Práctica 3: Algoritmos Greedy

Manuel Jesús Núñez Ruiz
Miguel Pedregosa Pérez

Índice

Problema Común: Viajante de comercio

Estrategia basada en cercanía

Elementos del algoritmo

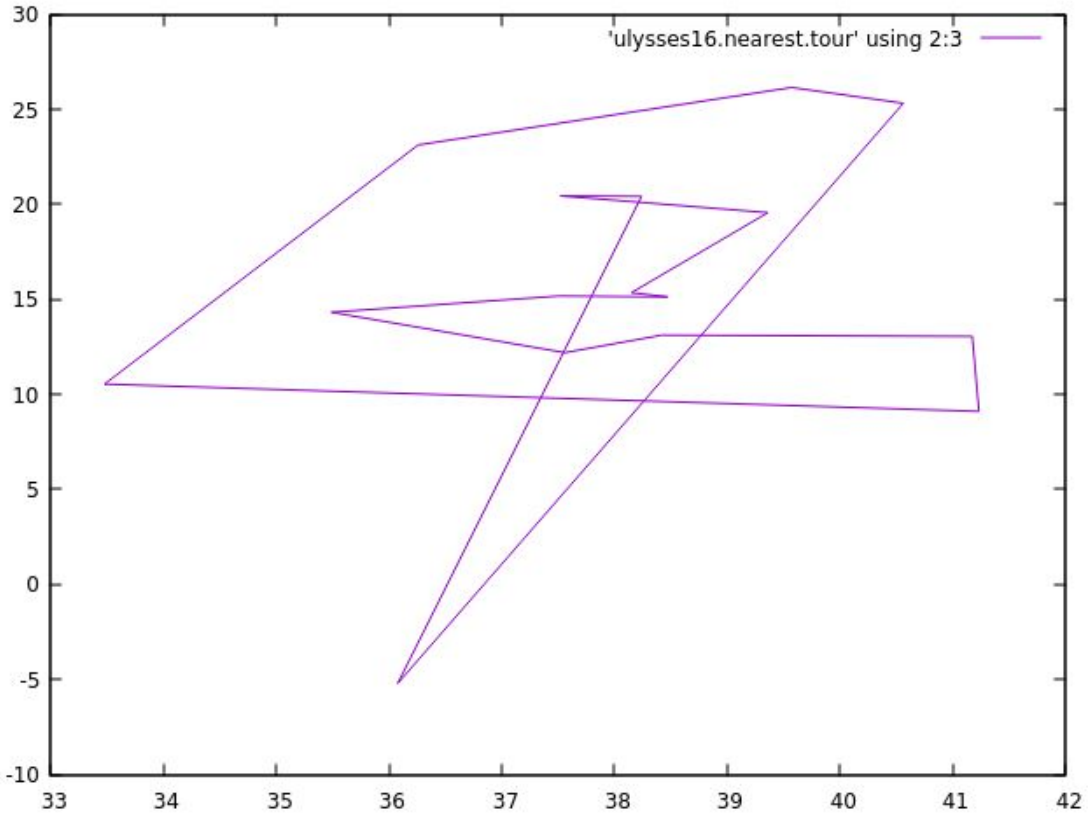
1. Conjunto de candidatos: todas las ciudades por las que tiene que pasar el viajante de comercio y la matriz con las distancias entre todas ellas.
2. Candidatos ya usados: ciudades por las que ya ha pasado.
3. Función solución: obtener un Circuito Hamiltoniano Minimal.
4. Criterio de factibilidad: al seleccionar una arista no se debe de formar ciclo y además tiene que cumplir la condición de no ser incidente en tercera posición al nodo escogido.
5. Función de selección: dada una ciudad v_0 se escoge como ciudad siguiente aquella v_i que se encuentre más cercana a v_0.
6. Función objetivo: la longitud del ciclo resultante ha de ser mínima.

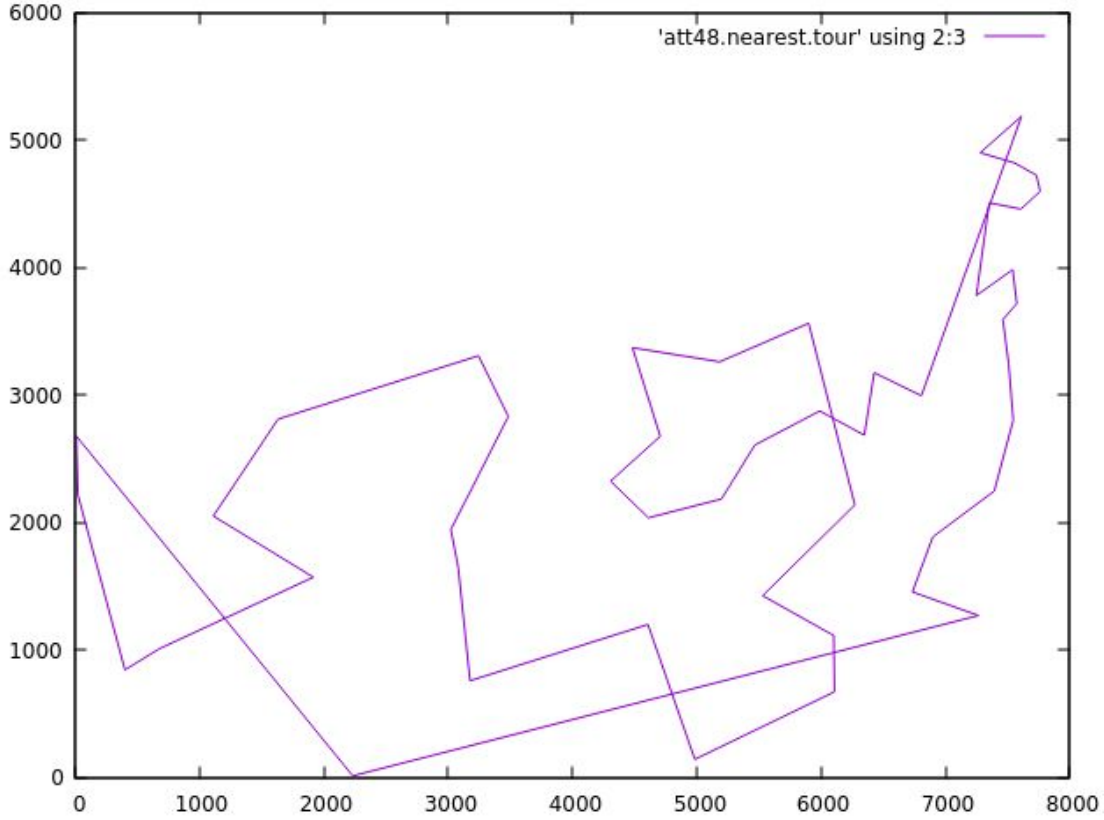
Pseudocódigo

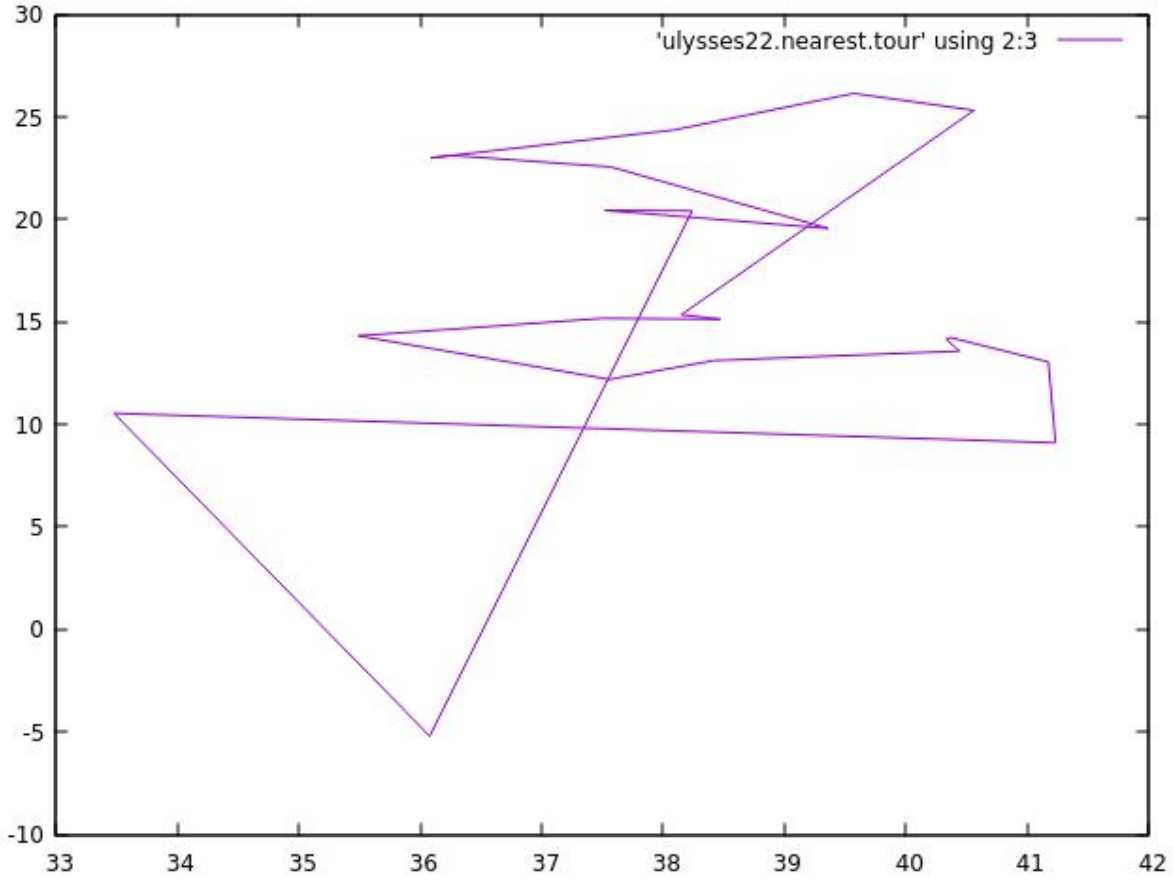
```
C= {0,1,2,...,N-1}
FUNCION TSP(C: CONJUNTO DE CIUDADES NO USADAS, L: MATRIZ DISTANCIAS, V0: CIUDAD INICIAL)
        C = C - (V0)
        P[0] = V0
        CIUDADACTUAL = V0

        REPETIR HASTA QUE C QUEDE VACIO
                PARA J=0 HASTA N-1 HACER:
                        SI D[CIUDADACTUAL][J] < MINIMO Y J ESTÁ EN C ENTONCES
                                MINIMO = D[CIUDADACTUAL][J]
                                PROXIMACIUDAD = J

                CIUDADACTUAL = PROXIMACIUDAD
                C = C - (CIUDADACTUAL)
                INTRODUCIR CIUDADACTUAL EN P

        INTRODUCIR P[0] AL FINAL DE P   <- Para formar el Ciclo Hamiltoniano

        DEVOLVER P
```

Donde P es un vector que contiene el ciclo hamiltoniano que devuelve el algoritmo.
Complejidad del algoritmo: **$O(n^2)$** debido a que hay un bucle que recorre las n columnas de una fila de la matriz de distancias anidado en otro que recorre todo C.

Resultados del algoritmo sobre los distintos conjuntos de ciudades:






'ulysses22.nearest.tour' using 2:3
30
25
20
15
10
5
0
-5
-10
33
34
35
36
37
38
39
40
41
42


'a280.nearest.tour' using 2:3
180
160
140
120
100
80
60
40
20
0
0
50
100
150
200
250
300

Estrategia basada en inserción económica

Elementos del algoritmo

1. Conjunto de candidatos: todas las ciudades por las que tiene que pasar el viajante de comercio y la matriz con las distancias entre todas ellas.
2. Candidatos ya usados: ciudades por las que ya ha pasado.
3. Función solución: obtener un Circuito Hamiltoniano Minimal.
4. Criterio de factibilidad: al seleccionar una arista no se debe de formar ciclo y además tiene que cumplir la condición de no ser incidente en tercera posición al nodo escogido.
5. Función de selección: dado un ciclo inicial, seleccionar una ciudad en una posición que menos incremente la longitud del circuito.
6. Función objetivo: la longitud del ciclo resultante ha de ser mínima.

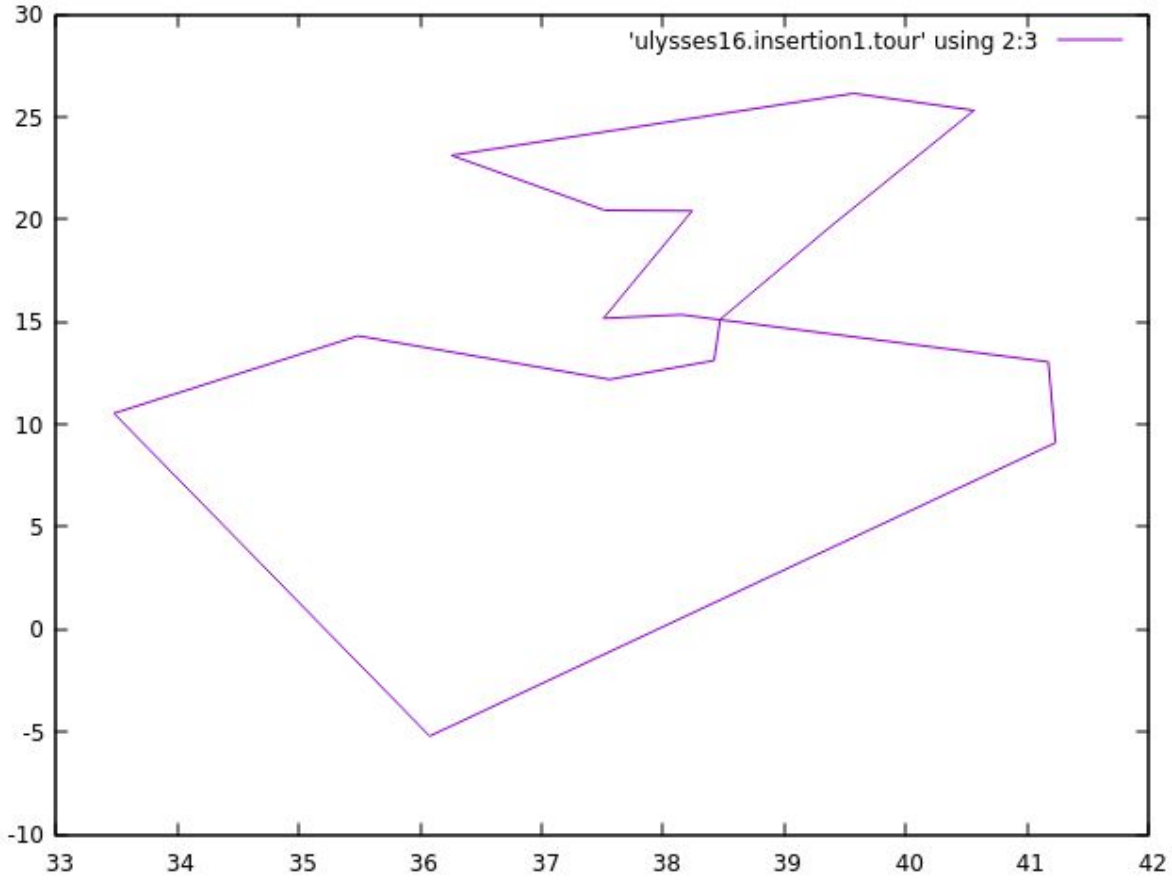
Pseudocódigo

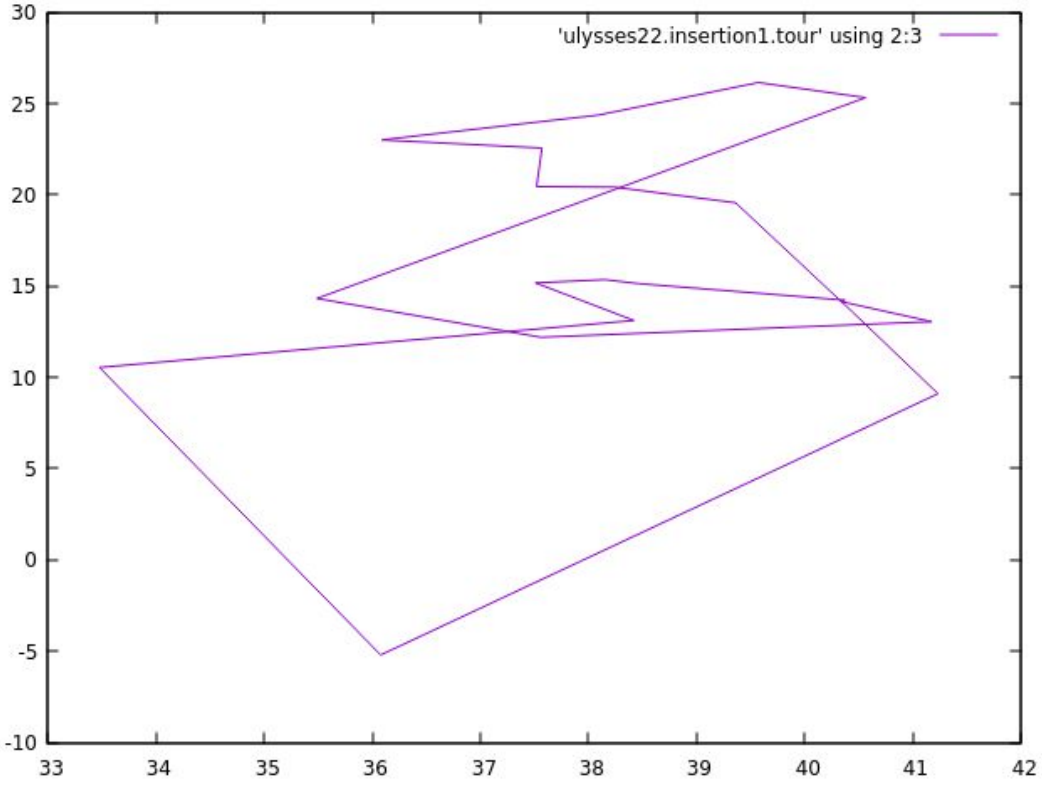
```
C = {0,1,2,3,...,N-1}
FUNCION TSP(C: CONJUNTO DE CIUDADES NO USADAS, L: MATRIZ DISTANCIAS)
        OESTE = CIUDAD CON MAYOR X
        ESTE = CIUDAD CON MENOR X
        NORTE = CIUDAD CON MAYOR Y
        INSERTAR OESTE EN P
        INSERTAR ESTE EN P
        INSERTAR NORTE EN P
        C = C - (NORTE) - (OESTE) - (ESTE)

        REPITE MIENTRAS C != Ø
                PARA I=0 HASTA N
                        CICLOMASCORTO = CalculaCicloMasCorto(P, L, C, I, DCICLO)

                        SI DCICLO < MINIMOHASTAHORA ENTONCES
                                MINIMOHASTAHORA = DCICLO
                                CICLIMIN = CICLOMASCORTO
                                POSCIUDADUSADA = I

                P = CICLOMIN
                C = C - (P[POSCIUDADUSADA])

        INSERTAR P[0] AL FINAL DE P       <- Para formar el Ciclo Hamiltoniano

        DEVOLVER P
```

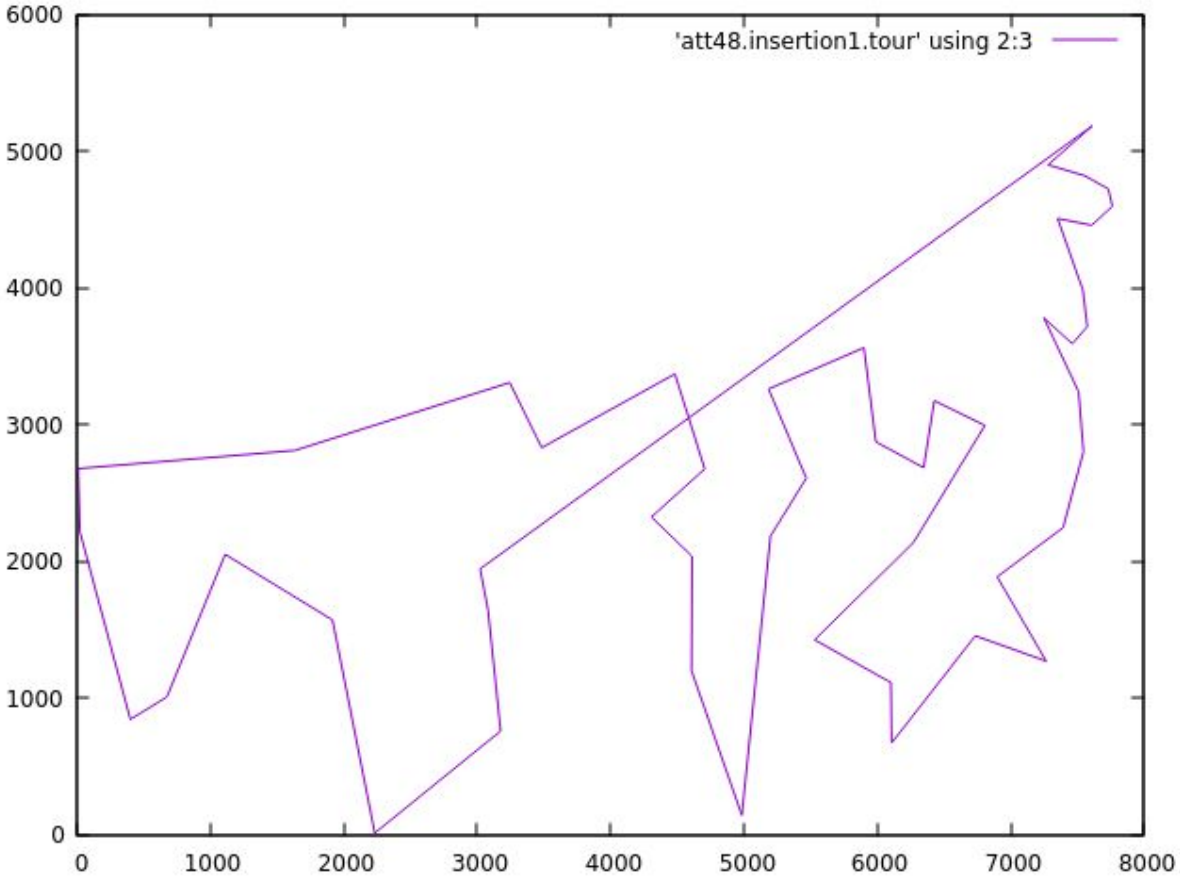
```
FUNCION CalculaCicloMasCorto(P: VECTOR QUE CONTIENE EL CICLO, L: MATRIZ DE DISTANCIAS, C: CIUDADES NO USADAS, I: POSICION, DCICLO&: DISTANCIA DEL CICLO RESULTANTE)
    PARA J=0 HASTA N-1
        INSERTA EN P[I] C[J] Y MUEVE LAS DEMÁS POSICIONES DEL VECTOR
        DISTANCIA = CalculaLongitudCiclo(P, L)

        SI DISTANCIA < MIN ENTONCES
            MIN = DISTANCIA
            CICLOMASCORTO = P

    DISTANCIA = MIN

    DEVOLVER CICLOMASCORTO


FUNCION CalculaLongitudCiclo(P: VECTOR QUE CONTIENE EL CICLO, L: MATRIZ DE DISTANCIAS)
    PARA I = 0 HASTA N-2
        DISTANCIA = DISTANCIA + L[P[I]][P[I+1]]

    DEVOLVER DISTANCIA
```

Complejidad del algoritmo: **$O(n^4)$** debido a que hay un bucle que se repite hasta que C está vacío y dentro de este otro que recorre todas las posiciones del vector, luego dentro de este hay otro bucle que recorre el conjunto de nodos no usados y dentro hay otro que recorre el camino parcial formado.

Resultados del algoritmo sobre los distintos conjuntos de ciudades


'ulysses16.insertion1.tour' using 2:3
30
25
20
15
10
5
0
-5
-10
33
34
35
36
37
38
39
40
41
42


'ulysses22.insertion1.tour' using 2:3
30
25
20
15
10
5
0
-5
-10
33
34
35
36
37
38
39
40
41
42


'att48.insertion1.tour' using 2:3
6000
5000
4000
3000
2000
1000
0
0
1000
2000
3000
4000
5000
6000
7000
8000


'a280.insertion1.tour' using 2:3
180
160
140
120
100
80
60
40
20
0
0
50
100
150
200
250
300

Estrategia basada en inserción por cercanía a cualquier nodo ya usado

Elementos del algoritmo

7. Conjunto de candidatos: todas las ciudades por las que tiene que pasar el viajante de comercio y la matriz con las distancias entre todas ellas.
8. Candidatos ya usados: ciudades por las que ya ha pasado.
9. Función solución: obtener un Circuito Hamiltoniano Minimal.
10. Criterio de factibilidad: al seleccionar una arista no se debe de formar ciclo y además tiene que cumplir la condición de no ser incidente en tercera posición al nodo escogido.
11. Función de selección: dado un ciclo inicial, seleccionar la ciudad más cercana a cualquiera que ya forma el ciclo.
12. Función objetivo: la longitud del ciclo resultante ha de ser mínima.

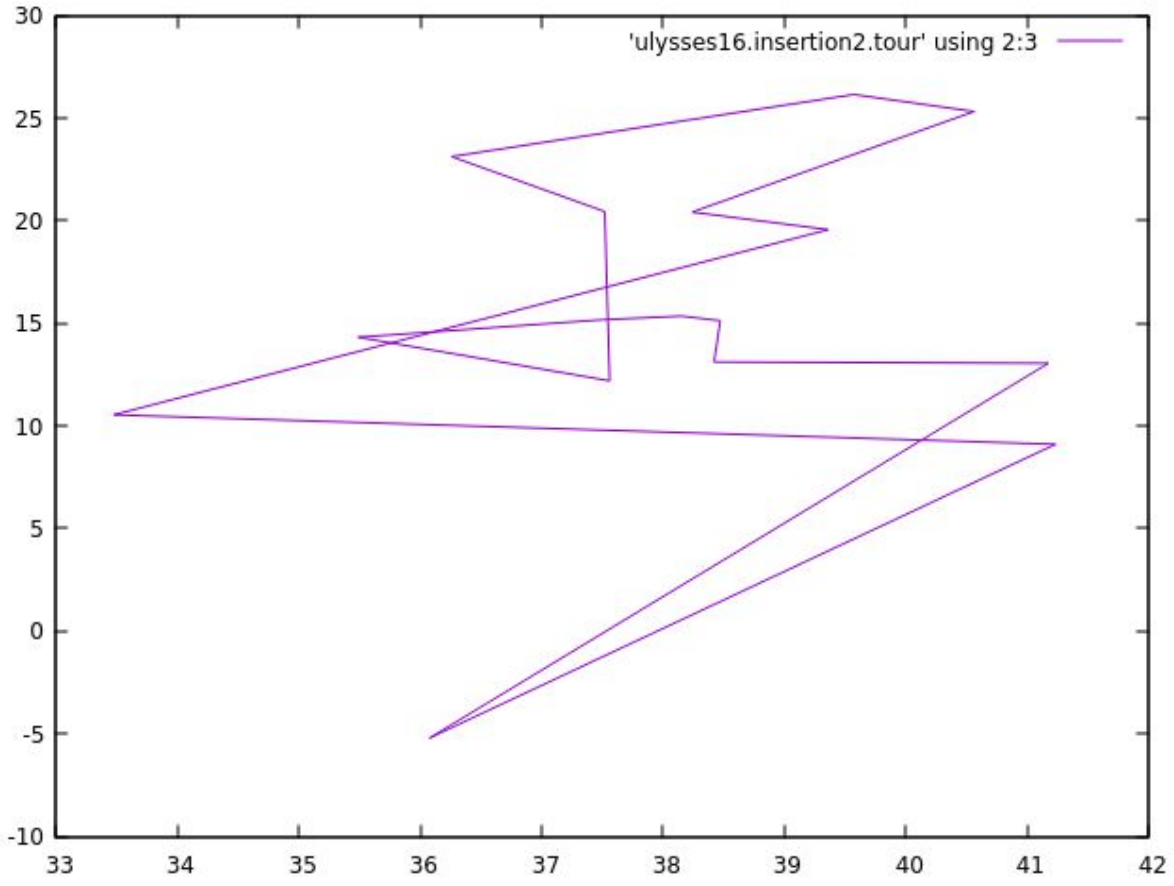
Pseudocódigo

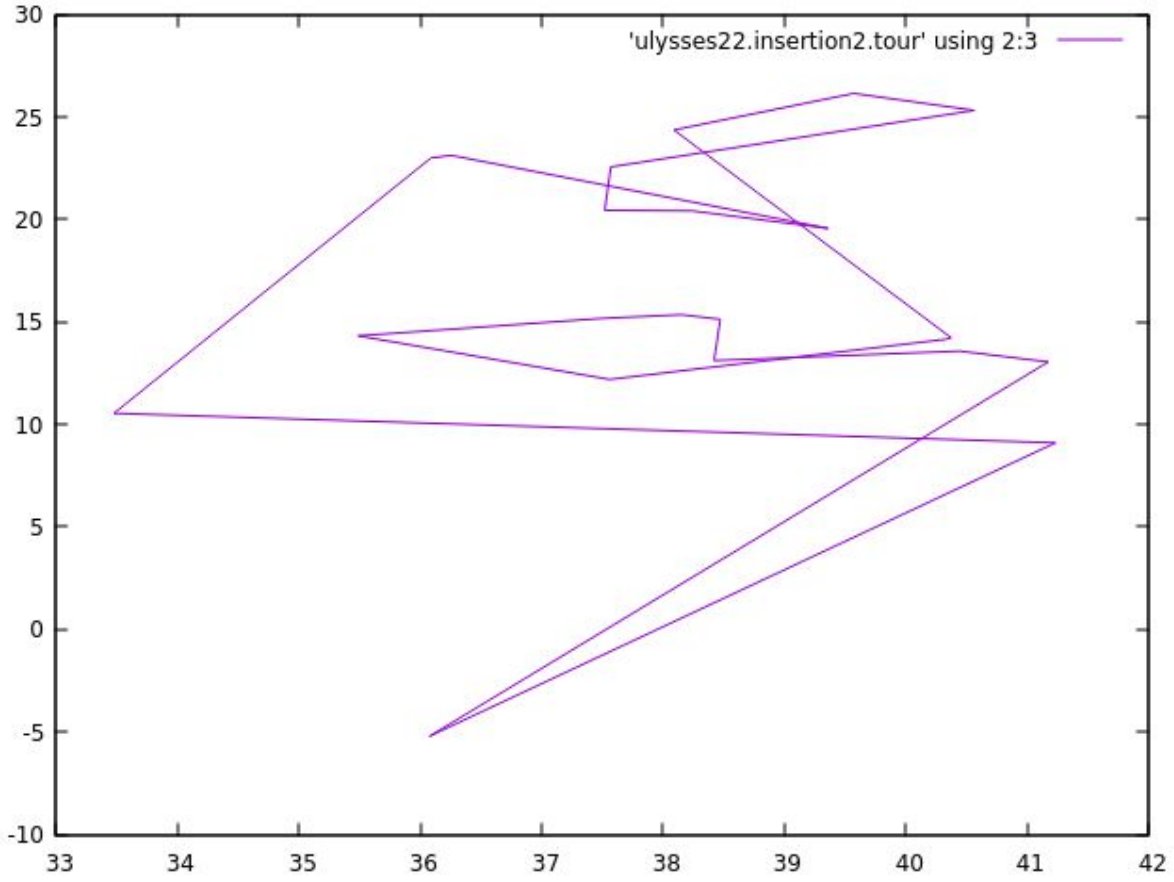
```
C = {0,1,2,3,...,N-1}
FUNCION TSP(C: CONJUNTO DE CIUDADES NO USADAS, L: MATRIZ DISTANCIAS)
        OESTE = CIUDAD CON MAYOR X
        ESTE = CIUDAD CON MENOR X
        NORTE = CIUDAD CON MAYOR Y
        INSERTAR OESTE EN P
        INSERTAR ESTE EN P
        INSERTAR NORTE EN P
        C = C - (NORTE) - (OESTE) - (ESTE)

        REPITE MIENTRAS C != Ø
                PARA I=0 HASTA N
                        CICLONUEVO = EncuentraVerticeMasCercano(P, L, C, I, DCICLO)

                        SI DCICLO < MINIMOHASTAHORA ENTONCES
                                MINIMOHASTAHORA = DCICLO
                                CICLIMIN = CICLOMASCORTO
                                POSCIUDADUSADA = I

                P = CICLOMIN
                C = C - (P[POSCIUDADUSADA])

        INSERTAR P[0] AL FINAL DE P       <- Para formar el Ciclo Hamiltoniano

        DEVOLVER P
```

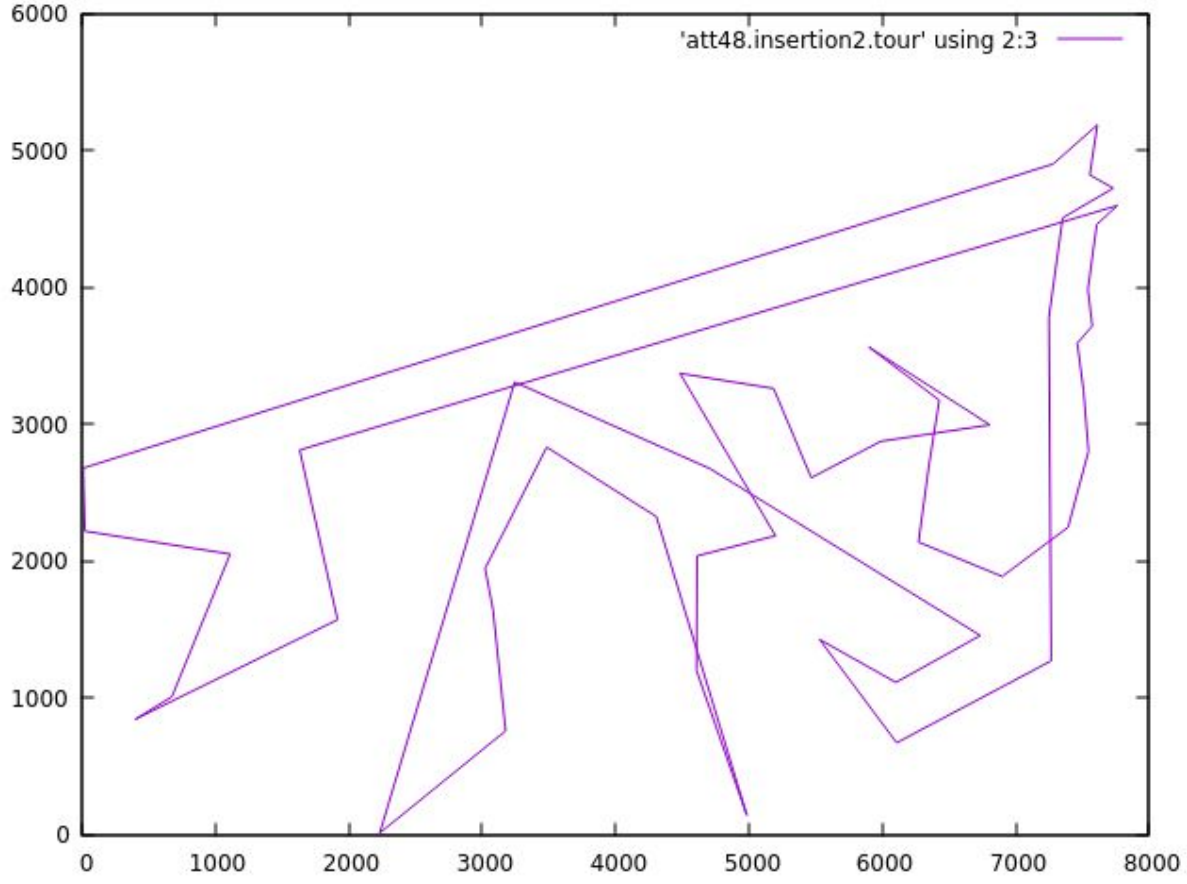
```
FUNCION EncuentraVerticeMasCercano(P: VECTOR QUE CONTIENE EL CICLO, L:
MATRIZ DE DISTANCIAS, C: CIUDADES NO USADAS, I: POSICION, DCICLO&:
DISTANCIA DEL CICLO RESULTANTE)
        PARA J=0 HASTA N-1
                INSERTA EN P[I] C[J] Y MUEVE LAS DEMÁS POSICIONES DEL VECTOR
                DISTANCIA = D[P[I-1]][P[I]]

                SI DISTANCIA < MIN ENTONCES
                        MIN = DISTANCIA
                        CICLONUEVO = P

        DISTANCIA = MIN

        DEVOLVER CICLONUEVO
```

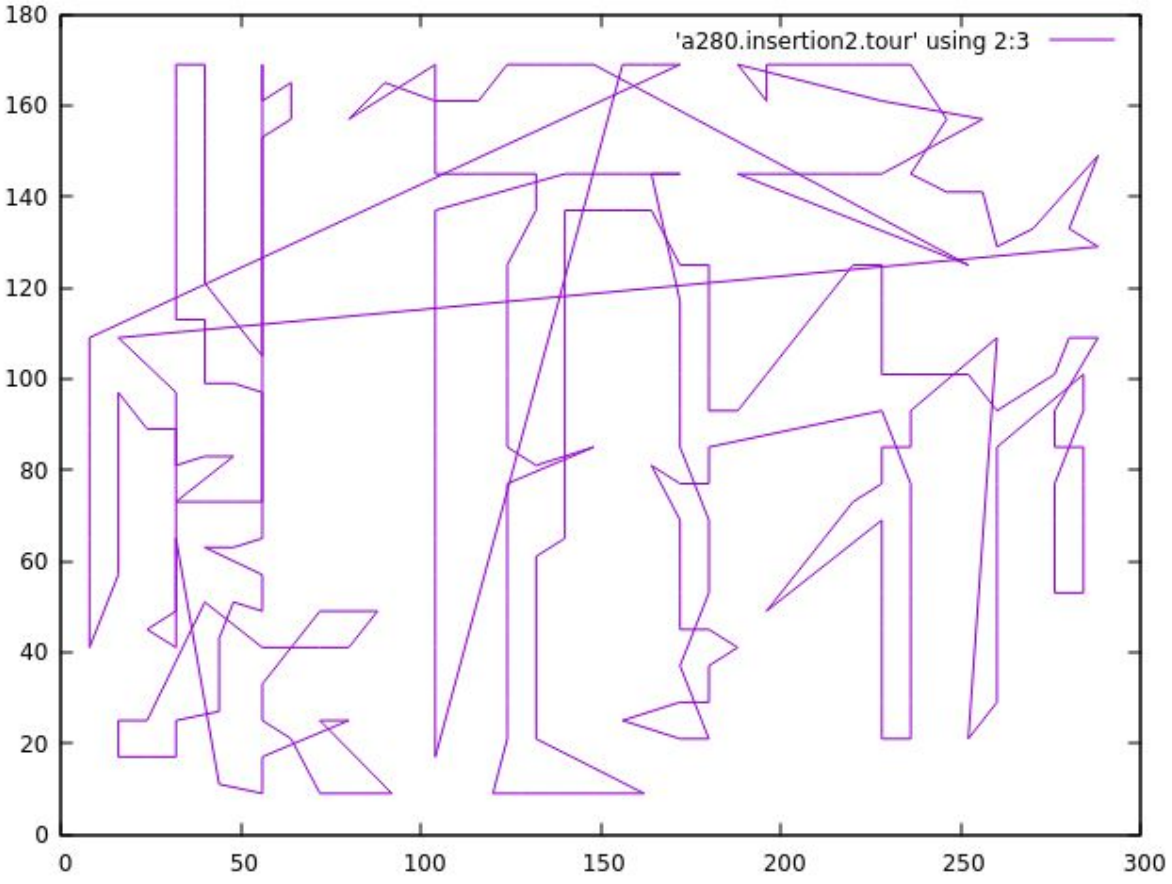
Complejidad del algoritmo: **O(n^3)** debido a que hay un bucle que se repite hasta que C está vacío y dentro de este otro que recorre todas las posiciones del vector, luego dentro de este hay otro bucle que recorre el conjunto de nodos no usados.

Resultados del algoritmo sobre los distintos conjuntos de ciudades




'ulysses22.insertion2.tour' using 2:3
30
25
20
15
10
5
0
-5
-10
33
34
35
36
37
38
39
40
41
42


'att48.insertion2.tour' using 2:3
6000
5000
4000
3000
2000
1000
0
0
1000
2000
3000
4000
5000
6000
7000
8000


'a280.insertion2.tour' using 2:3
180
160
140
120
100
80
60
40
20
0
0
50
100
150
200
250
300

Casos de ejecución

```
manueljnunez@ideapad:~/Asignaturas/ALG/Practicas/p3/codigo$ ./TSP ulysses16.tsp ulysses16.opt.
tour ulysses16.nearest.tour ulysses16.insertion1.tour ulysses16.insertion2.tour
Inserte la ciudad de inicio (número de 1 a 16): 1
Tiempo algoritmo cercanía: 74
Tiempo algoritmo inserción económica: 7601
Tiempo algoritmo inserción por cercanía: 341

Camino por cercanía: 1 8 16 13 12 14 15 6 7 10 9 5 4 2 3 11 1
Camino por inserción económica: 16 12 7 6 15 5 11 9 10 13 14 1 8 4 2 3 16
Camino por inserción por cercanía: 5 9 11 10 7 12 13 14 15 6 8 4 2 3 1 16 5

Coste del camino calculado por cercanía: 104.735
Coste del camino calculado por inserción económica: 75.2781
Coste del camino calculado por inserción por cercanía: 87.6251
Coste del camino optimo: 74.1087
manueljnunez@ideapad:~/Asignaturas/ALG/Practicas/p3/codigo$ ./TSP ulysses22.tsp ulysses22.opt.
tour ulysses22.nearest.tour ulysses22.insertion1.tour ulysses22.insertion2.tour
Inserte la ciudad de inicio (número de 1 a 22): 1
Tiempo algoritmo cercanía: 73
Tiempo algoritmo inserción económica: 16346
Tiempo algoritmo inserción por cercanía: 712

Camino por cercanía: 1 8 16 22 4 18 17 2 3 13 12 14 15 6 7 19 20 21 10 9 5 11 1
Camino por inserción económica: 15 6 10 20 21 12 13 14 7 5 11 9 19 16 1 8 22 18 4 17 2 3 15
Camino por inserción por cercanía: 5 9 11 10 19 7 12 13 14 15 6 20 21 17 2 3 22 8 1 16 4 18 5

Coste del camino calculado por cercanía: 89.6408
Coste del camino calculado por inserción económica: 84.8706
Coste del camino calculado por inserción por cercanía: 96.7126
Coste del camino optimo: 75.6651
manueljnunez@ideapad:~/Asignaturas/ALG/Practicas/p3/codigo$ ./TSP att48.tsp att48.opt.tour att
48.nearest.tour att48.insertion1.tour att48.insertion2.tour
Inserte la ciudad de inicio (número de 1 a 48): 1
Tiempo algoritmo cercanía: 187
Tiempo algoritmo inserción económica: 211131
Tiempo algoritmo inserción por cercanía: 4368

Camino por cercanía: 1 9 38 31 44 18 7 28 36 30 6 37 19 27 43 17 46 33 15 12 11 23 14 25 13 21
 47 20 40 3 22 16 41 34 29 5 48 39 32 24 10 42 26 4 35 45 2 8 1
Camino por inserción económica: 48 5 29 2 42 10 26 4 35 45 24 32 39 21 13 25 14 34 41 23 11 47
 20 12 15 33 46 40 3 22 16 1 8 9 38 31 44 36 18 7 28 30 6 37 19 27 43 17 48
Camino por inserción por cercanía: 45 35 10 26 4 42 24 37 6 28 7 18 44 31 38 9 40 15 33 20 46
12 11 47 21 23 14 34 41 25 39 48 5 29 2 32 13 1 22 3 16 8 36 30 19 27 17 43 45

Coste del camino calculado por cercanía: 40526.4
Coste del camino calculado por inserción económica: 39009.5
Coste del camino calculado por inserción por cercanía: 55662.5
Coste del camino optimo: 33523.7
```

```
manueljnunez@ideapad:~/Asignaturas/ALG/Practicas/p3/codigo$ ./TSP a280.tsp a280.opt.tour a280.
nearest.tour a280.insertion1.tour a280.insertion2.tour
Inserte la ciudad de inicio (número de 1 a 280): 1
Tiempo algoritmo cercanía: 3453
Tiempo algoritmo inserción económica: 301291026
Tiempo algoritmo inserción por cercanía: 653725

Camino por cercanía: 1 280 2 3 279 278 4 277 276 275 274 273 272 271 16 17 18 19 20 21 128 127
 126 125 30 31 32 29 28 27 26 22 25 23 24 14 13 12 11 10 8 7 9 6 5 260 259 258 257 254 253 208
 207 210 209 252 255 256 249 248 247 244 241 240 239 238 231 232 233 234 235 236 237 246 245 2
50 251 230 229 228 227 226 225 224 223 222 219 218 215 214 211 212 213 216 217 220 221 203 202
 200 144 143 142 141 140 139 138 137 136 135 134 270 269 268 267 266 265 264 263 262 261 15 13
3 132 131 130 129 154 155 153 156 152 151 177 176 181 180 179 178 150 149 148 147 146 145 199
198 197 194 195 196 201 193 192 191 190 189 188 187 185 184 183 182 161 162 163 164 165 166 16
7 168 169 101 100 99 98 93 94 95 96 97 92 91 90 89 109 108 104 103 102 170 171 172 173 106 105
 107 174 175 160 159 158 157 119 120 121 122 123 124 34 33 36 35 38 37 39 40 41 42 43 60 61 11
8 117 115 114 111 110 112 88 83 82 81 80 79 76 75 74 73 72 71 70 67 66 65 64 63 62 116 86 85 8
4 87 113 59 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 68 69 77 78 186 204 205 206 243 242 1

Camino por inserción económica: 258 257 256 251 250 249 248 247 246 245 244 243 242 241 240 23
9 238 237 236 235 234 233 232 231 230 229 228 227 226 225 224 223 222 221 220 219 218 217 216
215 214 213 212 211 210 209 252 255 254 253 208 207 206 205 204 203 202 201 198 197 196 195 19
4 193 192 191 190 189 188 187 186 185 184 183 174 173 172 171 170 168 167 166 165 164 163 162
161 175 160 159 176 181 182 180 179 149 148 147 146 145 199 200 144 143 142 141 140 139 133 17
 18 19 20 132 131 21 130 129 128 127 126 30 125 124 120 119 157 158 117 116 115 114 113 112 11
1 110 109 108 107 106 105 104 103 102 169 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85
 84 83 82 81 80 79 78 77 76 75 74 73 72 71 57 56 55 45 46 54 53 47 48 52 51 49 50 38 37 36 34
35 39 40 41 42 43 61 118 62 63 64 65 66 67 70 69 68 58 59 44 60 121 122 123 154 155 153 156 15
2 151 177 178 150 138 137 136 135 134 270 269 268 267 266 265 264 263 262 261 260 259 278 279
1 2 280 3 4 277 5 6 276 275 274 273 272 271 16 15 9 7 8 10 11 12 13 14 24 23 25 22 26 27 28 29
 31 32 33 258
Camino por inserción por cercanía: 69 70 71 72 73 74 75 77 78 76 67 66 65 85 86 116 84 87 113
114 115 117 118 62 63 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 61 60 43 42 41 40 39 35
36 38 37 111 112 88 110 108 109 90 91 92 93 98 97 96 95 94 89 104 105 106 173 174 107 103 102
169 168 167 166 170 171 172 101 100 99 83 82 81 79 80 64 68 2 280 1 3 279 4 277 276 6 7 8 10 1
1 12 13 15 14 9 5 275 274 273 272 271 16 278 22 26 27 28 29 31 33 34 32 30 125 126 127 128 129
 154 155 153 156 152 151 178 150 177 176 181 182 183 184 185 187 188 189 190 191 192 186 180 1
79 130 131 132 133 134 270 269 268 267 266 265 264 263 262 261 260 259 258 257 256 249 250 247
 244 243 242 245 246 239 238 237 236 235 234 233 232 231 240 241 251 230 229 228 227 226 225 2
24 223 248 255 252 253 208 206 205 204 203 207 212 213 216 217 220 221 222 219 218 215 214 211
 210 209 254 140 141 148 149 147 146 145 199 200 202 201 196 197 193 194 195 198 144 143 142 1
39 138 137 136 135 18 17 19 20 21 124 123 122 121 120 119 157 158 159 160 175 161 162 163 164
165 25 23 24 69

Coste del camino calculado por cercanía: 3148.11
Coste del camino calculado por inserción económica: 3113.58
Coste del camino calculado por inserción por cercanía: 4130.67
Coste del camino optimo: 2586.77
```

Comparativa de los algoritmos

Comparación eficiencia teórica

Teóricamente, el algoritmos más eficiente de estos tres es el de cercanía que es $O(n^2)$, el segundo más eficiente es el de inserción por cercanía a cualquier nodo ya usado que es $O(n^3)$ y el peor en cuanto a eficiencia es el de inserción económica siendo este $O(n^4)$.

Comparación tiempo de ejecución

	Cercanía	Inserción económica	Inserción por cercanía a cualquier nodo ya usado
Ulysses16	74µs	7601µs	341µs
Ulysses22	73µs	16346µs	712µs
att48	187µs	211131µs	4368µs
a280	3453µs	301291026µs	653725µs

En cuanto a tiempo de ejecución, el mejor es el de cercanía, el segundo mejor el de inserción por cercanía a cualquier nodo ya usado y el peor el de la inserción económica.

Comparación de eficiencia del resultado

	Cercanía	Inserción económica	Inserción por cercanía a cualquier nodo ya usado	Óptimo
Ulysses16	104.735	75.2781	87.6251	74.1087
Ulysses22	89.6408	84.8706	96.7126	75.6651
att48	40526.4	39009.5	55662.5	33523.7
a280	3148.11	3113.58	4130.67	2586.77

Como podemos ver en la tabla, el que consigue mejor resultado es el de inserción económica, aunque sea el más ineficiente en tiempo.

Problema Asignado: Contenedores en un barco

Algoritmo que maximiza el número de contenedores.

Hipótesis

El algoritmo más óptimo de maximización del número de contenedores es el siguiente:

1) Introduce en el barco el contenedor con menor peso de aquellos que quedan por introducir.
2) Si se ha sobrepasado la capacidad del barco, retirar el último contenedor y solución encontrada, si no se ha sobrepasado volver al paso 1.

Este algoritmo es greedy, ya que elige el elemento más óptimo de un conjunto de elementos disponibles en todo momento.

Definiciones

Sean:

- $K \in N$ carga máxima del barco
- C_i conjunto de contenedores en la i-ésima iteración del algoritmo
- P_i conjunto de pesos de los contenedores en la i-ésima iteración
- $p(c) : C \to N$ función que devuelve el peso de un contenedor c
- $min(C) \in C$ contenedor con peso mínimo en C
- Ω conjunto de todos los posibles conjuntos de contenedores
- $N(C,K) : \Omega \times N \to N$ número máximo de contenedores de C que caben en un barco de capacidad K
- Un subconjunto del conjunto inicial de contenedores es maximal en K si no existe ningún otro contenedor del conjunto inicial que quepa en K
- Un subconjunto del conjunto inicial de contenedores es EL conjunto maximal en K si no existe ningún otro conjunto maximal cuya cardinalidad sea mayor

Sabemos:

$$N(C,\ K_1) \leq N(C,\ K_2) \Leftrightarrow K_1 \leq K_2$$

Es decir, si el espacio disminuye, es imposible que la cardinalidad del conjunto maximal aumente.

Demostración

Antes de demostrar, definimos el problema de forma más simple. Ya que todas las inserciones de contenedores se realizan de la misma manera, basta con demostrar que el contenedor de menor peso siempre estará en el subconjunto maximal. Una vez insertado este contenedor, redefinimos el problema con K como el espacio restante y C como el conjunto inicial menos el contenedor insertado. Así, siempre insertamos el contenedor de menor peso del conjunto C.

Demostramos por contradicción.
Sean $K \in N; C, C_1, C_2 \in \Omega$ tal que $C_1, C_2 \subseteq C$ subconjuntos maximales en K, $c_1 \in C_1$, $c_1 \notin C_2$ y $c_1 = min(C)$. Sea $c_x \in C$ tal que $c_x \in C_2$ y $c_x \notin C_1$.
Suponemos que C_2 es el subconjunto maximal.
Hacemos dos experimentos mentales: en uno insertamos el primer elemento de C_1, el cual sigue el algoritmo siendo $min(C)$; en otro insertaremos un elemento de C_2, es arbitrario cuál, ya que el orden de inserción no afecta al resultado y en este segundo experimento no estamos siguiendo el algoritmo propuesto, insertaremos c_x. Ya que la cardinalidad del segundo conjunto es mayor que la del segundo, esperamos que estudiando la función N tras esta primera inserción, esta se mantenga de la siguiente manera:

$$N(C_1 \setminus \{c_1\}, K - p(c_1)) \leq N(C_2 \setminus \{c_x\}), K - p(c_x))$$

No obstante, por definición, ya que c_1 no está en C_2 ni c_x en C_1, sabemos que:

$$N(C_1 \setminus \{c_1\}, K - p(c_1)) = N(C_1 \setminus \{c_1, c_x\}, K - p(c_1))$$

y

$$N(C_2 \setminus \{c_x\}), K - p(c_1)) = N(C_2 \setminus \{c_x, c_1\}), K - p(c_x))$$

Así, estamos listos para encontrar la contradicción:

- Sabemos que $N(C, K_1) \leq N(C, K_2) \Leftrightarrow K_1 \leq K_2$
- Sabemos que $p(c_1) \leq p(c_x)$ ya que $c_1 = min(C)$
- Así, $K - p(c_1) \geq K - p(c_x)$
- Así, por la primera afirmación de esta lista:

$$N(C_1 \setminus \{c_1, c_x\}, K - p(c_1)) \geq N(C_2 \setminus \{c_x, c_1\}), K - p(c_x))$$

- Lo cual equivale a:

$$N(C_1 \setminus \{c_1\}, K - p(c_1)) \geq N(C_2 \setminus \{c_x\}), K - p(c_x))$$

Y así, llegamos a nuestra contradicción, de lo que obtenemos:

$$\neg \forall c \in C, c \neq min(C) \; tq \; c \in el\ conjunto\ maximal\ y\ min(C) \notin el\ conjunto\ maximal$$

Lo que es equivalente a:

$$min(C)\ estará\ en\ el\ subconjunto\ maximal\ de\ C\ para\ cualquier\ K \geq p(min(C))$$

qed

Complejidad

Ya que se ha escogido una búsqueda lineal para encontrar el contenedor mínimo, la complejidad en el caso peor (en el que introducimos todos los contenedores) será:

$$\sum_{i=1}^{n} n - i = n^2 - \frac{n(n+1)}{2} = \frac{n^2}{2} - \frac{1}{2} = O(n^2)$$

En el caso mejor (en el que no cabe ningún contenedor) únicamente se realizará una búsqueda, por lo que la complejidad en el caso mejor será:

$$\Omega(n)$$

Siendo n en ambos casos el número de contenedores.
Si se implementa un algoritmo de generación de contenedores ordenados, podríamos utilizar una búsqueda binaria y reducir los tiempos a:

$$\sum_{i=1}^{n} log(n - i) = O(n\ log(n))$$

$$\Omega(log(n))$$

Pseudocódigo

```
FUNCION MAXIMIZAR_CONTENEDORES

    k = capacidad máxima
    contenedores = {c1, c2, ..., cn}
    resultado = {}
    peso_cargado = 0
    min = 0

    REPETIR HASTA peso_cargado > k ó contenedores.vacio()
            min=contenedor_mas_pequeño(contenedores)
            resultado.insertar(min)
            contenedores.borrar(min)
            peso_cargado += peso(min)

    IF peso_cargado > k
            resultado.borrar(min)

    DEVOLVER resultado
```

Algoritmo que maximiza el peso de los contenedores cargados

Mientras que el anterior algoritmo se basaba en la elección del contenedor más pequeño, en este haremos lo contrario. Para maximizar la carga seleccionaremos en cada iteración del algoritmo el contenedor que más peso aporte, es decir, el contenedor de mayor peso de aquellos disponibles que cabe en el espacio libre. El algoritmo queda de la siguiente manera:

1) Introduce en el barco el contenedor más grande que quepa en el barco. Si este contenedor existe, repetir paso 1. Si no, solución encontrada.

La complejidad será la misma que para el anterior algoritmo.

Pseudocódigo

```
FUNCION MAXIMIZAR_CARGA

    k = capacidad máxima
    contenedores = {c1, c2, ..., cn}
    resultado = {}
    peso_cargado = 0
    min = 0

    REPETIR HASTA peso_cargado > k ó contenedores.vacio()
            min=contenedor_mas_pesado(contenedores)
            resultado.insertar(min)
            contenedores.borrar(min)
            peso_cargado += peso(min)

    IF peso_cargado > k
            resultado.borrar(min)

    DEVOLVER resultado
```

Caso de ejecución

```
miguel@miguel-UX305FA:~/Escritorio/ALG/practica3/programas$ ./contenedores
K = 1000
Contenedores:
[0 , 51]
[1 , 37]
[2 , 73]
[3 , 189]
[4 , 45]
[5 , 67]
[6 , 47]
[7 , 69]
[8 , 98]
[9 , 50]
[10 , 8]
[11 , 80]
[12 , 87]
[13 , 58]
[14 , 179]
[15 , 98]
[16 , 28]
[17 , 169]
[18 , 191]
[19 , 1]
[20 , 182]
[21 , 185]
[22 , 122]


MAXIMIZAMOS NUMERO DE CONTENEDORES
Solución: carga 897
[0 , 51]
[1 , 37]
[2 , 73]
[4 , 45]
[5 , 67]
[6 , 47]
[7 , 69]
[8 , 98]
[9 , 50]
[10 , 8]
[11 , 80]
[12 , 87]
[13 , 58]
[15 , 98]
[16 , 28]
[19 , 1]


MAXIMIZAMOS CARGA
Solución: carga 926
[3 , 189]
[14 , 179]
[18 , 191]
[20 , 182]
[21 , 185]
```