Laboratory work nr. 3
Greedy method and dynamic programming

Bîrcu Maxim

December 9, 2015

Student: Bîrcu Maxim
Instructor: Cojanu Irina

Work purpose

1. **Studying of the greedy technique**
2. **Study of Dynamic programming**
3. **Implementation of Kruskal algorithm**
4. **Implementation of the Floyd algorithm**
5. **Comparison of the algorithms (time complexity diagram/table/empirical)**

Results

I've studied all greedy algorithms that I've implemented applying them to different graphs with different number of vertices, and computed the execution time for each of them to place all this data on a chart to see the main differences of those algorithms.

1 Kruskal,Kruskal-Prim,Floyd and Djikstra algorithm implementation

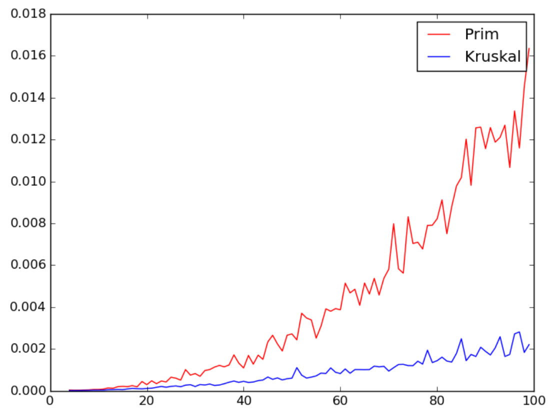
Source code in python

```python
import sys
INF = sys.maxint
#KRUSKCAL
###################################################################################
def kruskal(graph):
    #connected verticies
    connected = []
    #selected edges
    selected = []
    i = 0
    while (len(connected) != graph.n) and (i < len(graph.data)):
        e = graph.data[i]
        a = contains(connected,e.a) == -1
        b = contains(connected,e.b) == -1
        if (a != b) or len(connected) == 0:
            addIfNotExists(connected,e.a)
            addIfNotExists(connected,e.b)
            selected.append(e)
        i += 1
    return selected
###################################################################################


#KRUSKAL PRIM
###################################################################################
def prim(graph):
    #add 1 as connected !
    connected = [1]
    #selected edges
    selected = []
    edges = graph.data
    while len(connected) != graph.n:
        for e in edges:
            a = contains(connected,e.a)
            b = contains(connected,e.b)
            if ((a == -1 and b != -1)) or (b == -1 and a != -1):
```

```
                if a == -1:
                    connected.append(e.a)
                else:
                    connected.append(e.b)
                selected.append(e)
                edges.remove(e)
                break

    return selected
###############################################################################


#FLOYD WARHSALL
###############################################################################
def floydWarshall(matrix, numberOfVertices):
    distances = {0: matrix}
    for k in range(1,numberOfVertices+1):
        distances[k] = {}
        for i in range(1,numberOfVertices+1):
            for j in range(1,numberOfVertices+1):
                distances[k][i,j] = min(distances[k-1][i,j],distances[k-1][i,k
                    ] + distances[k-1][k,j])
    return distances[numberOfVertices]
###############################################################################

#FLOYD WARHSALL
###############################################################################
def dijkstra(matrix,n):
    distances = {2: matrix}
    for i in range(2,n+1):
        distances[i] = matrix[1,i]
    candidates = [i+2 for i in range(n-1)]
    while len(candidates) != 0 :
        current = 0
        min = INF
        for i in candidates:
            if min > distances[i]:
                current = i
                min = distances[i]
        for i in range(2,n+1):
            if (distances[i] > distances[current] + matrix[current,i]) :
                distances[i] = distances[current] + matrix[current,i]

        candidates.remove(current)
    return distances[n]
###############################################################################
```

2 Kruskal algorithm vs Kruskal-Prim

For better understanding of the results and to compare the algorithms easier I've computed the execution time for different graphs with different number of vertices and placed all this data on some graphs.

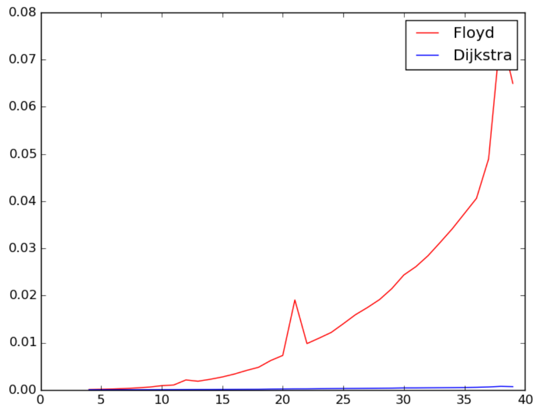


Vertices	Kruskal	Kruskal-Prim
4	1.3828277587890625e-05	4.100799560546875e-05
5	1.0013580322265625e-05	1.4066696166992188e-05
6	2.193450927734375e-05	4.601478576660156e-05
7	2.193450927734375e-05	4.315376281738281e-05
8	2.193450927734375e-05	6.103515625e-05

From the graph above we can notice that Kruskal algorithm works quite fast in comparison with the Kruskal-Prime algorithm.

3 Floyd algorithm vs Dijkstra

For better understanding of the results and to compare the algorithms easier I've computed the execution time for different graphs with different number of vertices and placed all this data on some graphs.



Vertices	Floyd	Dijkstra
4	1.2874603271484375e-05	3.886222839355469e-0
5	2.09808349609375e-05	2.09808349609375e-05
6	1.5020370483398438e-05	1.1920928955078125e-05
7	2.9087066650390625e-05	4.506111145019531e-05
8	2.5987625122070312e-05	6.890296936035156e-05

From the graph above we can notice that Dijkstra algorithm works quite fast in comparison with the Floyd algorithm.

4 Conclusion:

After all research and algorithms comparison I've observed that Kruskal and Dijkstra a quite fast algorithms for computing the shortest path, and also Kruskal-Prime and Floyd are a little bit slower, but they bring much information.