UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICA Y CIENCIA DE LA COMPUTACIÓN
LICENCIATURA EN CIENCIA DE LA COMPUTACIÓN
Profesor(es): Nicolas Thériault
Segundo Semestre de 2020

Programación Avanzada – Laboratorios 4 y 5

1. Objetivos

Objetivo principal: ordenar listados de puntos y encontrar el cruce de segmentos más corto definido por un conjunto de puntos.

Objetivos secundarios:

4.1 Implementar un algoritmo para ordenar datos en tiempo $O(n \log n)$.

4.2 Obtener una implementación eficaz aplicables a listados grandes.

4.3 Aplicar el algoritmo para obtener dos ordenes de puntos en $\mathbb{Z}^2$ (según x e y) y seleccionar sub-listados según los valores de una de las coordenadas.

5.1 Implementar el algoritmo Dividir-y-Conquistar de búsqueda del par de puntos más cercanos en $\mathbb{Z}^2$.

5.2 Adaptar el algoritmo para encontrar los k pares más cercanos, hasta encontrar el cruce de segmentos más corto.

5.3 Obtener una implementación eficaz aplicables a listados grandes.

Para los laboratorios, se puede trabajar en grupos de hasta 3 estudiantes, entregando el programa (parte práctica) en un archivo .c.

Para el estudio teórico (cátedra), el trabajo es individual y las respuestas (con desarrollo) de la parte teórica en formato pdf o jpg.

2. Programación

La programación debe ser en C (no en C++), utilizando a lo más las librerías estándares siguientes:

- stdio.h
- stdlib.h
- math.h
- time.h

Además, para evitar cualquier problemas de compatibilidad, se recomienda utilizar solamente la función *clock()* para la medición de tiempo, lo que servirá únicamente para determinar la eficiencia del programa obtenido.

Para generar ejemplos de archivos con los puntos a analizar, se puede utilizar el programa en "formato_lab4.c" (disponible en la página web del curso).

3. Problema de búsqueda geométrica

El algoritmo dividir-y-conquistar para la búsqueda del par más cercano en $\mathbb{R}^2$ puede servir de base a la elaboración de un algoritmo de tipo codicioso para elegir una malla triangular (en el plano) que mínimiza el largo total de las aristas en la malla.

Para mínimizar su complejidad, el algoritmo de búsqueda del par más cercano requiere ordernar el listado de puntos según cada una de las coordenadas, y además tener esos listados conectados entre si (en el sentido que la j-ésima entrada en un listado ordenado según una variable viene con su posición en el listado ordenado según la otra coordenada. Este último sirve a reducir los costos de selección de sub-listados para que siguen ordenados.

4. Algoritmos de ordenamientos

Existen varios algoritmos para ordenar un listado de n entradas en tiempo $O(n \log n)$. Los más conocidos son:

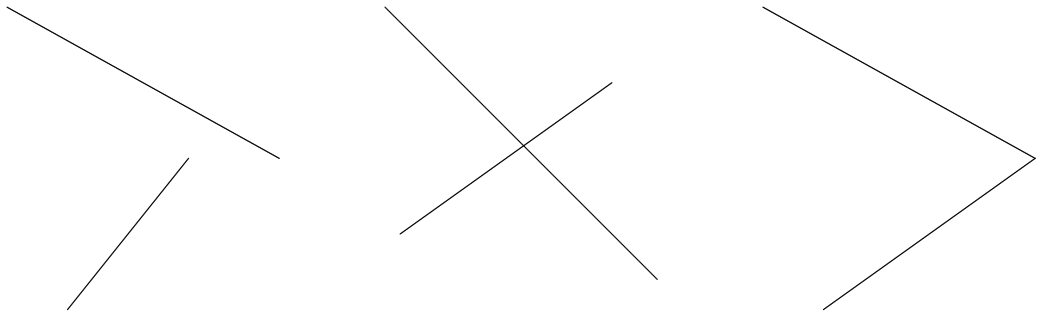
- Ordenar por mezcla (Mergesort)
- Quicksort
- Ordenar por montículos (Heapsort)
- Ordenar por árboles balanceados / AVL
- Ordenar por árboles 2-3

Los dos primeros son de tipo Dividir-y-Conquistar, mientrás que los tres últimos son de tipo Transformar-y-Conquistar (los montículos se basan en ordenar por selección, mientrás que los dos algoritmos de árboles se basan en ordenar por inserción).

5. Cruce más corto

Para aclarar el problema que se quiere resolver, primero hacemos unas definiciones:

Definición 1 *Un cruce consiste en dos pares de puntos* (x_1, y_1), (x_2, y_2), (x_a, y_a) *y* (x_b, y_b) *tales que los segmentos de rectas* $(x_1, y_1) - (x_2, y_2)$ *y* $(x_a, y_a) - (x_b, y_b)$ *se intersectan.*



No un cruce | Cruce | No un cruce

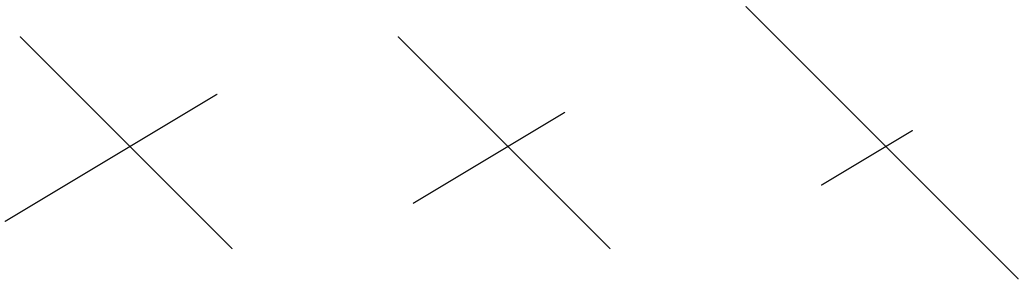
Hay varias maneras de determinar si dos segmentos de rectas forman un cruce, entre otras:

- Una opción consiste en encontrar el punto de intersección de las dos rectas (segmentos extendidos), resolviendo un sistema lineal, y luego verificar que la intersección esta adentro de ambos segmentos (por ejemplos utilizando las cotas en coordenadas x).
- Otra opción consiste en obtener la ecuación del segmento para un par de la forma

$$F_1(x, y) = Ay + Bx + C = 0$$

y verificar que F_1 cambia de signo cuando se evalúa en el otro par de punto, luego repetir para el segundo par. Los segmentos forman un cruce si y solo si ambas funciones cambian de signo (y nunca dan 0).

Definición 2 *El más corto de dos cruces es el que tiene el segmento más largo (de sus dos segmentos que se intersectan) de menor longitud. En el caso que ambos cruces tienen segmentos de la misma longitud (y solamente en este caso), el más corto es el que tiene el segundo segmento de menor longitud.*



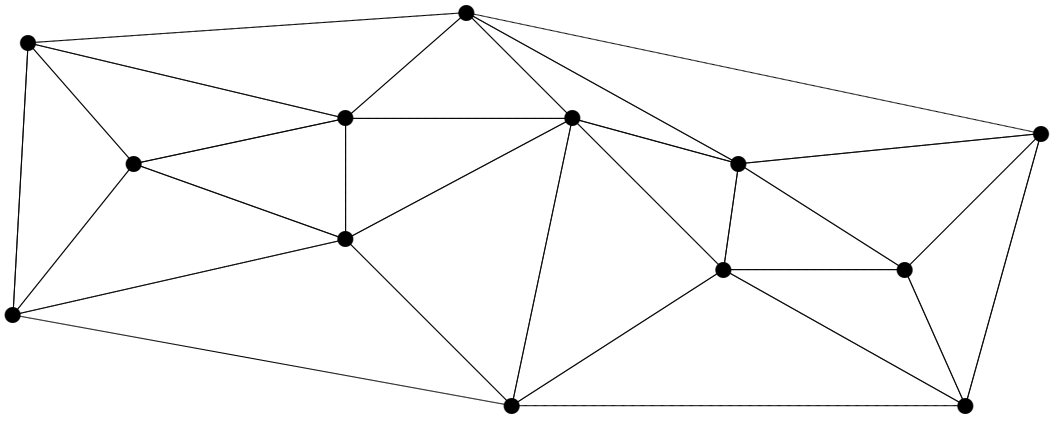
Cruce más corto

Algoritmo para encontrar el cruce más corto: Sean $\mathcal{P}$, el conjunto de puntos y $\mathcal{S}$, el listado de segmentos cortos encontrados (que se numerotarán en orden $S_0, S_1, S_2, \ldots S_{k-1}$),

- $\mathcal{S} \leftarrow \emptyset$ y $k \leftarrow 0$;
- $\mathcal{C} \leftarrow \emptyset$, los cruces encontrados
- Mientras $\mathcal{C} == \emptyset$, repetir:
  - $S_k \leftarrow$ el segmento más corto entre pares de puntos de $\mathcal{P}$ todavía no incluido en $\mathcal{S}$;
  - Para $i = 0$ hasta $k - 1$ mientras $\mathcal{C} == \emptyset$, hacer:
    - Si $\{S_i, S_k\}$ dan un cruces, entonces $\mathcal{C} \leftarrow \{S_1, S_k\}$;
  - Si $\mathcal{C} == \emptyset$, $k \leftarrow k + 1$ y adjuntar S_k al final de $\mathcal{S}$;
- Devolver $\mathcal{C}$.

6. Mallas triangulares

Definición 3 *Una malla triangular es una división del plano por segmentos de rectas entre puntos de un conjunto dado tal que cada región deliminada consiste en triángulos y tal que todo segmento de recta entre pares de puntos del conjunto está completamente incluido en la zona delimitada por los triángulos.*



7. Construcción de una mejor malla

Las mallas triángular sirven en varios contextos, por ejemplo en la representación tridimensional de una superficie (según las alturas en un conjunto de puntos), la interpolación de alguna propiedad observada en un conjunto limitado de puntos (interpolación y simulaciones con el *método de los elementos finitos*), etc.

Consideraremos como medición de un malla triangular la suma de las longitudes de todas sus aristas. En general, se considera que una malla triangular permite una mejor interpolación si su longitud total es menor. Una forma de construir una malla triangular consiste en agregar segmentos mientrás no hay intersecciones (cruces). Cuando se generan cruces, se elimina el segmento más largo.

Una manera ordenada (más controlada) de hacer eso consiste en agregar los segmentos desde el más corto hasta el más largo. Para eso, es recomendable no construir el listado de todos los segmentos y ordenarlos, ya que eso requiere $\Theta(n^2)$ memoria. Para controlar el uso de memoria, se extraen los segmentos desde el más corto aumentando progresivamente y eliminando segmento más largo del cruce más corto encontrado, hasta que no se encuentran más segmentos o cruces. El programa desarrollado en este trabajo se puede ver como un avance (parcial y más limitado en tiempo) hasta la búsqueda de una malla triangular completa.

8. Requerimientos del laboratorio 4

El programa deberá ser capaz de trabajar con listados de gran tamaño:

- Debe poder trabajar con archivos conteniendo 10^8 o más puntos.
- Puede utilizar las funciones *malloc*, *calloc* y/o *realloc* para asignar espacios de memoria, pero estos no deberán requerir más de 10^5 espacios de memoria consecutivos.

Como mínimo, crear funciones (no necesariamente separadas) para:

1. Leer archivos que contienen un listado de n puntos $(x, y) \in \mathbb{Z}^2$. La primera linea del archivo contiene la cantidad (de tipo int) de puntos en el listado; Todas las otras lineas contienen dos entradas de tipo long (de formato %ld) separadas por un espacio (y seguidos de un espacio y cambio de linea);
2. Ordenar los puntos según el valor de sus coordenadas x;
3. Ordenar los puntos según el valor de sus coordenadas y;
4. Tomar un listado ya ordenado (según una de las coordenadas) y seleccionar un sub-listado de los puntos que satisfacen algunas cotas en x e y, guardando el mismo orden.

Se podrán definir otras funciones (funciones auxiliares, etc) si deseado. Además se recomienda tener funciones para mostrar el listado cuando el tamaño del sistema es pequeño (hasta $n \leq 100$), o escribirlo a un archivo (para listados más grandes).

El programa debe utilizar un sistema de menú para el usuario / la usuaria, y debe liberar la memoria antes de cerrar el proceso. El menú debe permitir medir el tiempo de trabajo para un listado grandes (sin mostrar el resultado). El trabajo para generar un listado aleatorio, o leer y escribir archivos no se debe incluir en el tiempo de trabajo.

9. Requerimientos del laboratorio 5

El programa deberá ser capaz de encontrar el cruce más corto en listados de puntos (en dos coordenadas) de gran tamaño:

- Debe poder trabajar con archivos conteniendo 10^8 o más puntos.
- Puede utilizar las funciones *malloc*, *calloc* y/o *realloc* para asignar espacios de memoria, pero estos no deberán requerir más de 10^5 espacios de memoria consecutivos.

El programa debe utilizar un sistema de menú para el usuario / la usuaria, y debe liberar la memoria antes de cerrar el proceso.

Además de devolver el cruce más corto (dos pares de puntos), el menú debe permitir medir el tiempo de trabajo.

El tiempo para generar un listado aleatorio, o leer y escribir archivos no se debe incluir en el tiempo de trabajo.

10. Evaluación de la programación

La nota del laboratorio se calculará según la ponderación siguiente:

- Redacción [**10 %**]
  El programa está escrito de forma que puede ser leído y/o re-utilizado fácilmente por otros programadores: la redacción es limpia (con espacios y divisiones claras) y bien documentada, las sub-funciones y las variables tienen nombres naturales (que indican a que sirven) o van acompañadas de comentarios aclarando a que sirven.
- Ejecución [**15 %**]
  El programa compila sin advertencias. El ejecutable devuelve resultados correctos.
  Si el compilador no produce un ejecutable (por errores de compilación), la nota será 1.
- Eficiencia [**75 %**]
  Los tiempos de calculo de la multiplicación iterativa completa comparados con una implementación "básica" hecha por el profesor que satisface lo solicitado.
  Rangos de notas para la eficiencia:

- 1: no produce un ejecutable o no devuelve resultados correctos
- 1.1–2.4: más de 20 veces más lenta
- 2.5–3.9: 5 a 20 veces más lenta
- 4–5.4: 2 a 5 veces más lenta
- 5.5–6.9: a lo más 2 veces más lenta
- 7: tiempos parecidos o mejores