

Problema a resolver:

El problema esta dado por la siguiente situación: tenemos en un "lista" con una cantidad $3*n$ de números(n un número fijo).

Para i desde 0 a $n-1$, vamos a decir que la posición i en la lista va a ser *Izq* del edificio i -ésimo, $i+1$ va a ser *Alt* del edificio i -ésimo e $i+2$ va a ser *Der* del edificio i -ésimo.

A grandes rasgos vamos a tener una lista de n edificios (interpretamos a un edificio como una tupla $\langle Izq, Alt, Der \rangle$) con una base en común implícita que es 0 .

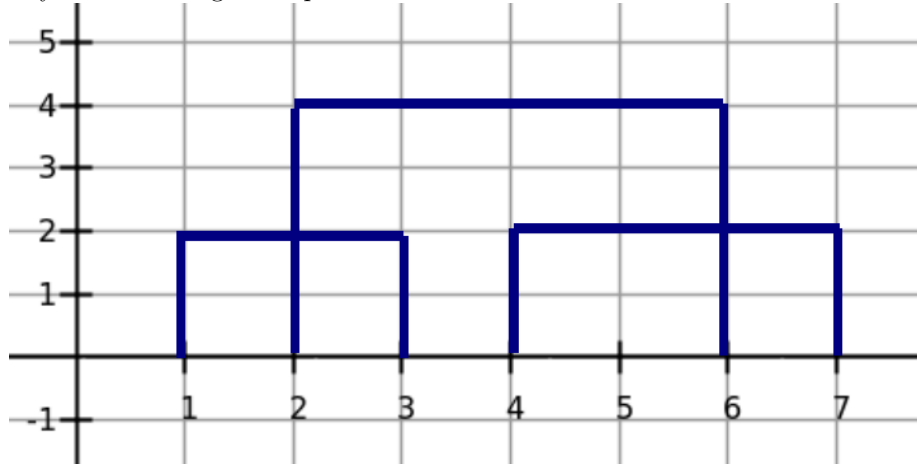
Vamos a utilizar esta notación para referirnos a los edificios.

Por ejemplo para un entrada de la forma:

$lista = 1, 2, 3, 4, 2, 7, 2, 4, 6$ y $n=3$ tendríamos:

$lista = \langle 1, 2, 3 \rangle, \langle 4, 2, 7 \rangle, \langle 2, 4, 6 \rangle$

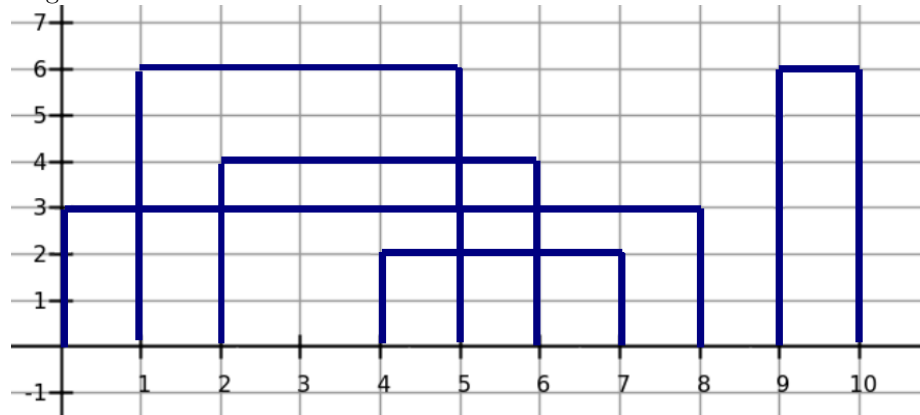
Proyectada en un gráfico quedaría:



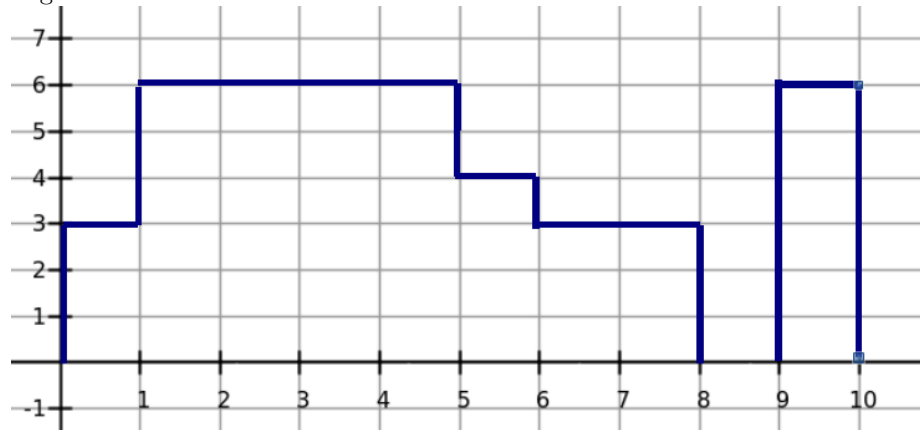
Lo que queremos hacer es "eliminar todas las líneas interiores del gráfico", quedarnos con su contorno (se obtiene el mismo resultado "siguiendo con el dedo el gráfico") para luego poder dar la solución final que explicaremos más adelante.

Veamos un ejemplo de eliminación de líneas interiores y el "procedimiento" de seguir con el dedo el gráfico

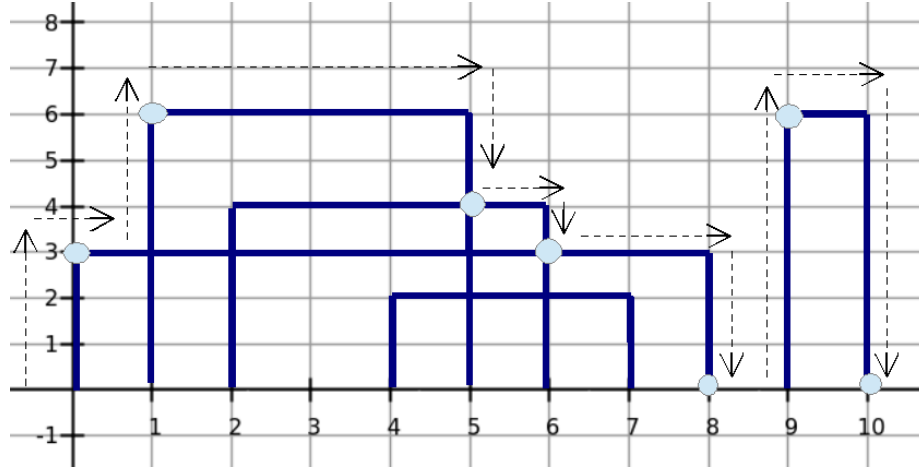
Para una $lista = \langle 0, 3, 8 \rangle, \langle 1, 6, 5 \rangle, \langle 2, 4, 6 \rangle, \langle 4, 2, 7 \rangle, \langle 9, 6, 10 \rangle$ con $n=5$, su gráfico es:



el gráfico de eliminar las líneas interiores es:



Mientras que el gráfico de "seguir con el dedo" es:



La salida para este ejemplo sería:

0,3,1,6,5,4,6,3,8,0,9,6,0

Lo que hago cuando "sigo con el dedo" es:

Empezar con el primer edificio y seguir el trazo, si me interseco con otro edificio seguir el trazo del edificio con el que me intersequé desde ese punto.

Si no me interseco con nadie, pero hay más edificios adelante "seguir con el dedo" los otros. Si no hay más edificios terminé.

Luego de ese contorno voy a obtener la solución final que son los puntos donde hay cambios de altura ($\uparrow \rightarrow$ y $\downarrow \rightarrow$).

Resolución:

Un panorama de la resolución es:

ordenar los edificios de menor a mayor por su *Izq* (en caso que tengan mismo *Izq* es menor el que tiene mayor *Alt*).

Agregar a la solución el primer punto del primer edificio de la lista de edificios ordenada.

Vamos a ir recorriendo los edificios, voy a tener un edificio *comparo* en este recorrido(al principio *comparo* es el primer edificio) y empiezo a recorrer desde el segundo, en caso de haber más de uno,el edificio por el que voy será *siguiente*. También tengo una *cola* de prioridad (organizada por mayor *Alt*) donde voy a ir metiendo edificios.

Si *comparo* interseca a *siguiente*:

agrego a la solución el cambio de altura si $\text{siguiente.Alt} > \text{comparo.Alt}$.

Si $\text{siguiente.Altura} \geq \text{comparo.Alt}$, ahora *comparo* es *siguiente* y encolo a *comparo* en la *cola*.

Si $\text{siguiente.Alt} < \text{comparo.Alt}$ encolo a siguiente en la *cola* y *comparo* queda donde estaba.

Si *comparo* no interseca a *siguiente*

Quiere decir que están "separados",pero puede que haya un edificio *e* tal que $e.Izq \leq \text{comparo.Izq}$, $e.Der > \text{comparo.Der}$ y $e.Alt \leq \text{comparo.Alt}$, si existiera *e* tendría que estar en la cola.)

Si la *cola* no está vacía voy a buscar a *e* en la *cola*(si existe, lo encolé cuando fui viendo los edificios anteriores a *siguiente*):

si el tope de la *cola* es *e*, ahora *e* es *comparo*, agrego a la solución la intersección por derecha de *e* con *comparo*, no lo desencolo porque pude que corte a otros edificios de más adelante.

Si el tope de la *cola* no es *e* desencolo y sigo buscando. Si la *cola* es vacía(quiere decir que no había ningún edificio que terminara después que *comparo*) agrego a la solución el final de *comparo* y el principio de *siguiente*, ahora *comparo* es *siguiente*.

Terminé de recorrer los edificios,pero puede que hayan quedado cosas dentro de la *cola* y parte de la solución no la esté dando.

Mientras la *cola* tenga edificios, voy a comparar el tope con *comparo*:

si $\text{tope.Der} < \text{comparo.Der}$ desencolo.

En caso contrario, ahora *comparo* es *siguiente* ,agrego a la solución la intersección por derecha entre el tope y *comparo*, desencolo.

Ya no hay más edificios en la *cola*. Agregar a la solución el último punto *comparo.Der,0* y retornar la solución.

Hay cosas que en esta descripción no tuve en cuenta, porque son muy específicas, para explicarlo profundamente lo hago con este pseudocódigo:

Sea lista: lista(números) de longitud $3*n$ y n la cantidad de edificios

```

1: procedure RESOLVEREDIFICIOS(lista,n)
2:   pasar a tuplas (<Izq,Alt,Der>) la lista
3:   ordenar los edificios por Izq, en caso de tener mismo Izq es menor el que tiene mayor Alt
4:    $comparo \leftarrow lista[0]$ 
5:    $cola \leftarrow vacía$ 
6:   imprimo el primer punto(comparo.Izq y comparo.Alt)
7:   for ( $i \leftarrow 1, n-1$ ) do //voy recorriendo los edificios a partir del segundo
8:      $siguiente \leftarrow lista[i]$ 
9:     if (se intersecan comparo y siguiente*0) then
10:      if ( $siguiente.Alt > comparo.Alt$  *1) then
11:        imprimir cambio de altura(siguiete.Der, siguiete.Alt)
12:        si la cola está vacia encolar comparo, sino, encolar comparo si el tope no es comparo
13:         $comparo \leftarrow siguiete$ 
14:      end if
15:      if ( $siguiente.Alt == comparo.Alt$  *2) then
16:        si la cola está vacia encolar comparo, sino, encolar comparo si el tope no es comparo
17:      end if
18:      if ( $siguiente.Alt < comparo.Alt$  *3) then
19:        encolar comparo
20:      end if
21:    else(no se intersecan comparo y siguiente*4)
22:      if (la cola no está vacía) then
23:        while (cola no vacía) do
24:          if ( $topeCola.Der < comparo.Der$  *5) then
25:            desencolar
26:          else
27:            imprimir intersección entre comparo y topeCola(comparo.Der,topeCola.Alt)
28:             $comparo \leftarrow topeCola$ 
29:            salir del while
30:          end if
31:        end while
32:      else //no pasé a nadie que cortaría a comparo, como no se intersecan comparo y siguiente imprimo ambos puntos
33:        imprimir comparo.Der, 0
34:        imprimir siguiete.Izq, siguiete.Alt
35:         $comparo \leftarrow siguiete$ 
36:      end if
37:    end if
38:  end for//terminé de iterar los edificios,puede que hayan quedado cosas en el cola *6,uso a comparo que es el último edificio visto con el que haya en el tope de la cola

```

```

39:   for (la cola no es vacía, desencolar) do
40:       if (comparo.Der <= topeCola.Der) then
41:           imprimir intersección por entre comparo y topeCola(comaro.Der,topeCola.Alt)
42:       end if
43:       desencolar
44:   end for
45:   al último punto no lo imprimo nunca *9)
46:   //lo imprimo acá
47:   imprimir comparo.Der y 0
48: end procedure

```

Complejidad:

Al principio del algoritmo paso la lista de edificios a una lista de tuplas: cada 3 números voy a formar una tupla (la cantidad de números totales es $3*n$), el costo de crear una tupla es c (de asignaciones y crear cosas que son 1 operación) y de ubicarla en una lista es también 1 operación.

Entonces recorro $3*n$ números y cada 3 creo un edificio, con lo cuál hago $3*n*(c+1)$ para convertir a tuplas los edificios.

Luego de convertirlos a tuplas, ordeno los edificios por Izq (en caso de igual Izq es menor el de mayor Alt), el costo de ordenarlos es $n*\log(n)$ (porque uso sort de la stl [según este link](#)).

Vamos a ver que la cantidad de veces que encolo es una función de n , y así poder ver que la cantidad de veces que desencolo es también una función de n porque no puedo desencolar más cosas de las que encolo. Vemos en el algoritmo que en $*1$ y $*2$, encola si está vacía o si el tope de la cola no es comparo. (**falta demostrar que si quiero encolar un elemento 2 veces el que quiero encolar de nuevo está en el tope, la idea la había sacado angel**) Luego en $*3$ encolo. Después a lo largo de todo el algoritmo no hago *nunca* un encolar.

Como todos estos casos son disjuntos (o son $<$, $>$ o $=$ Alt de *comparo* y *siguiente*) entonces hago 1 encolar en cada edificio en peor caso, con lo cual hago n encolar.

Veamos ahora la cantidad de operaciones del *while* dentro del *for* (de recorrer los edificios), en peor caso por cada edificio desencolo todos los edificios, eso da una complejidad $n*(n*(costo\ de\ desencolar))$. Pero si analizamos más finamente, nunca podría para cada paso desencolar todos los edificios.

Recorriendo los edificios ordenados (*línea 3 pseudo*), si voy por el edificio i (i entre 0 y $n-1$), tengo en la cola en peor caso i edificios (por lo explicado arriba de la cantidad de encolar), entro en el *while* (suponiendo que los edificios i e $i+1$ (*en caso de existir*) no se tocan) y desencolo i edificios (en peor caso desencolo todos los que encolé), sigo avanzando y llego a un edificio j (j entre i y $n-1$) ahora en la cola en peor caso tengo $j-i$ edificios (pues los edificios anteriores a i ya no están en la cola y encolé todos desde i hasta j), entro en el *while* (suponiendo que los edificios j y $j+1$ (*en caso de existir*) no se tocan) y desencolo en peor caso $j-i$ edificios. Llego al último edificio $n-1$ y en peor caso tengo $n-1-j$ edificios en la cola (porque los edificios hasta j no están en la cola porque los desencolé), entro al *while* y desencolo $n-1-j$ veces.

Si sumo la cantidad de veces que hice desencolar en el recorrido de los edificios es n (suma de los intervalos). Entonces hago n veces desencolar en peor caso. Relacionando la parte de encolar y desencolar en el primer *for* por cada edificio hago un encolar y una cantidad x de desencolar + c (*constante*) operaciones que tienen costo 1 (asignaciones y guardas).

Por lo visto anteriormente la suma de esos x es n , entonces el costo del *while* dentro de primer *for* es $n*(costo\ de\ encolar) + n*(costo\ de\ desencolar) + n*c(constante)$ operaciones

Para el segundo *for*, en peor caso tengo todos los edificios que son n en la

cola, y desencolo hasta que se vacia, entonces hace $n^*(costo\ de\ desencolar) + c1$ (constante de guardas y asignaciones)) operaciones

En la implementación usamos una priority-queue como cola, el costo de encolar(push) es $\log(n)$, desencolar(pop) es $\log(n)$ y tope(top) es 1. [LINKS](#)

Finalmente la complejidad es

$O(3^n * (c+1))$ de crear las tuplas

$+ n * \log(n)$ de ordenarlos

$+ n^*c + n[encolar] + n^*(\log(n))[desencolar]$ del primer for

$+ n^*c + n * \log(n)[desencolar]$ del segundo for) =

$O((3^n * (c+1) + c + 1 + c)^n + 3^n * (\log(n)))$

$\in O(n(\log(n)))$ (acoto por el máximo y saco las constantes)

Correctitud:

El algoritmo pone en la solución (*primerEdificio.Izq,primerEdificio.Alt*).

En la solución al problema, el primer "punto" es *Der, Alt del edificio* que tiene menor *Izq* y mayor *Alt* que todos, ya que es el primer cambio de altura. ¿El primer punto que pone mi algoritmo cumple con dichas propiedades? Veamos, paso los numeros de la entrada a tuplas, ordeno los edificios(en tuplas) por *Izq* (línea 3). Entonces el primer edificio va a ser el de menor *Izq* y el de mayor *Alt*. Con lo cual si, el primer punto cumple con esas propiedades.

Voy iterando los edificios mirando el edificio por el que voy(*siguiente*) y un edificio que voy marcando en ciertos casos, *comparo*. *Comparo* no necesariamente es uno menos que *siguiente*. Al principio del algoritmo *comparo* es el primer edificio y *siguiente* el segundo en caso de haber. Si *comparo* y *siguiente* se intersecan:

(graficos intersecciones)

si *comparo.Alt* < *siguiente.Alt*

(grafico A)

tengo que poner el punto de cambio de altura, *p*, en la solución

Supongamos no tengo que poner a *p* en la solución:

caso existe un edificio, *e* que cumple: *e.Izq* < *siguiente.Izq*, *e.Alt* > *siguiente.Alt* y *e.Der* >= *siguiente.Der*(este edificio tapa al punto que quiero agregar a la solución):

(ejemplo grafico C)

Si existiera tendría que haberlo puesto como *comparo*,pero estoy en el caso donde la altura de *comparo* es menor que la de *siguiente*, el edificio no puede existir. Entonces hay que poner a *p* en la solución.

Si no se intersecan *comparo* y *siguiente*:

(grafico no se intersecan)

voy a tener en la cola los edificios que tienen *Der* mayor a *comparo.Der* porque los fui encolando, en caso de existir, entre otros.

(grafico D)

si la cola está vacía quiere decir que no hay nadie que corte a *comparo*(Gráfico cola vacia grafico),entonces el edificio *comparo* sigue hasta su base y tengo que seguir con los demás edificios(en caso de tener más) desde *siguiente*, entonces voy a poner en la solución, el final de *comparo* y el principio de *siguiente*.

(cola vacia grafico)

falta demo

Si la cola no está vacía quiere decir que puede que tenga un edificio que corte a *comparo* por *Der*. Grafico que corte a *comparo*

Voy a buscar un edificio que interseque a *comparo* por *Der*(si lo encuentro no busco más),mirando el tope, en este caso no voy a desencolar. Si el tope no es lo que busco desencolo.

Por como es la cola este que quería encontrar es el que tiene mayor *Alt* de los edificios que tienen menor *Alt* que *comparo*. Entonces este punto,*p*, de intersección lo agrego a la solución.

Supongamos que no tengo que agregar a p a la solución.

caso existe un edificio, e , que cumple que: $e.Izq < comparo.Izq$, $e.Der > comparo.Der$ y

$comparo.Alt \leq e.Alt < topeCola.Alt$

(grafico E)

Entonces este si existiera, lo tendría que haber encolado cuando recorría los edificios y tendría que ser el edificio que me daría el tope de la cola Absurdo!.

Entonces tengo que agregar a p a la solución.

Siempre recorro los edificios mirando *comparo* y *siguiente*, en algún momento me voy a quedar sin edificios, *comparo* va a quedar en algún edificio y la cola va a tener algún estado (vacía o con edificios dentro)

ejemplos de estados de la cola

Ahora voy a trabajar con *comparo* y la *cola* (ahora en la cola están mis "siguientes") Si hay edificios en la cola: Quiere decir que puede que uno de esos edificios interseque en Der a *comparo*. Voy a buscar en la cola ,mirando el tope, un edificio que interseque a *comparo* Supongo que el punto no es solución blabla llego a absurdo y es punto. Ahora *comparo* es el que me intersecó Este nuevo *comparo*