Самойленко Марта

Український Католицький Університет

Лабораторна робота 1

Дискретна математика

Лютий, 2023

Суть та мета завдання

Дослідження та реалізація алгоритмів. На вибір дано алгоритми Прима та Краскала, Белмана-Форда та Флойда-Воршала.

Опис алгоритму Краскала

Вхідними даними є зважений неорієнтований граф. Суть алгоритму полягає в тому, що на початку сортуються ребра графа за вагою, а потім алгоритм жадібно обирає наступне ребро так, щоб утворилося дерево мінімальної ваги.

Код складається з таких функцій:

```
def find_set(vertex_list, vertex):
```

```
def union(vertex_list, deg_list, vertex_u, vertex_v):
```

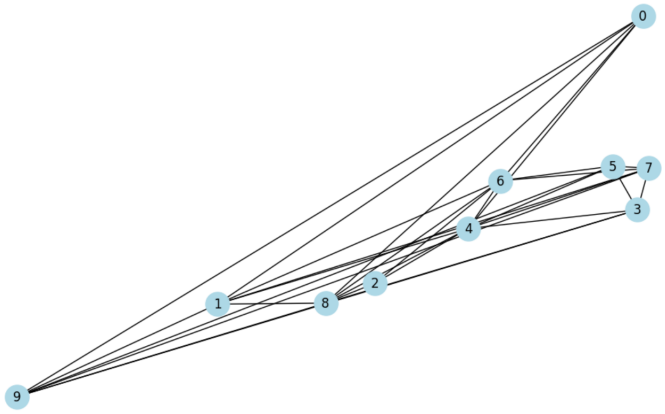
```
def kruskal_algorithm(G):
```

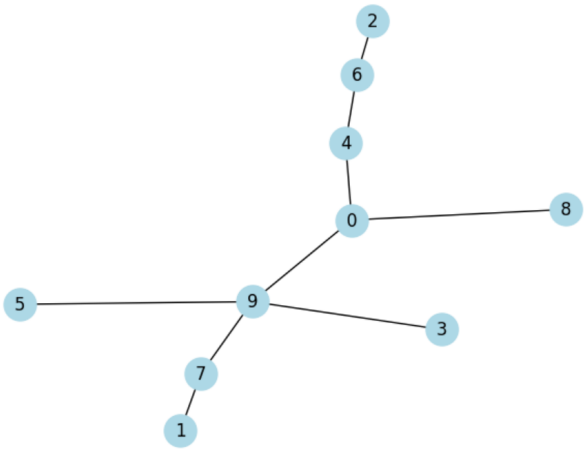
2

Програмний код:

```
def find_set(vertex_list, vertex):
    if vertex_list[vertex] == vertex:
        return vertex
    return find_set(vertex_list, vertex_list[vertex])

def union(vertex_list, deg_list, vertex_u, vertex_v):
    find_u = find_set(vertex_list, vertex_u)
    find_v = find_set(vertex_list, vertex_v)
    if deg_list[find_u] < deg_list[find_v]:
        vertex_list[find_u] = find_v
    elif deg_list[find_v] < deg_list[find_u]:
        vertex_list[find_v] = find_u
    else:
        vertex_list[find_v] = find_u
        deg_list[find_u] += 1

    return vertex_list, deg_list


def kruskal_algorithm(G):
    result = []
    vertex_list = []
    deg_list = []
    for vertex in G.nodes():
        vertex_list.append(vertex)
        deg_list.append(0)

    graph_start_list = []

    for edge in G.edges():
        graph_start_list.append([edge[0], edge[1], G.get_edge_data(edge[0], edge[1])['weight']])

    graph_list = list(sorted(graph_start_list, key = lambda x : x[2]))

    for edge in graph_list:
        u = edge[0]
        v = edge[1]
        if find_set(vertex_list, u) != find_set(vertex_list ,v):
            result.append([u, v, edge[2]])
            vertex_list, deg_list = union(vertex_list, deg_list, u, v)

    return result
```

На виході при вхідному графі G:



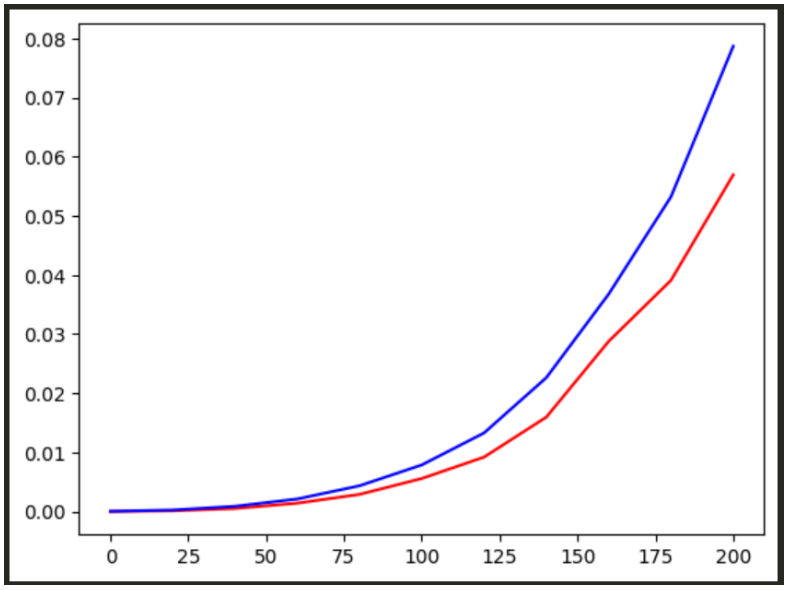


Експеримент

В експерименті я порівняла час виконання написаного алгоритму Краскала та вбудованого на різній кількості вершин (x - к-сть вершин, y - час у секундах)

Написаний алгоритм

Вбудований алгоритм



Висновок

Можемо побачити, що вбудований алгоритм працює трхи швидше. Як можна побачити, Графік наведено для кільеості вершин від 0 до 200 mod 20 (усі що діляться на 20)

Як на мене, алгоритм може працювати трохи сповільнено через використання рекурсії у (find_set()). На початку я також використовувала словники для зберігання даних про граф, проте потім змінила на списки. Час виконання тоді значно зменшився. Також є підстава вважати, що алгоритм Прима буде працювати швидше, проте менш точно на великій кількості вершин, адже він не сортує ребра.

Опис алгоритму Белмана-Форда:

Вхідними даними є зважений орієнтований граф без від'ємних циклів. Суть алгоритму полягає в тому, щоб знайти найкоротший шлях від конкретної вершини до всіх інших. Алгоритм є також жадібним.

Код складається з тиких функцій:

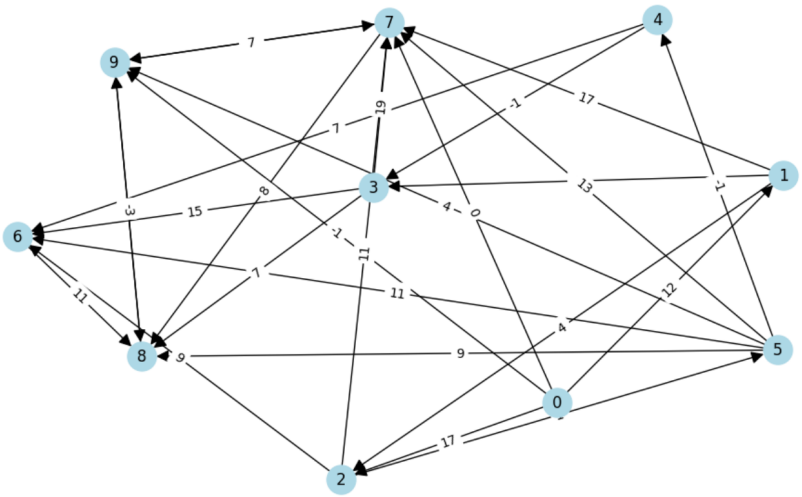
```
def relax(vertex_u, vertex_v, weight, distance_dict, pi_dict):
```

```
def bellman_ford(G, s):
```

Результати алгоритму мають вигляд:

```
Distance to 0 = 0
Distance to 1 = -5
Distance to 3 = -6
Distance to 4 = -8
Distance to 5 = -5
Distance to 6 = -12
Distance to 7 = -2
Distance to 8 = -7
Distance to 9 = -1
```

При вхідних даних:



Decision Tree Classifier

Загальний принцип побудови дерева:

Визначити для кожної вершини, за яким threshold та index варту аналізувати датасет, який передається. Тобто для тренування використовується датасетз даними класами, ми намагаємося поділити цей датасет так, щоб зробити ці частини як умога “чистішими”

Ми це зберігаємо і потім тестуємо його на датасеті вже без даних класів, потім порівнюючи точність вихідних даних зі справжніми класами.

```
[1, 2, 2, 0, 1, 2, 2, 2, 2, 2, 1, 0, 0, 2, 1, 1, 2, 0, 0, 2, 0, 1, 1, 0, 1, 0, 1, 2, 1, 2]
[2, 2, 2, 0, 1, 2, 2, 2, 2, 2, 0, 0, 0, 2, 1, 2, 2, 0, 0, 2, 0, 1, 1, 0, 1, 0, 1, 2, 1, 2]
0.9
```

```
[2, 2, 2, 0, 0, 0, 1, 1, 2, 2, 1, 2, 0, 1, 2, 0, 2, 1, 0, 1, 0, 2, 0, 0, 2, 2, 0, 0, 1, 0]
[2, 2, 2, 0, 0, 0, 1, 1, 2, 2, 1, 2, 0, 1, 2, 0, 2, 1, 0, 0, 0, 2, 0, 0, 2, 2, 0, 0, 1, 0]
0.9666666666666667
```

```
[2, 0, 0, 1, 0, 2, 0, 0, 1, 0, 1, 1, 2, 1, 0, 1, 0, 2, 0, 0, 2, 2, 0, 1, 0, 2, 0, 2, 0, 2]
[2, 0, 0, 1, 0, 2, 0, 0, 1, 0, 1, 1, 2, 1, 0, 1, 0, 2, 0, 0, 2, 2, 0, 1, 0, 2, 0, 2, 0, 2]
1.0
```

Для роботи з features зручно використовувати dataFrame.

```
df = pd.DataFrame(features, columns=[x for x in range(features_size)])
```

```
left_node_classes = df_copy['class'][:left_i].values.tolist()
right_node_classes = df_copy['class'][left_i:].values.tolist()
```