Source codes

1) dijkstra.cpp:

```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> pi;
#define INF 1e9
#define ll long long

vector<vector<pi>>adj(800902);
vector<int>d, p;
int n = 0;
void addEdge(int u, int v, int w) {
  adj[u].push_back({v, w});
  //adj[v].push_back({u,w});
}
ll dijkstra(int s, int dest) {
  d[s] = 0;
  priority_queue<pi, vector<pi>,
greater<pi>>pq;
```

```
  pq.push({0, s}); //storing dist as first
parameter so that pq gets arranged in
increaing order of dist
  while (!pq.empty()) {
    pi cur = pq.top();
    pq.pop();
    int u = cur.second;
    for (auto it = adj[u].begin(); it !=
adj[u].end(); it++) {
      int v = (*it).first;
      int wt = (*it).second;
      if (d[v] > d[u] + wt)
      {
        d[v] = d[u] + wt;
        //p[v]=u;
        pq.push({d[v], v});
      }
    }
  }
  ll ans;
  for (int i = 1; i <= n; i++) {
    if (i == dest) {
      ans = d[i];
      break;
```

```
    }
  }
  return ans;
}
int main(int argc, char **argv) {
  // n:no of vertices starting from 1 , m : no of
edges
  clock_t start, end;
  //start=clock();

  string s = argv[1];
  int destination = 0;
  for (int i = 0; i < s.length(); i++)
    destination = destination * 10 + s[i] - '0';
  //cout<<dest<<endl;
  ifstream infile("dataset.txt");
  char a;
  int u, v, dist;
  //astar::node<unsigned> nodes[3000];
  start = clock();
  while (infile >> a >> u >> v >> dist) {
    //cout<<u<<" "<<v<<" "<<dist<<endl;
    addEdge(u, v, dist);
    n = max(n, max(u, v));
```

```
  }

  d = vector<int>(n + 5, INF);
  int source = 1;
  ll distance = dijkstra(source, destination);
  end = clock();
  double time_taken = double(end - start) /
double(CLOCKS_PER_SEC);
  cout << "Source Node\tDestination Node\
tDistance\t\tTime Taken(in ms)\n";
  cout << source << "\t\t\t" << destination <<
"\t\t";
  if (distance >= INF)
    cout << -1;
  else
    cout << distance;
  cout << "\t\t\t" /*<< "Time taken by program
is : " */ << fixed
        << time_taken * 1000 <<
setprecision(10) << endl;
  //cout << "msec " << endl;
  cout << "\n\nNote:\n1.It is always guaranted
to get optimal path using dijkstra's
```

```
Algorithm.\n2.Distance=-1 means Path
Doesn't Exists.\n" << endl;
}
```

2)Astar.hpp

```
#pragma once
#include <vector>
#include <limits>
#include <queue>

//adding namespace for astar algorithm
namespace astar {
template<typename cost_type>//declaring a
template for cost_type
struct node;
template<typename cost_type>
using edge = std::pair<node<cost_type>*,
cost_type>;
template<typename cost_type>
//declaring a structure of nodes and edges
which will calc heuristic cost
struct node {
```

```
    std::vector<edge<cost_type>> edges;
    cost_type tentative, heuristic;
    node(
        decltype(edges) edges = {},
        decltype(tentative) tentative
        =
std::numeric_limits<decltype(tentative)>::ma
x(),
        decltype(heuristic) heuristic = 0
    ): edges(edges), tentative(tentative),
heuristic(heuristic) {}
};
template<typename cost_type>
cost_type path(node<cost_type>& start,
decltype(start) goal) {
    auto priority = [](
                    const node<cost_type>* lhs,
                    const node<cost_type>* rhs
    ) {
        return lhs->tentative
               + lhs->heuristic
               > rhs->tentative
               + rhs->heuristic;
    };
```

```
    //declaring a min heap
    std::priority_queue <
    node<cost_type>*,
        std::vector<node<cost_type>*>,
        decltype(priority)
        > open_set(priority);
    start.tentative = 0;
    open_set.push(&start);
    while (!open_set.empty()) {
        auto n = open_set.top();
        if (n == &goal) return goal.tentative;
        open_set.pop();
        for (auto& i : n->edges) {
            if (i.first->tentative
                    ==
std::numeric_limits<cost_type>::max()
                ) {
                  i.first->tentative = n->tentative
+ i.second;
                  open_set.push(i.first);
            }
        }
    }
    //returning the answer
```

```
    return
std::numeric_limits<cost_type>::max();
}
};
```

3) astar.cpp:

```
#include<bits/stdc++.h>
#include "astar.hpp"
using namespace std;
astar::node<unsigned> nodes[900223];
//Function to add edge between two nodes
of a graph
void addedge(astar::node<unsigned>& u,
astar::node<unsigned> &v, unsigned w) {
  u.edges.emplace_back(&v, w);
}
//Note:please provide destination node as
command line input to run this file
int main(int argc, char **argv) {
  clock_t start, end;
```

```
  //start=clock();

  string s = argv[1];
  int destination = 0;
  for (int i = 0; i < s.length(); i++)
    destination = destination * 10 + s[i] - '0';
  ifstream infile("dataset.txt");//opening
file "dataset.txt" in reading mode
  char a;
  int u, v, dist;
  start = clock(); //starting measuring time
  //reading file "datset.txt" and adding edge
between nodes of the graph
  while (infile >> a >> u >> v >> dist) {
    addedge(nodes[u], nodes[v], dist);
  }

  int source = 1;
  int distance = astar::path(nodes[source],
nodes[destination]); //finding the path
length
  end = clock(); //finishing measuring time
```

```
  double time_taken = double(end - start) / double(CLOCKS_PER_SEC); //total time taken
  cout << "Source Node\tDestination Node\tDistance\t\tTime Taken(in ms)\n";
  cout << source << "\t\t\t" << destination << "\t\t";
  if (distance == -1)
    cout << -1;
  else
    cout << distance;
  cout << "\t\t\t" /*<< "Time taken by program is : " */ << fixed
       << time_taken * 1000 << setprecision(10) << endl;
  //cout << "msec " << endl;
  cout << "\n\nNote:\nIt is NOT guaranted to always get optimal path using Astar Algorithm.\n\n2.Distance=-1 means Path Doesn't Exists.\n" << endl;

}
```

4) run.sh:

```
#!/bin/bash
echo -n  """>dijkstra.dat
echo -n "Collecting runtime measurement Data by using Dijkstra over Graph of 5 lac+ nodes and 20 lac+ edges...."
#running dijkstra algorithm to calculate time taken by algorithm to find shortest path to various nodes
g++ dijkstra.cpp
for(( i=1; i<=300000; i+=10000 ))
do
  ./a.out $i | sed -n 2p | sed -r 's/[[:blank:]]+/,/g'   | cut -f2,4 -d ',' | tr ',' '\t' >> dijkstra.dat
done
echo "ok Completed!"
#running dijkstra algorithm to calculate time taken by algorithm to find shortest path to various nodes
echo -n  """>astar.dat
```

```
echo -n "Collecting runtime measurement Data by using Astar Algorithm over Graph of 5 lac+ nodes and 20 lac+ edges...."
g++ astar.cpp
for(( i=1; i<=300000; i+=10000 ))
do
  ./a.out $i | sed -n 2p | sed -r 's/[[:blank:]]+/,/g'   | cut -f2,4 -d ',' | tr ',' '\t' >> astar.dat
done
echo "ok Completed!"
echo "Plotting Curve of No.of Nodes vs Time taken to calculate shortest path by both algorithms...."
sleep 4s
#plotting Curve of no. of nodes vs time taken
gnuplot run.p
#opening output file
eog output.png
#removing files no longer needed
rm dijkstra.dat
rm astar.dat
```

```
rm output.png
```

5) run.p:

```
set term png font arial 12 size 800,600
set output 'output.png'
set xlabel 'No. of Nodes'
set ylabel 'Time(in ms)'
p 'dijkstra.dat' u 1:2 w linespoints t
'Dijkstra','astar.dat' u 1:2 w linespoints t
'Astar'
replot
```