Análisis y diseño de software

dit
UPM

ejercicio 2

José A. Mañas

2.3.2018

ETSIT UPM

ejercicio 2

- algoritmos sobre grafos
  - BFS: camino mínimo (saltos)
  - Dijkstra: camino mínimo (distancias)
  - Edmonds-Karp: flujo máximo
- medir tiempos y validar complejidad

algoritmos

En un grupo de entrega
formado por 1, 2 o 3 alumnos,
se suma el último dígito del DNI de cada alumno

- si es par: algoritmo BFS
- si es impar: algoritmo Dijkstra
- opcionalmente se puede elegir Edmonds-Karp

Tareas:

1. medir tiempos
2. validar complejidad

preliminar

- los algoritmos usan profusamente los métodos
  - Node getNode(String name)
  - List<Link> getLinks(Node node)
- por lo que estos métodos deben ser razonablemente rápidos para que no contaminen la evaluación del algoritmo
- sugerencia: use diccionarios internamente
  - private Map<String, Node> nodeMap
  - private Map<Node, List<Link>> linkMap
  - de forma que los métodos citados sean O(1)
  - estos diccionarios se van cargando con addNode() y addLink()

medidas

- BFS
  - se espera complejidad lineal
  - sugerencia

```
for (int n = 1_000; n < 100_000; n+= 5_000) {
      Graph graph = new Graph();
      load(graph, n);
      long t = doit(graph);
      System.out.printf("%s %d%n", n, t);
   }
```

```
private static long doit(Graph graph) {
      BFS bfs = new BFS(graph);
      long t0 = System.currentTimeMillis();
      bfs.search(…);
      ….
      long t2 = System.currentTimeMillis();
      return t2 - t0;
   }
```

  - en doit() haga varias medidas para promediar
    - del nodo “0” al “1”, al “2”, …, al “5”

medidas

- Dijkstra
  - se espera complejidad cuadrática
  - sugerencia

```
for (int n = 1_000; n < 40_000; n += 2_000) {
        Graph graph = new Graph();
        load(graph, n);
        long t = doit(graph);
        System.out.printf("%s %d%n", n, t);
    }
```

```
private static long doit(Graph graph) {
        long t0 = System.currentTimeMillis();
        new Dijkstra(…);
        long t2 = System.currentTimeMillis();
        return t2 - t0;
    }
```

load()

- para generar un grafo del tamaño deseado
  1. genere N nodos
     - llámelos por su número de forma que siempre se puedan seleccionar nodos como
       - Node nodo0 = graph.getNode(“27”);
  2. para cada nodo, genere un número fijo, X, de enlaces a nodos elegidos aleatoriamente
     - por ejemplo, X = 5
  - esto nos deja un grafo donde probablemente todos los nodos esten enlazados

opcional

- si vamos a medir desde un nodo A, calcule cuantos nodos son alcanzables desde A

  creamos un conjunto vacío S

  creamos una cola de nodos, inicializada con A

  mientras la cola no está vacía

    se extre un nodo B de la cola

    si B ya está en S, continuamos

    se añade B a S

    para cada enlace desde B

      se añade a la cola el destino del enlace

- S contiene todos los nodos alcanzables, que usaremos como tamaño efectivo del grafo para las medidas

entrega

- package es.upm.dit.adsw.ej2
  - BFSMeter.java
  - DijkstraMeter.java
- fichero (word, pdf, … o similar)
  - medidas obtenidas
  - razonamiento de complejidad
  - capturas de correlator
    - coeficientes
    - gráfico seleccionado como representativo de la complejidad