Algorítmica

Entrega 3:

El problema de asignación cuadrática (QAP)

Javier Martín Gómez
Alfonso Soto López
Julián Torices Hernández
Antonio David Mota Martínez
Alberto Peinado Santana
Enrique Moreno Carmona

24 de mayo de 2017

Índice

Lista de Algoritmos

Resumen

En esta memoria se recoge la implementación y la ejecución de un algoritmo **greedy** para la resolución del problema de asignación cuadrática (QAP).

Se definirá el problema, se discutirá la implementación y se realizara un estudio teórico sobre la eficiencia.

1. Análisis del problema

2. Diseño de la solución

Diseño de la solución, describiendo cada una de las componentes de la misma relacionándola con las componentes de un algoritmo greedy.

La solución que hemos diseñado se resume en crear vectores de potenciales para la matriz de flujos y la de distancias y a partir de ahí aplicar greedy.

- Conjunto de elementos candidatos pendientes (C): El vector de **potenciales de distancias** (vector<float>pd), el vector de **potenciales de flujos** (vector<float>pf)
- Candidatos seleccionados para la solución (S): El **vector solución** donde el índice del mínimo actual del vector de distancias se empareja con el índice del máximo actual del vector de flujos

Se parte del conjunto C y se itera n veces donde n es el tamaño del problema.

En cada iteración se busca el máximo en el vector pf y el mínimo en el vector pd.

Una vez encontrados se meten los índices en el vector solución $S(solucion[indexF] = indexD)$ y se descartan del conjunto de candidatos $C\ (pf[indexF]* = -1; pd[indexD]* = -1)$

Y cuando el bucle ha acabado se calcula el coste total:

Algoritmo 1: Código del calculo de la solución

```

   ...

   float resultado = 0;
   for(int i = 0; i < tam; i++)
     for(int j = 0; j < tam; j++)
       resultado += flujo[i][j] * dist[solucion[i]][solucion[j]];

   ...
```

3. Implementación

Algoritmo 2: Pseudo-Codigo de un algoritmo greedy para la resolución del problema de QAP

```
greedyQAP(distancias, flujos, solucion){

    Calcular los potenciales f_i y d_k

    S <- []

    for( x=1 hasta n ){

        Escoger la unidad u_i no asignada con mayor valor de f_i

        Escoger la unidad l_k no asignada con menor valor de f_k

        solucion[u_i] = l_k

    }

    Calcular coste de solucion[]

    Devolver solucion y coste.

}
```

4. Explicación del funcionamiento del algoritmo

5. Uso en situaciones reales

Caso de ejemplo, habitaciones de un hospital.

El algoritmo greedy que resuelve el problema del QAP tiene muchas aplicaciones en la vida real, por ejemplo en un hospital, la asignación de pacientes a una habitación o otra dependiendo del flujo y la distancia de manera que los médicos recorran menos distancia y eviten problemas de logística. Por ejemplo los pacientes que entren y necesiten más cuidado, es decir tienen mayor numero de visitas, deben estar más cerca del medico para que así recorra menos espacio y tarde menos tiempo en llegar. De esta forma el medico tardara menos en llegar porque estarán más cerca y así tendrá más tiempo para curar a los demás pacientes.

El algoritmo se aplicaría siendo la gravedad del paciente el flujo puesto que si esta más enfermo necesitara más cuidados, y la distancia de las habitaciones será la distancia de las habitaciones de los pacientes.

6. Calculo teórico de la eficiencia

- La asignación de variables y llamada a métodos son operaciones elementales de eficiencia O(1)
- El bucle for de la línea 22 es un bucle homogéneo de eficiencia O(n)
- Los dobles bucles de las líneas: (39 a 47), (56 a 89), (95 a 97), (117 a 124) y (127 a 135), son bucles homogéneos que al ser dobles bucles, tienen eficiencia $O(n^2)$

Bucle 39-47 $\sum_{i=0}^{n}\sum_{j=0}^{n} c = cn^2 \Longrightarrow O(n^2)$

Bucle 56-89 $\sum_{k=0}^{n}\left(\sum_{i=0}^{n} c_1 + \sum_{j=0}^{n} c_2\right) = n\left(c_1 n + c_2 n\right) = n^2\left(c_1 + c_2\right) \Longrightarrow O(n^2)$

Bucle 95-97 $\sum_{i=0}^{n}\sum_{j=0}^{n} c_3 = c_3 n^2 \Longrightarrow O(n^2)$

- Aplicando la regla de la suma, la eficiencia sería el $\max\left(n^2, n, 1\right)$ que da como resultado que este código tiene de eficiencia $O(n^2)$

7. Instrucciones de compilación y ejecución

```
1  soto@soto-PC $ make
2          g++ -std=c++11 main.cpp -o main
3
4
5
6  soto@soto-PC $ ./main
7
8          Coste total: 4155
9
10         Oficinista 1 en Habitacion 5
11         Oficinista 2 en Habitacion 3
12         Oficinista 3 en Habitacion 2
13         Oficinista 4 en Habitacion 1
14         Oficinista 5 en Habitacion 4
```

Algoritmo 3: Ejemplo de compilación y ejecución del algoritmo