Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Guadalajara

Análisis de Algoritmos Avanzados

Actividad de Práctica

Andrés Martínez - A00227463

Grupo 502

Debido a la complejidad de los algoritmos se optó por dividir la entrega en dos carpetas: *act1* y *act2*, ambas contienen un archivo *main.cpp* el cual posee las soluciones descritas dentro de este documento. Esto sigue los lineamientos establecidos por la actividad y presenta una solución un tanto más organizada y formal.

Soluciones

Primer Ejercicio

Debido a su naturaleza de ser un problema relativamente ambiguo (no con los casos de prueba ni la explicación sino por lo abierto que son sus objetivos) resulta plausible que existan múltiples maneras de llegar a la solución que este problema desea. Como dato adicional, este problema me hizo investigar y aprender sobre diferentes temas referentes a la teoría de grafos, y me *"forzaron"* a utilizar un pizarrón para buscar una solución efectiva a este problema.

En primera instancia, tras una búsqueda exhaustiva sobre alguna herramienta que me podría ayudar para encontrar una solución a este problema encontré el concepto de Maximal Independent Set, es decir, un subset dentro de un set de nodos el cual no hay dos vértices que compartan el mismo eje, es decir, todos están separados por lo menos por un eje o más.

Para la solución de este problema, opté por crear un grafo unido **únicamente sí** la distancia entre estos nodos (medido utilizando la distancia euclidiana) es igual a la distancia previamente establecida como input.
A partir de este grafo nosotros utilizaremos un método recursivo para obtener el máximo set independiente, yendo por el primer nodo, buscando el máximo set independiente con y sin este primer nodo y de ahí llamando a la misma función con los vecinos del primer nodo, iteramos hasta obtener el máximo set independiente. Al obtener este, simplemente podemos restar el número de torres total menos el número de torres en el set del máximo set independiente para obtener nuestro resultado.

Debido a la capacidad recursiva de nuestro algoritmo, la complejidad de nuestro algoritmo es de $O(2^n)$, ya que de manera recursiva checamos todas las posibles combinaciones dentro de nuestro set. A su vez, nosotros guardamos nuestro grafo en una matriz de adyacencia, por lo que la complejidad espacial de nuestro algoritmo es de $O(n)$, donde n Es el número de nodos dentro de nuestra ecuación.

Casos de Prueba

Input	Output	Explicación
5 2 1 3 3 1 3 3 3 5 5 3	1	En este caso (originalmente utilizado de ejemplo) podemos ver que el nodo de (3,3) se encuentra en distancia d=2 con el resto de los nodos, por lo que solo es necesario eliminar ese en particular.

5 2 0 0 2 0 0 2 -2 0 0 -2	1	Este ejemplo funciona de una manera muy similar al anterior, en donde podemos apreciar que el nodo (0,0) es el único en el que tiene distancia d=2, por lo tanto es el único que debemos de remover.
5 2 0 0 a a 0 2 -2 0 0 -2	0	Nuestro algoritmo en particular no posee la función de discernir entre si ignorar input que no sea número, por lo tanto, la función "se quiebra" o deja de funcionar adecuadamente si es que se encuentra con caracteres o texto no válido.

Segundo Ejercicio

Para este ejercicio ocupamos crear una "valla" lo más chica posible que abarque a todas las posibles ovejas, es decir, un perímetro que abarque todos los posibles puntos dentro de un grafo con el perímetro más chico posible.
Considerando las limitantes dentro del problema (número no tan ilimitado de ovejas, son negligentemente pequeñas -es decir, su tamaño es irrelevante para este problema- así como que no se están moviendo), podríamos considerar que la solución para este problema se trata de *convex hull,* un set convexo que abarca todos los puntos, creando un polígono convexo.

En general, existen muchas maneras de llegar a este polígono, pero en lo particular opté por utilizar el algoritmo de gift wrapping ya que era relativamente sencillo de implementar, tenía un nivel aceptable de complejidad (considerando las limitantes así como el que se omite que tan importante es la complejidad para este problema en particular) y en general todo parecía indicar que este sería una solución adecuada para este problema.

Esta solución checa todos los otros nodos, eligiendo el que se encuentre más externo a su posición en base a la orientación que va cambiando clockwise. Es por esto mismo que su complejidad es de $O(n^2)$, ya que se comparan todos los nodos con los demás.

Casos de Prueba

Input	Output	Explicación
8 5 0 0 0 5 10 5 3 3 10 0 1 0 0	30.00 1 5 3 2 0.00 1 4.00 1 3 3.41 1 4 3	Este, siendo el caso prueba base, nos ayudó a confirmar que nuestro algoritmo funcionaba de manera adecuada. En este caso en particular podemos apreciar la creación de 8 polígonos convexos los cuales se definen mediante los input de las coordenadas obtenidas.

3 0 0 1 0 2 0 4 0 0 0 0 0 1 1 0 3 0 0 0 1 1 0 6 0 0 -1 -1 1 1 2 2 3 3 4 4 2 10 0 0 0 7 -3 -4 2 -3 4 3 -4 2 0 5 2 -3 -1 4	3.41 1 3 2 14.14 2 6 20.00 1 2 26.98 1 2 3 5 4	
1 1 0 0	0.00 1	Como se mencionó en las instrucciones de la actividad, al ser únicamente una sola oveja esta se “muere” o se queda sola y por lo tanto no es necesario construir ningún tipo de valla.
1 0 0 0	0.00	Al no haber ovejas, no es necesario escribir el número de “puntos” por los cuales se construye dicha vaya.

Conclusión

Fueron muchos los aprendizajes obtenidos a partir de esta actividad. En definitiva, fue una actividad que personalmente me forzó a pensar afuera de la caja e investigar por cuenta propia en busca de herramientas que me ayudarían a resolver esta actividad de manera efectiva.

Son muchas las soluciones y/o alternativas que existen para resolver problemas, sin embargo, consideraría esencial nuestra capacidad de interpretar problemas de tal manera que nos orientemos nosotros mismos a encontrar una solución simple y clara para nuestros problemas, así como aprovechamos del uso de la tecnología para pasar nuestras ideas del plano mental a algo físico y/o escrito en código.

Como dato adicional ajeno a la actividad, el desarrollo de esta actividad se realizó en un repositorio de Github el cual es *open source,* esto con el objetivo de documentar esta actividad e incitar a que otras personas intenten también resolverlo en el futuro.

Referencias Bibliográficas

- GeeksforGeeks. (2024a, marzo 7). Convex Hull using Jarvis' Algorithm or Wrapping. GeeksforGeeks. https://www.geeksforgeeks.org/convex-hull-using-jarvis-algorithm-or-wrapping/
- GeeksforGeeks. (2023a, marzo 15). Maximal Independent Set in an Undirected Graph. GeeksforGeeks. https://www.geeksforgeeks.org/maximal-independent-set-in-an-undirected-graph/