Лабораторна робона №1

Звіт

Кушнір Олександр

21st February, 2022

План

1. Опис постановки задачі та експерименту
2. Специфікація комп’ютера
3. Програмний код алгоритмів
    a. Алгоритм Крускала
    b. Алгоритм Прима
4. Програмний код проведення експериментів
5. Графіки експериментів
6. Загальний підсумок

Опис постановки задачі експерименту

В цьому експерименті наша команда порівняла ефективність алгоритму Краскала і Прима. Критерієм порівняння був час, за який алгоритм створює каркас графу розміром 10, 20, 50, 100, 200, 250 та різною повнотою 0.1, 0.5, 1. Результати експерименту представлені у висновку та графіках.

Специфікація комп’ютера

Усі експерименти проводилися на ноутбуці Macbook Air 2020 з процесором M1.

Кількість ядер: 8

Тактова частота: 3,20 ГГц

Пам’ять: 8 ГБ

ОС: MacOS

Програмний код алгоритмів

```python
import itertools

def kruskal(edges):

    edges = list(edges)
    edges.sort(key=lambda x: x[-1])
    edges = [(edge[0], edge[1]) for edge in edges]
    node_sets = [{i} for i in set(itertools.chain(*edges))]
    kruskal_edges = []

    while len(node_sets) != 1:

        for edge in edges:
            edge_sets = []
            for node_set in node_sets:
                if ({edge[0]}.issubset(node_set) and not {edge[1]}.issubset(node_set)) or ({edge[1]}.issubset(node_set) and not {edge[0]}.issubset(node_set)):
                    edge_sets.append(node_set)
                elif {edge[0]}.issubset(node_set) and {edge[1]}.issubset(node_set):
                    break

            if len(edge_sets) == 2:
                node_sets.append(edge_sets[0].union(edge_sets[1]))
                del node_sets[node_sets.index(edge_sets[0])]
                del node_sets[node_sets.index(edge_sets[1])]
                kruskal_edges.append((edge))
                del edges[edges.index(edge)]
                break

    return kruskal_edges
```

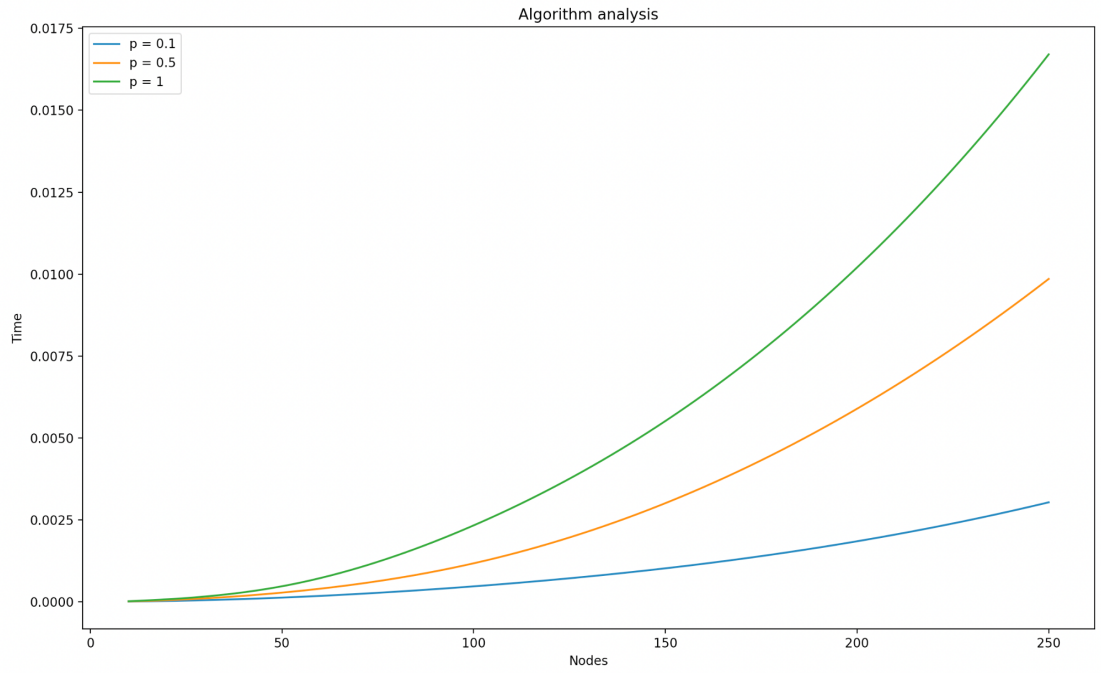
```python
import itertools
import random

def primo(edges):

    edges = list(edges)
    edges.sort(key=lambda x: x[-1])
    edges = [(edge[0], edge[1]) for edge in edges]
    picked_nodes = {random.choice(list(itertools.chain(*edges)))}
    primo_edges = []
    node_set = len(set(itertools.chain(*edges)))

    while len(picked_nodes) != node_set:
        for edge in edges:
            if (edge[0] in picked_nodes and not edge[1] in picked_nodes):
                primo_edges.append(edge)
                picked_nodes.add(edge[1])
                del edges[edges.index(edge)]
                break

            elif (edge[1] in picked_nodes and not edge[0] in picked_nodes):
                primo_edges.append(edge)
                picked_nodes.add(edge[0])
                del edges[edges.index(edge)]
                break

    return primo_edges
```

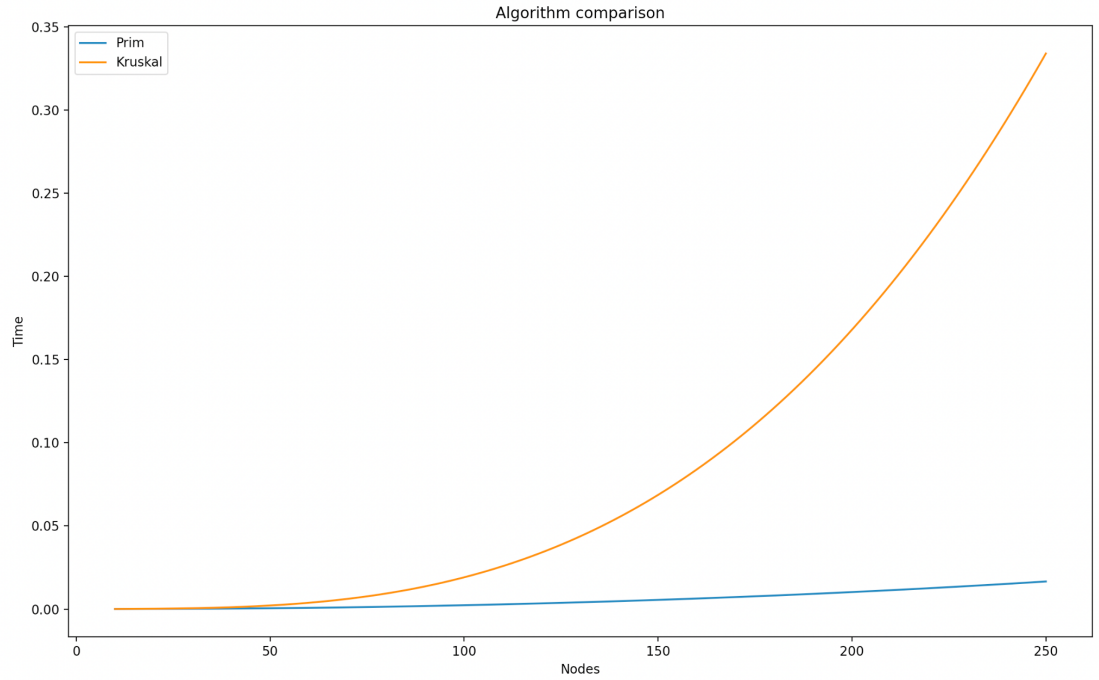
Програмний код експериментів

```
def analyse_alg(nodes, alg):
    #This function returns a graph with time/nodes axis of a given algorithm
    #for different values of p(fullness of graph)
    alg_values = []

    for prob in (0.1, 0.5, 1):
        for node in nodes:
            alg_values.append(measure_time(alg, node, prob))

    x = np.array(nodes)
    y_p01 = np.array(alg_values[:len(nodes)])
    y_p05 = np.array(alg_values[len(nodes):2*len(nodes)])
    y_p1 = np.array(alg_values[2*len(nodes):3*len(nodes)])

    cubic_interploation_model_p01 = interp1d(x, y_p01, kind = "cubic")
    cubic_interploation_model_p05 = interp1d(x, y_p05, kind = "cubic")
    cubic_interploation_model_p1 = interp1d(x, y_p1, kind = "cubic")

    X_=np.linspace(x.min(), x.max(), 500)
    Y_p01=cubic_interploation_model_p01(X_)
    Y_p05=cubic_interploation_model_p05(X_)
    Y_p1=cubic_interploation_model_p1(X_)

    plt.plot(X_, Y_p01)
    plt.plot(X_, Y_p05)
    plt.plot(X_, Y_p1)

    plt.legend(["p = 0.1", "p = 0.5", "p = 1"])
    plt.xlabel("Nodes")
    plt.ylabel("Time")
    plt.title("Algorithm analysis")
    plt.show()
```

```
def compare_algorirtms(prob, nodes_list):
    #Creates a graphic with time/nodes for Prim's and Kruskal's algorithms.
    prim_values = []
    krus_values = []
    for node in nodes_list:
        prim_values.append(measure_time(primo, node, prob))
        krus_values.append(measure_time(kruskal, node, prob))

    x = np.array(nodes_list)
    y_prim = np.array(prim_values)
    y_krus = np.array(krus_values)

    cubic_interploation_model_prim = interp1d(x, y_prim, kind = "cubic")
    cubic_interploation_model_krus = interp1d(x, y_krus, kind = "cubic")

    X_=np.linspace(x.min(), x.max(), 500)
    Y_prim=cubic_interploation_model_prim(X_)
    Y_krus=cubic_interploation_model_krus(X_)

    plt.plot(X_, Y_prim)
    plt.plot(X_, Y_krus)
    plt.legend(["Prim", "Kruskal"])
    plt.xlabel("Nodes")
    plt.ylabel("Time")
    plt.title("Algorithm comparison")
    plt.show()
```

Графіки експериментів





Підсумок

В цьому дослідженні наша команда провела два експерименти. В першому ми подивилися як змінюється час виконання при різних значеннях повноти графу , результати представлені на графіку.

У другому експерименті ми порівняли два алгоритми на повних графах з 10, 20, 50, 100, 200, 250 вершинами. За результатами експерименту алгоритм Прима трохи швидший ніж Краскала на кількості ребер < 100, далі алгоритм Прима майже не змінює час виконання, в той час як Краскала для к-сті ребер = 250 приблизно в 30р. повільніший.