- ☐ En el caso de los algoritmos *greedy*, esbozar un argumento claro que justifique la correctitud del procedimiento.

Para cada uno de los 6 problemas se provee una traducción y resumen del enunciado, asi como los links a UVa y vjudge (cuando este segundo esté disponible), desde donde se pueden submittear las soluciones[1]. **Ambos jueces soportan hasta C++11**.

También se provee un "Ejercicio 0" en forma de ejercicio guiado para ver como se submittea una solución a UVa. Este ejercicio piloto también contiene información relevante para el TP...

Nota: si bien el taller tiene 6 ejercicios, consideramos que los primeros 3 conforman un subconjunto mínimo y suficiente de ejercitación. Por lo tanto, sugerimos fervientemente terminar estos 3 en clase para poder consultar con los docentes cualquier duda pertinente.

[1] Damos ambos links para protegernos del caso en que la página UVa esté caída

Ejercicio 0: *Modex*[2]

Dados enteros x, y y n se nos pide calcular

$$x^y \mod n$$

Descripción de una instancia

La primera línea de la entrada indica la cantidad c de casos de test. Cada uno consiste de una línea con los 3 enteros x, y, n, donde $1 < x, n < 2^{15}$ y $0 \leq y < 2^{31}$.

La entrada termina con una línea con un 0.

Descripción de la salida

Para cada caso de test se debe imprimir una línea con el valor z que cumpla

$$z = x^y \mod n$$

y $0 \leq z < n$.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
2 2 3 5 2 2147483647 13 0	3 11

Solución Notemos que calcular primero x^y haciendo los y productos $x \times x \times \ldots \times x$ y luego tomar el módulo n es problemático:

- Por un lado, los resultados intermedios serán muy grandes, y no los podremos guardar en `int`s ni `long`s. Notemos que si $x = 2$ y $y = 2^{15}$ el valor x^y ya tiene más de 30000 dígitos en binario.
- Por otro, tal algoritmo hace una cantidad de operaciones por lo menos lineal en el exponente y (pues debemos hacer y productos). Por lo tanto, el algoritmo hará $\Omega(y)$ operaciones, que en el caso particular en que $y \sim 2^{31} \sim 10^9$ ya es muy prohibitivo teniendo en cuenta que el juez nos da 3 segundos para procesar todos los casos de test[3].

Ambos problemas se pueden resolver empleando *exponenciación binaria modular*. La idea es:

☐ Primero, ir aplicando el módulo tras cada producto, en vez de al final. Esto hace que el resultado intermedio siempre sea a lo sumo tan grande como n.

[2] UVa: https://onlinejudge.org/external/12/1230.pdf, vjudge: https://vjudge.net/problem/UVA-1230

[3] A efectos de este taller, podemos asumir que el juez realiza alrededor de 10^8 operaciones por segundo, considerando código escrito en C++.

- ☐ Segundo, realizar la exponenciación con un enfoque *Divide & Conquer*: como $x^y = (x^{\lfloor \frac{y}{2} \rfloor})^2 * x^{y \mod 2}$, podemos calcular x^y calculando primero $x^{\frac{y}{2}}$ y luego realizando unos pocos productos más.

Todo esto se resume en la siguiente función recursiva:

$$mod\_bin\_exp(x, y, n) = \begin{cases} 1 & y = 0 \\ (mod\_bin\_exp(x, \frac{y}{2}, n)^2) \,\%n & y = 0 \mod 2 \\ ((mod\_bin\_exp(x, \frac{y}{2}, n)^2) \,\%n) \times x) \,\%n & y = 1 \mod 2 \end{cases}$$

Es fácil ver, mediante herramientas como el Teorema Maestro, que esta función tiene una complejidad de $O(\log y)$ si asumimos que multiplicar y tomar módulo son operaciones que se pueden realizar en $O(1)$. Por lo tanto, si $y \sim 2^{31}$ nuestro algoritmo realizará alrededor de $\log(2^{31}) = 31$ operaciones por cada caso de test.

Tras completar en el archivo `modex.cpp` la función `mod_bin_exp` podemos submittear nuestro código a UVa clickeando submit arriba a la derecha del enunciado, y copiando nuestro código:

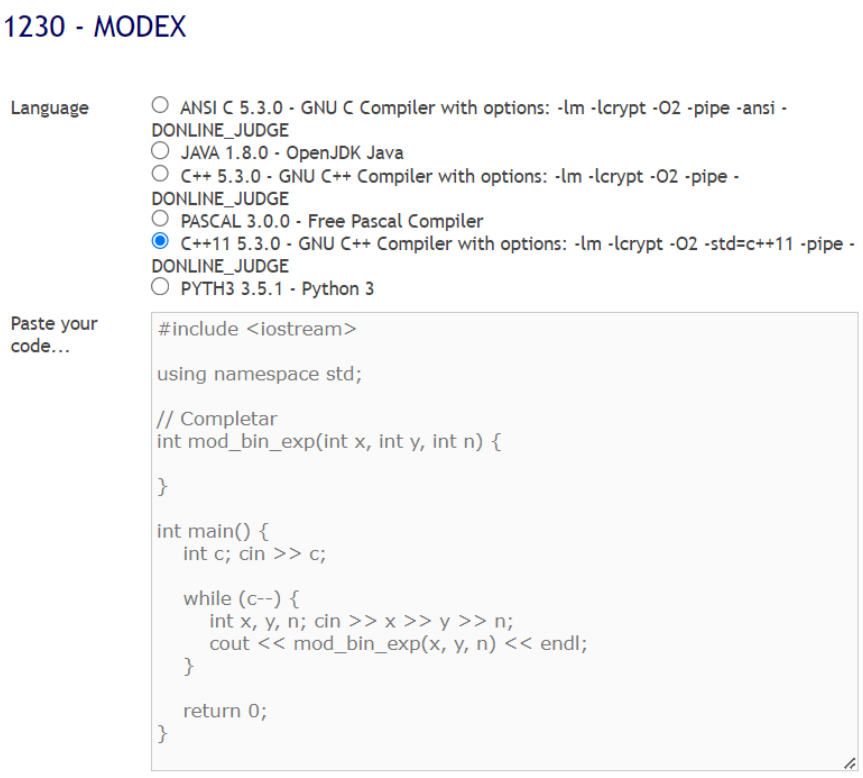

Figura 1: Ejemplo de submission en UVa

Finalmente, en My Submissions podremos ver el veredicto del juez:

Problem	Verdict	Language	Run Time
1230 MODEX	Accepted	C++11	0.000

Figura 2: Ejemplo de veredicto

En este caso podemos ver que el código tardó menos de un milisegundo en procesar todos los casos de test satisfactoriamente.

Ejercicio 1: *Shopaholic*[4]

Por Pascuas Tuki fue a comprar huevos de chocolate a la chocolatería. Una vez ahí, descubrió que, debido a una oferta, por cada 3 productos que compra solo debe pagar 2. Lamentablemente, los productos que no se pagan debido a la oferta son siempre los más baratos de la compra: por ejemplo, si Tuki compra huevos con precios de 100, 200 y 300 pesos, el monto final que deberá pagar es de 500 pesos, ya que el huevo de 100 pesos se lo lleva gratis. De la misma forma, si compra 6 productos de precios 100, 200, 300, 400, 500 y 600, el descuento total obtenido será solo de 300 pesos (debido a que no deberá pagar los productos con costo de 100 y 200 pesos).

Tuki cree que si realiza compras distintas va a poder aumentar el descuento que obtiene. Por ejemplo, en el caso anterior de 6 productos, Tuki podría realizar dos compras, primero adquiriendo los productos de 100, 200 y 300 pesos, y luego otra para el resto. En ese caso, obtendría un descuento de 500 pesos.

Tenemos que ayudar a Tuki a descubrir cuál es el máximo descuento que puede alcanzar dada la lista de productos que quiere comprar, teniendo en cuenta que puede realizar varias compras separadas.

Descripción de una instancia

La primera línea de la entrada indica la cantidad $1 \leq t \leq 20$ de casos de test. Cada uno consiste de una línea con un entero $1 \leq n \leq 20000$ denotando la cantidad de productos que Tuki quiere comprar, seguida de una línea con los n precios $1 \leq p_i \leq 20000$ correspondientes a los productos

Descripción de la salida

Para cada caso de test de sebe imprimir una línea con el máximo descuento que puede alcanzar Tuki teniendo en cuenta que debe comprar todos los productos y que puede realizar todas las compras separadas que hagan falta.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
1 6 100 200 300 400 500 600	500

[4] Uva: https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=2354, vjudge: https://vjudge.net/problem/UVA-11369

Ejercicio 2: *SuperSale*[5]

Luego de las fiestas, la chocolatería quitó la oferta del 3x2, pero puso todos sus productos en liquidación. Aparte, para evitar que una persona compre todos los objetos, estableció que cada individuo puede llevarse a lo sumo una copia de cada huevo.

Teniendo esto en cuenta, Tuki contactó a su grupo de amigos para poder aumentar la cantidad de objetos que se lleva. Como cada producto tiene un cierto peso W_i y cada amigo una capacidad C_j, podría no ser posible que todos se lleven una copia de cada objeto. Es por eso que vuelve a pedirnos ayuda: debemos ayudarlo a descubrir cuál es el máximo monto que su grupo puede pagar teniendo en cuenta los pesos de los objetos y las capacidades del grupo de amigos.

Vamos a considerar que hay suficientes copias de cada huevo como para que todos los amigos puedan comprar comprar todos los productos, si es que sus capacidades se lo permiten.

Descripción de una instancia

La primera línea de la entrada indica la cantidad $1 \leq T \leq 1000$ de casos de test.

Cada caso de test comienzo con una línea con un único entero $1 \leq N \leq 1000$, que denota la cantidad de objetos. Luego siguen N líneas, cada una con dos enteros P_i, W_i señalando el precio y el peso del i-ésimo objeto.

Luego sigue una línea con un único entero $1 \leq G \leq 100$ indicando la cantidad de personas en el grupo. Finalmente siguen G líneas con las capacidades C_j de cada persona.

Descripción de la salida

Para cada caso de test se debe imprimir una línea con el mayor monto que se puede pagar en total, teniendo en cuenta la capacidad para llevar objetos que tiene cada persona del grupo.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

[5] UVa: https://onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=1071, vjudge: https://vjudge.net/problem/UVA-10130

Entrada de ejemplo	Salida esperada de ejemplo
2 3 72 17 44 23 31 24 1 26 6 64 26 85 22 52 4 99 18 39 13 54 9 4 23 20 20 26	72 514

Ejercicio 3: *The Hamming Distance Problem*[6]

Dadas dos cadenas $x, y \in \{0,1\}^n$ (es decir, dos cadenas de longitud n cuyos caracteres son todos 0 o 1) se define la distancia de Hamming entre ambas como la cantidad de posiciones en las que difieren. Por ejemplo, la distancia entre 010 y 110 es 1, debido a que las cadenas solo difieren en la primera posición.

Dado un entero N y otro $H \leq N$ se nos pide imprimir todas las cadenas de longitud N que estén a distancia H de la cadena 0^N (es decir, la cadena de longitud N formada únicamente por 0s).

Descripción de una instancia

La primera línea contiene un número entero T, indicando la cantidad de casos de test.

Cada caso de test consiste en una línea con dos enteros $1 \leq N \leq 16$ y $1 \leq H \leq N$.

Descripción de la salida

Por cada caso de test con valores N y H se deben imprimir, en orden lexicográfico, todas las cadenas de longitud N a distancia H de la cadena 0^N. Aparte, se debe dejar una línea en blanco entre cada caso de test.[7]

[6] UVa: https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=670, vjudge: https://vjudge.net/problem/UVA-729

[7] Tener en cuenta que no debe haber una línea vacía debajo del último caso de test.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
2 3 2 4 4	011 101 110 1111

Ejercicio 4: *Factorial*[8]

El factorial de un número entero n mayor o igual a cero se define como $n! = \prod_{i=1}^{n} i$. Aquellos números M que se pueden escribir como el factorial de otro (i.e. $M = n!$ para algún n) se llaman *números factoriales*.

Dado un entero N, debemos decidir cuál es el mínimo k tal que exista una secuencia de números naturales $a_1 \dots a_k$ que cumpla

$$N = \sum_{i=1}^{k} a_i!$$

Descripción de una instancia

La entrada consiste de muchas líneas, con el valor $1 \leq N \leq 10^5$ por cada línea.

Descripción de la salida

Por cada caso de test de valor N se debe imprimir el menor número de números factoriales necesarios para sumar N.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
10 25	3 2

Nótese que $10 = 2 + 2 + 6 = 2! + 2! + 3!$ y $25 = 1 + 24 = 1! + 4!$.

Ejercicio 5: *The dominoes solitaire*[9]

Vamos a representar piezas de dominó como pares (l, r), con $0 \leq l, r \leq 6$. Decimos que una pieza (l_1, r_1) puede estar a la izquierda de otra (l_2, r_2) si y solamente si $r_1 = l_2$.

Inicialmente, colocamos dos piezas (i_1, i_2) y (d_1, d_2) en los bordes de una mesa. Queremos conectarlas mediante otras piezas de dominó, sabiendo que entre ambas hay espacio para colocar exactamente n. Aparte, conocemos las m piezas que tenemos disponibles.

[8] https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=4834
[9] https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=1444

La primera pieza (i_1, i_2) es colocada de tal forma que el valor i_2 queda a la derecha (y es por donde se debe empezar el camino), mientras que la última pieza (d_1, d_2) se pone con el lado d_1 hacia dentro de la mesa (y, por lo tanto, es en d_1 donde termina).

Debemos escribir un programa que decida, dada toda esta información, si es posible conectar las piezas iniciales.

Descripción de una instancia

La entrada consiste de muchos casos de test.

La primera línea contiene un entero que indica la cantidad n de espacios, mientras que la segunda tiene el entero m que denota la cantidad de piezas disponibles, con $1 \leq n \leq m \leq 14$. Sigue una línea con los valores i_1, i_2, y otra con los valores d_1, d_2. Luego hay m líneas con 2 enteros l_i, r_i cada una, describiendo las piezas disponibles.

La entrada termina en un 0.

Descripción de la salida

Para cada caso de test se deber responder una línea con la cadena YES si es posible formar el camino, y NO si no lo es.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
3 4 0 1 3 4 2 1 5 6 2 2 3 2 2 4 0 1 3 4 1 4 4 4 3 2 5 6 0	YES NO

En el primer caso de test se puede formar un camino entre $(0, 1)$ y $(3, 4)$ de longitud 3 usando las piezas $(1, 2)$, $(2, 2)$ y $(2, 3)$.

Ejercicio 6: *Radar Installation*[10]

Se quieren colocar radares para comunicar una serie de islas que están cerca de la costa. Para simplificar el modelo, se asume que la costa es una línea recta, representada por el eje x de un plano Cartesiano, utilizando como unidad el metro (para ambos ejes). Aparte, la posición de cada isla se representa con un par (x, y) que indica la coordenada usual.

Para proveer de comunicación a las islas se van a colocar radares sobre la línea de costa (es decir, sobre el eje x). Cada radar tiene un alcance de d metros, y necesitamos que cada isla esté cubierta por al menos un radar.

Debemos escribir un programa que dada la lista de posiciones de las islas y el alcance d de los radares devuelva la mínima cantidad de radares que deben construirse.

Descripción de una instancia

La entrada consiste en varios casos de test.

Cada uno comienza con una línea con dos enteros $1 \leq n \leq 1000$ y d, indicando la cantidad de islas y el alcance de los radares en metros, respectivamente. Siguen n líneas con dos enteros x_i e y_i cada una, denotando la posición de la i-ésima isla.

La entrada termina con una línea con dos ceros.

Descripción de la salida

Para el i-ésimo caso de test se debe imprimir una línea con el mensaje Case i: k, donde k es la mínima cantidad de radares necesarios para proveer de comunicación a todas las islas del caso de test i. En caso de que sea imposible cubrirlas se debe imprimir $k = -1$.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
3 2 1 2 -3 1 2 1 1 2 0 2 1 1 0 2 0 0	Case 1: 2 Case 2: 1 Case 3: -1

[10] UVa: https://onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=3634, vjudge: https://vjudge.net/problem/POJ-1328