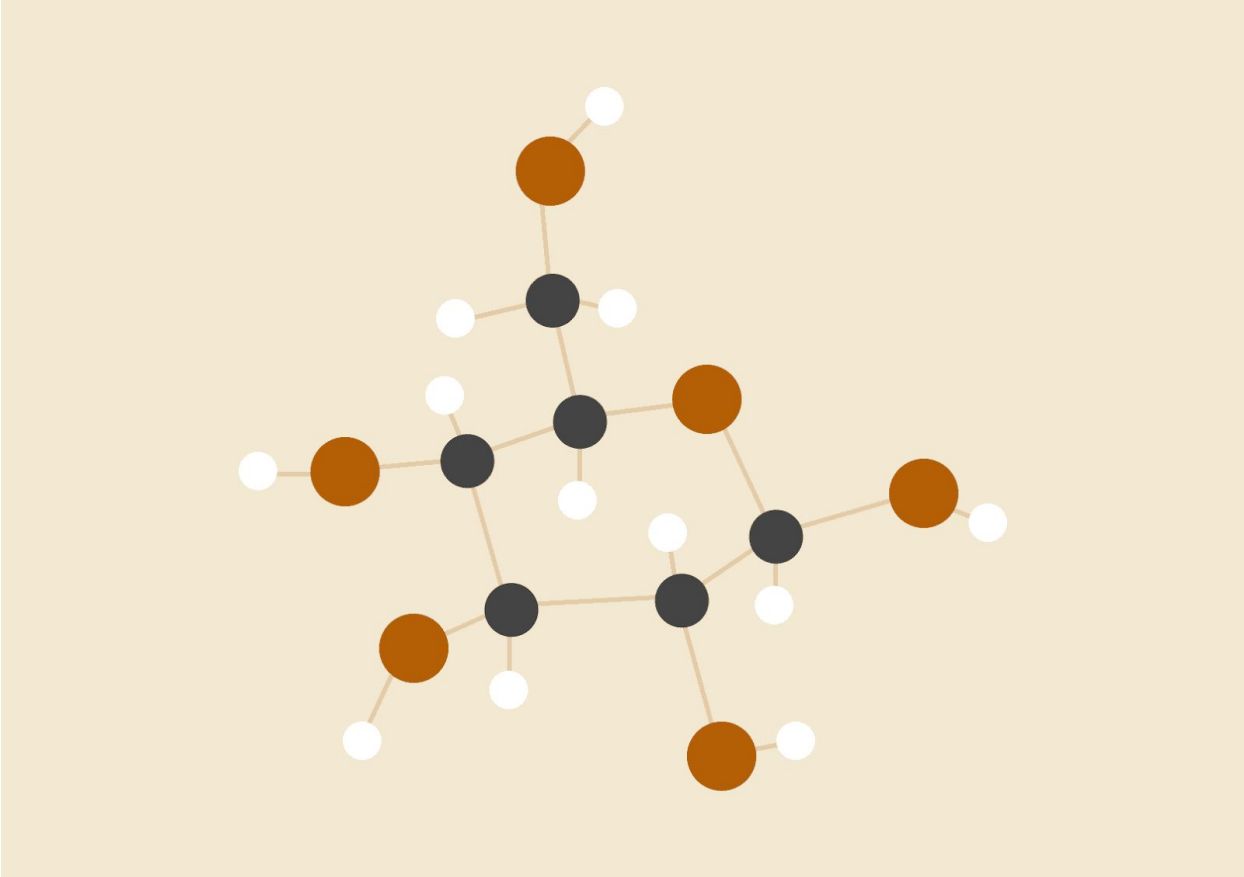
Algoritmos básicos en grafos

Respuestas a las cuestiones teóricas



Alfonso Camblor García

25/10/2020

Diseño y Análisis de Algoritmos - Práctica 1

Índice

Respuestas a las preguntas teóricas planteadas en la Práctica 1 de la asignatura Diseño y Análisis de algoritmos 2020-2021.

Cuestiones

Preguntas III-B: Dijkstra

1. Tanto el problema de distancias mínimas como el algoritmo Dijkstra se aplican en principio a grafos estándar, esto es, sin ramas que auto-apunten a un vértice y con a lo sumo una rama entre dos nodos distintos. ¿Cómo se definiría el problema de distancias mínimas en multigrafos? ¿Como habrá que modificar el algoritmo de Dijkstra si fuera necesario?

2. For a probability probability=0.5, apply the function time_dijkstra_mg, fit a linear model An2 log n + B to the times in the returned list and plot the real and fitted times discussing the results. To fit the model, use the template in the Scikit-learn section of the Python notes.

3. Para un grafo estándar generado con una probabilidad ρ, argumentar que podemos esperar que $|E| \simeq \rho * |V|^2$. Expresar el coste de Dijkstra sobre un grafo estándar en función del número de nodos = |V| y la probabilidad ρ del grafo en cuestión. Para un número de nodos fijo adecuado, ¿Cuál es el crecimiento del coste de Dijkstra en función de ρ? Ilustrar este crecimiento ejecutando Dijkstra sobre grafos con un número fijo de nodos y sparse factors = {0.1, 0.3, 0.5, 0.7, 0.9} y midiendo los correspondientes tiempos de ejecución.

4. ¿Cuál es el coste teórico del algoritmo de Dijkstra iterado para encontrar las distancias mínimas entre todos los vértices de un grafo estándar?

5. ¿Cómo se podrían recuperar los caminos mínimos si se utiliza Dijkstra iterado?

Respuestas III-B: Dijkstra

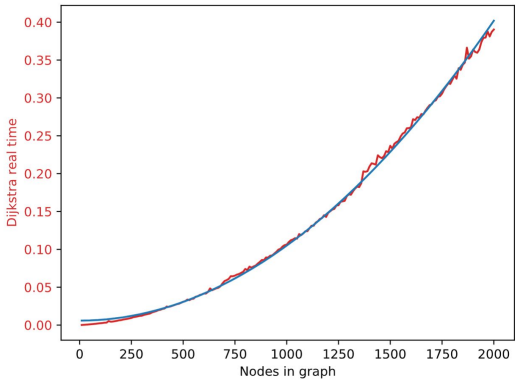
III-B.1

El problema cambia en lo que implica la iteración entre posibles ramas entre nodos. Ahora, se añade la posibilidad de que diferentes enlaces entre dos nodos tengan una diferencia de coste, por lo que la cuestión a resolver será la de escoger el camino con distancia mínima entre dos nodos, que dentro de su estructura, escoja el camino menos costoso entre vértices adyacentes.

No habría una modificación demasiado notoria, ya que cambiaría el analizar las ramas principales entre dos vértices, de forma que ahora se analizarán todas las ramas existentes entre los vértices seleccionados, lo que a pesar de su simple implementación, resultaría en un aumento del coste del algoritmo.

III-B.2

Dado que conocemos el orden del coste del algoritmo de Dijkstra para todos los nodos del grafo: $O(|V|^2 * log(|V|)$, entrenamos un modelo lineal $An^2 * log(n) + B$ con los resultados obtenidos del algoritmo, que ha sido puesto a funcionar en el rango desde n=10 hasta n=2000, con saltos de n=10, siendo los valores propuestos por el modelo los correspondientes a la línea azul.



En la gráfica, se puede observar una gran correlación entre los valores que hemos demostrado teóricamente sobre el algoritmo y los valores obtenidos en tiempo de ejecución. Las pequeñas variaciones en el tiempo real que ha tardado en ejecutarse se deben a eventos ocurridos en el ordenador donde ha tenido lugar la computación, pero no son lo suficientemente significativos como para invalidar los resultados teóricos, que se adaptan correctamente a los valores obtenidos.

III-B.3

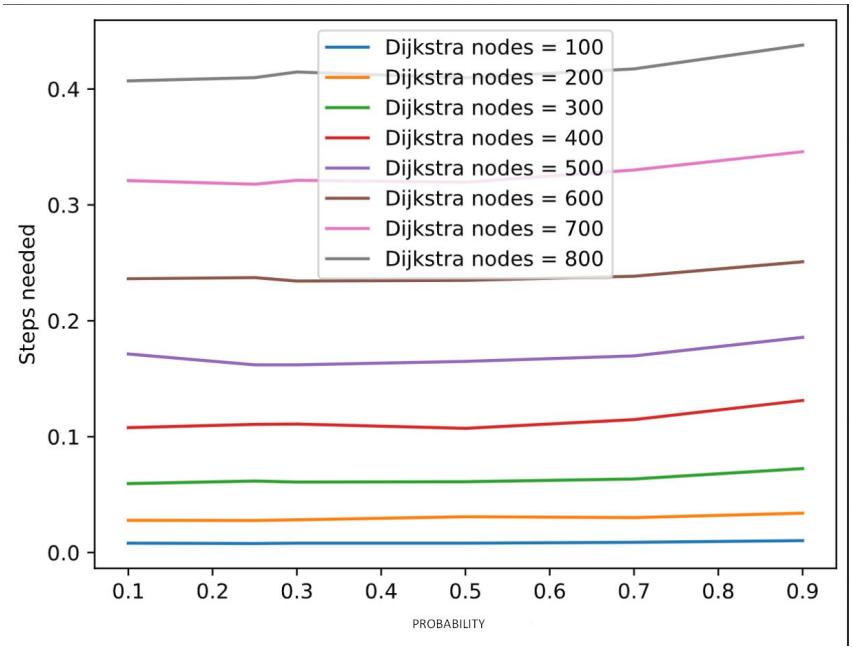
Inicialmente partimos del coste de Dijkstra: $O(|V|) + O(|E| * log(|E|))$.

Al aplicar la probabilidad p en la aparición de enlaces en los grafos, el coste de Dijkstra pasa a ser afectado por el número de enlaces, de forma que ahora definimos el número de enlaces como $|E^p| = |E| * p$, de forma que si sustituimos en la fórmula de Dijkstra, se cumple:

$$O(|V|) + O(|E^p| * log(|E^p|)\ ;\ O(|E^p| * log(|E^p|) = O(|E| * p * log(|E * p|)$$

$$Como\ |V| < |V|^2 * log(|V|) \Rightarrow Coste\ Dijkstra = O(|V^2| * p * log(|V| * p)$$

Para comprobarlo, se realiza la computación para los número de nodos desde 100 hasta 800 con saltos de 100, probando para cada uno de los valores en el rango seleccionado las probabilidades [0.1, 0.3, 0.5, 0.7, 0.9].



A partir de esta gráfica, comprobamos que diferentes valores de sparse factor afectan en el tiempo que tarda el algoritmo en realizar las operaciones necesarias para la resolución del problema, tal y como describimos anteriormente, como un multiplicador sobre el coste de Dijkstra. El tiempo que Dijkstra tarda es proporcional al valor de la probabilidad de aparición de enlaces en el grafo.

III-B.4

El coste teórico de Dijkstra es $O((vértices + enlaces) * log(vértices))$, pero podemos comprimirlo en la expresión: $O(E * log(V))$.

A partir de ahora, $vértices = v$, $enlaces = e$.

El tipo de estructura de datos que empleamos para la resolución del problema también afecta. Dado que utilizamos una cola de prioridad, esto genera un coste $O(v * log(v))$ teniendo en cuenta que almacenará los vértices a analizar y el coste de sacarlos será log(v).

La operación más restrictiva es $O(e * log(v))$, la cual proviene de la búsqueda del camino más corto, dado que el hecho de encontrar el camino más corto habiendo especificado el vértice, es $O(log(v))$. Sin embargo, esta operación se repetirá para todos los enlaces, por lo que el coste pasa a ser $O(e^* log(v))$.

En total, el coste de Dijkstra iterado es, hallando para cada vértice el camino mínimo y utilizando una cola de prioridad, es:

$$O(v * e * log(v)) = O(V^3 * log(V))$$

III-B.5

En el caso de Dijkstra, una vez se haya encontrado el camino más corto, se puede hacer backtracking desde el nodo destino hacia sus predecesores, encontrando de esta forma un camino desde el nodo destino hacia el nodo origen escogidos, que podemos invertir para hallar el camino mínimo desde el nodo origen a el nodo destino. Se implementaría recorriendo la lista de predecesores y guardando el predecesor de cada nodo, empezando por el nodo destino.

Preguntas IV-B: Floyd-Warshall

1. ¿Cuál es el coste teórico del algoritmo Floyd–Warshall para la misma tarea?

2. Suponiendo que se va a trabajar con grafos con una probabilidad ρ, ¿para que valores de ρ y del número de nodos N debería ser Floyd–Warshall más competitivo que Dijkstra iterado? Contrastar las conclusiones obtenidas con ejemplos y gráficas concretos obtenidas mediante grafos de diferentes tamaños , N y ρ = 0,5.

3. ¿Cómo se podrían recuperar los caminos mínimos si se utiliza Floyd–Warshall?

Respuestas IV-B: Floyd-Warshall

IV-B.1

Para computar Floyd-Warshall utilizamos una matriz, en la que el coste de encontrar todos los n^2 $de\ M_k\ desde\ M_{k-1}$ $\Rightarrow$ $2n^2\ operaciones$.

Comenzamos teniendo $M_0\ hasta\ M_n$ $\Rightarrow n * 2n^2\ operaciones = 2n^3\ operaciones$.

Es por esto que Floyd-Warshall tiene un coste $O(n^3)$. Más adelante comprobaremos que esta “estabilidad” en el coste provoca que sea más eficiente en grafos densos que Dijkstra.

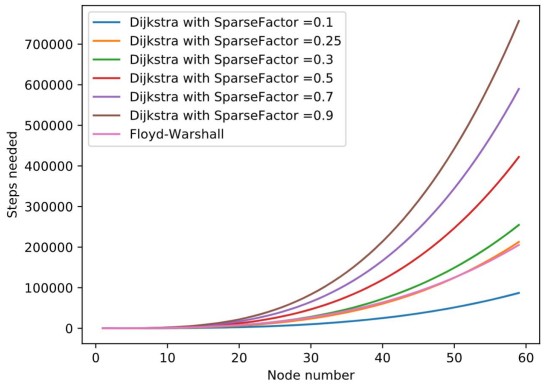
IV-B.2

Si partimos de fijar el número de nodos:

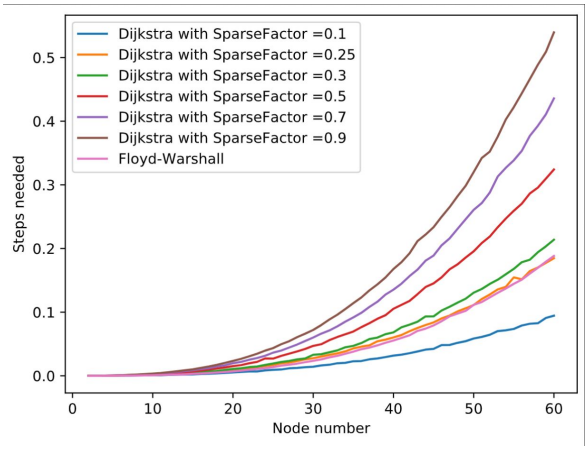
$V^3 \rho\ log(V) \geq V^3 \Leftrightarrow \rho \geq 1/log(V)$, es decir, el factor de densidad debe ser inversamente proporcional al resultado del logaritmo sobre el número de nodos.

Si ahora fijamos $\rho = 0.5:\ 0.5 \geq 1/log(V) \Leftrightarrow V \geq 4$, se cumple a partir de grafos de 4 nodos.

A continuación se muestran dos gráficas. La primera corresponderá a las estimaciones teóricas, mientras que la segunda corresponderá a los resultados reales a la hora de ejecutar los algoritmos.



En esta primera gráfica ya comprobamos que para grafos densos, es decir, con muchos enlaces, el algoritmo de Floyd-Warshall supera en términos de eficiencia al algoritmo de Dijkstra. Sin embargo, en grafos cuya probabilidad de enlace es pequeña, concretamente, a partir de un valor entre 0.25 y 0.3, el algoritmo de Floyd-Warshall comienza a ser más eficiente que el algoritmo de Dijkstra.



Comprobamos que los valores obtenidos, aplicando la probabilidad de enlace junto con la variación del número de enlaces por nodo para generar los resultados correctamente, corresponden a los valores teóricos anteriormente calculados, encontrándose Floyd-Warshall entre los tiempos de Dijkstra correspondientes a probabilidades de entre 0.25 ± 0.02 y 0.3, demostrando la hipótesis inicial:

En grafos densos ($|E| \geq 0.23 * |V|^2$), Floyd-Warshall es más eficiente que Dijkstra.

IV-B.3

Para la reconstrucción de caminos en Floyd-Warshall es necesaria la construcción de una segunda matriz *path_to* cuyo contenido sean los índices de los vértices (Esta matriz debe estar inicializada a un valor que no genere conflicto, por ejemplo, NULL) y en el momento de realizar el cálculo correspondiente a comparar caminos en Floyd Warshall, añadir también el nodo anterior a la nueva matriz.

[...]

```
if dist[i][j] > dist[i][k] + dist[k][j] :
        dist[i][j] = dist[i][k] + dist[k][j]
        path_to[i][j] = path_to[i][k]
        path_to[i][k] = path_to[k][j]
```

[...]

Ahora, tenemos preparada la matriz de reconstrucción de caminos, por lo que para recorrerla, el algoritmo de obtención de caminos sería:

```
reconstruir_Camino (inicio, fin) :
        if next[fin][inicio] = NULL :
                return [] # No hay nodo anterior. Camino reconstruido.
        path_to = [fin] # Iniciamos la reconstrucción
        while inicio != fin :  # Mientras no se llegue al nodo inicial
                fin = path_to[inicio][fin] # Obtener el nodo anterior
                path_to.append(fin) # Añadimos a la lista el nodo anterior
        return reverse(path_to)
```

De esta forma, obtenemos el camino mínimo desde el nodo inicial al final a partir del algoritmo Floyd-Warshall.