Algoritmul lui Rosenstiehl

Calin Mihai
CEN 1.2A Slytherin

Year 2017 - 2018
Semester II

Problem Statement:

The Rosenstiehl algorithm builds a chain L which will become an eulerian cycle using for it a stack as an auxiliar data structure.

It starts from a random node. It is advancing while it is still possible: for the current node u se search search an incident edge with it and which was not travelede at one of the steps before. If there exists such an edge, then we save in the stack the u node, and the node v becomes the current node.

In the moment in which the current node does not have visited edges, we add it to the chain L, and we extract form the stack the previous node. The vector *visit* has the purpose to keep the edges situation.

Application Design:

Rosenstiehl Algorithm to determine an eulerian cycle in a conex graph.

Pseudocode algorithm:

```
1: function ROSENSTIEHL(u, G)
2:     for e ∈ E do
3:         vizit_e ← 0
4:     end for
5:     S ⇐ u                                          ▷ Se inserează pe stivă nodul u
6:     while (S ≠ ∅) do
7:         S ⇒ u                                      ▷ Se extrage din stivă nodul curent
8:         while (∃e = [u, v] ∈ E) ∧ (vizit_e = 0)) do
9:             vizit_e ← 1                            ▷ Se marchează muchia e ca fiind utilizată
10:            S ⇐ u                                  ▷ Se salvează pe stivă nodul curent u
11:            u ← v                                  ▷ Nodul curent devine nodul v
12:        end while
13:        L ⇐ u                                      ▷ Se adaugă la lista L nodul curent
14:    end while
15:    return L
16: end function
```

To continue , because there is no more unvisited edges, we will extract the elements from the stack S, one at each step, and they'll be inserted in the list L.

Implementation of the algorithm in C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

struct g_node{
    int info;
    struct g_node *next;
};

void push_element_end(struct g_node *head ,int new_element_value){
    struct g_node *new_element = malloc(sizeof(struct g_node));
    struct g_node *iterator = head;
    struct g_node *last_element;

    while (iterator->next != NULL) {
        iterator = iterator->next;
    }
    last_element = iterator;

    last_element->next = new_element;
    new_element->info = new_element_value;
    new_element->next = NULL;
}

int pop_element_end(struct g_node *head){
    struct g_node *poped_element;
    struct g_node *iterator = head;
    int aux;

    while (  iterator -> next -> next != NULL) {
        iterator = iterator->next;
    }

    poped_element = iterator->next;
    aux = poped_element->info;
    iterator->next = poped_element->next;

    free(poped_element);

    return aux;
}
```

```
42
43  void print_list(struct g_node *head){
44      struct g_node *iterator = head;
45
46      while (iterator->next != NULL) {
47          printf("%d ", iterator->next->info);
48          iterator = iterator->next;
49      }
50      printf("\n");
51  }
52
53  int return_no_elements(struct g_node *head){
54      int no_elements ;
55      struct g_node *iterator;
56      iterator = head;
57      no_elements = 0;
58
59      while (iterator->next != NULL) {
60          iterator = iterator->next;
61          ++no_elements;
62      }
63      return no_elements;
64  }
65
66  void rosenstiehl(int n, int ma[6][6], int u, struct g_node *head, struct g_node *
67      int v;
68      push_element_end(head, u);
69      while (return_no_elements(head)) {
70          u = pop_element_end(head);
71          v = 0;
72          while (v < n) {
73              if (ma[u][v] == 1) {
74                  ma[u][v] = 0;
75                  ma[v][u] = 0;
76                  push_element_end(head, u);
77                  u = v;
78                  v = 0;
79              }
80              else v++;
81          }
82          push_element_end(head2, u);
83      }
```

```
85
86
87  int main(){
88      struct g_node *head = malloc(sizeof(struct g_node));
89      head->next = NULL;
90      struct g_node *head2 = malloc(sizeof(struct g_node));
91      head2->next = NULL;
92
93
94      int aux, aux2;
95      int i, j;
96      int ma[6][6] = {0, 1, 1, 1, 1, 0,
97                      1, 0, 1, 1, 1, 0,
98                      1, 1, 0, 1, 0, 1,
99                      1, 1, 1, 0, 0, 1,
100                     1, 1, 0, 0, 0, 0,
101                     0, 0, 1, 1, 0, 0};
102
103     for(i = 0; i < 6; i++){
104         for(j = 0; j < 6; j++){
105                 printf("%5d ",ma[i][j]);
106         }
107         printf("\n");
108     }
109     rosenstiehl(6, ma, 0, head, head2);
110     print_list(head2);
111
112     free(head);
113     free(head2);
114
115     return 0;
116 }
117
```

The input will be taken from a file:

6
0 1 1 1 1 0
1 0 1 1 1 0
1 1 0 1 0 1
1 1 1 0 0 1
1 1 0 0 0 0
0 0 1 1 0 0

In the first row we have the number of nodes (cardinal of V), and from the second row we have the values of the adjacency matrix, separated by spaces having each line on a row.

Conclusions:

I've learned how to implement an interesting algorithm for finding Eulerian Cycles.

The Bibliography

http://en.wikipedia.org/wiki/ Standard_Template_Library
http://www.sgi.com/tech/stl/
http://www.sgi.com/tech/stl/stack.html
http://www.sgi.com/tech/stl/List.html
http://www.sgi.com/tech/stl/Vector.html