

Slow Convex Hull

Ejercicio 1: Explicar la complejidad del algoritmo SlowConvexHull presentado en clase.

Denotemos el tamaño del conjunto de puntos de los que se desea obtener su Convex Hull como 'N' (se asume que no hay 2 puntos iguales en el conjunto).

Consideremos los pasos principales del algoritmo:

- a) Fijar todos los pares de puntos P,Q tal que $P \neq Q$ para probar la validez de la línea que forman. La cantidad de puntos P's que podemos fijar son N (podemos tomar cada punto del conjunto como P una vez). Una vez fijado el P, quedan N-1 posibles candidatos para Q. Como vamos a fijar $O(N)$ puntos para P, y para cada uno de ellos también estamos fijando $O(N)$ puntos para Q, la complejidad de este paso es $O(N^2)$. La cantidad exacta de pares sería $N(N-1) = N^2 - N$, que de igual modo es del orden **$O(N^2)$** .
- b) Para cada par de puntos (P, Q), vamos a probar si forman una línea válida iterando sobre todos los puntos R diferentes de P y Q. Como ya se han fijado 2 puntos P y Q, existen N-2 posibles candidatos para R. Como validar cada punto individual es $O(1)$ (esto puede realizarse utilizando producto cruz de vectores), y debemos validar $O(N)$ posibles puntos R, la complejidad total de este paso es **$O(N)$** .
- c) Denotemos la cantidad de líneas que forman el Convex Hull del conjunto de puntos dado como 'H'. Para obtener los puntos propios del Convex Hull en sentido horario, el algoritmo presentado en clase itera sobre todas las líneas obtenidas, y para cada línea L (denotada por un punto de inicio P y un punto final Q), busca la siguiente línea K que tenga al punto Q como punto de inicio. Tendremos que revisar $O(H)$ líneas L, y para cada línea L tendremos que hacer un segundo recorrido sobre todas las líneas para encontrar la línea K correspondiente, que también es $O(H)$. Por lo tanto tendremos $H*H = O(H^2)$ operaciones. Sin embargo, podemos notar que H está acotado por $O(N)$, por lo que este procedimiento es **$O(N^2)$** en el peor caso.

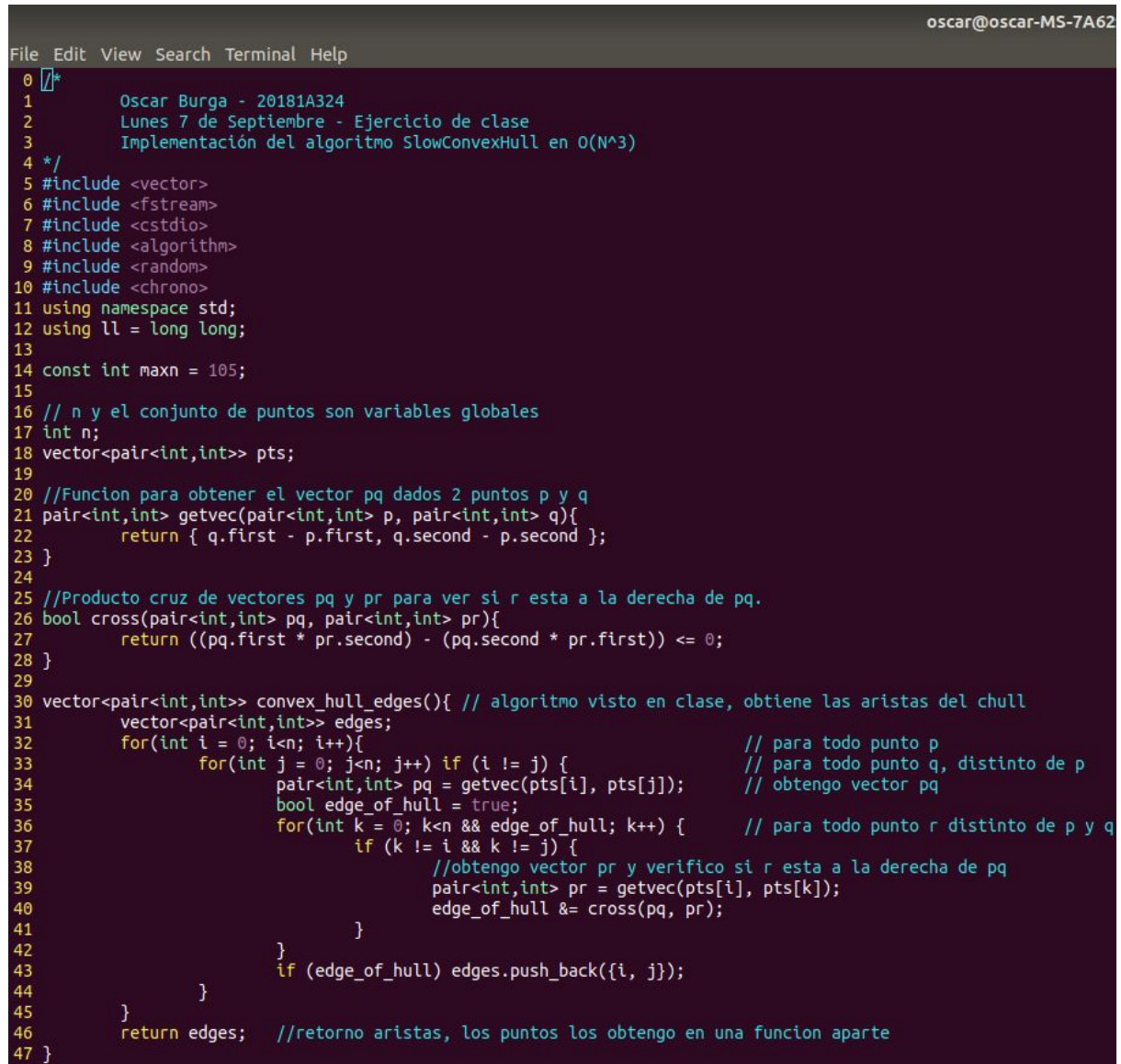
Según **(a)**, debemos probar la validez de la línea formada por cada par (P, Q) del conjunto, y hay $O(N^2)$ de estos pares, y según **(b)**, probar la validez de cada par (P, Q) tiene un costo de $O(N)$. Por lo tanto, la complejidad del algoritmo hasta el momento es $N^2 * N = O(N^3)$. Finalmente, debido a **(c)**, debemos realizar un procedimiento adicional en $O(N^2)$, pero como la primera parte de nuestro algoritmo ya es $O(N^3)$, esto no afecta la complejidad total. **Por ello, la complejidad final del algoritmo es $O(N^3)$.**

Ejercicio 2: Realizar la implementación del algoritmo SlowConvexHull en cualquier lenguaje de programación.

La solución de este ejercicio se encuentra en la siguiente página. Los enlaces para el acceder directamente al código se encuentran al final del documento.

Escogí el lenguaje C++ para realizar este ejercicio ya que es el lenguaje con el cual me siento más cómodo. Las capturas de pantalla de la parte principal del código se encuentran en las siguientes 2 páginas.

Al final del documento se encuentran los enlaces a la implementación completa (incluyendo generación aleatoria de puntos, graficador, etc.)



```
File Edit View Search Terminal Help
0  /*
1     Oscar Burga - 20181A324
2     Lunes 7 de Septiembre - Ejercicio de clase
3     Implementación del algoritmo SlowConvexHull en O(N^3)
4  */
5  #include <vector>
6  #include <fstream>
7  #include <cstdio>
8  #include <algorithm>
9  #include <random>
10 #include <chrono>
11 using namespace std;
12 using ll = long long;
13
14 const int maxn = 105;
15
16 // n y el conjunto de puntos son variables globales
17 int n;
18 vector<pair<int,int>> pts;
19
20 //Funcion para obtener el vector pq dados 2 puntos p y q
21 pair<int,int> getvec(pair<int,int> p, pair<int,int> q){
22     return { q.first - p.first, q.second - p.second };
23 }
24
25 //Producto cruz de vectores pq y pr para ver si r esta a la derecha de pq.
26 bool cross(pair<int,int> pq, pair<int,int> pr){
27     return ((pq.first * pr.second) - (pq.second * pr.first)) <= 0;
28 }
29
30 vector<pair<int,int>> convex_hull_edges(){ // algoritmo visto en clase, obtiene las aristas del chull
31     vector<pair<int,int>> edges;
32     for(int i = 0; i<n; i++){
33         for(int j = 0; j<n; j++) if (i != j) { // para todo punto p
34             pair<int,int> pq = getvec(pts[i], pts[j]); // para todo punto q, distinto de p
35             bool edge_of_hull = true; // obtengo vector pq
36             for(int k = 0; k<n && edge_of_hull; k++) { // para todo punto r distinto de p y q
37                 if (k != i && k != j) {
38                     //obtengo vector pr y verifico si r esta a la derecha de pq
39                     pair<int,int> pr = getvec(pts[i], pts[k]);
40                     edge_of_hull &= cross(pq, pr);
41                 }
42             }
43             if (edge_of_hull) edges.push_back({i, j});
44         }
45     }
46     return edges; //retorno aristas, los puntos los obtengo en una funcion aparte
47 }
```

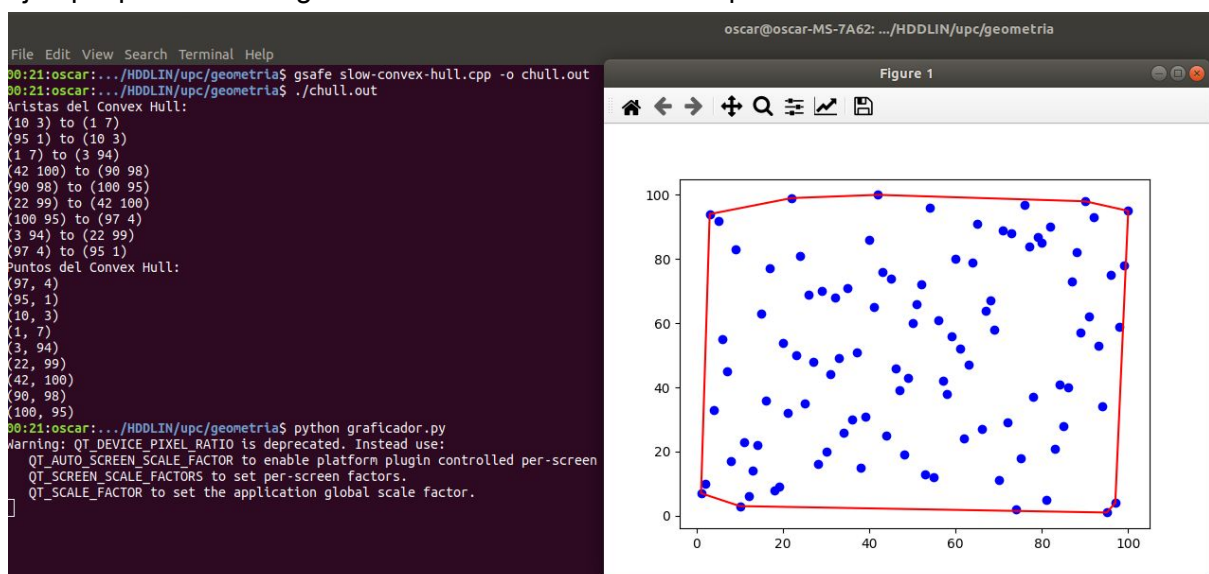
La función **get_chull_points** se encuentra en la siguiente página. Esta se utiliza para obtener los puntos propios del Convex Hull a partir de las aristas generadas por **convex_hull_edges**.

```

File Edit View Search Terminal Help
56
55 vector<pair<int,int>> get_chull_points(vector<pair<int,int>> edges){
54     vector<pair<int,int>> chull;
53     chull.push_back(pts[edges.back().first]);
52     chull.push_back(pts[edges.back().second]);
51     pair<int,int> cur = chull.back();
50     //reconstruyo el chull en O(N^2) hasta que el punto inicial vuelva a aparecer.
49     while(cur != chull.front()){
48         for(pair<int,int>&p: edges) if (pts[p.first] == cur) {
47             chull.push_back(pts[p.second]);
46             cur = chull.back();
45             break;
44         }
43     }
42     chull.pop_back(); //eliminar punto repetido (es el mismo que el primer punto)
41     return chull;
40 }

```

Ejemplo para un caso generado aleatoriamente de 100 puntos no-colineales:



El código fuente de mi implementación del algoritmo y graficador pueden encontrarse en los siguientes enlaces:

[Google Colab con Algoritmo + Graficador](#) (seguir instrucciones adjuntas).

[Implementación solo del algoritmo](#) en C++.

[Implementación del graficador](#) en Python usando matplotlib.