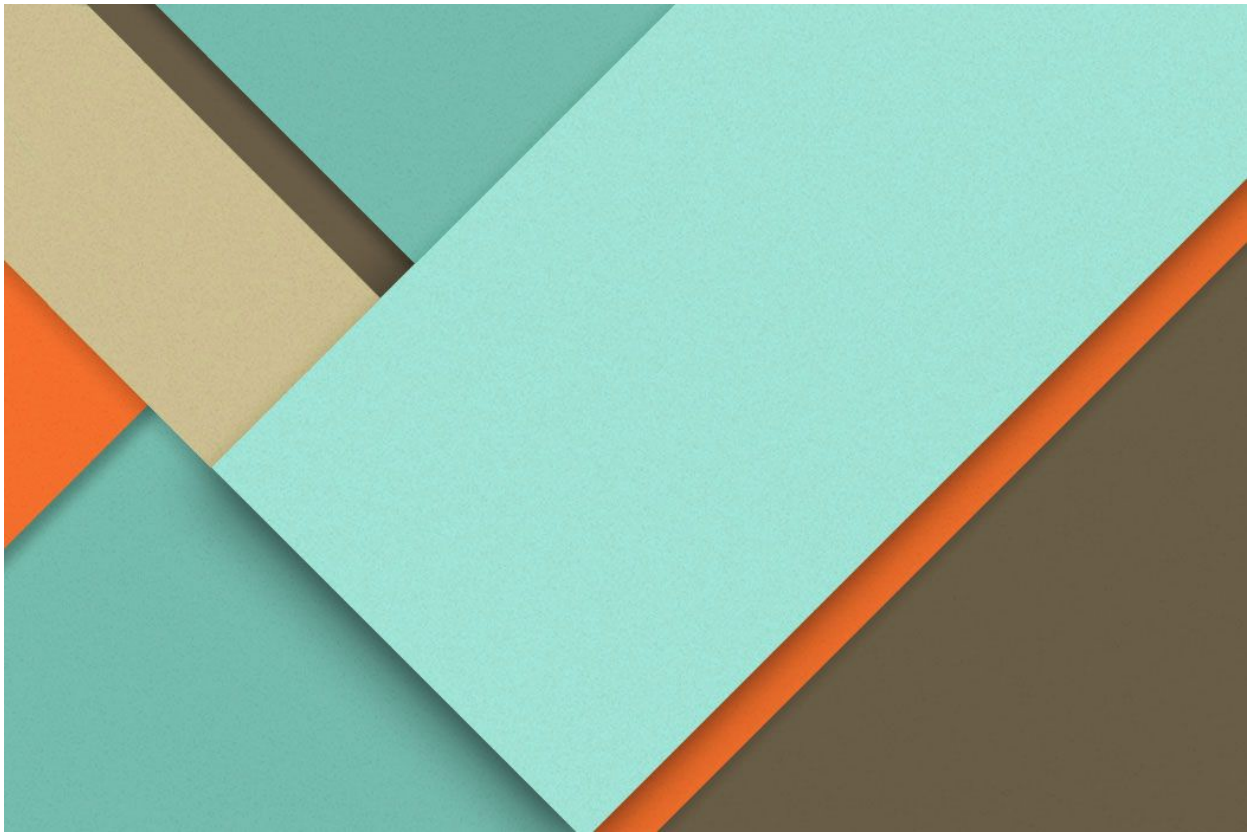


Programación 3 - TUDAI 2019

Trabajo Práctico Especial

19/06/2019

Cánneva, Juan Bernardo
jcanneva@gmail.com
https://github.com/jcanneva/Programacion-3/

Introducción

Continuado con la primera parte realizada, ahora para esta segunda entrega se implementaron distintas soluciones para el problema del agente viajero en base a los aeropuertos y rutas dados anteriormente. Para lograr una solución óptima y con una salida correcta se modificaron dichas ruta y aeropuertos, acotando el dado en un principio.

Objetivos

1. Implementar una solución utilizando Backtracking
2. Implementar una solución utilizando Greedy
3. Analizar las características de cada algoritmo implementado

Análisis del problema

El problema del agente viajero consiste en que, dado un nodo origen, en este caso un aeropuerto, se desea obtener un recorrido, el cual tenga un costo mínimo, es decir la menor distancia posible, y que a su vez pase por todos los nodos que componen el grafo, regresando al final del recorrido al origen.

Para resolver el problema utilizando Backtracking, se generan todas las soluciones posibles, y de todas ellas se obtiene la mejor solución, la cual en este caso sería la de menor distancia.

En cambio para resolverlo con Greedy, lo que se hace es dado el origen, obtener el destino del origen con menor costo. Luego a partir del destino volver a obtener el próximo con menor distancia hasta que ya no quede ningún nodos.

Características de los algoritmos

I. Backtracking

Una característica fundamental de este algoritmo es que consume muchos recursos al ser ejecutado, ya que encuentra todas las soluciones posibles que existen para un problema dado (en este caso el problema del agente viajero).

A su vez, al encontrar todas las soluciones posibles, permite que se encuentre la mejor solución, siempre que esta exista. Esto hace que siempre que haya, encuentre una respuesta óptima.

Dicho esto este algoritmo es implementado cuando se quiere encontrar al menos una solución, o todas ellas, o la mejor de todas las soluciones, siempre que haya una.

II. Greedy

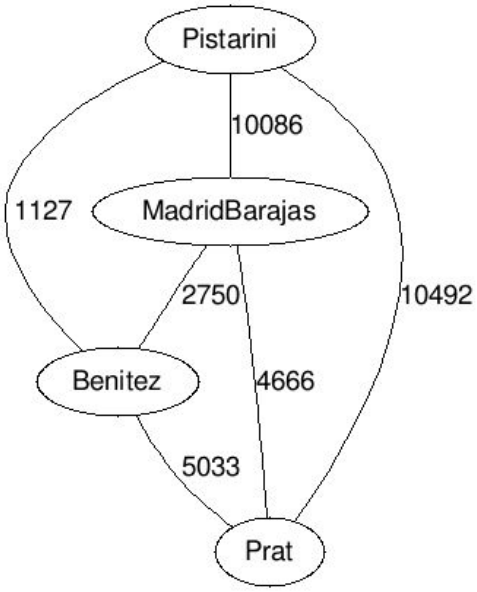
Una de las características principales de Greedy es que obtiene soluciones eficientes sin consumir demasiados recursos, pero no siempre se asegura que se obtenga una solución, aún cuando esta exista.

Su implementación es relativamente sencilla, para ello el algoritmo en cada estado toma la mejor decisión posible y no se replantea las decisiones ya tomadas. Por esto es que, en ciertas circunstancias, no obtiene ninguna solución, aun existiendo, ya que la búsqueda de un óptimo local no implica encontrar un óptimo global. Aunque a veces, se da que encuentra una solución óptima, esto no es garantizado.

Generalmente es aplicado a problemas de optimización, esto quiere decir que se aplica cuando se necesita obtener una solución eficiente, sin consumir demasiados recursos, que no sea la mejor de todas las soluciones posibles pero aún así sea considerada como una solución competente.

Implementación

Como se mencionó en un principio se modificó el Dataset dado por la cátedra, acotando el mismo para poder obtener y realizar un seguimiento más acorde, obteniendo como resultado el siguiente grafo.



I. Backtracking

Para implementarlo , se buscó una solución que genere todas las alternativas posibles en el grafo, dado un vértice de origen, pudiendo así obtener el recorrido de menor distancia. Para esto, en cada estado se pregunta si todos los vértices han sido visitados, y en caso de ser así, se pregunta si el recorrido es solución, es decir si desde el vértice actual se puede llegar al origen, si esto se cumple se guarda el recorrido con menor distancia hasta el momento. Si no es un estado final, es decir todos los vértices no han sido visitados, el algoritmo se ejecuta recursivamente en los arcos de los vértices adyacentes al actual, agregando el arco o ruta, a la lista del recorrido, es decir de solución, que el algoritmo va generando, pero antes de hacerlo se fija si el recorrido actual es menor al que ya se ha obtenido, acotando así el árbol de exploración.

Una vez que retorna de la recursión se vuelve al estado en que el algoritmo estaba antes.

Pseudocódigo del algoritmo:

```
back(universo, actual, solucion, destino) {
        si universo es vacio {
                si es solucion {
                        ruta = getRuta(actual, destino);
                        solucion.add(ruta);
                        solucionBack = ruta;
                        solucion.remove(ruta);
                }
        } sino {
                para cada rutal del actual {
                        si el estado del destino de la ruta es NO_VISITADO {
                                actual = ruta.destino;
                                actual.estado = VISITADO;
                                universo.remove(actual);
                                solucion.add(ruta);
                                if (esMenor(solucion))
                                        back(universo, actual, solucion, destino);
                                solucion.remove(ruta);
                                actual.estado = NO_VISITADO;
                                universo.add(actual);
                        }
                }
        }
}
```

Salida por pantalla con origen Pistarini

```
Ingrese aeropuerto origen:
Ministro Pistarini
Tiempo de ejecucion 3.5838
Mejor solucion:
Ministro Pistarini -- El prat -- Madrid-Barajas -- Comodoro Benitez -- Ministro Pistarini
19035.0
```

Salida por pantalla con origen Benitez

```
Ingrese aeropuerto origen:
Comodoro Benitez
Tiempo de ejecucion 1.0262
Mejor solucion:
Comodoro Benitez -- Ministro Pistarini -- El prat -- Madrid-Barajas -- Comodoro Benitez
19035.0
```

II. Greedy

Para implementar este algoritmo, dado un vértice de origen, se decidió iterar mientras haya vértices sin visitar y origen no sea nulo. Seguido de esto se obtiene el vértice más cercano al origen, es decir el de menor distancia. Luego se agrega el arco entre dichos vértices una lista solución en donde se guarda el recorrido. Después se elimina el vértice del universo y el origen se vuelve el vértice más cercano, volviendo a ejecutar las mismas sentencias en otra iteración.

Una vez finalizado el algoritmo se corrobora que la solución obtenida sea válida.

Pseudocódigo del algoritmo:

```
public List<Ruta> greedy(origen, universo, solucion) {
        si origen no es nulo {
                para cada aeropuerto del universo
                        aeropuerto.estado = NO_VISITADO;
                mientras (universo no sea vacio y origen no sea nulo) {
                        origen.estado = VISITADO;
                        tmp = getAeropuertoCercano(origen);
                        si (tmp no es nulo) {
                                ruta = getRuta(origen, tmp);
                                solucion.add(ruta);
                        }
                        universo.remove(origen);
                        origen = tmp;
                }
                si es solucion
                        return solucion;
                } sino
                        return nulo;
        } sino
                return nulo;

}
```

Salida por pantalla con origen Pistarini

```
Ingrese aeropuerto origen:
Ministro Pistarini
Tiempo de ejecucion 0.3377
Ministro Pistarini -- Comodoro Benitez -- Comodoro Benitez -- Madrid-Barajas -- Madrid-Barajas -- El prat -- El prat -- Ministro Pistarini
Distancia:  19035.0
```

Salida por pantalla con origen Benitez

```
Ingrese aeropuerto origen:
Comodoro Benitez
Tiempo de ejecucion 0.3285
Comodoro Benitez -- Ministro Pistarini -- Ministro Pistarini -- Madrid-Barajas -- Madrid-Barajas -- El prat -- El prat -- Comodoro Benitez
Distancia:  20912.0
```

Conclusiones

Al terminar este trabajo se pudo lograr cumplir los objetivos propuestos en un principio.

Se logró además, poder observar de forma práctica cómo funcionan los algoritmos de Greedy y Backtracking, viendo así las ventajas y desventajas que la aplicación de cada uno de ellos conlleva.

Por un lado Backtracking ocasiona un costo computacional muy grande ya que explora todas las permutaciones posibles, pero al mismo tiempo posibilita encontrar una solución óptima siempre que exista.

En cambio Greedy no garantiza que se encuentre una solución, pero cuando la encuentra, lo hace de forma eficiente, sin un costo alto de tiempo de ejecución, es decir que bajo ciertas circunstancias ofrece una solución aproximada, sin garantías de que sea la óptima, y sin originar un costo computacional elevado.

Cabe aclarar además que a la técnica de Backtracking se le pueden reducir los costos empleando una estrategia de cotas o poda, minimizando así el espacio de búsqueda y por ende el costo computacional.