UNIVERSIDAD DE GRANADA

E.T.S. DE INGENIERÍAS INFORMÁTICA y DE TELECOMUNICACIÓN

Departamento de Ciencias de la Computación e Inteligencia Artificial

Algorítmica

Guión de Prácticas

Práctica 2: Algoritmos Greedy

Curso 2011-2012

Grado en Informática

Práctica 2

Algoritmos Greedy

1 Objetivos y Evaluación

El objetivo de la práctica es que el alumno comprenda y asimile el funcionamiento de la técnica de diseño de algoritmos"greedy". Esta técnica consiste básicamente en obtener una solución para el problema tratado mediante un proceso iterativo en el que se va construyendo una solución parcial. En cada paso del proceso constructivo se escoge la mejor decisión posible para la componente actual de la solución considerada, teniendo en cuenta las decisiones tomadas hasta el momento (la solución parcial) y las opciones no escogidas aún.

La práctica se evalúa sobre **0.5 puntos**. Se valorará la correcta implementación del algoritmo. La no consideración de las especificaciones de formato y demás cuestiones expresadas en este guión podrán influir negativamente en la evaluación de la práctica.

2 El Algoritmo de Caminos Mínimos de Dijkstra

2.1 Descripción del Problema

Dado un grafo ponderado con pesos positivos $G(V, A)$, donde V es el conjunto de vértices y A el de aristas, y dado un vértice s de dicho grafo, el problema consiste en encontrar el camino de coste mínimo para llegar desde s a todos los demás vértices de G. El coste de un camino se define como la suma de los pesos de los arcos que lo componen.

2.2 El algoritmo de Dijkstra

Existen diversos algoritmos para resolver el problema de caminos mínimos. Uno de los más extendidos es el propuesto por Edsger Dijkstra en 1959. El método está basado en la técnica greedy de diseño de algoritmos. En cada paso del algoritmo, se consideran dos conjuntos de vértices, los **escogidos** S, es decir, aquellos vértices ya seleccionados y que son actualmente considerados en los caminos mínimos desde el origen, y los **candidatos** V/S, es decir, aquellos que no han sido seleccionados aún y para los que no se conoce todavía el camino de distancia mínima desde el vértice origen. De este modo, los candidatos

componentes de la solución son los vértices y la función de selección escoge en cada paso el vértice de V/S con menor distancia al vértice origen s.

El algoritmo emplea un vector D de dimensión n (el número de vértices) para almacenar las distancias del camino mínimo que lleva del origen s a cada vértice. Inicialmente, $D[i] = \infty, \forall i \neq s$; $D[s] = 0$ y $S = \{s\}$. En cada paso, se escoge el vértice candidato $u \in V/S$ que menor distancia presente al origen s. Una vez incluido dicho vértice u en S, se estudia si procede la actualización de las distancias de s a los nodos adyacentes de u considerando caminos que pudieran pasar por u de acuerdo a la siguiente operación:

Para todo $v \in V$ tal que v es adyacente de u, si la distancia actual de s a v, $D[v]$, es mayor que la distancia de ir de s a u y después de u a v, $D[u] + A[u, v]$, entonces se actualiza $D[v]$ como $D[v] = D[u] + A[u, v]$.

Una vez finalizada la ejecución del algoritmo, es decir, tras n pasos, momento en el que $S = V$, el vector D que será devuelto como salida incluye las distancias correspondientes a los caminos mínimos para ir de s a cualquier otro vértice del grafo G. Si además se quieren almacenar dichos caminos, se usa un vector adicional de dimensión n, denominado P, de tal modo que $P[i]$ guarda el vértice predecesor (el padre) de i en el camino de mínima distancia que lleva hasta s. El vector P está inicializado de forma que $pred[i] = null$ y se actualiza a la vez que D de forma que $P[i] = x$ cuando $D[v] > D[x] + A[x, v]$.

2.3 Implementación considerando una Cola con Prioridad

La implementación más eficiente del algoritmo de Dijkstra considera el uso de una cola con prioridad Q para almacenar en cada paso los vértices candidatos de V/S, ordenados de acuerdo a su distancia al vértice origen s. El pseudocódigo del algoritmo es el mostrado a continuación:

```
1  DIJKSTRA (Grafo G(V,A), vertice-origen s)
2          Para cada vertice u de V hacer
3              D[u] = INFINITO
4              P[u] = NULL
5          D[s] = 0
6
7          Para cada vertice u de V hacer
8              Insertar(Q,(v,D[v]))
9
10         Mientras que Q no es vacia hacer
11             u = Extraer-minimo(Q)
12             Para cada vertice v adyacente a u hacer
13                 Si D[v] > D[u] + A[u,v] hacer
14                     D[v] = D[u] + A[u,v]
15                     P[v] = u
16                     Actualizar(Q,(v,D[v]))
17
18         Devolver D y P
```

La eficiencia de este algoritmo que considera la implementación usando cola con prioridad es $O((|E| + |V|)log|V|)$, donde E es el número de arcos y V el número de vértices.

3 Tareas a Realizar

Se proporciona al alumno un código que incluye la clase $grafo$ que almacena un grafo como una matriz de adyacencia. Se han implementado las operaciones de asignación y recuperación del peso de un arco (*asignar_peso* y *devolver_peso*), así como las de recuperación del tamaño del grafo (*size*) y obtención del primer y último vértice adyacentes de uno dado (*begin_ady* y *end_ady*). Por otro lado, se dispone de la clase *heap* que implementa una cola con prioridad con sus operaciones habituales: inserción (*insert*), extracción del elemento situado en la primera posición (*erase_min*), comprobación de si la cola está vacía (*empty*), comprobación de si un elemento está incluido (*en_heap*) y actualización de los valores de prioridad de los elementos incluidos en la cola (*actualizar_heap*). Igualmente, se proporciona una implementación del algoritmo de Prim para la creación de árboles generadores minimales que hace uso de las clases anteriores. El algoritmo de Prim es otro método que trabaja sobre grafos que también está basado en la filosofía greedy.

El trabajo del alumno consiste en realizar una implementación del algoritmo de Dijkstra basada en las clases anteriores y en ejecutarla sobre dos casos del problema. Para realizar dicha implementación, se puede tomar como base la existente para el algoritmo de Prim. Como resultado de esta práctica, el alumno entregará el código realizado así como los resultados obtenidos en los dos casos, es decir, los vectores D y P para cada uno de ellos. Los casos a considerar son los siguientes:

1. Ftv70.atsp: Grafo de tamaño 71.

2. Kro124p.atsp: Grafo de tamaño 100.

Ambos casos están disponibles en ficheros de texto comprimidos en el directorio de la asignatura y comparten el mismo formato. El mostrado a continuación es el correspondiente al caso $Kro124p$:

```
NAME: kro124p
TYPE: ATSP
COMMENT: Asymmetric TSP (Fischetti)
DIMENSION:  100
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION
A[1,1] \dots A [1,124] \dots A[2,1] \dots A[100,100]
EOF
```

Los únicos campos tiles para esta práctica son $DIMENSION$, que indica el número de vértices del grafo y $EDGE\_WEIGHT\_SECTION$, que almacena la matriz de adyacencia de forma explícita. Será responsabilidad del alumno el implementar una pequeña función para leer los datos desde fichero y almacenarlos en un objeto de la clase grafo.

4 Documentación y Entrega de la Práctica

La práctica deberá entregarse antes del **MIERCOLES 16 DE MAYO DE 2012**. La entrega se realizará telemáticamente a través de la Web de la asignatura accesible desde `https://decsai.ugr.es`.

El nombre del fichero que contiene toda la documentación se compondrá añadiendo los apellidos y nombre del alumno separados por el símbolo de subrayado `_`, sin tildes y sustituyendo la letra 'ñ' por la letra 'n'. Por ejemplo, el alumno José Caños Pérez subirá un fichero de nombre `Canos_Perez_Jose_practica_1.zip` que a su vez contendrá los siguientes ficheros:

```
dijkstra.cpp
greedy_dijkstra.pdf
```

El primero de ellos corresponde a la implementación realizada. El segundo será la memoria de la práctica en la que se describirá dicha implementación así como su ejecución sobre los dos casos mencionados y se incluirá un apartado con las soluciones obtenidas en cada uno de ellos, es decir, los vectores D y P.