Technical University of Moldova
Inginerical department S.A.

Report

№: 3

OOP

Subject: Greedy algorithms

Author: Terman Emil FAF161
Prof: I. Costiuc

Chisinau 2017

Objectives:

- study Greedy algorithms;
- implementation of Greedy algorithms: Kruskal and Primal MST

Greedy algorithm:

Greedy algorithms are used in optimization problems , as to find the shortest path in a graph , etc. Greedy algorithms build the solution piece by piece, and such an example is computing the spanning tree.

These types of algorithms, have the following steps:

1. a candidate set, from which a solution is created;
2. a selection function, which chooses the best candidate to be added to the solution;
3. an objective function, which assigns a value to the solution, or a partial solution;
4. a solution function, which will indicate when the solution is completely computed;

1 Kruskal algorithm

1.1 Complexity

Small number of vertexes: $O(m \log_2 n)$
Large number of vertexes: $O(n^2 \log_2 n)$
Where m is the number of edges

1.2 Python implementation

Pyhon Code 1: Kruskal

```python
# A tree is composed of a list of tuples of size 3, where the last element
# in the tuple represents the length: ['A', 'B', 20]
def MinSpanTreeKruskal(tree):
    tree = sorted(tree, key=lambda i: i[2])

    nodeNames = list(set([i[0] for i in tree] + [i[1] for i in tree]))
    nodeGroups = [[i] for i in nodeNames]

    result = []
    for start, end, length in tree:
        startGroup = next(group for group in nodeGroups if start in group)
        endGroup = next(group for group in nodeGroups if end in group)

        if startGroup != endGroup:
            newGroup = startGroup + endGroup
            startGroup.extend(endGroup)
            nodeGroups.remove(endGroup)
            result.append((start, end, length))

        if len(nodeGroups) == 1:
            break

    return result
```

2 Prim's algorithm

2.1 Complexity

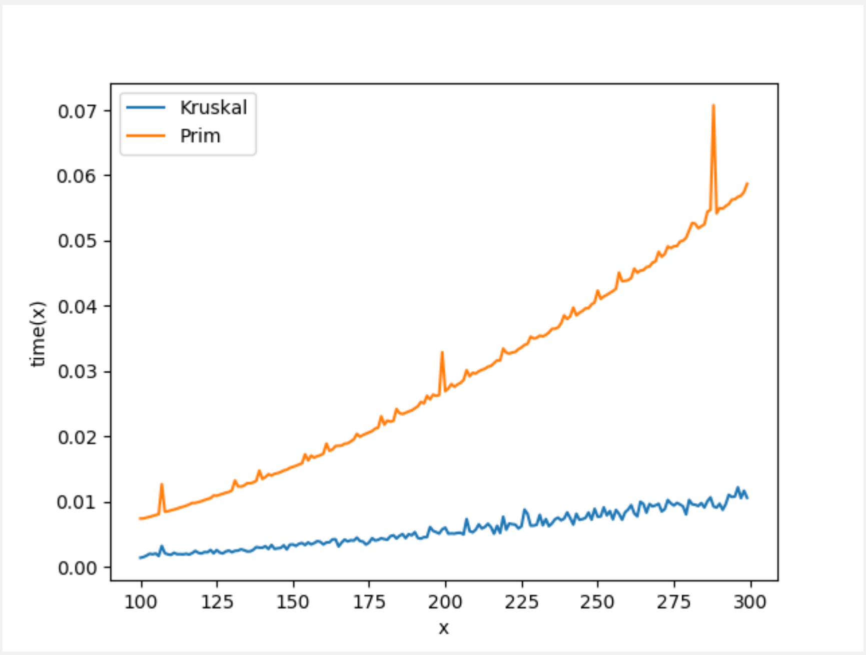
Small number of vertexes: $O(n^2)$
Large number of vertexes: $O(n^2)$

2.2 Python implementation

Pyhon Code 2: Prim

```python
import math

def MinSpanTreePrim(tree):
    nodeNames = list(set([i[0] for i in tree] + [i[1] for i in tree]))
    heapMap = dict((nodeName, math.inf) for nodeName in nodeNames)
    heapMap[tree[0][0]] = 0

    treeDict = dict()
    for start, end, length in tree:
        setKey = frozenset(sorted([start, end]))
        if setKey in treeDict:
            if length < treeDict[setKey]:
                treeDict[setKey] = length
        else:
            treeDict[setKey] = length

    vertsAndEdges = dict()
    while len(heapMap) != 0:
        minElement = min(heapMap, key=heapMap.get)
        del heapMap[minElement]

        for node in heapMap.keys():
            setKey = frozenset(sorted([node, minElement]))
            if setKey in treeDict and treeDict[setKey] < heapMap[node]:
                heapMap[node] = treeDict[setKey]
                vertsAndEdges[node] = [minElement, node, treeDict[setKey]]

    return sorted(list(vertsAndEdges.values()), key=lambda x: x[2])
```

3 Time diagram



Img 1: Kruskal and Prim Time Diagram

4 Conclusion

In this laboratory work, two Greedy algorithms for findning the minimal spaning tree were implemented: Kruskal and Prim's. From the time diagram, it can be clearly seen that Kruskal is much faster and it is also simplier to implement. But for large numbers, Prim's algorithm is said to work faster of graphs.