Centro de Investigación y de Estudios Avanzados
del IPN

Proyecto 1. Viajando por México

Matemáticas Computacionales

Miembros del equipo:
Hernán Guillermo Dulcey Morán
Karla Jacquelin Guzmán Sánchez

25 de Octubre de 2019

Introducción

Se requiere comparar dos tipos de algoritmos (búsqueda exhaustiva y algoritmo voráz), implementados en una aplicación web que determina la mejor ruta para visitar ciudades de México. Se debe explicar que tecnología se usó, además de considerar sus complejidades y determinar que ventajas tienen uno respecto al otro.

Búsqueda Exhaustiva

El algoritmo de búsqueda exhaustiva consiste en recorrer y generar todas las soluciones posibles a un problema (en este caso, generar rutas de viaje).

Pseudocódigo

Inicio

1. Recibir la ubicación inicial y las ubicaciones seleccionadas a las que se desea viajar.
2. Crear una variable auxiliar que almacenará la *mejor ruta* encontrada (inicializada en vacio).
3. Realizar todas las **permutaciones** posibles entre las ubicaciones seleccionadas y almacenarlas en un arreglo de soluciones, con los atributos "ruta" y "distancia".
4. **Para cada** posible solución i:
   (a) **Añadir** la ubicacion inicial al inicio y final de la ruta i como parte de la solución.
   (b) **Calcular la distancia** de forma ordenada entre las ubicaciones de cada solución i y almacenarla en el atributo distancia de la solución i.
   (c) **Si** la solución i actual es mejor (posee menor distancia) que la solución almacenada en la variable auxiliar *mejor ruta*, **entonces** la solución i actual es ahora la mejor ruta.
5. **Mostrar** la mejor ruta encontrada y las demás soluciones posibles calculadas.

Fin

Explicación

El algoritmo consiste en crear un vector de soluciones que contiene las listas resultantes de permutar las ubicaciones seleccionadas, es decir, cada elemento del vector de soluciones contiene una posible ruta de viaje. Además, el vector de soluciones tiene un atributo llamado distancia, donde se almacenará la distancia resultante de visitar las ubicaciones en el orden indicado.

Cada elemento i del vector de soluciones tiene entonces la siguiente forma:

$$\text{solución } i = \{ \text{ lista de ubicaciones, distancia } \}$$

Una vez que se crean todas las permutaciones, y por lo tanto, todas las rutas (soluciones) posibles, se procede a calcular la distancia entre las ubicaciones. Esto sería:

Para cada elemento i del vector de soluciones, se calcula la distancia entre todas las ubicaciones de la *lista de ubicaciones* que contiene, en el orden en el cual están enlistadas y dicha distancia se almacena en el atributo *distancia* del vector de soluciones.

Cuando termina de calcular la distancia entre las ciudades de la solución i, antes de calcular la siguiente distancia (solución $i + 1$), el algoritmo compara si la distancia que recien calculó es más corta que la distancia que antes consideraba mejor (*mejor ruta*), y en caso de que la distancia actual sea mejor que la anterior, la distancia actual pasa a ser la *mejor ruta* y con esto, la mejor solución.

Nota: En la primer iteración, la *mejor ruta* siempre se reemplaza con la distancia de la solución actual.

Al terminar de calcular las distancias para cada elemento del vector de soluciones, el algoritmo ya conoce cual es la mejor ruta y todas las posibles soluciones.

Complejidad computacional

El algoritmo puede dividirse en 2 partes fundamentales:

a) Realizar las permutaciones de las ubicaciones.

b) Calcular la distancia de las rutas y determinar la mejor ruta.

Siendo n la cantidad de ciudades seleccionadas, tenemos que:

La primer parte del algoritmo (realizar las permutaciones), realiza operaciones de complejidad $n!$ porque nos da como resultado $n!$ soluciones.

La segunda parte del algoritmo (calcular las distancias), se ejecuta $n + 2$ veces para cada solución porque es necesario calcular la distancia entre las n ciudades, y además entre *dos* puntos más que serían la ubicacion inicial y la ubicación final, siendo ambas la misma ubicación. Como resultado de las permutaciones de las ciudades se tienen $n!$ soluciones. Por lo tanto, esta segunda parte ejecuta $n! * (n + 2)$ operaciones.

La complejidad total del algoritmo está dada por la función $n! + n! * (n + 2)$. Utilizando la notación Big-O, concluimos que la complejidad del algoritmo de búsqueda exhaustiva es de **O(n!)**.

Algoritmo voráz

El algoritmo utilizado fue una búsqueda en dos direcciones. Primero se va a considerar las dos ciudades más cercanas al punto de origen, creando una ruta de ida y otra ruta de regreso. De esta manera se calculan las distancias más cortas en un sentido casi circular del recorrido.

Pseudocódigo

1. busquedaVoraz(ubicacionInicial, ubicaciones)

    2. solucion = $\emptyset$
    3. ordenCiudades = $\emptyset$

4. regresoCiudades = ∅
5. ordenCiudades.añadir(menorDistancia(ubicacionInicial,ubicaciones))
6. ubicaciones = quitarUbicacion(ordenCiudades[0], ubicaciones)
7. si(ubicaciones != ∅)
    8. regresoCiudades.añadir(menorDistancia(ubicacionInicial,ubicaciones))
    9. ubicaciones = quitarUbicacion(regresoCiudades[0], ubicaciones)
10. mientras(indice < ubicaciones.longitud)
    11. ordenCiudades.añadir(menorDistancia(ubicacionInicial,ubicaciones))
    12. ubicaciones = quitarUbicacion(ordenCiudades[indice], ubicaciones)
    13. regresoCiudades.añadir(menorDistancia(ubicacionInicial,ubicaciones))
    14. ubicaciones = quitarUbicacion(regresoCiudades[indice], ubicaciones)
15. mientras(indice < regresoCiudades.longitud)
        //Añadir primero los últimos términos
    16. ordenCiudades.añadir(regresoCiudades[indice])
17. solucion = convertirASolucion(ordenCiudades)

Explicación

Este algoritmo recibe la ubicación inicial y la lista de ubicaciones seleccionadas por el usuario. Inicialmente se busca la ciudad más cercana al punto inicial, si es la única ciudad seleccionada entonces termina la ruta, en caso contrario va a seleccionar otra ciudad cercana al punto inicial (a esta ruta se le denomina regreso). Este proceso se va a repetir por cada ubicación que el usuario haya seleccionado hasta que ya no hayan más ubicaciones para elegir. Como se puede observar tiene dos ciclos, el primer ciclo va a recorrer las ubicaciones dadas por el usuario y empezará a poblar los dos arreglos (orden de ida y regreso). Este proceso le tomará máximo n/2 iteraciones, donde n es el número de ubicaciones seleccionado, puesto que por cada recorrido se están seleccionando dos objetos del arreglo de ubicaciones. Al finalizar el primer ciclo debe unir los arreglos para dar la solución final, solamente que el último elemento del arreglo de regreso es la continuación del último elemento del arreglo de ida. por lo que la concatenación debe hacerse al revés. Este proceso se realiza como mucho n/2 veces puesto que la mitad de las ubicaciones ya estan en el arreglo final. Finalmente se añaden la posición inicial al principio y al final del arreglo resultante para aparezcan en la impresión que se le hace al usuario.

Complejidad computacional

Debe considerarse que al calcular la menor distancia, el algoritmo tiene que compararse con la lista de ubicaciones. La lista se va a ir decrementando a medida que se seleccionan ciudades, pero su ecuación sigue siendo de orden n, por lo tanto este proceso toma O(n^2). Al tener 2 ciclos, uno de (n^2)/2 y otro de n/2, el algoritmo se ejecuta como mucho n^2 veces por lo que su complejidad es $\mathbf{O}(n^2)$.

Tecnologías utilizadas

AngularJS

AngularJS es un framework de desarrollo web creado por google para codificar lógica del lado del cliente. El framework utiliza javascript como lenguaje principal aunque sus archivos son en su mayoría typescript (para integración con arquitectura JSON). Angular está diseñado para implementar aplicaciones web con fácilidad, por lo que fue elegido por la necesidad del proyecto.

Conclusión

El algoritmo voráz es capaz de encontrar una solución en un tiempo razonable pero no necesariamente es la mejor solución. El algoritmo exhaustivo es capaz de encontrar la mejor solución pero como debe analizar cada una de ellas, le toma mucho tiempo para un espacio de búsqueda muy grande. En este caso se puede afirmar que cada algoritmo sacrifica una caracteristica para poder compensar lo que le hace falta. Cabe resaltar que en ciertos casos el algoritmo voráz puede encontrar la mejor solución o una solución muy cercana a la mejor, por lo que generalmente se recomienda utilizar una busqueda voráz en casos que no dependan de la exactitud del resultado.