XV Concurso de Programación de la UAM
"Luis Erick González Moreno" - Solucionario

June 23, 2020

Problema A. Uniendo Puntos

Este problema consiste en calcular la longitud total, que resulta de sumar las longitudes de las lineas que se forman al unir los puntos de manera consecutiva, es decir P_1 con P_2, P_2 con P_3, etc; hay que tener en cuenta que al llegar al último punto, su siguiente será P_1, siendo esta la última longitud que sumaremos. El plano es $\mathbb{R}2$, por lo que cada punto será dado con sus respectivas coordenadas (x, y). En la Figura 1 se aprecian las distintas lineas que se forman al unir los puntos, por lo tanto la respuesta es:

$$longitud_{total} = \sum_{i=1}^{n} l_i \qquad 1 \leq i \leq n \tag{1}$$

donde l_i es la longitud entre los puntos $(P_i,\ P_{i+1})$, esta se puede calcular facilmente con la formula para distancia entre 2 puntos, recordemos que $P_i = (x_i, y_i)$:

$$\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \tag{2}$$

en c++ tenemos una función que nos calcula esto la cual es hypot() y recibe como argumentos la diferencia en x y la diferencia en y.

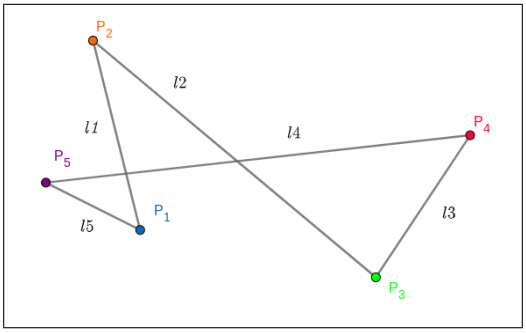


Figure 1: Lineas que se forman al unir los puntos

Por último el problema nos pide usar 2 decimales de precisión, esto se puede lograr de la siguiente forma

```
printf("%.2f", longitud_total); /* para c */
cout << fixed << setprecision(2) << longitud_total; /* para c++ */
```

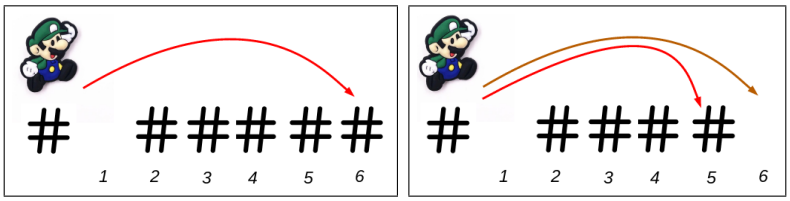
Para usar setprecision() hay que añadir en la cabecera a <iomanip> y para usar hypot() a <cmath>.

Complejidad del algoritmo O(n)

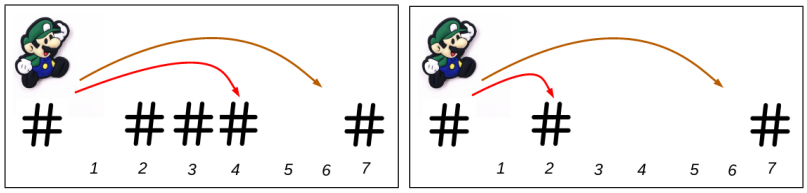
Problema B. Super UAMKid Run

Como queremos minimizar la cantidad de saltos, comenzaremos caminando y avanzaremos hacia la derecha siempre que sea posible, ya que cuando nos encontremos un espacio tendremos que forzosamente saltar, cuando lleguemos al espacio en caso de existir siempre intentaremos saltar el máximo posible por lo que siempre buscaremos saltar 5, aqui hay varias situaciones:

- Si al saltar llegamos al último caracter o lo sobrepasamos (que sería equivalente a saltar menos pero justo en el caracter final), hemos terminado.



- Si al saltar llegamos a un espacio entonces tendremos que reducir nuestro salto en 1 hasta que sea #, ya que tenemos que caer en un # para poder continuar, en el peor caso nuestro salto será de longitud 1 ya que saltaremos únicamente el espacio que encontramos en un inicio. Después procedemos a seguir caminando hasta que forzosamente tengamos que volver a saltar o lleguemos al caracter final.



- Tambien se puede dar el caso de que no sea necesario saltar, ahí la solucion es 0.

Complejidad del algoritmo O(n)

Problema C. Omitiendo la intersección

Hay varias formas de resolver este problema dependiendo de como se manipulen los conjuntos, a continuación se presenta una forma, primero encontraremos el conjunto intersección entre A y B al cual llamaremos C, para lograr esto primero ordenaremos ambos conjuntos, de esta forma comparandolos de izquierda a derecha podemos verificar cuales pertenecen a C, c++ tiene una funcion que hace esto, set_intersection(), como más adelante necesitaremos recordar la posición original de los elemenos, haremos una copia de los arreglos antes de ordenarlos.

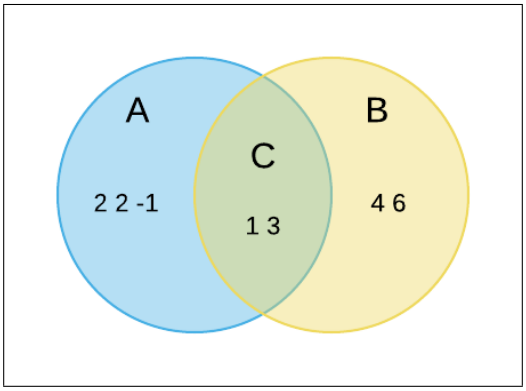


Figure 2: Representación de los conjuntos

El problema nos pide imprimir los elementos que no pertenecen al conjunto intersección según el orden en el que fueron leídos desde la entrada, por esta razón iteraremos sobre nuestra copia del array, es decir el arreglo original. Tomamos el primer elemento de izquierda a derecha, y verirficamos si esta en el conjunto C, para evitar la busqueda lineal y que nuestro algoritmo sea lento, podemos usar busqueda binaria sobre C o en un inicio utilizar "set" de c++ el cual es un árbol binario balanceado, por lo que para saber si un elemento se encuentra con la llamada a sus métodos count o find será suficiente para verificarlo esto se hará en log_2(n) por lo que será eficiente, entonces si el elemento es encontrado lo omitimos en caso contrario lo imprimimos.

Complejidad del algoritmo O($nlog_2n$)

Problema D. ¿Qué cursos puedo inscribir?

Para empezar requerimos separar los requisitos suaves de los duros para cada curso, lo haremos haciendo uso de una estructura auxiliar llamada nodo, donde cada nodo consta de un id y de 2 vectores, que representan el id del curso, asi como sus requisitos suaves y duros, un vector para cada tipo de requisito. Creando un arreglo de nodos podemos entonces almacenear todos nuestros cursos junto a sus requisitos. Como nos solicitan que la salida sea presentada en orden, ordenaremos nuestro arreglo de nodos según su id, una vez que ya tenemos ordenados los cursos, iteramos sobre el arreglo, primero verificamos si el curso ya fue aprobado en ese caso no hacemos nada ya que solo nos interesan los que se pueden inscribir, si el curso no ha sido aprobado hacemos lo siguiente.