Neslihan Çekiç 150160060

AoA2 –HW2

Compile: g++ -std=c++11 150160060.cpp

Run: ./a.out text1.txt

CLASSES

-
  ```
  class Edge {  //Hold the edges info for creating graph
          int source;
          int dest;
          int weight;
  };
  ```

-
  ```
  class Graph {
     //Creating vector of vectors of pair(for holding dest and weight)
     vector<vector<pair<int, int>>> adjList;
     int node; //total node count of graph
  };
  ```

-
  ```
  class Path { //To hold path info
  public:
          int parent; //the node info
          int time; // time to going that node
  };
  ```

FUNCTIONS

```
//Printing Path (Node,Time)
```

-
  ```
  void printPath(vector<Path> e) {…}
  ```

```
//Printing Final Result (J's and L's paths)
```

-
  ```
  void PrintResult(vector<Path> JGo, vector<Path> JReturn, vector<Path> LGo,
  vector<Path> LReturn) {…}
  ```

```
//Creating paths compatible of homework
```

-
  ```
  void createPath(vector<Path> parent, int j, vector<Path>& path){…}
  ```

```
//Dijkstra algorithm
//src->source node
//dst->destination node
//cost->starting cost
//forbidden->forbidden node for collision(forbidden==-1->no forbidden node)
//forbidden->(act like the node visited)
```

-
  ```
  vector<Path> dijkstra(Graph graph, int src, int dst, int cost, int forbidden)
  {…}
  ```

```
//Detecting Node-base collision
```

-
  ```
  int CollisionDetect(vector<Path> &J, vector<Path> &L) {…}
  ```

  ->In this function I compared J’s and L’s every node and return the node which caused collision. If there are no collision return the (-1).

```
//Detecting Stop-base collision + change the path causing collision
```

-
  ```
  void CollisionDetect2(Graph graph, vector<Path>& J, vector<Path>& L) {…}
  ```

  ->In this function I compared J’s nodes and L’s stop time and vice versa. If there any collision change the path on the move.

```
//Changing paths for node-base collision(4-cases)
```

-
  ```
  void changePath(vector<Path>& J, vector<Path>& L, vector<Path>& JTemp, vector<Path>& LTemp) {…}
  ```

  ->In this function I controlled the alternative paths for 4 cases and update the paths accordingly.

- Case1: No alternative path for both of them (exit the program)
- Case2: One of them has alt. Path and it is collision free.
- Case3: Both of them has alt. Path but one of them collision free.
- Case4: Both of them has alt. Path and they are collision free. (For extra I checked if collision occurs in both alt.paths and exit the program)

PROGRAM FLOW

->In main function these are the steps of program:

1. Read the file and creating edges and hold them in EdgeList (vector<Edge> type) .
2. Creating graph using EdgeList.
3. Creating going paths like so:

   ```
   vector<Path> *JGo = new vector<Path>(dijkstra(graph, JH, JD, 0,-1));
   vector<Path> *LGo = new vector<Path>(dijkstra(graph, LH, LD, 0,-1));
   ```

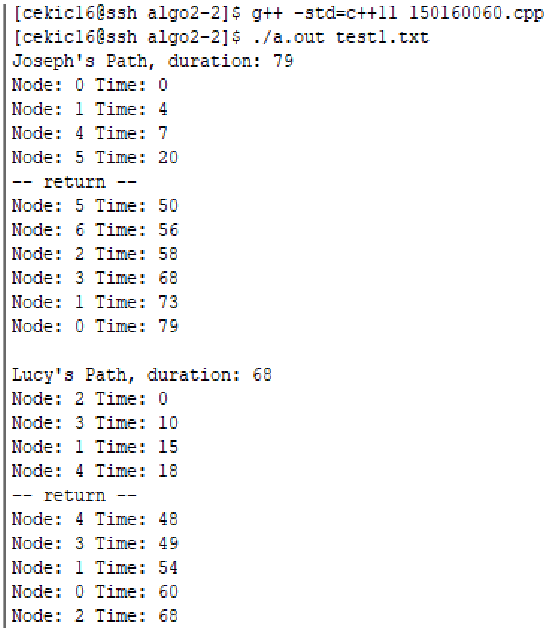
4. CollisionDetect() between JGo and LGo
5. If collision occurs create alternative paths for both of them forbid the node that caused collision. Then changePath() according the cases.
6. CollisionDetect2() between JGo and LGo stoptime and vice versa. Then find the node that caused collision and change the person's path which still moving.
7. After the CollisionDetect2() check there is any collision between new paths. If collision occurs that means no solution exit the program.
8. After finishing going paths calculate the return path starting cost like so:

   ```
   int jtime = JGo->back().time + 30;
   int ltime = LGo->back().time + 30;
   ```

9. Redo the 3 to 7 steps for return paths
10. After the finding return paths print final result using PrintResult().

SAMPLE OUTPUTS

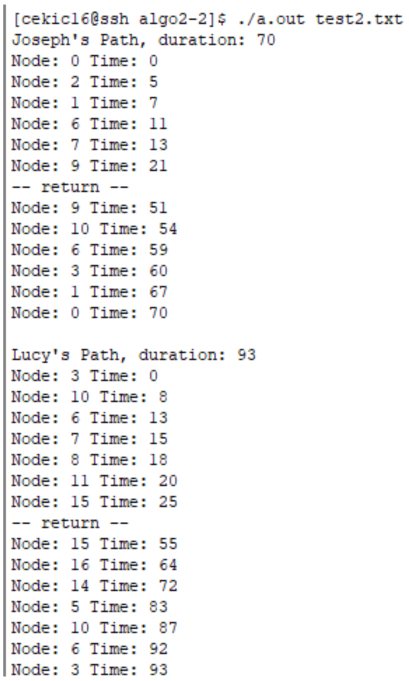
TEXT1



```
[cekic16@ssh algo2-2]$ g++ -std=c++11 150160060.cpp
[cekic16@ssh algo2-2]$ ./a.out test1.txt
Joseph's Path, duration: 79
Node: 0 Time: 0
Node: 1 Time: 4
Node: 4 Time: 7
Node: 5 Time: 20
-- return --
Node: 5 Time: 50
Node: 6 Time: 56
Node: 2 Time: 58
Node: 3 Time: 68
Node: 1 Time: 73
Node: 0 Time: 79

Lucy's Path, duration: 68
Node: 2 Time: 0
Node: 3 Time: 10
Node: 1 Time: 15
Node: 4 Time: 18
-- return --
Node: 4 Time: 48
Node: 3 Time: 49
Node: 1 Time: 54
Node: 0 Time: 60
Node: 2 Time: 68
```

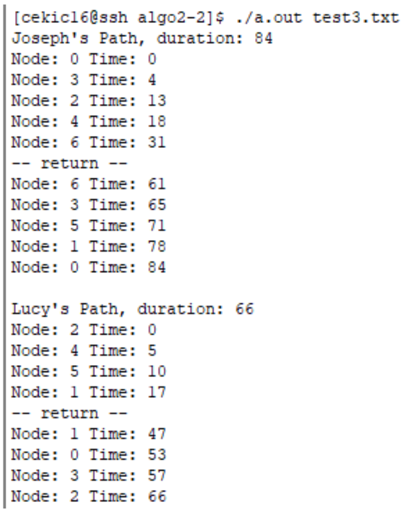
TEXT2



```
[cekic16@ssh algo2-2]$ ./a.out test2.txt
Joseph's Path, duration: 70
Node: 0 Time: 0
Node: 2 Time: 5
Node: 1 Time: 7
Node: 6 Time: 11
Node: 7 Time: 13
Node: 9 Time: 21
-- return --
Node: 9 Time: 51
Node: 10 Time: 54
Node: 6 Time: 59
Node: 3 Time: 60
Node: 1 Time: 67
Node: 0 Time: 70

Lucy's Path, duration: 93
Node: 3 Time: 0
Node: 10 Time: 8
Node: 6 Time: 13
Node: 7 Time: 15
Node: 8 Time: 18
Node: 11 Time: 20
Node: 15 Time: 25
-- return --
Node: 15 Time: 55
Node: 16 Time: 64
Node: 14 Time: 72
Node: 5 Time: 83
Node: 10 Time: 87
Node: 6 Time: 92
Node: 3 Time: 93
```

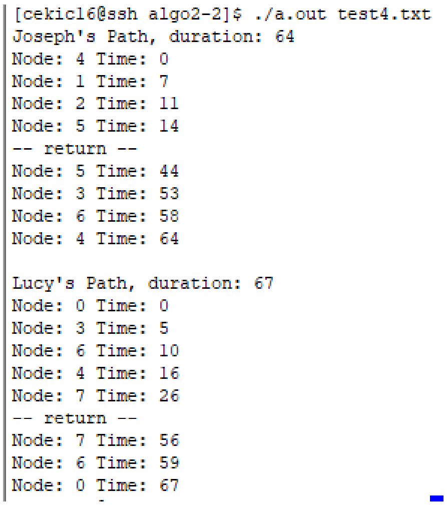
TEXT3



```
[cekic16@ssh algo2-2]$ ./a.out test3.txt
Joseph's Path, duration: 84
Node: 0 Time: 0
Node: 3 Time: 4
Node: 2 Time: 13
Node: 4 Time: 18
Node: 6 Time: 31
-- return --
Node: 6 Time: 61
Node: 3 Time: 65
Node: 5 Time: 71
Node: 1 Time: 78
Node: 0 Time: 84

Lucy's Path, duration: 66
Node: 2 Time: 0
Node: 4 Time: 5
Node: 5 Time: 10
Node: 1 Time: 17
-- return --
Node: 1 Time: 47
Node: 0 Time: 53
Node: 3 Time: 57
Node: 2 Time: 66
```

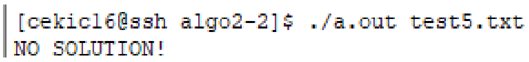
TEXT4



```
[cekic16@ssh algo2-2]$ ./a.out test4.txt
Joseph's Path, duration: 64
Node: 4 Time: 0
Node: 1 Time: 7
Node: 2 Time: 11
Node: 5 Time: 14
-- return --
Node: 5 Time: 44
Node: 3 Time: 53
Node: 6 Time: 58
Node: 4 Time: 64

Lucy's Path, duration: 67
Node: 0 Time: 0
Node: 3 Time: 5
Node: 6 Time: 10
Node: 4 Time: 16
Node: 7 Time: 26
-- return --
Node: 7 Time: 56
Node: 6 Time: 59
Node: 0 Time: 67
```

TEXT5



```
[cekic16@ssh algo2-2]$ ./a.out test5.txt
NO SOLUTION!
```