UNIVERSIDAD
DE SANTIAGO
DE CHILE

Departamento de Matemática y Ciencia de la Computación

Laboratorio 3

Convex Hull vs Quick Hull

Segundo Semestre 2016

Programación Avanzada 26106

Licenciatura en Ciencia de la Computación

Silvana Cerda Aravena
Alonso Maripi Vallejos
silvana.cerda@usach.cl / alonso.maripi@usach.c

1 Introducción

La diferencia de complejidad entre algoritmos es algo muy importante ya que nos demuestra la eficiencia del codigo, en este caso se realizaron 2 formas diferentes de encontrar la figura mas grande formada por una cantidad de puntos ingresada, debemos resolver este problema por medio de fuerza bruta y de Divide et Impera.

2 Procedimiento

Dado una cantidad de puntos con sus coordenadas debemos analizar cual es el poligono mas grande que podemos formar segun estos puntos entregados por el usuario, de entrada recibiremos la cantidad de puntos a ingresar , luego valores de x e y . EL algoritmo de fuerza bruta recorre todas las distancias entre la totalidad de los puntos ingresados, este tarda mas que el Quickhull ya que la manera de recorrer los puntos es totalmente distinta.

2.1 Restricciones

Se debe ingresar datos de tipo entero, para evitar problemas con los calculos de las distancias se utilizo variables de tipo double.

3 Estructuras de Datos Utilizadas

Para este problema utilizamos struct para las coordenadas en el caso del Quick-Hull.

4 Algoritmo

Algorithm Convex Hull B(n,P[n]).

Input: n: A positive Integer, P[n]: A list of points in the X-Y axis.
Output: V[*cont*]: A list of points that enclose the rest of points ingressed.

```
1  for i ← 0 to n do
2      for j ← i + 1 to n do
3          work (P[i], P[j], P, n)
```

Algorithm Work(P1,P2,n,P[n])

```
1   flag ← 0
2   for m ← 0 to n do
3       if El punto P[n] es distinto a P1 y P2
4           if P1 y P2 son distintos en el eje X e Y
5               A ← El valor del punto P[m] evaluado en la recta generada por P1 y P2
6               if A > 0
7                   if flag = 0 || flag == 1
8                       flag = 1
9                   else
10                      flag = −1
11              else
12                  if A <= 0
13                      if flag = 0 || flag == 2
14                          flag = 2
15                      else
16                          flag = −1
17                  else
18                      flag = −1
19          else
20              if P1 y P2 son distintos en el eje Y pero iguales en el X
21                  if P[m] en el eje X > P1 en el eje X
22                      if flag = 0 || flag == 1
23                          flag = 1
24                      else
25                          flag = −1
26                  else
27                      if P[m] en el eje X <= P1 en el eje X
```

```
28                          if flag = 0  ||  flag == 2
29                              flag = 2
30                          else
31                              flag = −1
32                      else
33                          flag = −1
34              else
35                  if P1 y P2 son distintos en el eje X pero iguales en el Y
36                      if P[m] en el eje Y > P1 en el eje Y
37                          if flag = 0  ||  flag == 1
38                              flag = 1
39                          else
40                              flag = −1
41                      else
42                              if P[m] en el eje Y <= P1 en el eje Y
43                                  if flag = 0  ||  flag == 2
44                                      flag = 2
45                                  else
46                                      flag = −1
47                              else
48                                      flag = −1
49              end If
50          end If
51      end If
52  end For
53  if flag! = −1
54      if P1 no existe en el conjunto V
55          V[cont] en el eje X ← P1 en el eje X
56          V[cont] en el eje Y ← P1 en el eje Y
57          cont ← cont + 1
58      end If
59      if P2 no existe en el conjunto V
60          V[cont] en el eje X ← P2 en el eje X
61          V[cont] en el eje Y ← P2 en el eje Y
62          cont ← cont + 1
63      end If
64  end If
```

Convex Hull de fuerza bruta realiza la busqueda de uno en uno, es decir, toma dos puntos de los que se ingreasron y los evalua con el resto, teniendo en cuenta que aquellos elementos no se evaluen consigo mismos y de ahi repite el mismo procedimiento con otros puntos hasta que todos hayan sido evaluados. Al final el programa almacena los puntos que encapsulan todos los demas en una lista. Ambos algortimos poseen la misma forma en que recibe los parametros.
El funcionamiento del algoritmo Quick Hull consiste en separar el largo del ar-

reglo de puntos por la mitad y enviar las posiciones obtetidas recursivamente a si mismo con el fin de ordenar nuestros puntos y analizar la distancia entre ellos para mostrar la figura formada.

4.1 Análisis de Complejidad

El algoritmo propuesto tiene una complejidad de θ n cubica,mientras que Quick Hull posee una complejidad de nlogn.

5 Implementación

Para solucionar el problema se eligio el lenguaje C.

/home/mjaripi/Documents/New/ch_i.c

```
/*
 * FILE: ch_i.c
 *
 * DESCRIPTION: From 'N' points, search for those that enclose the rest of them, genetaring a polygon.
 *
 * AUTHOR: Alonso Maripi Vallejos & Silvana Cerda Aravena
 *
 * LAST REVISED: Santiago de Chile, 04/11/2016
 *
 */

 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <time.h>
 #include <math.h>
 #define MAX 110L

 struct point// Basic estructure of a point in the X-Y axis
 {
    double x;
    double y;
  };

struct point S[MAX];
int cont = 0;

int exist(struct point j){//Function that determinate if a point already exist in the list that determinate the polygon.
    int i;
    for(i = 0;i < cont;i++){
        if(S[i].x == j.x && S[i].y == j.y)
          return 1;

    }
    return 0;
}

double evaluate(struct point i,struct point j,struct point m){ //Function that get the value of the point evaluated in the l
ine generated by other two points.
    return ((m.x - i.x)/(j.x - i.x)) - ((m.y - i.y)/(j.y - i.y));
}

void work (struct point i,struct point j,struct point M[MAX],int n){ //Function that set if a point is part of the polygon t
hat enclose the rest of them.
    int m,flag = 0;
    double A;

    for(m = 0;m < n; m++){

        if (M[m].x != j.x && M[m].y != j.y && M[m].x != i.x && M[m].y != i.y){

            if(j.x - i.x != 0 && j.y - i.y != 0){ //When both points that are evaluating the rest of them have diferent valu
es on the X and Y axis.

                A = evaluate(i,j,M[m]);
                if(A > 0){
                    if(flag == 0 || flag == 1)
                        flag = 1;
                    else
                        flag = -1;
                }
                else{
                    if(A <= 0){
                        if(flag == 0 || flag == 2)
                            flag = 2;
                        else
                            flag = -1;
                    }
                    else
                        flag = -1;
                }
            }
            else{
                if(j.x - i.x == 0 && j.y - i.y != 0){ // When both point have the same value on the X axis.
                    if(M[m].x > i.x){
                        if(flag == 0 || flag == 1)
                            flag = 1;
                        else
                            flag = -1;
                    }
                    else{
                        if(M[m].x <= i.x){
                          if(flag == 0 || flag == 2)
                              flag = 2;
                          else
                              flag = -1;
                        }
                        else
```

```c
/*
 * FILE: ch_r.c
 *
 * DESCRIPTION: From 'N' points, search for those that enclose the rest of them, genetaring a polygon.
 *
 * AUTHOR: Alonso Maripi Vallejos & Silvana Cerda Aravena
 *
 * LAST REVISED: Santiago de Chile, 04/11/2016
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define MAX 50

struct point
{
   double x;
   double y;
 };

void find(struct point *p,int num);
int ordenax(const void *,const void *);
double dist(struct point a,struct point b);
void find2(struct point *p,int num);

struct point c1,c2,c3,c4; // puntoscercanos y lejanos.
double mindist=1e10; // mínima distancia.
double maxdist=-1e10; //maxima distancia.
int main()
{
   int a,num;  //cantidad de puntos
   double x,y; //parametros de (x,y)
   struct point p[MAX];

   for(scanf(" %d",&num),a=0;a<num;a++) // Coge la entrada
   {
      scanf(" %lf %lf",&x,&y);
      p[a].x=x;
      p[a].y=y;
    }

   find(p,num); // Hacer la primera búsqueda.
   find2(p,num);

   printf("Distancia minima: %.3lf\n",mindist);
   printf("Distancia maxima   %3lf\n",maxdist);
   printf("Puntos: (%.3lf,%.3lf) y (%.3lf,%.3lf)\n",c1.x,c1.y,c2.x,c2.y);

   return(0);
 }

void find(struct point *p,int num)
{
   double d;
   int start,end,a,b;

   if(num<=1) // Si no hay pares de puntos:
      return; // salir.
   // Ordenar los puntos por la coordenada x.
   qsort(p,num,sizeof(struct point),ordenax);

   find(p,num/2); //busca por la izquierda

   find(p+num/2,(num+1)/2); //busca por la derecha

   // Hallar los límites del conjunto central
   for(start=num/2; end>0 && p[num/2].x-p[start].x<mindist; start--);
   for(start=num/2; end<num-1 && p[end].x-p[num/2].x<mindist; end++);


```

5.1 Plataforma Computacional

El programa fue ejecutado en 2 computadores con las siguientes características:

- **Computador n1**
- **Procesador:** Intel(R) Core(TM) i7 2.9 GHz
- **Memoria RAM:** 8192MB RAM
- **Sistema Operativo:** Ubuntu Gnome 16.04

- **Computador n2**
- **Procesador:** Inter(R) Core(TM) i3-4005U 1.70 GHz
- **Memoria RAM:** 6144MB RAM
- **Sistema Operativo:** Ubuntu 15

6 Resultados Experimentales

Para analizar la comparaciń entre el algortimo con variables globales y el algortimo con variables locales hemos graficado por medio de gnuplot el tiempo que tarda el programa en realizar los ciclos y en crear el arreglo en ambos computadores.

Computador n1

Analizando la gráfica podemos observar que la diferencia de tiempo es pequeña, en este caso el computador tarda menos tiempo utilizando el algoritmo con variables globales, esto se puede observar en la línea de color rojo.
En cuanto los ciclos podemos observar que la diferencia es pareja entre ambos programas , por lo que realizan la misma cantidad de ciclos que correspondería a n-1 del n ingresado por el usuario.

Computador n2

Trabajando con el mismo valor máximo podemos observar que el algortimo con variables locales tarda un poco más .

Se cumple entre ambos programas que los ciclos son similares.