Rosenstiehls' Algorithm

Perisoara Mihai Cezar
CEN 1.2B 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.

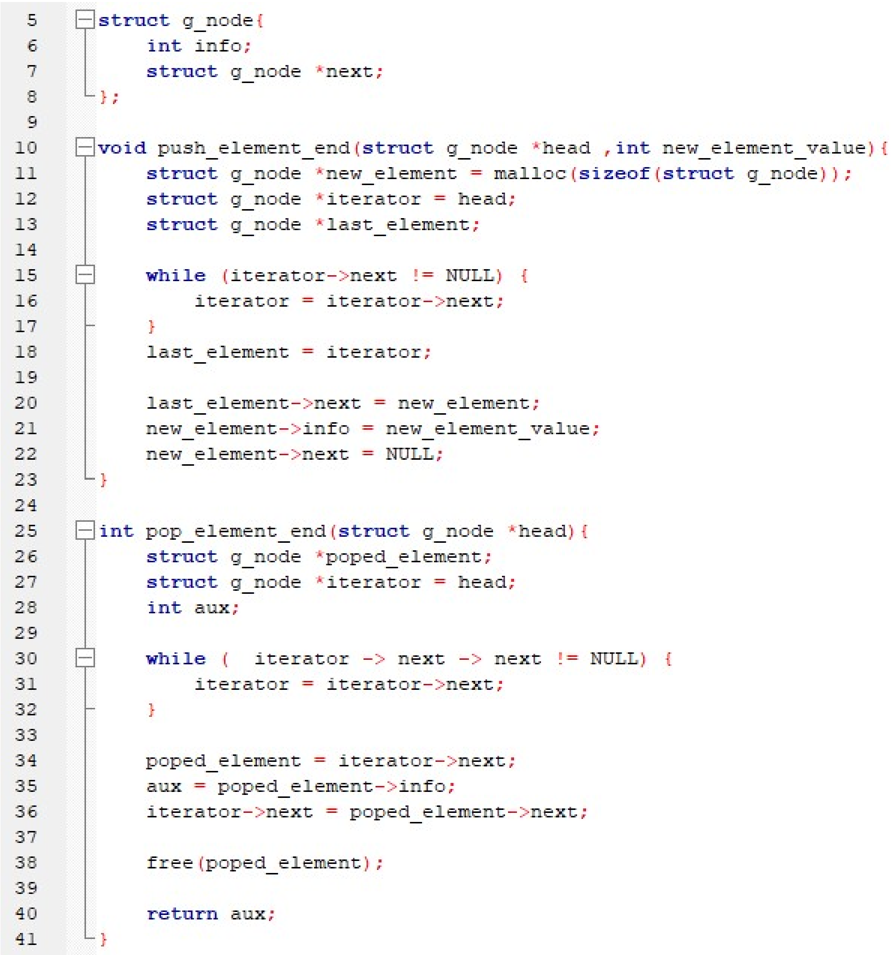
It starts from a random node. It is advancing while it is still possible: for the current node u se search search an incident edge with it and which was not travelede at one of the steps before. If there exists such an edge, then we save in the stack the u node, and the node v becomes the current node.

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.

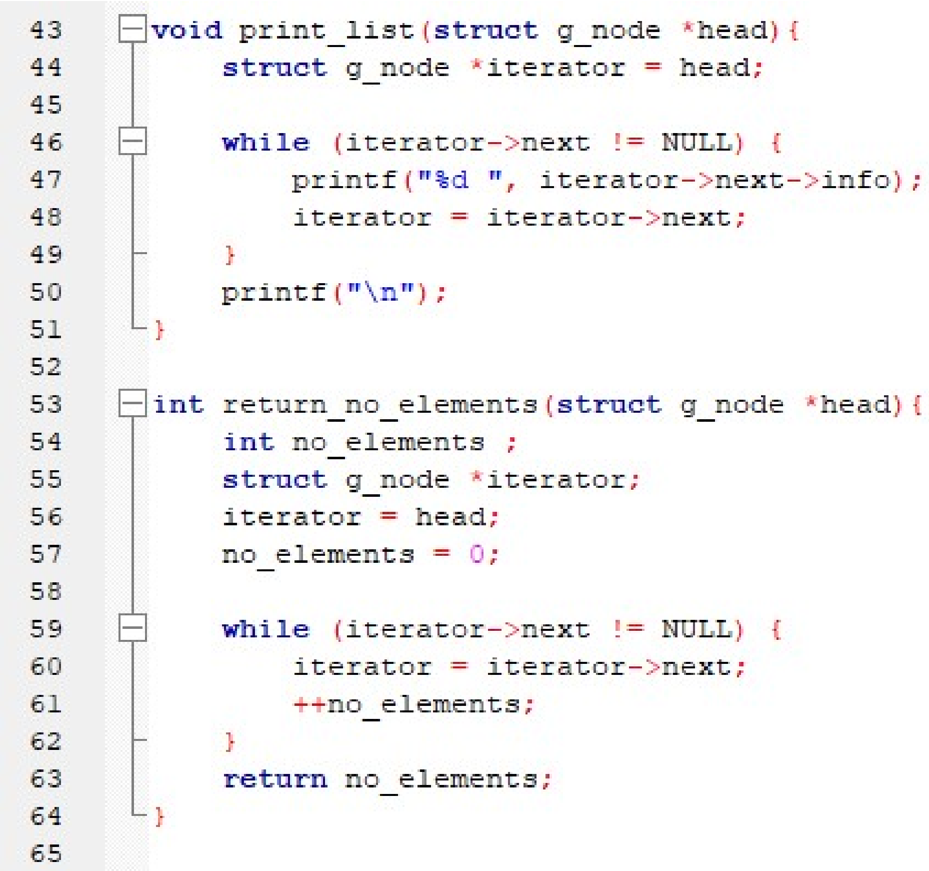
Pseudocode algorithm:

```
function Rosenstiehl($u$, $G$)
    for $e \in E$ do
        $vizit_e \leftarrow 0$
    end for
    $S \Leftarrow u$                                          ▷ Se inserează pe stivă nodul $u$
    while $(S \neq \emptyset)$ do
        $S \Rightarrow u$                                      ▷ Se extrage din stivă nodul curent
        while $(\exists e = [u, v] \in E) \wedge (vizit_e = 0))$ do
            $vizit_e \leftarrow 1$                 ▷ Se marchează muchia $e$ ca fiind utilizată
            $S \Leftarrow u$                         ▷ Se salvează pe stivă nodul curent $u$
            $u \leftarrow v$                                 ▷ Nodul curent devine nodul $v$
        end while
        $L \Leftarrow u$                                ▷ Se adaugă la lista $L$ nodul curent
    end while
    return $L$
end function
```

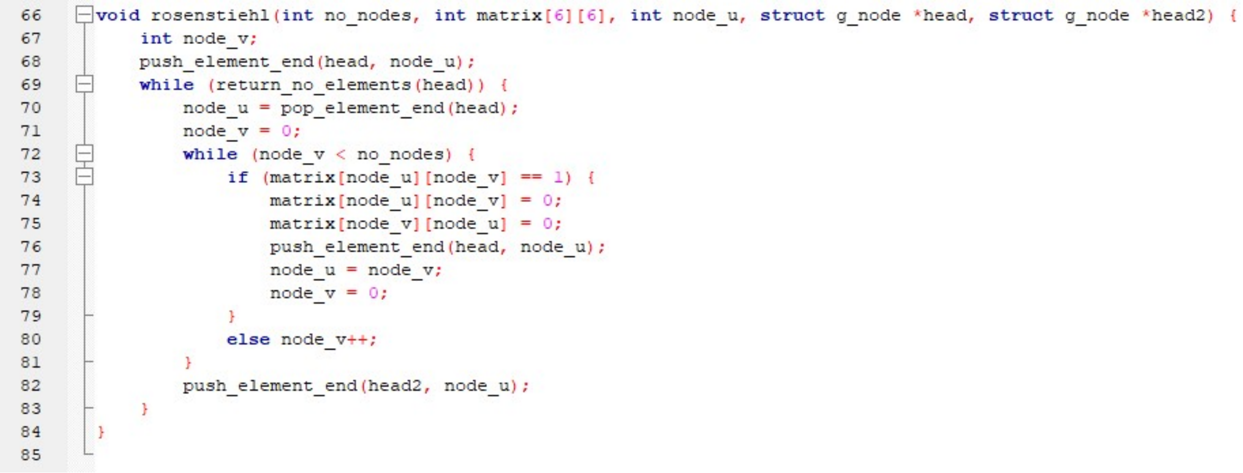
Implementation of the algorithm in C:



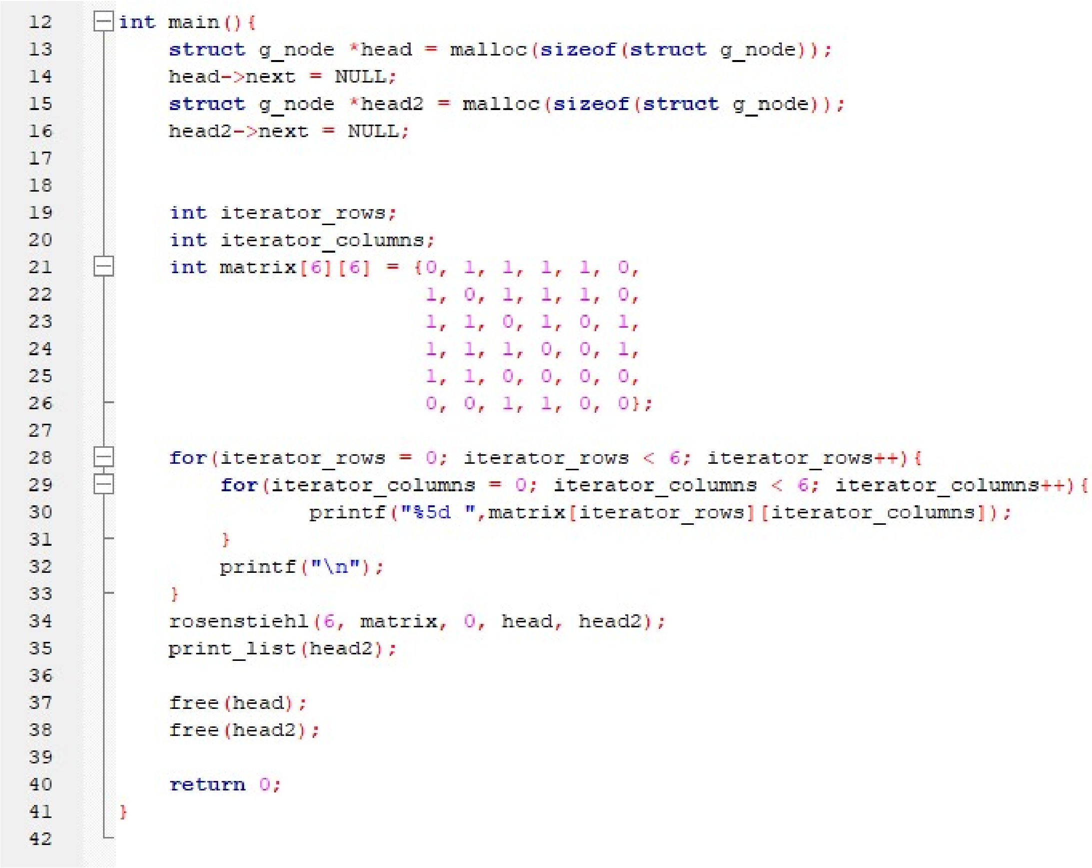
```c
struct g_node{
    int info;
    struct g_node *next;
};

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;
}
```



```c
void print_list(struct g_node *head){
    struct g_node *iterator = head;

    while (iterator->next != NULL) {
        printf("%d ", iterator->next->info);
        iterator = iterator->next;
    }
    printf("\n");
}

int return_no_elements(struct g_node *head){
    int no_elements ;
    struct g_node *iterator;
    iterator = head;
    no_elements = 0;

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

```



```c
void rosenstiehl(int no_nodes, int matrix[6][6], int node_u, struct g_node *head, struct g_node *head2) {
    int node_v;
    push_element_end(head, node_u);
    while (return_no_elements(head)) {
        node_u = pop_element_end(head);
        node_v = 0;
        while (node_v < no_nodes) {
            if (matrix[node_u][node_v] == 1) {
                matrix[node_u][node_v] = 0;
                matrix[node_v][node_u] = 0;
                push_element_end(head, node_u);
                node_u = node_v;
                node_v = 0;
            }
            else node_v++;
        }
        push_element_end(head2, node_u);
    }
}

```



```
int main(){
    struct g_node *head = malloc(sizeof(struct g_node));
    head->next = NULL;
    struct g_node *head2 = malloc(sizeof(struct g_node));
    head2->next = NULL;


    int iterator_rows;
    int iterator_columns;
    int matrix[6][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};

    for(iterator_rows = 0; iterator_rows < 6; iterator_rows++){
        for(iterator_columns = 0; iterator_columns < 6; iterator_columns++){
               printf("%5d ",matrix[iterator_rows][iterator_columns]);
        }
        printf("\n");
    }
    rosenstiehl(6, matrix, 0, head, head2);
    print_list(head2);

    free(head);
    free(head2);

    return 0;
}

```

The input data 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

On the first line we have the number of vertices (the cardinal of V) and starting with the second line we have the values of the adjacency matrix, separated by spaces, a row on each line.

Conclusions:

This algorithm is quite an interesting one.

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