Прізвище: Пастернак
Ім’я: Вероніка
Група: КН-406
Варіант: 24
GitHub: https://github.com/veronikalpnu/8semest-kriviy
Кафедра: САПР
Дисципліна: Дискретні моделі САПР
Перевірив: Кривий Р.З.

ЗВІТ

до лабораторної роботи №1

на тему: “Алгоритм побудови дерев”

Мета роботи: вивчити алгоритмів рішення задач побудови остових дерев.

Теоретичні відомості:

ПОЛОЖЕННЯ ТЕОРІЇ ГРАФІВ

Графом *G* називають скінчену множину *V* з нерефлексивним симетричним відношенням *R* на *V*. Визначим *E* як множину симетричних пар в *R*. Кожний елемент *V* називають вершиною. Кожний елемент *E* називають ребром, а *E* множиною ребер *G*.

Граф називається зв’язним, якщо в ньому для будь-якої пари вершин знайдеться ланцюг, який їх з’єднує, тобто, якщо по ребрах (дугах) можна попасти з будь-якої вершини в іншу.
Цикл - це ланцюг, в якого початкова і кінцева точки співпадають.
Дерево - це зв’язний граф без циклів.
Покриваючим деревом графа називається любе дерево, що утворене сукупністю його ребер(дуг), які включають всі вершини графа.
Лісом називається будь-яка сукупність дуг (ребер) інцидентних до вершин, які не містять циклів. Таким чином, ліс складається з одного або більше дерев.
Остовним деревом графа називається будь-яке дерево, яке утворене сукупністю дуг, які включають всі вершини графа. Будь-який зв’язний граф має остовне дерево.
Коренем орієнтованого дерева (прадерева) називається його вершина, в яку не входить жодна з дуг.
Орієнтований ліс визначається як звичайний, тільки складається не зпростих дерев, а орієнтованих.
Остовним орієнтованим деревом називається орієнтоване дерево, яке одночасно є і остовим деревом.
Остовним орієнтованим лісом називається орієнтований ліс, який включає всі вершини відповідного графа.
Вага дерева - це сума ваг його ребер.

Поставимо у відповідність кожній дузі *(x, y)* графа *G* вагу *a(x, y)*. Вага орієнтованого лісу (або орієнтованого дерева) визначається як сума ваг дуг, що входять в даний ліс (дерево).

Максимальним орієнтованим лісом графа *G* називається орієнтований ліс графа G з максимально можливою вагою.
Максимальним орієнтованим деревом графа *G* називається орієнтоване дерево графа *G* з максимально можливою вагою. Мінімальні орієнтовані ліс і дерево визначаються аналогічним чином.
Кущ(букет) - зв'язний фрагмент графа.
Представлення графів в ЕОМ, в більшості випадків, здійснюється з допомогою матриць: суміжності, зв'язності, інцидентності і ваг.

ПОШУК ОСТОВОГО ДЕРЕВА

Нехай *G = (V, E)* - зв'язний граф, у якому кожне ребро *(u, v)* позначено числом *c(u, v)*, що називається вагою ребра. Остовним деревом графа *G* називається дерево, що містить всі вершини *V* графа *G*. Вага остового дерева обчислюється як сума ваг всіх ребер, що входять у це дерево.

Існують різні методи побудови максимальних остових дерев. Багато з них ґрунтуються на наступній властивості максимальних остових дерев . Нехай *G = (V, E)* - зв'язний граф із заданою функцією вартості, що задана на множині ребер. Позначимо через *U* підмножину вершин *V*. Якщо *(i, v)* - таке ребро найбільшої вартості, що й належить *U* і *v* належить *V \ U*, тоді для графа *G* існує максимальне остове дерево, що містить ребро *(i, v)*.

Існує декілька популярних алгоритмів побудови максимального остового дерева для позначеного графа *G = (V, E)*, що використовують описану властивість.

Алгоритм Борувки.
Це алгоритм знаходження мінімального остового дерева в графі. Вперше був опублікований в 1926р. 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:

1. Спочатку, нехай *T* - порожня множина ребер (представляє собою остовий ліс, до якого кожна вершина входить в якості окремого дерева).
2. Поки *T* не є деревом (поки число ребер у *T* менше, ніж *V-1*, де *V* - кількість вершин у графі):
   a. 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).
   b. Додамо всі знайдені ребра в множину *T*.
3. Отримана множина ребер *T* є мінімальним остовим деревом вхідного графа.

Алгоритм Крускала
Найбільш відомий алгоритм Крускала був придуманий автором у 1956р. Основна стратегія цього алгоритму така: ребра упорядковуються за вагою; на кожному кроці до споруджуваного остового дерева додається найлегше ребро, яке з'єднує вершини з різних компонент.

Таким чином на кожному кроці побудована множина складається з однієї або більше нетривіальних компонент, кожна з яких є підграфом деякого мінімального кістяка. Час роботи алгоритму Крускала становить *O(ElogE)* при використанні для зберігання компонент зв'язності системи непересічних множин з об'єднанням за рангом і стиском шляхів (найшвидший відомий метод). Більша частина часу йде на сортування ребер.

Побудова максимального остового дерева алгоритмом Прима

Цей алгоритм названий на честь американського математика Роберта Прима, який відкрив цей алгоритм у 1957р. Втім, ще в 1930р. цей алгоритм був відкритий чеським математиком Войтеком Ярніком. Крім того, Едгар Дейкстра в 1959р. також винайшов цей алгоритм, незалежно від них.

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).

Індивідуальне завдання:

1) Отримати у викладача індивідуальне завдання.
2) Підготувати програму для вирішення виданого завдання.
3) Запустити на покрокове виконання програму побудови мінімального покриваючого дерева і максимального покриваючого дерева.
4) Здійснити перевірки роботи програм з результатами розрахунків проведених вручну.
5) Зафіксувати результати роботи.
6) Оформити і захистити звіт.

Ручні обчислення:

Так як я Варіант 24 у списку групи я виконую - **Алгоритм Борувки** (1_3.txt):
Це алгоритм знаходження мінімального остового дерева в графі.
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:

1. Спочатку, нехай *T* - порожня множина ребер (представляє собою остовий ліс, до якого кожна вершина входить в якості окремого дерева).
2. Поки *T* не є деревом (поки число ребер у *T* менше, ніж *V-1*, де *V* - кількість вершин у графі):
   a. 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).
   b. Додамо всі знайдені ребра в множину *T*.
3. Отримана множина ребер *T* є мінімальним остовим деревом вхідного графа.

	1	2	3	4	5	6	7	8
1	0	0	38	95	0	1	57	0
2	0	0	0	0	79	0	36	19
3	38	0	0	51	0	0	44	0
4	95	0	51	0	0	44	0	0
5	0	79	0	0	0	93	41	48
6	1	0	0	44	93	0	1	0
7	57	36	44	0	41	1	0	0
8	0	19	0	0	48	0	0	0

Рис. 1 – Вхідні дані (1_3.txt)

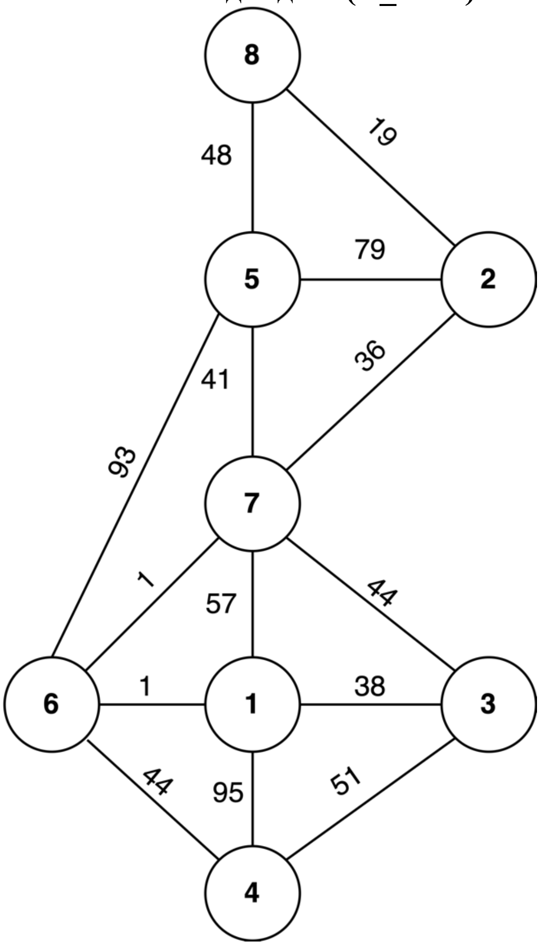
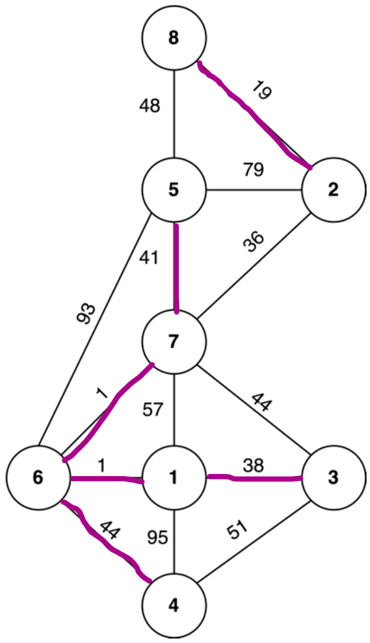


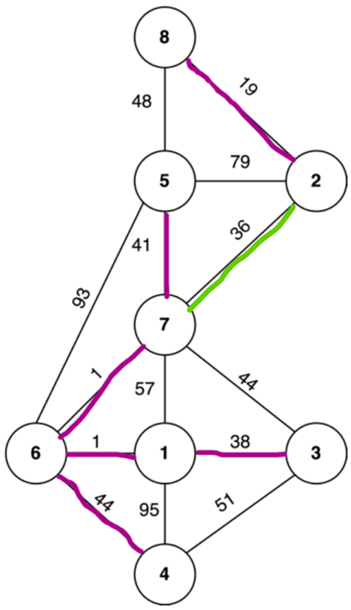
Рис. 2 – Графічне представлення графа

Пошук мінімального остового дерева:

Перебираємо всі вершини і вибираємо для кожної інциденте ребро з мінімальними вагами.
Вершина 1 – ребро [1; 6] з вагою 1,

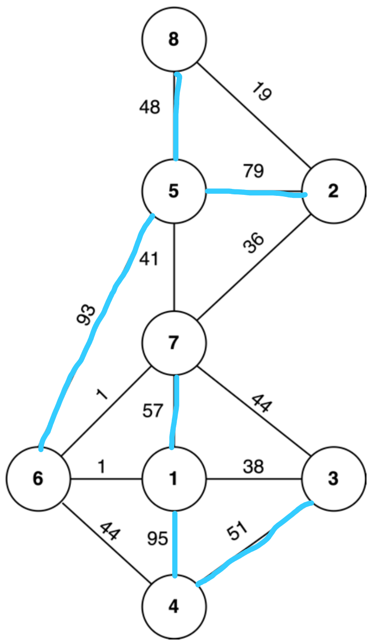
Вершина 2 – ребро [2; 8] з вагою 19,
Вершина 3 – [1; 3] – з вагою 38,
Вершина 4 – [4; 6] – з вагою 44,
Вершина 5 – [5; 7] – з вагою 41,
Вершина 6 – [1; 6] або [6; 7] – з вагою 1,
Вершина 7 – [6; 7] – з вагою 1,
Вершина 8 – [2; 8] – з вагою 19.
Об’єднуємо кожну компоненту зв’язності і получаємо два дерева [{2, 8}, {4, 6, 1, 3, 7, 5}].
Повторяємо алгоритм доти доки не отримаємо мінімальне остове дерево. Якщо ребра повторюються або мають обидві вершини, що вже включені в дерево, то їх відкидаємо.
В результаті отримуємо мінімальне остове дерево: {8, 2, 7, 5, 6, 1, 3, 4}

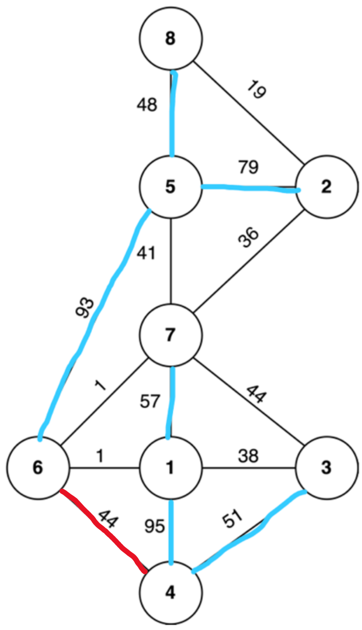




Сума ваг дерева = 44+1+38+1+41+36+19 = 180

Пошук максимального остового дерева:
Перебираємо всі вершини і вибираємо для кожної інциденте ребро з максимальними вагами.
Повторяємо алгоритм доти доки не отримаємо максимальне остове дерево. Якщо ребра повторюються або мають обидві вершини, що вже включені в дерево, то їх відкидаємо.
В результаті отримуємо максимальне остове дерево: {8, 2, 7, 5, 6, 1, 3, 4}





Сума ваг дерева = 44+95+51+57+93+79+48 = 467

Програмна реалізація (python):

```
def boruvkamin(self):
        component_size = []
        mst_weight = 0

        minimum_weight_edge = [-1] * self.m_v

        for node in range(self.m_v):
            self.m_component.update({node: node})
            component_size.append(1)

        num_of_components = self.m_v

        print("---------Min Tree------------")
        while num_of_components > 1:
            for i in range(len(self.m_edges)):

                u = self.m_edges[i][0]
                v = self.m_edges[i][1]
                w = self.m_edges[i][2]

                u_component = self.m_component[u]
                v_component = self.m_component[v]

                if u_component != v_component:
                    if minimum_weight_edge[u_component] == -1 or \
                            minimum_weight_edge[u_component][2] > w:
                        minimum_weight_edge[u_component] = [u, v, w]
                    if minimum_weight_edge[v_component] == -1 or \
                            minimum_weight_edge[v_component][2] > w:
                        minimum_weight_edge[v_component] = [u, v, w]

            for node in range(self.m_v):
```

```python
            if minimum_weight_edge[node] != -1:
                u = minimum_weight_edge[node][0]
                v = minimum_weight_edge[node][1]
                w = minimum_weight_edge[node][2]

                u_component = self.m_component[u]
                v_component = self.m_component[v]

                if u_component != v_component:
                    mst_weight += w
                    self.union(component_size, u_component, v_component)
                    print("Added edge [" + str(u) + " - "
                          + str(v) + "]\n"
                          + "Added weight: " + str(w) + "\n")
                    num_of_components -= 1

        minimum_weight_edge = [-1] * self.m_v
    print("----------------------------------")
    print("The total weight of the minimal spanning tree is: " + str(mst_weight))
```

Результати виконання програми:

```
>>>
============================== RESTART: /Users/verikpaster/Documents/NULP/4.2/САПР/1/code.py ==============================
---------Min Tree------------
{0: 5, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7}
Added edge [0 - 5]
Added weight: 1

{0: 5, 1: 7, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7}
Added edge [1 - 7]
Added weight: 19

{0: 5, 1: 7, 2: 5, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7}
Added edge [0 - 2]
Added weight: 38

{0: 5, 1: 7, 2: 5, 3: 5, 4: 4, 5: 5, 6: 6, 7: 7}
Added edge [3 - 5]
Added weight: 44

{0: 5, 1: 7, 2: 5, 3: 5, 4: 6, 5: 5, 6: 6, 7: 7}
Added edge [4 - 6]
Added weight: 41

{0: 5, 1: 7, 2: 5, 3: 5, 4: 5, 5: 5, 6: 5, 7: 7}
Added edge [5 - 6]
Added weight: 1

{0: 5, 1: 5, 2: 5, 3: 5, 4: 5, 5: 5, 6: 5, 7: 5}
Added edge [1 - 6]
Added weight: 36

----------------------------------
The total weight of the minimal spanning tree is: 180
---------Max Tree------------
{0: 3, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7}
Added edge [0 - 3]
Added weight: 95

{0: 3, 1: 4, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7}
Added edge [1 - 4]
Added weight: 79

{0: 3, 1: 4, 2: 3, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7}
Added edge [2 - 3]
Added weight: 51

{0: 3, 1: 4, 2: 3, 3: 3, 4: 4, 5: 4, 6: 6, 7: 7}
Added edge [4 - 5]
Added weight: 93

{0: 3, 1: 4, 2: 3, 3: 3, 4: 4, 5: 4, 6: 3, 7: 7}
Added edge [0 - 6]
Added weight: 57

{0: 3, 1: 4, 2: 3, 3: 3, 4: 4, 5: 4, 6: 3, 7: 4}
Added edge [4 - 7]
Added weight: 48

{0: 4, 1: 4, 2: 4, 3: 4, 4: 4, 5: 4, 6: 4, 7: 4}
Added edge [3 - 5]
Added weight: 44

----------------------------------
The total weight of the maximal spanning tree is: 467
>>>
```

Висновок:

В ході даної лабораторної роботи я ознайомилась з алгоритмами побудови остового дерева. Провела ручні обчислення та реалізувала алгоритм Борувки згідно індивідуального завдання.