Operating Systems Laboratory

Optimized Code Report

Group 31

February 27, 2023

1 Introduction

In the non-optimized case we are using Multisource Dijkstra and computing shortest distance to a node from those $\frac{N}{10}$ nodes in a consumer set. This has a time complexity of $O(Elog(E))$ and is still better than multiple single source Dijkstra(though this gives all the shortest distances from each of $\frac{N}{10}$ nodes alloted to that consumer, we need only the shortest of shortest distances to a node from those $\frac{N}{10}$ nodes), which has a time complexity of $O(NElogE)$. But even when we use multisource dijkstra we are recomputing many distances, which may not have been changed and we try to come up with a strategy to reduce those computations.

2 Observations

The key observations used are:

- The weight/cost between 2 consecutive nodes is 1, which drives us to Breadth First Search.
- When we use BFS and if a node doesn't get it's distance updated, the child will also won't get updated, so no need to traverse the whole graph and some pruning can be done.
- In the website from where data is being taken, it's mentioned that shortest distance between two nodes is 8. Further, when we find shortest distance from set of $\frac{N}{10}$ nodes, the maximum shortest distance is coming as 5. This gives us that in the next iteration the maximum distance can increase by at most 1 and eventually the increase in size will be at most k, for k iterations.

3 Strategy

From observation 1,2 and 3 we can conclude that we can prune the graph after at most k+5 levels till we come to iteration k.The pruning can be done even earlier as probability for increase of 1 is still very less as there are very less nodes with maximum shortest distance of 5 in the first iteration, and new node getting randomly attached to these nodes is less. Mathematically, Let p be the probability of selecting a node which has maximum shortest distance of 5. So probability that new node generated will be attached to all the nodes with shortest distance 5 in worst case is p^x if there are x neighbours of that node, which is still very less. And this effect makes even increase in 1 in every iteration less probable.

All new nodes added by producer in one iteration can be divided into 2 categories : New-source nodes and New-non-source nodes. We first performed **Pruned Multisource Breadth First Search** from the New-source nodes. This pruning happens because we already have previous shortest distances to nodes and we try to update and push only if we find the current distance is shorter than the previous one. In naive multisource BFS, distances are initialised to infinity, here, as they are already initialised to values between 1 to max-shortest-distance, so effective number of operations will be less.

Next we need to consider New-non-source nodes. We can find their distances by just considering their neighbours. After this, it is possible that distance values for some neighbours might have changed due to addition of these New-non-source nodes. Hence, we need to update their values by considering updated value of New-non-source nodes. Further, we need to propagate this effect using **Pruned Multisource Dijkstra**. Even this will be pruned because of the above observation that the probability in increase of distance is small and even if it gets increased it will at most k increase for k iterations.

4 Analysis

It is clear that above algorithm will perform good in average cases. So, we measured time taken by piece of code written in original Dijkstra by each process and average time taken in 10 iterations of optimised code by each process. Here are the results :

NOTE: All the times were measured using std::chrono library.
When we ran **Multisource Dijkstra**(Even in Optimised case we will run Mulitsourse Dijkstra for first iteration)

```
Process#0 Iteration#1: 6270
Process#3 Iteration#1: 6285
Process#4 Iteration#1: 6417
Process#7 Iteration#1: 6412
Process#2 Iteration#1: 6902
Process#8 Iteration#1: 7195
Process#1 Iteration#1: 7773
Process#9 Iteration#1: 7229
Process#5 Iteration#1: 5112
Process#6 Iteration#1: 4897
```

These are the average running times for **first 10 iterations** for each process

```
Avg time by Process#6 in first 10 iterations = 458
Avg time by Process#4 in first 10 iterations = 143
Avg time by Process#1 in first 10 iterations = 273
Avg time by Process#5 in first 10 iterations = 844
Avg time by Process#9 in first 10 iterations = 964
Avg time by Process#2 in first 10 iterations = 301
Avg time by Process#0 in first 10 iterations = 171
Avg time by Process#7 in first 10 iterations = 823
Avg time by Process#8 in first 10 iterations = 598
Avg time by Process#3 in first 10 iterations = 147
```

These are the average running times for **first 20 iterations** for each process

```
Avg time by Process#3 in first 20 iterations = 91
Avg time by Process#0 in first 20 iterations = 106
Avg time by Process#5 in first 20 iterations = 464
Avg time by Process#7 in first 20 iterations = 462
Avg time by Process#4 in first 20 iterations = 82
Avg time by Process#8 in first 20 iterations = 351
Avg time by Process#6 in first 20 iterations = 272
```

When ran for the second time

These are the running times for **Multisource Dijkstra**

```
Process#0 Iteration#1: 6311
Process#3 Iteration#1: 6326
Process#5 Iteration#1: 6209
Process#4 Iteration#1: 7525
Process#8 Iteration#1: 6279
Process#7 Iteration#1: 6412
Process#6 Iteration#1: 4955
Process#1 Iteration#1: 5260
Process#2 Iteration#1: 5748
Process#9 Iteration#1: 5461
```

These are the average running times for **first 10 iterations**

```
Avg time by Process#4 in first 10 iterations = 75
Avg time by Process#8 in first 10 iterations = 535
Avg time by Process#2 in first 10 iterations = 126
Avg time by Process#9 in first 10 iterations = 335
Avg time by Process#0 in first 10 iterations = 92
Avg time by Process#6 in first 10 iterations = 312
Avg time by Process#1 in first 10 iterations = 141
Avg time by Process#7 in first 10 iterations = 505
Avg time by Process#5 in first 10 iterations = 625
Avg time by Process#3 in first 10 iterations = 131
```

These are the average running times for **first 20 iterations**

```
Avg time by Process#9 in first 20 iterations = 209
Avg time by Process#5 in first 20 iterations = 354
Avg time by Process#0 in first 20 iterations = 66
Avg time by Process#8 in first 20 iterations = 302
Avg time by Process#6 in first 20 iterations = 178
Avg time by Process#2 in first 20 iterations = 88
Avg time by Process#7 in first 20 iterations = 282
```