Tarea 4 - Problemas Conceptuales - Sección 5

Mauricio Urrego (202211641)

27 de octubre del 2023

1 Bicoloring

1.1 Descripción

Este ejercicio es una instancia del problema de la coloración de grafos, el cual se refiere al problema de colorear los vértices de un grafo de tal manera que no haya dos vértices adyacentes que tengan el mismo color. En este caso este ejercicio nos restringe a máximo dos colores. De esta manera, ya que hablamos de máximo dos colores, el problema se puede reformular a si el grafo es bipartito o no.

1.2 Algoritmos para resolver el problema

La solución para este problema se puede abordar con BFS o DFS, en particular escogeremos DFS para la implementación.

1.3 Aplicación del algoritmo

Basados en [1], podemos describir el algoritmo de la siguiente manera

1. **Paso 1:** Para realizar un recorrido DFS, necesitamos un nodo inicial y una matriz para rastrear los nodos visitados. Sin embargo, en lugar de utilizar una matriz visitada, utilizaremos una matriz de colores donde a cada nodo se le asigna inicialmente el valor -1 para indicar que aún no se ha coloreado.
2. **Paso 2:** En la llamada a la función DFS, es importante pasar el valor de color asignado y almacenarlo en la matriz de colores. Intentaremos colorear los nodos usando 0 y 1, aunque también se pueden elegir colores alternativos. La clave es garantizar que los nodos adyacentes tengan el color opuesto a su nodo actual.
3. **Paso 3:** Durante el recorrido DFS, exploramos vecinos sin color utilizando la lista de adyacencia. Para cada nodo sin color encontrado, le asignamos el color opuesto a su nodo actual.
4. **Paso 4:** Si, en algún momento, encontramos un nodo adyacente en la lista de adyacencia que ya ha sido coloreado y comparte el mismo color que el nodo actual, implica que no es posible colorear.

En consecuencia, devolvemos $False$, lo que indica que el gráfico dado no es bipartito. De lo contrario, continuamos devolviendo $True$.

1.4 Estructuras de datos para representar el problema

Para este problema utilizaremos una matriz de colores y lista de adyacencia, la primera para representar los nodos visitados y la útlima para representar el grafo.

1.5 Orden de complejidad

Dado que, en el peor de los casos, DFS tiene que considerar todas las rutas a todos los nodos posibles, la complejidad temporal de la búsqueda es $O(|V| + |E|)$ donde $|V|$ y $|E|$ es la cardinalidad del conjunto de vértices y aristas respectivamente.

1.6 Conclusión

Depth-First Search (DFS) ofrece un importante enfoque para determinar si un grafo puede ser coloreado con dos colores de manera que no haya vértices adyacentes con el mismo color. La complejidad temporal del algoritmo es razonable en función del número de vértices y aristas en el grafo

2 Sending email

2.1 Descripción

Este problema es un ejemplo del problema de encontrar el camino de costo mínimo, en donde dado un grafo ponderado no dirigido, la tarea es encontrar el costo mínimo del camino desde del nodo de origen al nodo de destino a través de nodos intermedios, en este caso los enrutadores representarían los nodos.

2.2 Algoritmos para resolver el problema

Este problema puede ser abordado con el algortimo de Dijkstra. Este algoritmo es un algoritmo de búsqueda de grafos ampliamente utilizado que encuentra el camino más corto entre nodos en un grafo ponderado. Es particularmente útil cuando desea encontrar la ruta más corta desde un nodo de origen a todos los demás nodos en un grafo con pesos de aristas no negativos. El algoritmo de Dijkstra garantiza que encontrará el camino más corto siempre que los pesos de las aristas no sean negativos.

2.3 Aplicación del algoritmo

El algoritmo comienza estableciendo la distancia del nodo de origen en 0 y las distancias de todos los demás nodos en infinito. Luego selecciona iterativamente el nodo con la distancia tentativa más pequeña, lo marca como visitado y actualiza las distancias a sus vecinos no visitados. Este proceso continúa hasta que se hayan visitado todos los nodos o no haya más nodos con distancias tentativas finitas. El algoritmo garantiza encontrar el camino más corto siempre que los pesos de las aristas no sean negativos. De esta manera calcula de manera eficiente las rutas más cortas desde una única fuente hasta todos los demás nodos del gráfico. Por último, retornamos la distancia hasta el nodo de destino

2.4 Estructuras de datos para representar el problema

Para este ejercicio utilizaremos una cola de prioridad usando un *heap*. Asimismo, haremos uso de una matriz de visitados y una lista de adyacencia en representación del grafo.

2.5 Orden de complejidad

La complejidad temporal del algoritmo de Dijkstra es $O(V^2)$, pero con una cola de prioridad mínima, que es la que utilizaremos en la implementación, desciende a $O(V + E \log V)$.

2.6 Conclusión

En resumen, este ejercicio aborda la búsqueda del camino de costo mínimo en un grafo ponderado no dirigido, donde se debe encontrar la ruta más corta desde un nodo de origen al nodo de destino a través de nodos intermedios, en este caso representados como enrutadores. La solución se implementa utilizando el algoritmo de Dijkstra, una técnica de búsqueda de grafos que encuentra la ruta más corta en un grafo ponderado.

3 Wormholes

3.1 Descripción

Este problema es un ilustración del problema de encontrar ciclos negativos en un grafo. Dado un gráfico ponderado dirigido, la tarea es encontrar si el gráfico dado contiene algún ciclo de peso negativo o no.Es de

aclarar que un ciclo de peso negativo es un ciclo en un gráfico cuyas aristas suman un valor negativo.

3.2 Algoritmos para resolver el problema

Para este problema utilizaremos el algoritmo de Bellman-Ford, el cual es un algoritmo de grafos versátil que se utiliza no sólo para encontrar los caminos más cortos en gráficos ponderados sino también para detectar ciclos negativos dentro de ellos.

3.3 Aplicación del algoritmo

Funciona relajando las aristas de forma iterativa, actualizando las distancias provisionales entre los nodos y garantizando al mismo tiempo que se considere el camino más corto. Para detectar ciclos negativos, el algoritmo realiza un seguimiento de las distancias en cada iteración. Si después de completar todas las iteraciones las distancias continúan disminuyendo, indica la presencia de un ciclo negativo.

3.4 Estructuras de datos para representar el problema

Para nuestra implementación haremos uso de una matriz de adyacencia en representación del grafo, un arreglo de distancias y un arreglo de visitados.

3.5 Orden de complejidad

La complejidad temporal del algoritmo es de $O(V \cdot E)$, donde V y E son el número de vértices en el gráfico y aristas respectivamente.

3.6 Conclusión

Bellman-Ford es una opción confiable para identificar ciclos negativos en gráficos, lo cual es crucial en aplicaciones como la optimización de redes y la búsqueda de rutas óptimas en los sistemas de transporte, o en este caso encontrar agujeros de gusano que permitan estudiar el origen del universo.

4 Freckles

4.1 Descripción

Este ejercicio es una instancia del problema del árbol de expansión mínima (MST), el cual es un problema fundamental de la teoría de grafos que busca encontrar el árbol más pequeño posible que abarque todos los nodos de un gráfico ponderado y conectado sin formar ningún ciclo.

4.2 Algoritmos para resolver el problema

Dos algoritmos bien conocidos para resolver este problema son el algoritmo de Kruskal y el algoritmo de Prim, los cuales seleccionan iterativamente aristas que contribuyen al árbol de expansión mínimo. En este caso escogeremos el algoritmo de Kruskal.

4.3 Aplicación del algoritmo

En primer lugar, el algoritmo calcula la distancia entre los puntos, posteriormente comienza con un conjunto vacío de aristas y progresivamente agrega aristas para formar el MST evitando la creación de ciclos. Funciona de la siguiente manera: Inicialmente, todos las aristas del grafo se ordenan en orden no decreciente de sus pesos. Luego, el algoritmo de Kruskal itera a través de estas aristas ordenados y los agrega al MST si no forman un ciclo con las aristas ya seleccionados. Esta verificación del ciclo se logra utilizando una estructura de datos de conjuntos separados (*Union-Find*). El proceso continúa hasta que hay $V - 1$ aristas en el MST, donde V es el número de vértices en el grafo.

4.4 Estructuras de datos para representar el problema

La estructura que principalmente mejora la eficiencia en este algoritmo es el conjunto disjunto ($Union - Find$), asimismo utilizaremos una matriz que representa las aristas del grafo.

4.5 Orden de complejidad

La complejidad de este algoritmo es de $O(E \cdot \log E)$.

4.6 Conclusión

El algoritmo de Kruskal es un algoritmo ávaro ampliamente conocido que se utiliza para encontrar el árbol de expansión mínimo (MST) de un gráfico ponderado y conectado. La utilización de este algoritmo satisface la solución del ejercicio propuesto.

5 Virtual friends

5.1 Descripción

5.2 Algoritmos para resolver el problema

5.3 Aplicación del algoritmo

5.4 Estructuras de datos para representar el problema

5.5 Orden de complejidad

5.6 Conclusión

6 Internet bandwidth

6.1 Descripción

6.2 Algoritmos para resolver el problema

6.3 Aplicación del algoritmo

6.4 Estructuras de datos para representar el problema

6.5 Orden de complejidad

6.6 Conclusión

References

[1] URL: https://www.geeksforgeeks.org/bipartite-graph/.