Algorithms 2021/22

Degree in Computer Science Engineering

Practical 4

Submission deadline: Tuesday, 7th December 2021 at 23:59

Prim's Algorithm:

```
function Prim ( M[1..n, 1..n] )
    T := ∅ ; /* T is initially empty */
    MinimumDistance[1] = -1 ;
    for i := 2 to n do
        Closest[i] := 1 ;
        MinimumDistance[i] := M[i,1]
    end for ;
    repeat n-1 times /* greedy loop */
        min := ∞ ;
        for j := 2 to n do
            if 0 <= MinimumDistance[j] < min then
                min := MinimumDistance[j] ;
                k := j
            end if
        end for ;
        T := T ∪ {(Closest[k], k)} ;
        MinimumDistance[k] := -1;
        for j := 2 to n do
            if M[j, k] < MinimumDistance[j] then
                MinimumDistance[j] := M[j, k] ;
                Closest[j] := k
            end if
        end for
    end repeat ;
    return T
end function
```

1. Implement this algorithm in C in such a way that it returns the edges that form the minimum spanning tree (figure 1) in a queue.

   Create a array-based circular implementation of the queue, as seen in the theory class.

2. Check that the algorithm works correctly. Three test cases are proposed in figures 2, 3 and 4.

3. Using the functions of figure 5 to randomly generate complete undirected graphs, calculate the computational complexity of the algorithm empirically for the calculation of the spanning tree.

4. Submit the files with the C source code and the report (`.txt`) by means of the task *Practical 4 Submission* at the Algorithms page in `https://campusvirtual.udc.gal`. Remember that the deadline to complete the task is Tuesday, 7th December, at 23:59 and, once uploaded, files cannot be changed. **All the members of each team must submit the work.**

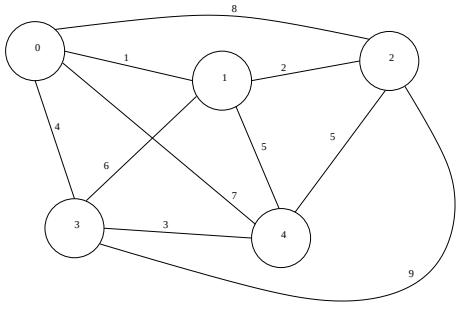
```
#define MAX_SIZE 1600

typedef int ** matrix;
typedef struct {
  int x, y, weight;
} edge;

typedef edge element_type;
typedef struct {
  int front_index, rear_index, size;
  element_type vector[MAX_SIZE];
} queue;
void create_queue(queue *);
int empty_queue(queue);
void enqueue(element_type, queue *);
element_type dequeue(queue *);
element_type front(queue);
void show_queue(queue);


void prim(matrix m, int nodes, queue *edges) {
  /* calculate the minimum spanning tree returning
     the edges of the tree in the 'edges' queue */
  int min, i, j, k=0;
  edge a;
  int *closest = (int *) malloc(nodes*sizeof(int));
  int *minDistance = (int *) malloc(nodes*sizeof(int));
  create_queue(edges);
  minDistance[0] = -1;
  for(i = 1; i < nodes; i++) {
    closest[i] = 0;
    minDistance[i] = m[i][0];
  }
   /*
    ...
    */
  free(closest);
  free(minDistance);
}
```

Figure 1: Part of the implementation of function `prim`



(a) Graph

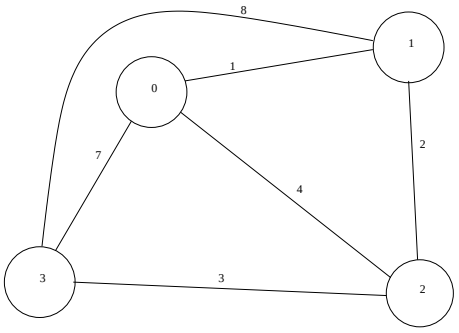
$$\begin{pmatrix} 0 & 1 & 8 & 4 & 7 \\ 1 & 0 & 2 & 6 & 5 \\ 8 & 2 & 0 & 9 & 5 \\ 4 & 6 & 9 & 0 & 3 \\ 7 & 5 & 5 & 3 & 0 \end{pmatrix}$$

(b) Adjacency matrix

Edges: $(0,1),(1,2),(0,3),(3,4)$
Weight: 10

(c) Minimum spanning tree

Figure 2: First example



(a) Graph

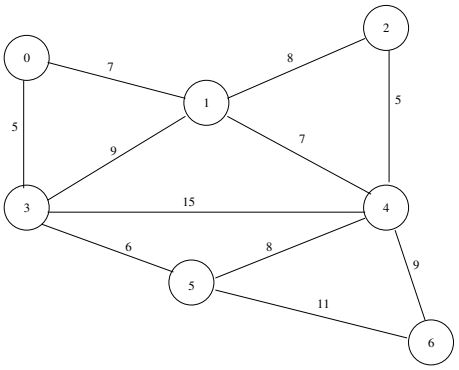
$$\begin{pmatrix} 0 & 1 & 4 & 7 \\ 1 & 0 & 2 & 8 \\ 4 & 2 & 0 & 3 \\ 7 & 8 & 3 & 0 \end{pmatrix}$$

(b) Adjacency matrix

Edges: $(0,1),(1,2),(2,3)$
Weight: 6

(c) Minimum spanning tree

Figure 3: Second example



(a) Graph

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

(b) Adjacency matrix

Edges: $(0,3),(3,5),$
$(0,1),(1,4),$
$(4,2),(4,6)$
Weight: 39

(c) Minimum spanning tree

Figure 4: Third example

```
matrix create_matrix(int n) {
  int i;
  matrix 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 init_matrix(matrix m, int n) {
  /* Creates an undirected complete graph with random values between 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 free_matrix(matrix m, int n) {
  int i;
  for (i=0; i<n; i++)
    free(m[i]);
  free(m);
}
```

Figure 5: Functions `create_matrix`, `init_matrix` and `free_matrix`