Practica 1B - AMC

Algoritmos Voraces

Nombre: Saúl Rodríguez Naranjo

ÍNDICE

Algoritmo Dijkstra

Coste de las soluciones óptimas

berlin52.tsp

ch130.tsp

ch150.tsp

- **Para el archivo berlin52, el coste de la solución óptima es 570. El archivo en el que se encuentra es berlin52.opt.tour**
- **Para el archivo ch130, el coste de la solución óptima es 662. El archivo en el que se encuentra es ch130.opt.tour**
- **Para el archivo ch150, el coste de la solución óptima es 857. El archivo en el que se encuentra es ch150.opt.tour**

Análisis de Complejidad

Algoritmo Dijkstra

El pseudocódigo de nuestro algoritmo de Dijkstra es el siguiente:

Como podemos observar, al tener un bucle while y un bucle for, nuestra complejidad seria de $O(n^2)$.

No obstante, si implementaramos una cola de prioridad para almacenar las rutas mas cortas, obtendríamos una complejidad O(|A| + |V| log |V|). Donde A son las Aristas del Grafo y V los Nodos/Vértices.

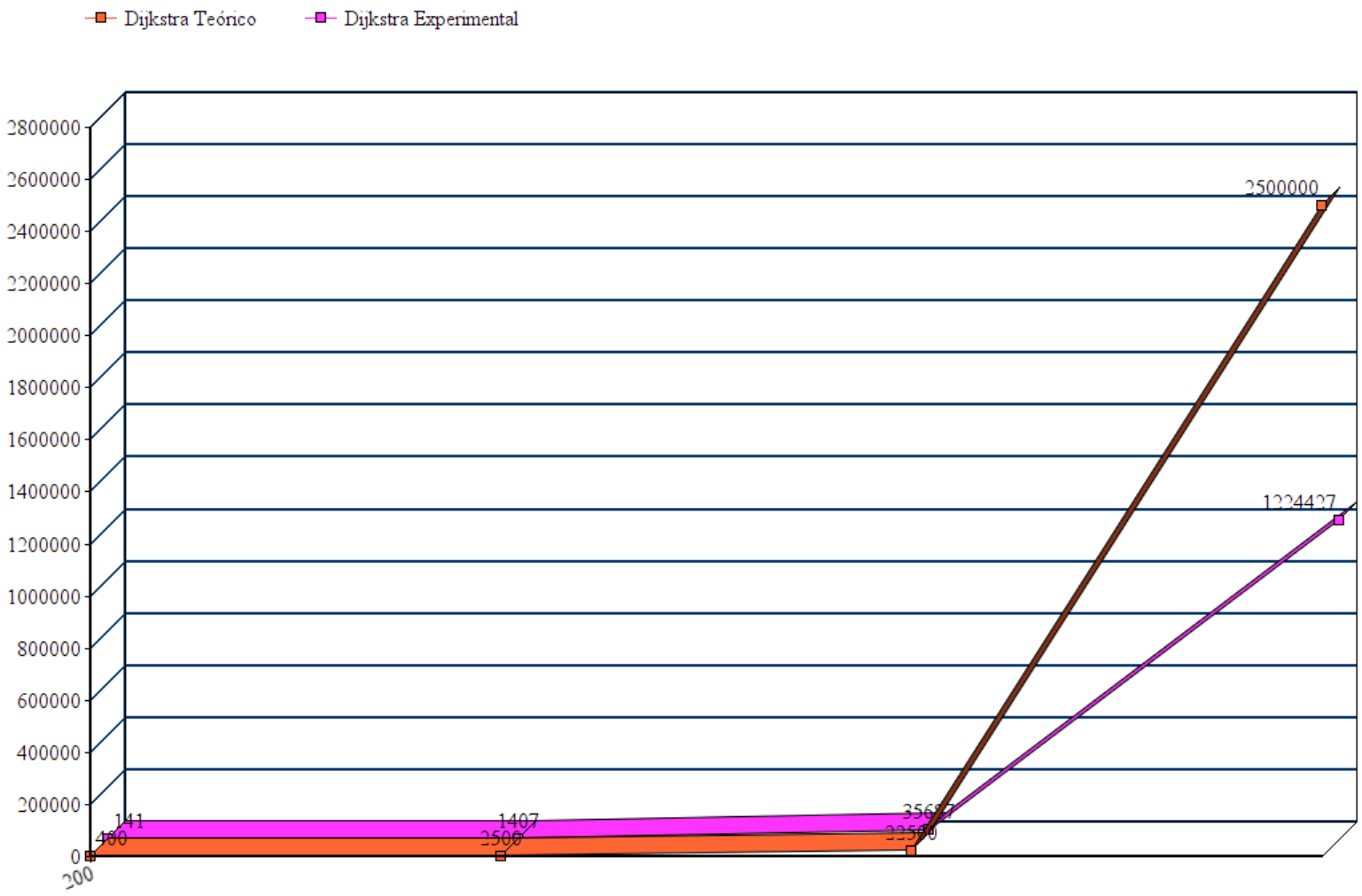
```
funcion Dijkstra(Nodo inicio) return Rutas

        Diccionario Rutas = nuevo Diccionario()

        Diccionario Visitados := nuevo Diccionario()

        Para i := 0 Hasta numero_de_Nodos
            poner_en_Visitados(obtenerNodo(i), falso)
        fPara

        Diccionario Distancias := nuevo Diccionario()

        Para i := 0 Hasta numero_de_Nodos
            aux := obtenerNodo(i)
            poner_en_Distancias(aux, infinito);
        fPara


        Distancias.reemplaza(inicio, 0)
        permanente := inicio
        Visitados.reemplaza(inicio, verdadero)
        Ruta.añade(inicio)
        A_visitados.quita(inicio);
        poner_en_Rutas(permanente, Ruta)

        Mientras A_visitados no sea vacio Hacer

            Aristas := permanente.obtenerAristas()

            Para i := 0 Hasta tamañoAristas Hacer
                distancia_preliminar := Distancias.obtener(permanente);
                examinar := Aristas.obtener(i).obtenerDestino();
                Si no_he_visitado(examinar)
                    distancia_preliminar = distancia_preliminar + Aristas.get(i).getPeso();

                    Si distancia_preliminar < Distancias.obtener(examinar)
                        Distancias.reemplazar(examinar, distancia_preliminar)

                        Ruta := Rutas.obtener(permanente)

                        Ruta.añadir(examinar)

                        poner_en_Rutas(examinar, Ruta)
                    fSi
                fSi
            fPara

            permanente := this.getMenorDistancia(Distancias, Visitados)
            A_visitados.eliminar(permanente)
            Visitados.reemplazar(permanente, verdadero)
        fMientras

        return Rutas
ffuncion
```

Comparación de los resultados teóricos y experimentales

Tallas de:
200
500
1500
5000

Talla	Algoritmo de Dijkstra (Teórico)	Algoritmo de Dijkstra (Experimental)
200	400 ms	141 ms
500	2500 ms	1407 ms
1500	22500 ms	35687 ms
5000	2 500 000 ms	1 224 427 ms

Se han realizado 10 ejecuciones para cada talla y cogido la media de ellos redondeándola a un valor entero.


Dijkstra Teórico
Dijkstra Experimental
2800000
2600000
2400000
2200000
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
200
141
1407
2500000
1224427