UNIVERSIDAD NEBRIJA

Escuela Politécnica Superior

Estructuras de datos y algoritmos

Caso práctico

El Problema del Viajante de Comercio (*Traveling Salesman Problem - TSP*) forma parte de los problemas de mayor complejidad conocidos, catalogado como NP-Completo porque no existe un algoritmo capaz de resolver este problema con complejidad polinomial. El Problema del Viajante de Comercio se describe fácilmente, pero es muy difícil de resolver. El problema consiste en encontrar el recorrido (*tour*) más corto que visite un conjunto de ciudades para las que se conoce la distancia entre ellas. Durante el *tour* se debe visitar cada ciudad una sola vez y regresar a la ciudad origen del viaje. Es decir, si un viajante parte de una ciudad y conoce las distancias entre las distintas ciudades que debe visitar, ¿cuál es el *tour* óptimo que visita todas las ciudades una sola vez y vuelve a la ciudad de partida?

El origen de este problema es incierto. Se conoce un manual para viajantes, de 1832, que menciona el problema con ejemplos de viajes por Alemania y Suiza, pero no incluye una formulación matemática. Entre 1856 y 1858, los matemáticos Thomas Kirkman y William R. Hamilton plantean cuestiones relacionadas con este problema. Entre 1931 y 1932, la forma general del Problema del Viajante de Comercio es estudiada por primera vez por los matemáticos Karl Menger en Viena y en Harvard y por Hassler Whitney y Merril Flood en Princeton[1].

Desde su formulación matemática en los años 30[2], es muy relevante en la investigación operativa y en las ciencias de la computación. El Problema del Viajante de Comercio se hizo muy popular tres décadas más tarde.

---

[1] Schrijver, A. On the History of Combinatorial Optimization (till 1960). Handbooks in Operations Research and Management Science, 2005, vol. 12, p. 1-68

[2] Lawler, E. L., et al. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. 2007

En 1962, Procter and Gamble (P&G) promocionó un concurso dotado con un premio de 10.000 dólares para quien encontrara la ruta más corta entre 33 ciudades de Estados Unidos[3].

El Problema del Viajante de Comercio tiene muchas aplicaciones prácticas en el ámbito de logística o de la industria electrónica. Se puede aplicar para calcular la ruta óptima de reparto de mercancías entre distintas ciudades o para minimizar los desplazamientos de las máquinas que fabrican circuitos impresos, por poner solo algunos ejemplos.

Heurísticas para el Problema del Viajante de Comercio

El caso práctico consiste en desarrollar dos heurísticas para resolver el Problema del Viajante de Comercio.

La primera heurística es el algoritmo del vecino más próximo. Dado un vértice v de partida, selecciona la arista con el coste mínimo que conecta con un vértice w que no forma parte del *tour*. Una vez seleccionado w, se utiliza como punto de partida y se busca nuevamente una arista que conecte con un vértice aún sin seleccionar. Esto se repite sucesivamente hasta que todos los vértices se han seleccionado. Finalmente, el último vértice se une con el vértice v de partida para completar el *tour*.

La segunda heurística es una variación del algoritmo de Prim para calcular el árbol de recubrimiento de coste mínimo de un grafo. Cada vez que se selecciona la arista de coste mínimo, se comprueba que el grado de sus dos vértices es menor que 2 para evitar un ciclo. Una vez que se tiene el árbol de recubrimiento de coste mínimo, basta con añadir la arista que une los vértices que tienen grado menor que 2.

[3] Cook, W. J. In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton, NJ. Princeton University Press, 2011

Diseño del programa

La clase Grafo

La clase `Grafo` utiliza una matriz de adyacencias para representar los vértices y el coste de las aristas del grafo. Los vértices de un grafo se identifican empezando por 1. El vértice 1 corresponde con la posición cero de la matriz de adyacencias.

```
class Grafo {

public:

   Grafo(int v);
   Grafo(int v, std::vector<std::vector<int>> costes);
   ~Grafo();

   int coste(int v, int w);
   void inserta(int v, int w, int coste);
   int totalVertices();

   static int* dijkstra(Grafo G);
   static Grafo prim(Grafo G);

   static std::list<AristaGrafo> viajanteComercioVecino(Grafo G, int v);
   static std::list<AristaGrafo> viajanteComercioPrim(Grafo G, int v);


   std::string imprime(std::string s);
   std::string profundidad(int inicio);

   static std::string imprimeVector(std::string s,
      int* vector, int n);

private:

   static const int INFINITO =
      std::numeric_limits<unsigned short int>::max();

   int vertices;
   int **aristas;

   static void costeMinimoArista(Grafo G, Conjunto U,
      Conjunto W, int &u, int &w);

   int verticeNoVisitado(bool *visitado, int vertices);
   std::string profundidad(int v, bool *&visitado);

};
```

La clase Conjunto

La clase `Conjunto` ofrece operaciones de unión, intersección o resta de conjuntos.

```
class Conjunto {

public:

   Conjunto(int c);
   Conjunto(int c, bool pertenece);
   ~Conjunto();

   bool vacio() const;
   void inserta(int e);
   void elimina(int e);
   bool pertenece(int e);
   bool esIgual(Conjunto A);

   static Conjunto une(Conjunto A, Conjunto B);
   static Conjunto resta(Conjunto A, Conjunto B);
   static Conjunto intersecta(Conjunto A, Conjunto B);

   std::string imprime(std::string s);

private:

   int cardinalidad;
   bool *elementos;

};
```

La clase AristaGrafo

```
class AristaGrafo {

public:

   int v, w, coste;

   AristaGrafo(int v1, int v2, int c) {
      v = v1;
      w = v2;
      coste = c;
   }

};
```

Heurísticas voraces para el Problema del Viajante de Comercio

Desarrolle dos heurísticas, la del vecino más próximo y una variación del algoritmo de Prim.

El vecino más próximo

```
std::list<AristaGrafo> viajanteComercioVecino(Grafo G, int v) {

   inicializa el conjunto V con todos los vértices del grafo G

   inicializa el conjunto T (el tour) con el vértice de partida v

   inicializa el conjunto W = V – T (los vértices no seleccionados)

   inicializa la lista aristas del tour

   // u toma el valor de v, el vértice de partida

   u = v

   // mientras el conjunto de vértices de T (el tour) no contiene todos
   // los vértices del grafo

   while (T != V)

      // busca la arista de coste mínimo A(u, w) tal que w pertenece W

      // añade la arista A(u, w) a la lista de aristas del tour
      // añade el vértice w al conjunto T de vértices del tour
      // elimina el vértice w del conjunto W

      // el vértice u toma el valor de w, el último vértice seleccionado

   end wuile

   // finalmente, se añade la arista (w, v) para completar el tour con el
   // vértice de partida v

   // devuelve la lista de aristas del tour

};
```

Variación del algoritmo de Prim

```
std::list<AristaGrafo> viajanteComercioPrim(Grafo G, int v) {

   inicializa el conjunto V con todos los vértices del grafo G

   inicializa el conjunto T (el tour) con el vértice de partida v

   inicializa el conjunto W = V – T (los vértices no seleccionados)

   inicializa a cero el vector D que almacena el grado de los vértices

   inicializa la lista aristas

   // mientras el conjunto de vértices T (el tour) no contiene todos los
   // vértices del grafo

   while (T != V)

      // busca la arista de coste mínimo A(u, w) tal que el vértice u pertenece
      // al conjunto T, el vértice w pertenece al conjunto W y el grado de los
      // vértices u y w es menor que 2

      // incrementa el grado de los vértices u y w
      // añade la arista A(u, w) a la lista de aristas del tour

      // añade el vértice w al conjunto T de vértices del tour
      // elimina el vértice w del conjunto W

   end while

   // finalmente, selecciona la arista A(u, w) cuyos vértices tienen grado
   // menor que 2 en D y la añade a las aristas del tour

   // devuelve la lista de aristas del tour

};
```

Ejemplo de la salida por la consola

El siguiente ejemplo muestra cómo se podrían mostrar los resultados:

```
Tour del viajante de comercio del Vecino próximo, desde A Coruña

A Coruña - León             334
León - Oviedo               118
Oviedo - Valladolid         252
Valladolid - Zamora          96
Zamora - Salamanca           62
Salamanca - Cáceres         210
Cáceres - Sevilla           264
Sevilla - Córdoba           138
Córdoba - Málaga            187
Málaga - Murcia             407
Murcia - Valencia           241
Valencia - Zaragoza         326
Zaragoza - Soria            157
Soria - Burgos              141
Burgos - Bilbao             158
Bilbao - San Sebastián      119
San Sebastián - Pamplona     92
Pamplona - Madrid           407
Madrid - Barcelona          621
Barcelona - A Coruña       1118

El coste total del tour es 5448


Tour del viajante de comercio basado en el algoritmo de Prim, desde A Coruña

A Coruña - León             334
León - Oviedo               118
Oviedo - Valladolid         252
Valladolid - Zamora          96
Zamora - Salamanca           62
Salamanca - Cáceres         210
Cáceres - Sevilla           264
Sevilla - Córdoba           138
Córdoba - Málaga            187
Málaga - Murcia             407
Murcia - Valencia           241
Valencia - Zaragoza         326
Zaragoza - Soria            157
Soria - Burgos              141
Burgos - Bilbao             158
Bilbao - San Sebastián      119
San Sebastián - Pamplona     92
Pamplona - Madrid           407
Madrid - Barcelona          621
Barcelona - A Coruña       1118

El coste total del tour es 5448
```

Memoria

La memoria de la práctica debe incluir los resultados del grafo de 20 ciudades de la tabla 1.

Calcule el tour del viajante de comercio con ambas heurísticas utilizando todas las ciudades como punto de partida. ¿Desde qué ciudad se debe comenzar el viaje para obtener el coste menor? Compare los resultados de las heurísticas propuestas. ¿Con cuál se obtiene el coste más bajo? ¿Cuáles son las características y las limitaciones de las heurísticas voraces?

Puntuación

Este caso práctico es obligatorio y se realiza en grupos de dos alumnos. Si el caso práctico no se entrega se puntúa con 0. La nota mínima de este trabajo es 5,0.

El trabajo debe ser realizado por los dos alumnos del grupo. En caso de que uno de los alumnos no sepa responder a las preguntas planteadas, la práctica se considera suspensa para ambos alumnos.

El criterio de puntuación es el siguiente:

- [4 puntos] Funcionamiento correcto de los algoritmos propuestos para el Problema del Viajante de Comercio
- [4 puntos] Estructura del programa, diseño adecuado de las clases y métodos. Claridad del código y comentarios en el código
- [2 puntos] Presentación de la memoria: resultados, conclusiones, calidad de la redacción y uso correcto de los términos y conceptos desarrollados en clase

Entrega

La práctica se debe entregar en un fichero ZIP en el Campus Virtual antes del 24 de mayo a las 22:00 h. El fichero ZIP debe incluir el código fuente y la memoria en formato PDF.

Tabla 1. Distancias en km. entre ciudades españolas

	A Coruña	Barcelona	Bilbao	Burgos	Cáceres	Córdoba	León	Madrid	Málaga	Murcia	Oviedo	Pamplona	Salamanca	San Sebastián	Sevilla	Soria	Valencia	Valladolid	Zamora	Zaragoza
A Coruña		1118	644	535	683	995	334	609	1153	1010	340	738	473	763	947	676	961	455	411	833
Barcelona	1118		620	583	918	908	784	621	997	590	902	437	778	529	1046	453	349	663	759	296
Bilbao	644	620		158	605	795	359	395	939	796	304	159	395	119	993	257	633	280	376	324
Burgos	535	583	158		447	637	201	237	781	638	322	203	237	232	775	141	517	122	218	287
Cáceres	683	918	605	447		319	407	297	506	654	525	650	210	679	264	490	636	325	272	622
Córdoba	995	908	795	637	319		733	400	187	444	851	807	529	869	138	631	545	578	616	725
León	334	784	359	201	407	733		333	877	734	118	404	197	433	671	342	685	134	135	488
Madrid	609	621	395	237	297	400	333		544	401	451	407	212	469	538	231	352	193	248	325
Málaga	1153	997	939	781	506	187	877	544		407	995	951	756	1013	219	775	648	737	792	869
Murcia	1010	590	796	638	654	444	734	401	407		852	714	613	807	534	589	241	594	649	539
Oviedo	340	902	304	322	525	851	118	451	995	852		463	315	423	789	463	803	252	253	604
Pamplona	738	437	159	203	650	807	404	407	951	714	463		440	92	945	176	501	325	421	175
Salamanca	473	778	395	237	210	529	197	212	756	613	315	440		469	474	325	564	115	62	482
San Sebastián	763	529	119	232	679	869	433	469	1013	807	423	92	469		1007	268	594	354	450	268
Sevilla	947	1046	993	775	264	138	671	538	219	534	789	945	474	1007		769	697	589	536	863
Soria	676	453	257	141	490	631	342	231	775	589	463	176	325	268	769		376	210	306	157
Valencia	961	349	633	517	636	545	685	352	648	241	803	501	564	594	697	376		545	600	326
Valladolid	455	663	280	122	325	578	134	193	737	594	252	325	115	354	589	210	545		96	367
Zamora	411	759	376	218	272	616	135	248	792	649	253	421	62	450	536	306	600	96		463
Zaragoza	833	296	324	287	622	725	488	325	869	539	604	175	482	268	863	157	326	367	463	