ADA : Primera entrega

Nil Domene Esteban, Diego Velazquez Dorta
1425988,1429086

Universitat Autònoma de Barcelona

Resumen Primera entrega de la práctica de ADA, donde se nos pide implementar dos algoritmos: el algoritmo Dijkstra y una modificación de este, el Dijkstra con cola de prioridades que disminunye la complejidad temporal.

Índice general

1. Algoritmo Dijkstra

1.1. Definición:

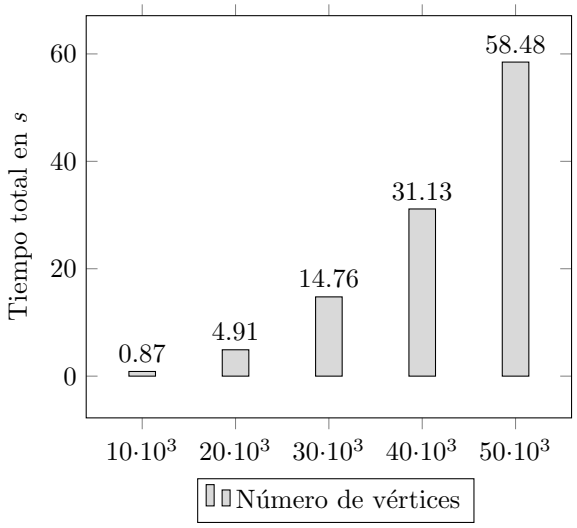
És un algoritmo que se usa para la determinación de los caminos más cortos entre un vértice origen, y el resto de vértices. El funcionamiento de este algoritmo es el siguiente, va calculando los costes asociados a cada uno de los vecinos del vértice actual, y actualiza su valor, si tenian un coste mayor asociado.Una vez hecho esto, seleccionamos el vértice con la menor distancia asociada.
La complejidad temporal de este algoritmo es: $O(V^2)$, donde V es el número de vértices, su complejidad espacial en también es $O(V^2)$.

1.2. Pseudocódigo:

Pseudocódigo genérico del algoritmo Dijkstra:

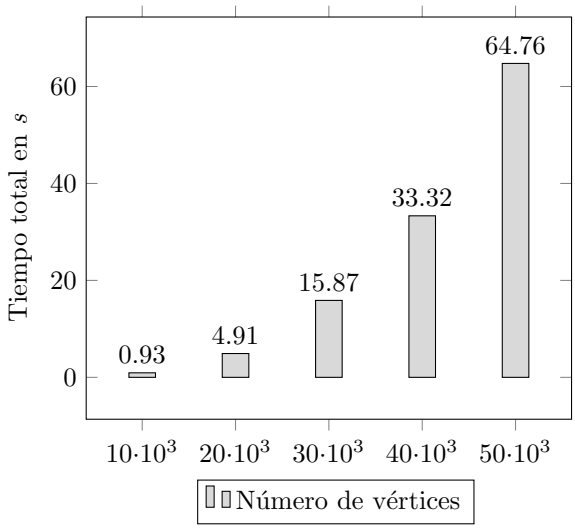
```
function Dijkstra(Grafo, origen):

    creamos el conjunto de vértices Q

    para cada v en Grafo:
        dist[v] = INFINITO
        prev[v] = NULO
        añadimos v a Q

    dist[origen] = 0

    mientras Q no esté vacía:
        u = vértice en Q con menor dist[u]
        quitamos u de Q
        para cada vecino  v de u:
            alt = dist[u] + distancia_euclidea(u, v)
            si alt < dist[v]:
                dist[v] = alt
                prev[v] = u

    retorno dist[], prev[]
```

1.3. Gráficas:

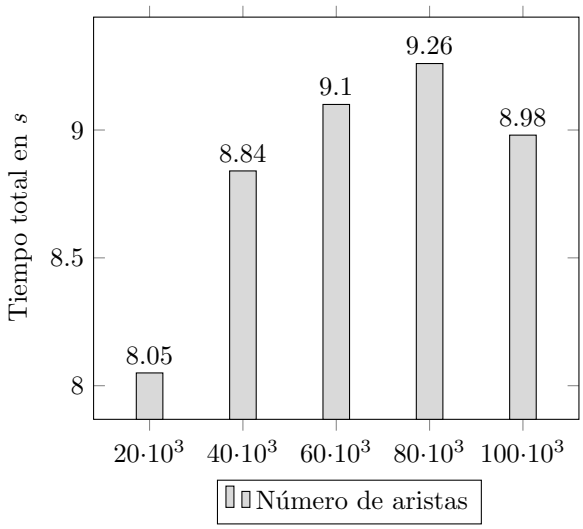


Este grafo es el resultado de aplicar el algoritmo sobre grafos generados aleatoriamente con un total de $25{\cdot}10^3$ aristas como parámetro fijo.Aquí se muestra

como el tiempo crece de forma quadrática en funcion de los vértices generados, comprobamos que a partir de $120{\cdot}10^3$ vértices el tiempo que tardaba era demasiado grande como para poder representarlo en un mismo diagrama de barras

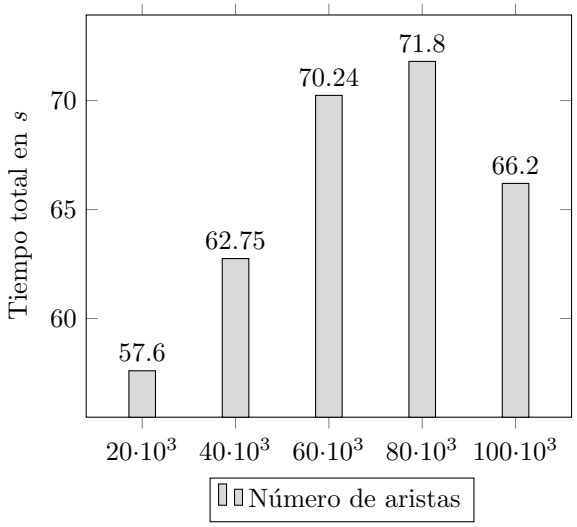


Aquí se analizan siguiendo el mismo estándar que en el gráfico anterior, pero el número total de aristas que contienen los grafos aleatorios ahora son $50 \cdot 10^3$, como se muestra en el diagrama, un número de aristas bastante más elevado que el anterior no influye de forma significativa en el comportamiento del algoritmo.



Se puede ver, que una vez que fijamos el total de vértices, los tiempos se mantienen prácticamente constantes, se ha ampliado el gráfico para que se puedan

apreciar las pequeñas diferencias, ya que sino eran prácticamente 4 líneas iguales. Para este diagrama de barras, hemos usado grafos con 25 $\cdot 10^3$ aristas.



De nuevo, se ve que aquí los valores no fluctuan en exceso dependiendo del número de vértices que se tengan, es cierto que si que hay diferencia entre usar $20 \cdot 10^3$ vértices y $70 \cdot 10^3$, pero no es realmente significativa ya que puede deberse a factores externos como podrían ser los accesos a memoria que realiza el sistema.

2. Algoritmo Dijkstra con cola de prioridades

2.1. Definición:

El funcionamiento de este algortimo, es el mismo que el Dijkstra original, con la diferencia de que aquí usamos una estructura auxiliar que nos va a permitir aumentar en gran medida la eficiencia del algoritmo .La complejidad del algoritmo on esta mejora es, $O(|E|log|V|)$, donde E son el número de aristas y V el número de vértices.Incluso si se produciera el peor caso su complejidad no seria cúadratica sino que quedaría en quasilinear $(O(|E| + |V| \cdot log(|V|)))$. La complejidad espacial del algoritmo sigue siendo $O(V^2)$.
En el apartado de gráficas podemos ver cambios muy significativos en los tiempos asociados a la resolución de grafos de grandes dimensiones.

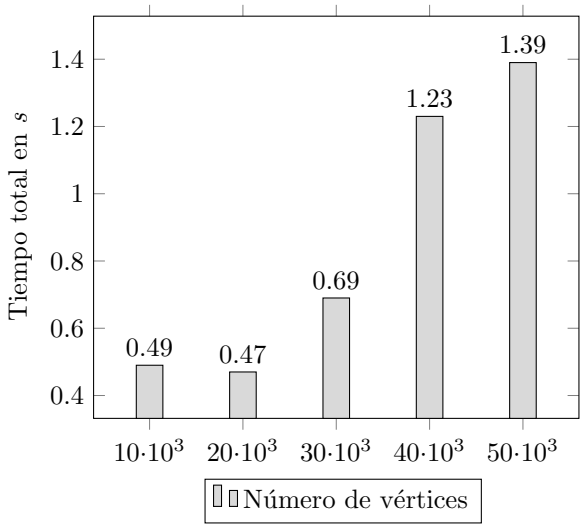
2.2. Pseudocódigo:

Pseudocódigo genérico del algoritmo Dijkstra con cola de prioridades:

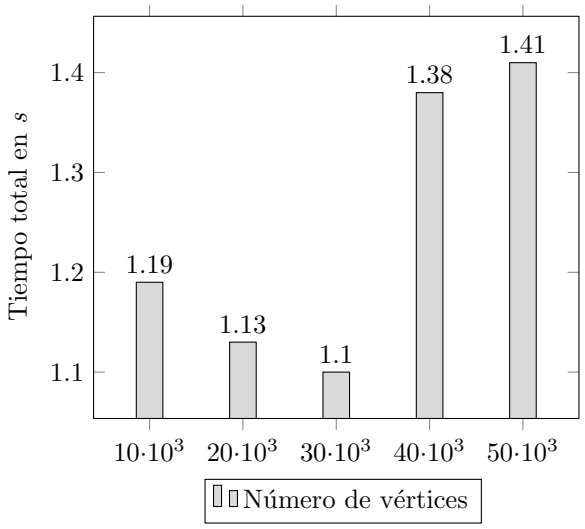
```
function Dijkstra(Grafo, origen):

    dist[origen] = 0
```

```
crear el conjunto de vérices Q

para cada vértice v en Grafo:
    if v != origen
        dist[v] = INFINITO
        prev[v] = NULO
    Q.añadir_con_prioridad(v, dist[v])

mientras Q no esté vacía:
    u = Q.extraemos_mínimo()
    para cada vecino v de u:
        alt = dist[u] + distancia_euclidea(u, v)
        if alt < dist[v]
            dist[v] = alt
            prev[v] = u
    Q.decrementamos_prioridad(v, alt)

retorno dist[], prev[]
```

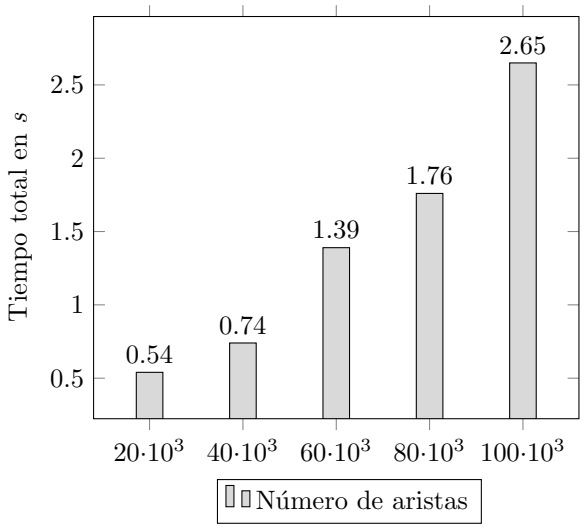
2.3. Gráficas:



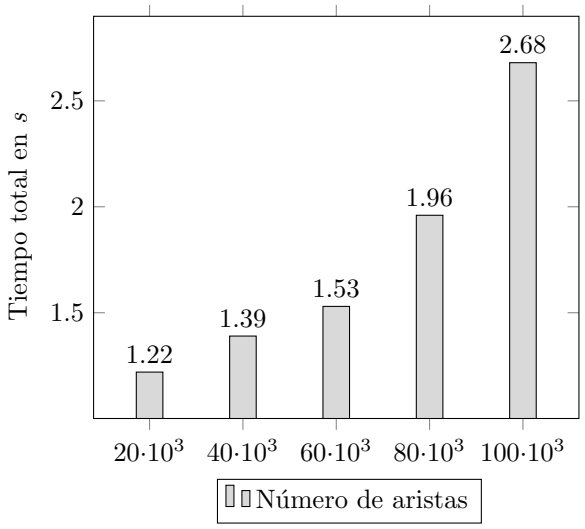
Mirando el gráfico nos podemos dar cuenta, que esta modificación del algoritmo no se ve afectada en exceso por el número de vértices, aquí se habían fijado el número de aristas a $25 \cdot 10^3$.



En este otro diagrama, se puede observar como el valor que influye realmente en el funcionamiento del algoritmo es el número de aristas que tiene el grafo, ya que todos los tiempos sufren un incremento del 100 % prácticamente en valores relativamente pequeños, a grandes cantidades de vértices tiende a seguir el comportamiento del diagrama anterior. Estos cálculos se han hecho sobre grafos con $50 \cdot 10^3$ aristas.



En este gráfico se puede ver la evolución quasilinear del algoritmo, ya que ahora si que vemos unas variaciones significativas de los tiempos del algoritmo, el tiempo que tarda se ve afectado directamente por los cambios de proporción de aristas en el grafo, este análisi se ha hecho sobre un número fijo de $25 \cdot 10^3$ vértices.



En este grafo se puede observar que el número de aristas es realmente el parámetro influyente, si nos fijamos podemos ver que el tiempo se mantiene más o menos constante independientemnete del número de vértices(hasta que se convierte en un valor muy grande y es penalizado por el tiempo de acceso a memoria), podemos ver que el valor limitante del rendimiento siempre es el número de aristas. Estos cálculos se han hecho sobre grafos con $25 \cdot 10^3$ vértices.

3. Datos sobre el ordenador donde se han obtenido los resultados:

Los algoritmos han sido ejecutados en un máquina virtual (VirtualBox) en un MacBook Pro de principios de 2015, las características de la máquina virtual son las siguientes:

- Sistema Operativo: Windows 10 (64 bits)
- Memoria RAM: 5120 MB
- Memoria de vídeo: 128 MB
- Tipo de almacenamiento : Almacenamiento reservado dinámicamente
- Tamaño virtual: 32 GB
- Versión de Visual Studio: Visual Studio Community 2017

Al ejecutar los algoritmos en una máquina virtual, el resultado puede no ser el más óptimo posible, con esto nos referimos, a que los tiempos pueden llegar a variar considerablemente si no se estuviera trabajando sobre un entorno virtual, pero esto no cambia la interpretación de los resultados ya que la complejidad del problema en ningún momento se ve afectada por el sistema sobre el cual se trabaja, y es por tanto que el comportamiento de estos seguirián el mismo patrón en cualquier sistema dónde sea ejecutado, conservando una relación de proporcionalidad.

4. Conclusiones:

Si bien es cierto, que el algoritmo de dijkstra con cola de prioridades tiene una complejidad espacial mayor, esto no nos importa ya que no la tenemos en cuenta de cara al análisis.
Una vez dicho esto, creemos que importante destacar el gran impacto que tiene la cola de prioridades en la eficiencia del algoritmo, en un principio nos asustamos un poco porque con un número reducido de vértices y aristas, la versión del algoritmo con cola de prioridad no mejoraba el rendimiento, y incluso provocaban un empeoramiento cuando las dimensiones eran extremadamente reducidas.
Esta práctica nos ha ayudado a familiarizarnos con la libreria STL de C++, ya que no la habíamos utilizado nunca, y hemos visto que su uso es muy importante en la resolución de problemas que tengan grandes dimensiones, ya que su implementación facilita mucho la realización de los algoritmos. Otra cosa que nos ha llamado la atención, es que mientras buscabamos el pseudocódigo del algoritmo de cara a implementarlo, hemos visto que se puede implementar usando montículos y auque es cierto que la complejidad usandolos no mejora (su complejidad equivale a $0(|E| \cdot log(|V| + |E|))$ donde E es el número de aristas y V el número de vértices del grafo), si que podría ser interante comprender como funciona el algoritmo en otras estructuras.