Tarea 2

Algoritmo de Kruskal

Profesores: Benjamín Bustos, Gonzalo Navarro
Auxiliares: Sergio Rojas, Pablo Skewes

1. Indicaciones administrativas

- Fecha de entrega: jueves 12 de junio a las 23:59
- Grupos de a lo más 3 personas **de la misma sección**.
- Lenguaje a utilizar: C, C++ o Java.

2. Contexto

En el curso se revisó el algoritmo de Kruskal para obtener el árbol cobertor mínimo de un grafo. En esta tarea probaremos la eficiencia de este algortimo en distintos contextos.

2.1. Algoritmo de Kruskal

El algoritmo de Kruskal para encontrar el árbol cobertor mínimo de un grafo crea una estructura con todas las aristas que facilita la extracción de la arista con menor peso, y parte con un bosque $T = \emptyset$ (visto como conjunto de aristas). Luego va tomando cada arista, y si no forma ciclo en el bosque, la agrega a T, **terminando cuando $|T| = n - 1$** y el bosque se ha convertido en un árbol.

En el apunte del curso se menciona la interfaz Union-Find para poder realizar de forma eficiente la verificación de la no formación de ciclos. En esta explicación se señala que hay una optimización que se realiza para obtener los costos deseados, la cual se realiza en `find`:

Cuando se realiza `find(u)`, lo que se hace es buscar el nodo raíz del árbol que contiene a u. Para esto, se parte de u y se va subiendo por los padres hasta llegar a la raíz. La optimización consiste en hacer que todos los nodos que se visitan durante esta búsqueda apunten directamente a la raíz. Esto reduce el tiempo de búsqueda para futuras consultas.

3. Descripción de la tarea

En el contexto de esta tarea, tendrán que implementar el algoritmo de Kruskal, tanto con la optimización de `find` como sin ella, como también definiendo la estructura que facilita la extracción del mínimo, la cual será un arreglo de aristas ordenado, o un heap clásico. En otras palabras, tendrán que realizar los siguientes algoritmos:

- Algoritmo de Kruskal con la optimización de `find` y usando un arreglo de aristas ordenado.
- Algoritmo de Kruskal con la optimización de `find` y usando un heap clásico.
- Algoritmo de Kruskal sin la optimización de `find` y usando un arreglo de aristas ordenado.
- Algoritmo de Kruskal sin la optimización de `find` y usando un heap clásico.

3.1. Objetivos

- Implementar el algoritmo de Kruskal, con las 4 variaciones señaladas.
- Comparar la eficiencia de estas 4 variaciones en términos del tiempo de ejecución.
- Analizar los resultados obtenidos y discutir sobre ellos.

3.2. Metodologías obligatorias

Si bien hay libertad para implementar lo pedido en la tarea, se espera que se sigan ciertas metodologías obligatorias. Estas son:

3.2.1. Creación de la estructura de extracción de aristas

Pueden utilizar la implementación de heap en la librería estándar de C++. Para el arreglo ordenado, simplemente creen un arreglo de aristas y usen la función `sort` de la librería estándar de C++. El tiempo de ejecución de la creación de la estructura se debe considerar en el tiempo total del algoritmo. En Java también se pueden usar las implementaciones de la librería estándar para ordenar (por ejemplo, `Arrays.sort` o `Collections.sort`) y para heap (con `PriorityQueue`).

3.2.2. Valor de los nodos y pesos

Los nodos serán representados por puntos en un plano bidimensional, de tal forma de que cada nodo tendrá un par de coordenadas (x, y), donde x e y son números reales de 64 bits en el rango $[0, 1]$. El peso de la arista entre dos nodos será el cuadrado de la distancia euclidiana entre ellos, es decir, $d = (x_1 - x_2)^2 + (y_1 - y_2)^2$.

3.2.3. Implementación de Union-Find

Inicialmente guardaremos la información de los nodos en un arreglo, donde cada nodo se representará como un árbol con solo una raíz. Las aristas serán un par de punteros a los nodos que conectan.

Cuando se extraiga una arista de la estructura de extracción, se deberá realizar `find` a los nodos apuntados por los punteros de la arista. Si ambos nodos apuntan a la misma raíz, se debe descartar la arista. Si no, se incluye la arista en T y se realiza `union` entre los nodos raíces encontrados con `find`, de tal forma de que la raíz del árbol más pequeño quede como hijo de la raíz del árbol más grande (es fácil mantener un contador que señale la cantidad de nodos que tiene un árbol).

3.3. Experimentación

Se deberá generar 5 secuencias de N puntos en el plano, con $N \in \{2^5, 2^6, ..., 2^{12}\}$ (es decir, 5 secuencias de 2^5 puntos, 5 secuencias de largo 2^6, ...) y calcular las $\frac{N(N-1)}{2}$ aristas posibles con sus respectivos pesos. Luego, se deberán realizar los siguientes experimentos:

- Obtención de los tiempos de ejecución para el algoritmo de Kruskal con la optimización de `find` y usando un arreglo de aristas ordenado.
- Obtención de los tiempos de ejecución para el algoritmo de Kruskal con la optimización de `find` y usando un heap clásico.
- Obtención de los tiempos de ejecución para el algoritmo de Kruskal sin la optimización de `find` y usando un arreglo de aristas ordenado.
- Obtención de los tiempos de ejecución para el algoritmo de Kruskal sin la optimización de `find` y usando un heap clásico.

Se deberá crear un gráfico con los resultados obtenidos, donde el eje x será el tamaño de la entrada y el eje y será el tiempo de ejecución. Ya como se realizarán 5 pruebas para cada uno de los tamaños de entrada, se deberán mostrar los promedios de los tiempos obtenidos.

4. Entregables

Se deberá entregar el código y un informe donde se explique el experimento en estudio. Con esto se obtendrá una nota de código ($NCod$) y una nota de informe ($NInf$). La nota final de la tarea será el promedio simple entre ambas notas ($NT_2 = 0.5 \cdot NCod + 0.5 \cdot NInf$).

4.1. Código

La entrega de código debe ser hecha en C, C++ o Java. Tiene que contener:

- **(0.3 pts)** README: Archivo con las instrucciones para ejecutar el código, debe ser lo suficientemente explicativo para que cualquier persona solo leyendo el README pueda ejecutar la totalidad de su código (incluyendo librerías no entregadas por el equipo docente que potencialmente se deban instalar).
- **(0.2 pts)** Firmas: Cada estructura de datos y función debe tener una descripción de lo que hace y una descripción de sus parámetros de entrada y salida.
- **(0.5 pts)** Estructuras de datos: Los nodos y las aristas utilizan el sistema de punteros explicado en este enunciado.
- **(1.5 pts)** Implementación de la interfaz Union-Find: Debe ser creada por ustedes basándose en las metodologías explicadas en este enunciado y en el apunte del curso.
- **(1.0 pts)** Implementación de Kruskal sin optimización: Debe ser creada por ustedes basándose en las metodologías explicadas en este enunciado. La interfaz Union-Find no aporta puntaje a este apartado.
- **(1.0 pts)** Implementación de la optimización de Kruskal: Este puntaje va asociado netamente a la optimización, el resto de la evaluación del algoritmo se encuentra en el punto anterior. Se espera que la optimización sea creada por ustedes basándose en las metodologías explicadas en este enunciado.
- **(0.5 pts)** Experimento: Creación de las secuencias de N puntos, junto a las aristas con sus pesos.
- **(0.5 pts)** Obtención de resultados: La forma en el que se obtienen los resultados (tiempos de ejecución) es correcta.
- **(0.5 pts)** Main: Un archivo o parte del código (función main) que permita ejecutar la construcción y experimentos.

4.2. Informe

El informe debe ser claro y conciso. Se recomienda hacerlo en LaTeX o Typst. Debe contener:

- **(0.8 pts)** Introducción: Presentación del tema en estudio, resumir lo que se dirá en el informe y presentar una hipótesis.
- **(0.8 pts)** Desarrollo: Presentación de algoritmos, estructuras de datos, cómo funcionan y por qué. Recordar que los métodos ya son conocidos por el equipo docente, lo que importa son sus propias implementaciones.
- **(2.4 pts)** Resultados: Especificación de los datos que se utilizaron para los experimentos, la cantidad de veces que se realizaron los tests, con qué inputs, que tamaño, etc. Se debe mencionar en qué sistema operativo y los tamaños de sus cachés y RAM con los que se ejecutaron los experimentos. Se deben mostrar gráficos/tablas y mencionar solo lo que se puede observar de estos, se deben mostrar los valores y parámetros que se están usando.

- **(1.2 pts)** Análisis: Comentar y concluir sus resultados. Se hacen las inferencias de sus resultados.
- **(0.8 pts)** Conclusión: Recapitulación de lo que se hizo, se concluye lo que se puede decir con respecto a sus resultados. También ven si su hipótesis se cumplió o no y analizan la razón. Por último, se menciona qué se podría mejorar en su desarrollo en una versión futura, qué falta en su documento, qué no se ha resuelto y cómo se podrían extender.

Todo lo mencionado debe estar en sus informes en las secciones en las que se señalan, la falta de algún aspecto o la presencia de algún aspecto en una sección equivocada hará que no se tenga la totalidad del puntaje.