LULEÅ UNIVERSITY OF TECHNOLOGY

Algorithms and Data Structures - Lab 3

Course: D0012E

Marcus Lund (amuulo-4)

Edvin Åkerfeldt (edvker-4)

Samuel Karlsson (samkar-4)

January 7, 2016

First submission.

Abstract

In this report the runtime of Dijkstra's algorithm is analysed with different sizes of the graph and different number of connections on on each node. The trend shows that a bigger graph results in a longer runtime.

Generally the tests we ran were very case sensitive. The runtime differs a lot between worst and best case. The time we used was an average of multiple testes to minimize the effect from best and worst case tests.

Contents

1 Introduction

In this report a lab on Dijkstra's algorithm is presented. The lab was done in the course "algorithms and computer structure". The goal for the lab was to see how the runtime is effected on different sizes of graphs and different number of connections to each node.

1.1 Language

We chose to use Java as our programming language of choice over other languages, because it is a language the we are comfortable to work with. Java is also one of the better optimized languages.

1.2 Implementation

Implementing Dijkstra's algorithm is a challenge because the graphs representation have a huge influence on Dijkstra's algorithms runtime.

We chose to use a class representation of the graph. The class implementaion is in some regard straight forward and logical but has a clear disadvantages when it comes to memory consumption. When the graph grows its memory usage grows very fast.

The hardest part of implementating the Dijkstra's algorithm is to find a way to add the new nods to the added ones without adding the same node twice. Every node can be in one out of three stages: not known, known but don't added or known and added. The nodes in the third stage is easily stored a d-array Heap. The nodes in the second stage can be stored in many ways. It is preferable to store it in such a way that it is easy to add new nodes and find the minimum, this can be achived with a min heap for example.

2 Test procedure

The tests were ran several times with different sizes of nodes, d and edges. The numbers were doubled in hope of making nice looking graphs.

The tests started at 20000 nodes, d at 4 and minimum edges equal to the number of nodes. When increasing the three different sizes were doubled individually and 80 different graphs were generated to get an average for the specific test (nodes, d and edges relation).

The tests that were run are listed in 3.1.

3 Result

When increasing the value d we see an increment in time. Also with more edges and more nodes we get a bigger graph and thus it takes a longer time to calculate.

In graph 3.1 there is 4 different colored bars where blue is $d = 4$, orange $d = 8$, grey $d = 16$ and yellow is $d = 31$. The bottom axis describes '$nodes$ & $edges$' and the vertical axis describes

Table 3.1: Testing sets

Nodes	Edges	D	Average time
20000	20000	4	644
20000	20000	8	673
20000	20000	16	7609
20000	20000	32	8975
20000	40000	4	2030
20000	40000	8	1800
20000	40000	16	9188
20000	40000	32	9444
20000	80000	4	2059
20000	80000	8	2497
20000	80000	16	10401
20000	80000	32	10001
40000	40000	4	2478
40000	40000	8	6324
40000	40000	16	2232
40000	40000	32	2663
40000	80000	4	2478
40000	80000	8	6324
40000	80000	16	8673
40000	80000	32	8528

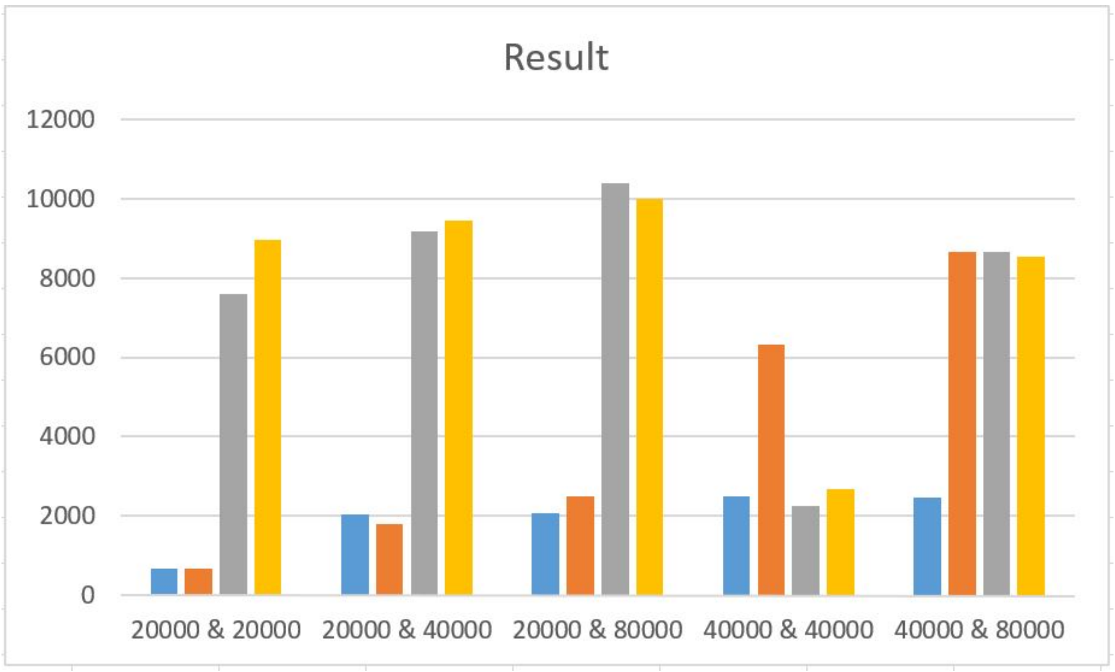

Figure 3.1: Result

the average time. We can see from the graph that d has an impact of the average time when it is increased. But the number of edges has an even greater impact.

4 Discussion

We had some problems with the implementation of the algorithm. It was in the representation of the different stages we got a problem. We got errors because all the nodes did not connect to the same tree. To be sure that the graph was a single connected graph we manually designed a start of the graph such that all nodes were connected.

The test result is really inconsistent. Since we are building a random graph and search the shortest path between to random nods, it makes the result differ a lot. The time it take when the two nods are close to each outer is fast no mater how huge the graph is. To minimize this difference we ran the test multiple times and took an average of the run times for each test.

In theory we thought that the number of edges was to be the biggest factor of impact closely followed by the d value. If d is large we get more possible ways to take from each node and potentially makes the time increase because the algorithm could take the wrong path from each node.

The trend in the result tells that it is the number of edges and number of nodes that provide the big different in the runtime. If the graph holds more nodes there is more incorrect nodes and the operation that is adding a new nod to the path takes more time.

The size of d does not affect the runtime directly. Though the size of d has a big impact on the probability for a best case runtime. On the other hand, a big d will more often then not result in worse preformance in average case due to the higher chance that the method will have to check more nodes and paths.