D0012E Laboration 1

Grupp 27
Magda Larsson Andersson, magany-7@student.ltu.se
Andreas Form, forane-9@student.ltu.se
Marcus Asplund, marasp-9@student.ltu.se

November 2020

1 Laboration 1 del 2

1.1 Introduktion

Labbens syfte var att jämföra olika algoritmer för samma problem. Problemet var att räkna ut ett minimalt uppspänt träd med prims algoritm. Två olika data typer användes för detta Linked list och matrix. Vi gjorde också två till varianter av vadera datatyper med "heap". Heap är en specialiserad datastruktur som är gjord för träd.

1.2 Adjacent list no heap

För att utföra Prims algoritm med adjacent lista så utgick vi från en startnod som vi la till i en lista som håller koll på noder som blivit besökta. Sedan kollade vi på alla noder som fanns i denna lista för att undersöka deras grannar och hitta den minsta vikten, när vi funnit den så la vi till grannen i listan för besökta noder. Algoritmen var klar när alla noder blivit besökta. Vår uträknade kostnaden för algoritmen blev $O(E * N)$ vilket är en nära kurva jämnfört med graf 1

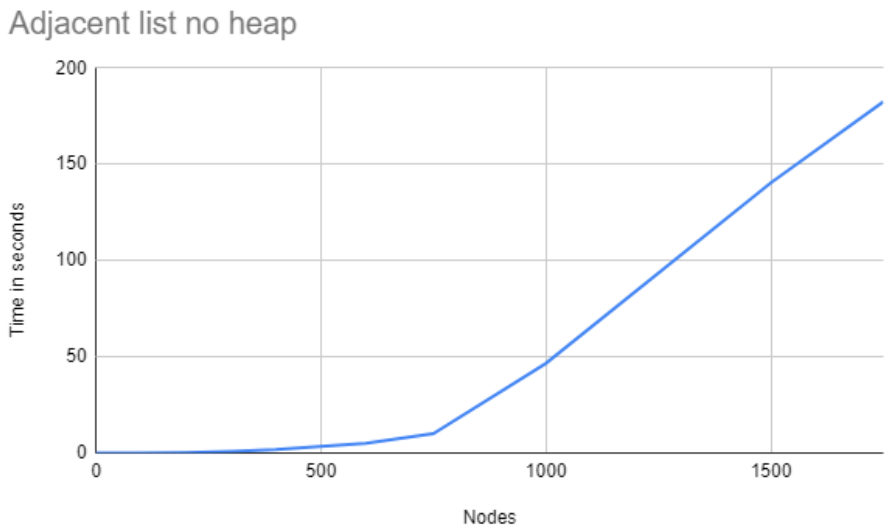

Figure 1: $T(n)$ Tagen tid beroende på noder

1.3 Adjacent list heap

När vi utförde Prims algoritm med adjecent lista och med minimum heap så fyllde vi på en heap lista som höll våra nycklar(vikter) med infinity värden förutom första noden som fick värde 0. Sedan tog vi ut noden som hade den minsta nyckeln från listan och kollade på nodens grannar för att hitta minsta vikten. Kostnaden för för listan med heap blev $O(E * log(N))$ vilket är en marginell skillnad med utan heap. Om man kollar på graf 2 så stämmer kostnaden relativt överns med mäta tiden. Man kan se att kurvan planar ut i slutet. Något som märks är att kruvan är ojämn vilket är pågrund av att antalet kanter var framslumpade och påverkade grafens kurva mycket mer än i graf 1

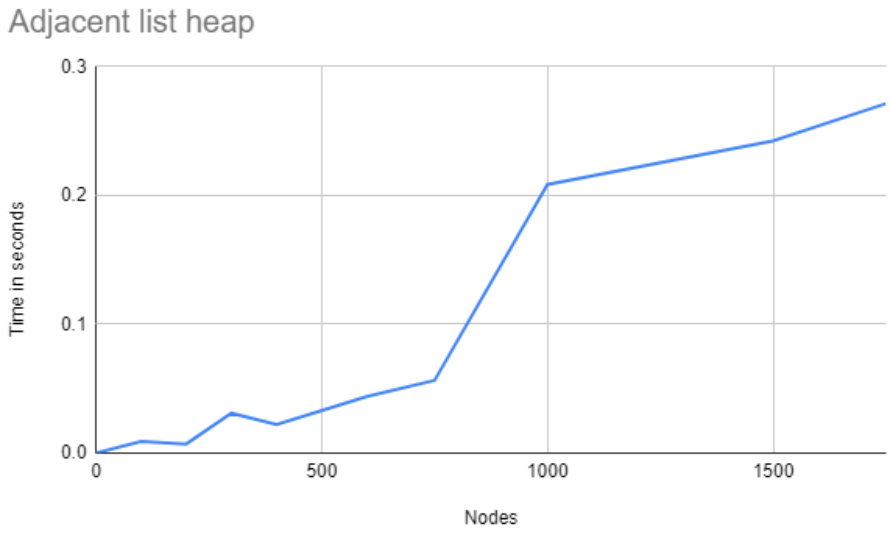

Figure 2: $T(n)$ Tagen tid beroende på noder

1.4 Matrix no heap

För att utföra Prims algoritm med adjacent matrix så utgick vi från en startnod. Med starnoden kollade vi på alla dess vägar att gå och kollade på alla som var över 0 och sparade den minsta som sedan läggs till som visited. Sedan kollar vi igenom från den noden på alla vägar den har minus startnoden som är visited och fortsätter tills alla noder är tagna. Vår uträknade kostnaden för algoritmen blev $O(N^3)$ vilket stämmer ganska bra överenes med vårt resultat då det är en kraftigt exponentiellt kurva. 3

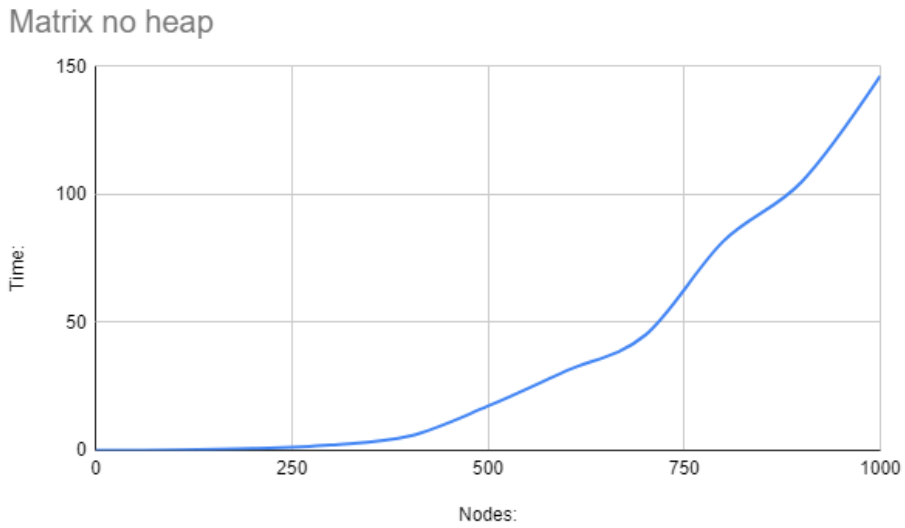

Figure 3: $T(n)$ Tagen tid beroende på noder

1.5 Matrix heap

När vi utförde Prims algoritm med adjecent matrix och med minimum heap så fyllde vi på en heap lista som höll våra nycklar(vikter) med infinity värden förutom första noden som fick värde 0. Sedan tog vi ut noden som hade den minsta nyckeln från listan och kollade på nodens väger som är större än 0. Kostnaden för matrix med heap blev $O(n^2)$ vilket är en marginell skillnad med utan heap. Detta för att gå igenom alla vikter i en matrix är n^2 så även om sorteringen är mindre än så spelar det ingen roll. Om man kollar på graf 4 så stämmer kostnaden relativt överns med mäta tiden. Man kan se att kurvan ökar exponentiellt.

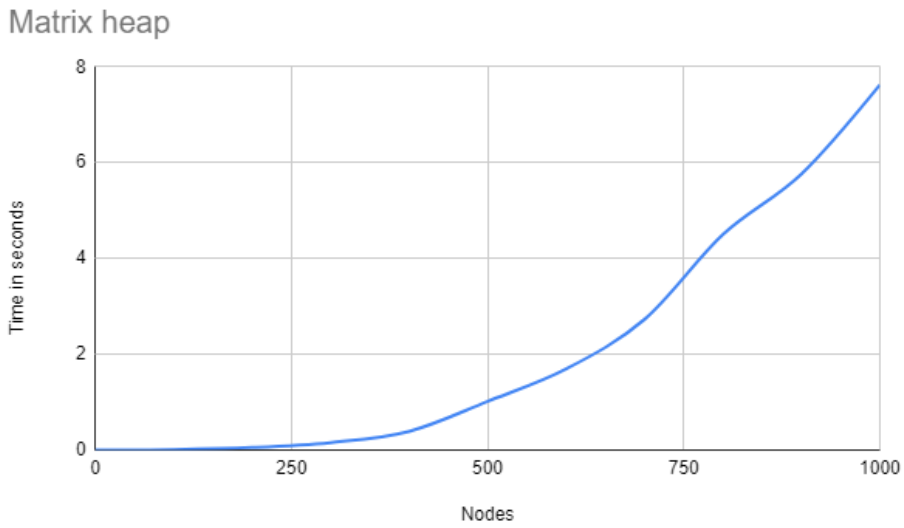

Figure 4: $T(n)$ Tagen tid beroende på noder