Universidad de Concepción
Departamento de Ingeniería Informática y Ciencias de la Computación

501.404 **Análisis de Algoritmos** **2025A**

Jérémy Barbay

Tarea 1: Min Distance

(Fecha de entrega: 21 de Abril 2025 [2025-04-21])

Dado un conjunto $S = \{(x_i, y_i)\}_{i \in [1..n]}$ de n puntos en el plano, considere el problema de encontrar la distancia euclidiana mínima entre dos puntos.

1 Fuerza Bruta

Implemente un algoritmo `brute_force` calculando las $n(n-1)$ distancias de manera a seleccionar tal distancia mínima. Realice un análisis de correctitud del algoritmo. Defina y demuestre su complejidad computacional.

2 Dividir para Vencer

Diseñe o busque en Internet (cite su fuente) un algoritmo `divide_and_conquer` usando dividir para vencer para encontrar tal distancia en tiempo en $o(n^2)$. Realice un análisis de correctitud del algoritmo. Defina y demuestre su complejidad computacional. (KPIs: {1.3,7.1})

3 Implementación

Implemente su algoritmo `divide_and_conquer`, comparando sus respuestas con las de su implementación de `brute_force`. (KPIs: {6.4})

4 Análisis Experimental

Diseñe y realice un análisis experimental calculando los tiempos de ejecución en nanosegundos de sus dos soluciones para un conjunto de n puntos cuyas coordenadas enteras están elegidas al azar en un cuadrado de 100×100, variando la cantidad n de puntos entre las potencias de dos de $2^3 = 8$ a $2^9 = 512$. (KPIs: {6.3})

5 Mejora

Observando el valor de la distancia calculada para valores grandes de n, proponga y evalúe una versión trivialmente mejorada de los dos algoritmos. (KPIs: {7.3})

6 Gráfico

Construya un gráfico que muestre cómo varían los tiempos de ejecución de sus cuatros soluciones en nanosegundos, variando la cantidad n de puntos entre las potencias de dos de $2^3 = 8$ a $2^9 = 512$.

Instructions

La tarea se realiza en grupos de a lo más 3 alumnos, y el reporte debe especificar para cada pregunta el nombre de un alumno quien "lideró" el trabajo y la redacción de la respuesta correspondiente, con cada miembro del grupo liderando al menos una pregunta. Sólo una persona del grupo debe enviar la entrega por Canvas, procurando que los integrantes del grupo estén detallados en su informe. No se aceptan grupos de una persona.

Se usará C++ para los programas y LaTeX para la redacción de informes. Pueden usar fuentes (i.e. artículos wikipedia) y otras herramientas (i.e. Herramientas de IA generativa como ChatGPT) para apoyar el desarrollo de su tarea siempre que: 1) sean citadas explícitamente, dando una explicación de cómo fueron utilizadas, y 2) demuestren un entendimiento acabado de la solución producida a través de respuestas orales en una interrogación o examinación escrita.

Se debe entregar (a través de Canvas) el informe con sus respuestas en formato pdf de un máximo de 6 paginas, cual incluye los fragmentos relevantes de sus programas pero no el código completo de tales programas, y un archivo `.zip` con los archivos o programas que utilice para sus respuestas. Es decir, en su entrega de Canvas deberá aparecer un archivo `pdf` (informe) y un archivo `zip` con los códigos o material que referencie en su informe.

Sugerencias

a. El código siguiente permite calcular la distancia euclidiana entre dos puntos en `C++`:

```
double calculateDistance(pair<double, double> p1, pair<double, double> p2) {
    return sqrt(pow(p1.first - p2.first, 2) + pow(p1.second - p2.second, 2));
}
```

b. Para medir el tiempo de ejecución usando `uhr`, disponible en `https://github.com/leonardlover/uhr`, hacer lo siguiente:

   (a) Clonar el repositorio,

   (b) Leer el `README.md` para ver cómo compilar y ejecutar el programa de medición,

   (c) En el mismo directorio que `uhr` escribir el código que se desea probar, digamos que en `foo.cpp`,

   (d) En el archivo `uhr.cpp`, incluir *todos* los archivos que se quieren probar, debajo de la línea 22 colocar `#include "foo.cpp"`, por ejemplo,

   (e) Configurar cada prueba debajo de la línea 58, *usando como variable a medir* `n`. Por ejemplo, si se quiere un arreglo de `n` enteros, se debería inicializar así `int data[n]`. Es en este punto en donde toda la información necesaria para lo que se quiera probar debe ser inicializada.

(f) Ejecutar la función a medir en la línea 66. Es *muy importante* que todo el código medido se encuentre entre las asignaciones de las variables `begin_time` y `end_time`.

c. Para generar números aleatorios en `uhr`, ya hay una distribución uniforme de enteros inicializada. Esta se llama `u_distr` y para generar un entero aleatorio hay que hacer `u_distr(rng)`.

Si se quiere generar enteros en un rango, hay que modificar la inicialización de `u_distr` en la línea 44. Digamos que queremos generar enteros desde 0 hasta 99, *inclusive*. Lo que hay que hacer es cambiar la línea 44 a

```
std::uniform_int_distribution<std::int64_t> u_distr(0, 99);
```

Ahora, llamando a `u_distr(rng)` se generan enteros en el rango deseado.

d. El código siguiente en `Python` genera automáticamente un gráfico con 2 curvas representando las columnas 2 y 3 de un archivo `data.csv`.

```
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
# open a csv file
df = pd.read_csv('data.csv')
# graphic each line
# plot(x-values, y-values, label, points marker)
plt.plot(df['n'], df['brute_force'], label = 'Brute Force',
    marker = 'o')
plt.plot(df['n'], df['divide_conquer'], label = 'Divide and Conquer',
    marker = 's')
# x-labels and y-labels
plt.xlabel('Running time (nanoseconds)')
plt.ylabel('Input time (log)')
# log-scale
plt.yscale('log')
# show legend
plt.legend()
#save graphic in a file
plt.savefig('plot.png')
```