

Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

ЗВІТ
З ПРАКТИЧНОЇ РОБОТИ №11
дисципліна: «Алгоритмізація та програмування»

Виконав: студент 2 курсу групи КС22
Спеціальності 122 «Комп'ютерні науки»
Скрипняк Тарас Артемович
Прийняв: викладач
Олешко О.І.

Завдання №1: Реалізувати алгоритм Прима знаходження мінімального кістякового дерева. Для представлення графа використовувати матрицю суміжності, для черги з пріоритетами – масив.
Виконати порівняльний аналіз ефективності алгоритму для розріджених та щільних графів.

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#include <time.h>

int minKey(int V, int key[], bool mstSet[])
{
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

void printMST(int V, int parent[], int graph[V][V])
{
    printf("Edge \tWeight\n");
    for (int i = 1; i < V - 1; i++)
        printf("%d - %d \t%d \n", parent[i] + 1, i + 1, graph[i][parent[i]]);
}

clock_t primMST(int V, int graph[V][V])
{
    clock_t start = clock();
    int parent[V];
    int key[V];
    bool mstSet[V];

    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(V, key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }
    clock_t end = clock();
    printMST(V, parent, graph);
    return end - start;
}

int maxMatrix(int n, int info[n][3]) {
    int max = 0;
    for (int i = 0; i < n; i++) if (max < info[i][0] || max < info[i][1])
        max = info[i][0] > info[i][1] ? info[i][0] : info[i][1];
    return max + 1;
}
```

```

}

void printMatrix(int rows, int cols, int matrix[rows][cols]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%7d ", matrix[i][j]);
        }
        printf("\n");
    }
}

void test(int n, int info[n][3]) {
    int q = maxMatrix(n, info);
    int graph[q][q];
    for (int i = 0; i < q; i++)
        for (int j = 0; j < q; j++)
            graph[i][j] = 0;

    for (int i = 0; i < n; i++) {
        int u = info[i][0], v = info[i][1], weight = info[i][2];
        graph[u-1][v-1] = graph[v-1][u-1] = weight;
    }

    printf("MATRIX:\n");
    printMatrix(q, q, graph);

    clock_t time = primMST(q, graph);

    printf("Time for calc: %ld microseconds\n", time);
}

int main() {
    // start, end, weight
    int info1[11][3] = {
        {1, 2, 7},
        {2, 3, 8},
        {3, 5, 5},
        {5, 2, 7},
        {1, 4, 5},
        {4, 2, 9},
        {4, 5, 15},
        {4, 6, 6},
        {6, 5, 8},
        {5, 7, 9},
        {6, 7, 11},
    };

    test(11, info1);

    // start, end, weight
    int info2[21][3] = {
        {1,2,3}, {1,3,10}, {1,4,5},
        {1,5,8}, {1,6,7}, {1,7,6},
        {2,3,12}, {2,4,9}, {2,5,7},
        {2,6,4}, {2,7,10}, {3,4,11},
        {3,5,6}, {3,6,8}, {3,7,9},
        {4,5,7}, {4,6,3}, {4,7,2},
        {5,6,4}, {5,7,5}, {6,7,1},
    };

    test(21, info2);

    return 0;
}

```

Лістинг - вихідний код програми

```
● bouncytorch@AORUS:~/Repos/homework-c/pr11/tarik$ ./a.out
MATRIX:
    0      7      0      5      0      0      0      0
    7      0      8      9      7      0      0      0
    0      8      0      0      5      0      0      0
    5      9      0      0     15      6      0      0
    0      7      5     15      0      8      9      0
    0      0      0      6      8      0     11      0
    0      0      0      0      9     11      0      0
    0      0      0      0      0      0      0      0

Edge   Weight
1 - 2   7
5 - 3   5
1 - 4   5
2 - 5   7
4 - 6   6
5 - 7   9
Time for calc: 3 microseconds

MATRIX:
    0      3     10      5      8      7      6      0
    3      0     12      9      7      4     10      0
   10     12      0     11      6      8      9      0
    5      9     11      0      7      3      2      0
    8      7      6      7      0      4      5      0
    7      4      8      3      4      0      1      0
    6     10      9      2      5      1      0      0
    0      0      0      0      0      0      0      0

Edge   Weight
1 - 2   3
5 - 3   6
7 - 4   2
6 - 5   4
2 - 6   4
6 - 7   1
Time for calc: 2 microseconds
```

Рисунок 1 - результат виконання програми

Завдання №2: Реалізувати алгоритм Крускала знаходження мінімального кістякового дерева. Виконати порівняльний аналіз ефективності з алгоритмом Пріма на графах для попереднього алгоритму (для розріджених та щільних графів).

```
// C code to implement Kruskal's algorithm

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Comparator function to use in sorting
int comparator(const void* p1, const void* p2)
{
    const int(*x)[3] = p1;
    const int(*y)[3] = p2;

    return (*x)[2] - (*y)[2];
}

// Initialization of parent[] and rank[] arrays
```

```

void makeSet(int parent[], int rank[], int n)
{
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}

// Function to find the parent of a node
int findParent(int parent[], int component)
{
    if (parent[component] == component)
        return component;

    return parent[component]
        = findParent(parent, parent[component]);
}

// Function to unite two sets
void unionSet(int u, int v, int parent[], int rank[], int n)
{
    // Finding the parents
    u = findParent(parent, u);
    v = findParent(parent, v);

    if (rank[u] < rank[v]) {
        parent[u] = v;
    }
    else if (rank[u] > rank[v]) {
        parent[v] = u;
    }
    else {
        parent[v] = u;

        // Since the rank increases if
        // the ranks of two sets are same
        rank[u]++;
    }
}

// Function to find the MST
void kruskalAlgo(int n, int edge[n][3])
{
    // First we sort the edge array in ascending order
    // so that we can access minimum distances/cost
    qsort(edge, n, sizeof(edge[0]), comparator);
    clock_t start = clock();

    int parent[n];
    int rank[n];

    // Function to initialize parent[] and rank[]
    makeSet(parent, rank, n);

    // Array to store the edges in the MST
    int mstEdges[n][3];
    int mstEdgeCount = 0;
    int mstWeight = 0;

    for (int i = 0; i < n; i++) {
        int u = edge[i][0];
        int v = edge[i][1];
        int wt = edge[i][2];

        int v1 = findParent(parent, u);
        int v2 = findParent(parent, v);
    }
}

```

```

    // If the parents are different, union the sets and add the edge to MST
    if (v1 != v2) {
        unionSet(v1, v2, parent, rank, n);
        mstEdges[mstEdgeCount][0] = u;
        mstEdges[mstEdgeCount][1] = v;
        mstEdges[mstEdgeCount][2] = wt;
        mstEdgeCount++;
        mstWeight += wt;
    }
}
clock_t end = clock();
printf("Time for calc: %ld microseconds\n", end - start);

// Print the edges and total weight of the MST after timing ends
printf("Edge \tWeight\n");
for (int i = 0; i < mstEdgeCount; i++) {
    printf("%d - %d \t%d\n", mstEdges[i][0], mstEdges[i][1], mstEdges[i]
[2]);
}
printf("Total weight of MST: %d\n", mstWeight);
}

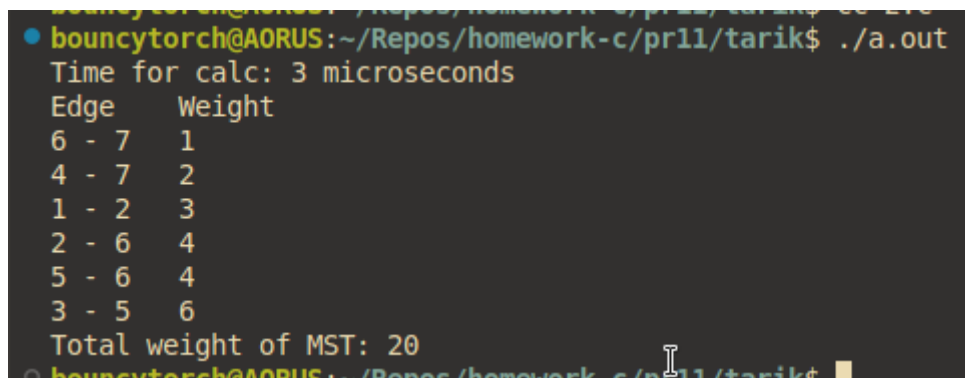
// Driver code
int main()
{
    // start, end, weight
    int info2[21][3] = {
        {1, 2, 3}, {1, 3, 10}, {1, 4, 5},
        {1, 5, 8}, {1, 6, 7}, {1, 7, 6},
        {2, 3, 12}, {2, 4, 9}, {2, 5, 7},
        {2, 6, 4}, {2, 7, 10}, {3, 4, 11},
        {3, 5, 6}, {3, 6, 8}, {3, 7, 9},
        {4, 5, 7}, {4, 6, 3}, {4, 7, 2},
        {5, 6, 4}, {5, 7, 5}, {6, 7, 1},
    };

    kruskalAlgo(21, info2);

    return 0;
}

```

Лістинг - вихідний код програми



```

bouncytorch@AORUS:~/Repos/homework-c/pr11/tarik$ ./a.out
Time for calc: 3 microseconds
Edge    Weight
6 - 7    1
4 - 7    2
1 - 2    3
2 - 6    4
5 - 6    4
3 - 5    6
Total weight of MST: 20
bouncytorch@AORUS:~/Repos/homework-c/pr11/tarik$

```

Рисунок 1 - результат виконання програми

Алгоритм Прима: Для графа, представленого у вигляді матриці суміжності, використовується масив. Часова складність пошуку мінімального ребра для кожної вершини становить $O(V^2)$, де V — кількість вершин. У розріджених графах (де мало ребер) алгоритм менш

ефективний через квадратичну складність, незалежно від кількості ребер. У щільних графах (з великою кількістю ребер) він працює добре, оскільки перевірка кожної пари вершин неминуча.

Алгоритм Крускала: Складність сортування ребер дорівнює $O(E \log E)$, де E — кількість ребер. Перевірка циклів виконується за допомогою "системи непересічних множин" із складністю $O(E \cdot \alpha(V))$, де α — обмежена функція від розміру множин. Для розріджених графів алгоритм ефективний, оскільки кількість ребер невелика. У щільних графах його ефективність дещо знижується через необхідність сортування великої кількості ребер, але він може бути кращим, якщо $E \log E$ менше, ніж V^2 .

Висновок: Алгоритм Крускала виявляється більш вигідним, оскільки сортування ребер менш затратне за ресурсами, ніж перевірка кожної вершини і ребра у Примі. Проте алгоритм Прима може бути кращим для дуже щільних графів, де кількість ребер наближається до V^2 .