Algoritmos 2021/22

Grado en Ingeniería Informática

Práctica 4

Límite para la entrega: martes 7 de diciembre, a las 23:59

Algoritmo de Prim:

```
función Prim ( M[1..n, 1..n] )
    T := ∅ ; /* T está vacío al inicio */
    DistanciaMinima[1] = -1 ;
    para i := 2 hasta n hacer
        MasProximo[i] := 1 ;
        DistanciaMinima[i] := M[i,1]
    fin para ;
    repetir n-1 veces /* bucle voraz */
        min := ∞ ;
        para j := 2 hasta n hacer
            si 0 <= DistanciaMinima [j] < min entonces
                min := DistanciaMinima [j] ;
                k := j
            fin si
        fin para ;
        T := T ∪ {(MasProximo[k], k)} ;
        DistanciaMinima[k] := -1;
        para j := 2 hasta n hacer
            si M[j, k] < DistanciaMinima[j] entonces
                DistanciaMinima[j] := M[j, k] ;
                MasProximo[j] := k
            fin si
        fin para
    fin repetir ;
    devolver T
fin función
```

1. Implemente en C este algoritmo de modo que devuelva las aristas que forman el árbol de recubrimiento mínimo (figura 1) en una cola.

   Haga una implementación circular de la cola en base a vectores, tal como se vio en clase de teoría.

2. Compruebe que el algoritmo funciona correctamente. En las figuras 2, 3 y 4 se proponen tres casos de prueba.

3. Usando las funciones de la figura 5 para generar aleatoriamente grafos completos no dirigidos, calcule empíricamente la complejidad computacional del algoritmo para el cálculo del árbol de recubrimiento.

4. Entregue los ficheros con el código C y el fichero `.txt` con el informe por medio de la tarea *Entrega Práctica 4* en la página de Algoritmos en `https://campusvirtual.udc.gal`. Se recuerda que el límite para completar la tarea es el martes 7 de diciembre a las 23:59, y una vez subidos los archivos no se podrán cambiar. **Todos los compañeros que forman un equipo tienen que entregar el trabajo**.

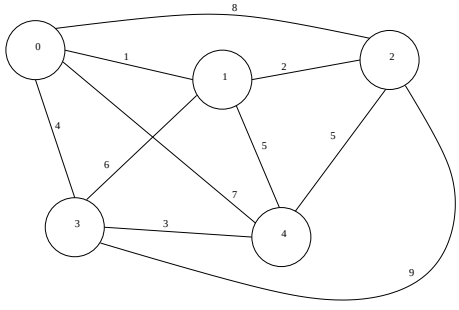
```
#define TAM_MAX 1600

typedef int ** matriz;
typedef struct {
  int x, y, peso;
} arista;

typedef arista tipo_elemento;
typedef struct {
  int cabeza, final, tamano;
  tipo_elemento vector[TAM_MAX];
} cola;
void crear_cola(cola *);
int cola_vacia(cola);
void insertar(tipo_elemento, cola *);
tipo_elemento quitar_primero(cola *);
tipo_elemento primero(cola);
void mostrar_cola(cola);


void prim(matriz m, int nodos, cola *aristas) {
  /* calcular el árbol de recubrimiento mínimo devolviendo
     las aristas del arbol en la cola 'aristas' */
  int min, i, j, k=0;
  arista a;
  int *masProximo = (int *) malloc(nodos*sizeof(int));
  int *distanciaMinima = (int *) malloc(nodos*sizeof(int));
  crear_cola(aristas);
  distanciaMinima[0] = -1;
  for(i = 1; i < nodos; i++) {
    masProximo[i] = 0;
    distanciaMinima[i] = m[i][0];
  }
   /*
    ...
    */
  free(masProximo);
  free(distanciaMinima);
}
```

Figura 1: Parte de la implementación de la función `prim`



(a) Grafo

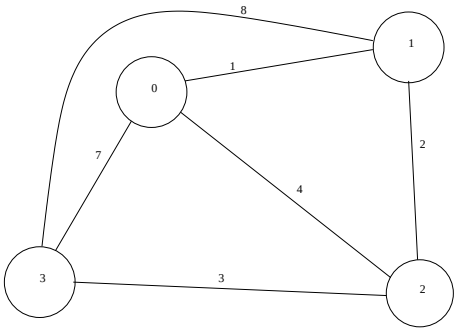
$$\begin{pmatrix} 0 & 1 & 8 & 4 & 7 \\ 1 & 0 & 2 & 6 & 5 \\ 8 & 2 & 0 & 9 & 5 \\ 4 & 6 & 9 & 0 & 3 \\ 7 & 5 & 5 & 3 & 0 \end{pmatrix}$$

(b) Matriz de adyacencia

Aristas: $(0,1),(1,2),(0,3),(3,4)$
Peso: 10

(c) Árbol de recubrimiento mínimo

Figura 2: Primer ejemplo



(a) Grafo

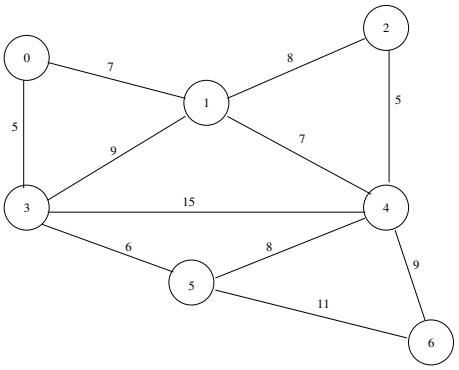
$$\begin{pmatrix} 0 & 1 & 4 & 7 \\ 1 & 0 & 2 & 8 \\ 4 & 2 & 0 & 3 \\ 7 & 8 & 3 & 0 \end{pmatrix}$$

(b) Matriz de adyacencia

Aristas: $(0,1),(1,2),(2,3)$
Peso: 6

(c) Árbol de recubrimiento mínimo

Figura 3: Segundo ejemplo



(a) Grafo

$$\begin{pmatrix} 0 & 7 & 99 & 5 & 99 & 99 & 99 \\ 7 & 0 & 8 & 9 & 7 & 99 & 99 \\ 99 & 8 & 0 & 99 & 5 & 99 & 99 \\ 5 & 9 & 99 & 0 & 15 & 6 & 99 \\ 99 & 7 & 5 & 15 & 0 & 8 & 9 \\ 99 & 99 & 99 & 6 & 8 & 0 & 11 \\ 99 & 99 & 99 & 99 & 9 & 11 & 0 \end{pmatrix}$$

(b) Matriz de adyacencia

Aristas: $(0,3),(3,5),$
$(0,1),(1,4),$
$(4,2),(4,6)$
Peso: 39

(c) Árbol de recubrimiento mínimo

Figura 4: Tercer ejemplo

```
matriz crear_matriz(int n) {
  int i;
  matriz aux;
  if ((aux = malloc(n*sizeof(int *))) == NULL)
    return NULL;
  for (i=0; i<n; i++)
    if ((aux[i] = malloc(n*sizeof(int))) == NULL)
      return NULL;
  return aux;
}

void inicializar_matriz(matriz m, int n) {
  /* Crea un grafo completo no dirigido con valores aleatorios entre 1 y n */
  int i, j;

  for (i=0; i<n; i++)
    for (j=i+1; j<n; j++)
      m[i][j] = rand() % n + 1;

  for (i=0; i<n; i++)
    for (j=0; j<=i; j++)
      if (i==j)
        m[i][j] = 0;
      else
        m[i][j] = m[j][i];
}

void liberar_matriz(matriz m, int n) {
  int i;
  for (i=0; i<n; i++)
    free(m[i]);
  free(m);
}
```

Figura 5: Las funciones `crear_matriz`, `ini_matriz` y `liberar_matriz`