E.T.S.I. Telecomunicación
Departamento de Ingeniería Electrónica

Herramientas para el diseño electrónico

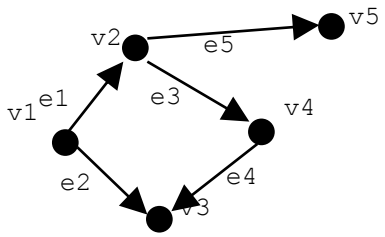
Curso 2010-11

Ejercicio nº 1 **Fecha de entrega: Viernes 25 Febrero 2011**

A. Grafos

En este primer ejercicio, se va a poner en práctica algunos de los conceptos sobre grafos que se han visto durante la primera clase.

Dado el siguiente grafo:



Conteste a las siguientes cuestiones:

- ¿Es un grafo conectado?
- ¿Es un grafo completo?
- ¿Es un grafo ponderado?
- ¿Es un grafo dirigido?
- ¿Existe algún clique que contenga más de dos nodos?

Represente la lista de adyacencia y la matriz de adyacencia del grafo.

Diga el orden de nodos que seguirían los algoritmos Depth-First Search y Breadth-First Search si partieran del vértice “v1”.

B. Simulated Annealing

Con este ejercicio se pretende “hacer manos” sobre la utilización del lenguaje C y los conceptos básicos involucrados en el algoritmo de optimización *Simulated Annealing*.

Considere la siguiente aplicación del algoritmo *Simulated Annealing*. Tenemos un problema de *"partitioning"* con N objetos (N par), y queremos N/2 objetos en cada lado de la partición. Los módulos están conectados por hilos y vamos a utilizar un método simple para medir las conexiones a través de la partición: una matriz de conectividad C_{ij} modela los costes de cada conexión módulo a módulo. Nuestro objetivo es colocar los módulos en el lado apropiado a fin de minimizar las conexiones entre ambos lados.

A continuación aparece una realización del algoritmo *Simulated Annealing* para este problema en forma de pseudo-código C. Por desgracia, parece que no funciona muy bien. El trabajo a realizar es la depuración de este código: señalad lo que está equivocado, decid por qué lo está y cómo solucionarlo.

Los errores pueden estar en cualquier parte, pero ignorad variables sin declarar, puntos y coma que falten, etc. Algunas funciones no se muestran detalladamente. Se debe suponer que funcionan como allí se explica.

```
/* definición de algunas constantes útiles */


#define  HOT       1000.0     /* temperatura inicial */
#define  N         100        /* número de módulos */
#define  ATTEMPTS  100        /* número de movimientos por
                                 módulo permitidos a cada
                                 temperatura */
#define  COOLING   0.9        /* velocidad de enfriamiento */


/* algunas variables globales */

float T;                      /* la temperatura */
int   left[N/2], right[N/2];  /* left[i] = j significa que el
                                 módulo j está en la partición
                                 izquierda ahora */

/* algunas funciones útiles */

eval_cost_actual() {
     /* devuelve el coste total (número de conexiones
        a través de la partición) para la partición actual */
     return(coste_total);
}

random_int(low, high) {
     /* devuelve un entero aleatorio (distribución uniforme)
        entre low y high */
     return(random_int);
}

random_float() {
     /* devuelve un float aleatorio (distribución uniforme)
        entre 0 y 1 */
     return(random_float);
}

random_exponencial() {
     /* devuelve un número positivo aleatorio con distribución
        exponencial, p.e., Prob(num < X) = 1 - exp(-X) */
     return(random_numero);
}

eval_cambio_coste(l, r) {
     /* devuelve el cambio en el coste total (ganacia) cuando
        el módulo en left[l] es cambiado con el módulo right[r]
        Esta función NO mueve los módulos */
     return(delta_cambio_coste);
}
```

```
/* aquí empieza lo interesante */

anneal() {

     for (move = 1; move <= N; move++) {
         /* prueba de movimiento */
         ll = random_int(1, N);
         rr = random_int(ll, N);

         /* moverlos: intenta el movimiento y evalua */
         left[ll] = right[rr];
         right[rr] = left[ll];
         delta = eval_cambio_coste();

         if (delta > 0) {

            R = random_exponencial();
            threshold = exp(T/delta);
            if (R < threshold) {
               /* huy!!, deshacer el cambio */
               left[ll] = right[rr];
               right[rr] = left[ll];
               T = COOLING * T;
             }
             else {
               T = COOLING * T;
             }
          }
     } /* end for */
}  /* end anneal */


main() {

     /* inicializar */
     T = HOT;       /* temperatura es variable global!! */
     oldestcost = oldcost = currentcost = 999999999;
     done = 0;

     while (!done) {

           anneal();

           /* 3 seguidas sin mejora ??? */
           oldestcost = oldcost;
           oldcost = currentcost;
           currentcost = eval_coste_actual();
           if (oldestcost == oldcost && oldcost == currentcost)
              done = 1;
      }
} /* end main */
```