<table>
<tr><td colspan="2">НУЛП, ІКНІ, САПР</td><td>Тема</td><td>оцінка</td><td>підпис</td></tr>
<tr><td>КН-414</td><td>1</td><td rowspan="4">АЛГОРИТМ ПОБУДОВИ ДЕРЕВ</td><td rowspan="3"></td><td rowspan="3"></td></tr>
<tr><td colspan="2">Юнусов Ю. М.</td></tr>
<tr><td colspan="2">№ залікової: 16083062</td></tr>
<tr><td colspan="2">Дискретні моделі в САПР</td><td colspan="2">Викладач:
к.т.н., асистент
Кривий Р.З.</td></tr>
</table>

Мета:

Вивчення алгоритмів рішення задач побудови остових дерев.

Завдання: Написати програму для побудови мінімального та максимального покриваючого дерева.

Варіант 1. Алгоритм Борувки.

Теоретичні відомості:

Максимальне остове дерево.

Даний зважений неорієнтований граф з вершинами і ребрами. Потрібно знайти таке піддерево цього графа, яке б з'єднувало всі його вершини, і при цьому мало найбільшу можливу вагою (тобто сумою ваг ребер). Таке піддерево називається максимальним остовим деревом.

У природному постановці ця задача звучить наступним чином: є міст, і для кожної пари відома вартість з'єднання їх дорогою (або відомо, що з'єднати їх не можна). Потрібно з'єднати всі міста так, щоб можна було доїхати з будь-якого міста в інший, а при цьому вартість прокладання доріг була б максимальною. Сам алгоритм має дуже простий вигляд. Шуканий максимальний кістяк будується поступово, додаванням до нього ребер по одному. Спочатку остов покладається складається з єдиної вершини (її можна вибрати довільно). Потім вибирається ребро максимальної ваги, що виходить з цієї вершини, і додається в максимальне остове дерево. Після цього остов містить уже дві вершини, і тепер шукається і додається ребро максимальної ваги, що має один кінець в одній з двох обраних вершин, а інший - навпаки, у всіх інших, крім цих двох. І так далі, тобто щоразу шукається максимальне по вазі ребро, один кінець якого - вже взята в остов вершина, а інший кінець - ще не взята, і це ребро додається в остов (якщо таких ребер кілька, можна взяти будь-яке). Цей процес повторюється до тих пір, поки остов не стане містити всі вершини (або, що те ж саме, ребро). У результаті буде побудований остов, що є максимальним . Якщо граф був спочатку не зв'язний, то остов знайдений не буде (кількість вибраних ребер залишиться менше).

Алгоритм Борувки.

Це алгоритм знаходження мінімального остового дерева в графі. Вперше був опублікований в 1926 році Отакаром Борувкой, як метод знаходження оптимальної електричної мережі в Моравії. Робота алгоритму складається з декількох ітерацій, кожна з яких полягає в послідовному додаванні ребер до остового лісу графа, до тих пір, поки ліс не перетвориться на дерево, тобто, ліс, що складається з однієї компоненти зв'язності. У псевдокоді, алгоритм можна описати так: Спочатку, нехай T - порожня множина ребер (представляє собою остовий ліс, до якого кожна вершина входить в якості окремого дерева). Поки T не є деревом (поки число ребер у T менше, ніж V-1, де V - кількість вершин у графі): Для кожної компоненти зв'язності (тобто, дерева в остовому лісі) в підпункті з ребрами T, знайдемо ребро найменшої ваги, що зв'язує цю компоненту з деякої іншої компонентою зв'язності. (Передбачається, що ваги ребер різні, або як-то додатково впорядковані так, щоб завжди можна було знайти єдине ребро з мінімальною вагою). Додамо всі знайдені ребра в множину T. Отримана множина ребер T є мінімальним остовим деревом вхідного графа.

Програмна реалізація основного методу:

```
while (numTree > 1) {
            System.out.println("Number of Vertices:" + numTree);

            //Reset the cheapest values every iteration
            for (int i = 0; i < vertNum; i++) {
                cheapest[i] = -1;
            }

            //Iterate over all edges to find the cheapest
            //edge of every subtree
            for (int i = 0; i < edgeNum; i++) {

                //Find the subsets of the corners of the edge
                int set1 = find(subsets, edges[i].getSrc());
                int set2 = find(subsets, edges[i].getDest());

                //If the two corners belong to the same subset,
                //ignore the current edge
                if (set1 != set2) {

                    //If they belong to different subsets, check which
                    //one is the cheapest
                    if (cheapest[set1] == -1 || edges[cheapest[set1]].getWeight() >
edges[i].getWeight()) {
```

```
                        cheapest[set1] = i;
                    }

                    if (cheapest[set2] == -1 || edges[cheapest[set2]].getWeight() >
edges[i].getWeight()) {
                        cheapest[set2] = i;
                    }
                }
            }

            //Add the cheapest edges obtained above to the MST
            for (int j = 0; j < vertNum; j++) {

                //Check if the cheapest for current set exists
                if (cheapest[j] != -1) {
                    int set1 = find(subsets, edges[cheapest[j]].getSrc());
                    int set2 = find(subsets, edges[cheapest[j]].getDest());

                    if(set1 != set2){
                        MSTweight += edges[cheapest[j]].getWeight();
                        System.out.println("Edge ("+
vertNames[edges[cheapest[j]].getSrc()] + ", " +
vertNames[edges[cheapest[j]].getDest()]+") added to the MST");
                        uniteSubsets(subsets, set1, set2);
                        numTree--;
                    }
                }
            }
        }
     System.out.println("Final weight of MST :" + MSTweight);
```

Результати роботи програми:

Вхідні данні:

```
4 5
0 A
1 B
2 C
3 D
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
```

Вхідні данні у вигляді текстового файлу, де: перша стрічка від повідає кількості вершин (4) та кількості ребер (5).

```
Initializing Boruvka's MST
Number of Vertices:4
Edge (A, D) added to the MST
Edge (A, B) added to the MST
Edge (C, D) added to the MST
Final weight of MST :19
```

Рис.1. Результат роботи програми

Висновок: На цій лабораторній роботі роботі було здійснено ознайомлення з алгоритмами побудови остових дерев, програмно реалізувано роботу алгоритму Борувки.