Robotic Delivery Algorithm

Bhawna (180010009)

Yashi Agarwal (180010030)

Indian Institute of Technology, Goa

Index

Sr. no.	Table of Contents
1	Problem Description
2	Inputs and Outputs
3	How the problem is converted into a technical problem?
4	The technical part that we have addressed
5	Solution to the problem (algorithm)
6	Analysis (time and space complexity)
7	Scope, Limitations, Future works
8	Additional reading Disjoint sets

Abstract

We have designed an algorithm, which can be fed to robots and they will decide automatically, which path is better. We have also designed a client side (in the same program, we have implemented both company and client side interchangeably), which asks them to place orders and their parcels will be delivered by robots in at least time.

In our implementation both client's and company sides are implemented. Firstly the company is choosing a location from [1 to 18], then the customer is asked to place an order, then the company employer is asked whether he wants to assign orders to robots or are there more customers who want to place orders.

When the company employer says yes, he wants robots to move now for delivering the orders, the path will be calculated and fed to robots. Then the robot will deliver the order and return back to the company location for its next order. One who is returning first, gets a recent order.

Then we had analysed which all locations are not reachable currently from company location, and we have suggested these places suitable for next headquarter of company.

Our algorithm tells user the following informations:

- Which robot (robot ID) is going to deliver his/her parcel.
- Path taken by the robot from company location to hotel and then customer’s location.
- Time taken by the robot to reach the customer's location.
- If in case all the robots are working, the customer will be informed but his/her order will be taken.
- Where to open the next headquarters.
- Pending orders
- Total orders which a robot is taking.

Problem Statement

The food industry has gone through a significant revolution over the recent past. Customers can now easily order food online and have it delivered to their homes within the shortest time possible. In fact, the growth rate for online food ordering and restaurant delivery has been over 20% over the last five years. Although the estimates differ across locations, online food delivery rates are expected to grow to more than $220 billion by 2025. This translates to about 40% of the total restaurant sales. But in pandemic situations like COVID-19, it is not recommended to have people for home delivery, and even people are not allowed to go to market by themselves because they might be infected and they can spread exponentially. In such situations people could not buy groceries and other necessary items.

This problem would not have occurred if we would have robots for delivery. These robots can be sanitized after one delivery.

Inputs

- **Customer details**
    1. ID (assigned by company)
    2. Location
    3. Hotel Location (optional, in case you have a specific choice for hotel)
    4. Membership Plan (extra delivery charges for fast delivery)
- **Location details**
    1. Location of hotels in the area offering food delivery
    2. Road availability (taken as a graph)
    3. Time to reach from one location to another (adjacency list entries, taken according to distance between the places and traffic)
    4. Company location

Graph (implemented as a adjacency matrix):

- Nodes are customer's location and hotel locations.
- Edges are direct roads between the two locations.

- Cost of edges is the time taken between them depending on the distance and traffic on the road. If there is no direct road between two locations then its cost is taken as 10,000 (considered as infinity).

Outputs

- Robot details
    1. ID
    2. Path from company location to customer's hotel location
    3. Full path from company location to customer's location
    4. Time taken by the robot to reach customer's location
    5. Number of orders collectively delivered
- Details provided to the customer after order is placed
    1. Which robot (robot's ID) is going to deliver his/her order
- Details for company
    1. Currently assigned robot's ID for delivering the packet
    2. Total number of orders pending

3. In case the company wants to open a new branch, then the suitable location of the branch.

How the problem is converted into a technical problem?

We have firstly collected data for a city. Collected data contains some places from where people place orders, some famous hotels and road networks of the city. Then we have drawn the graph, places are taken as its vertices and the linkage by roads as edges between the two vertices. Time to reach from one place to another is taken as the cost of each edge.

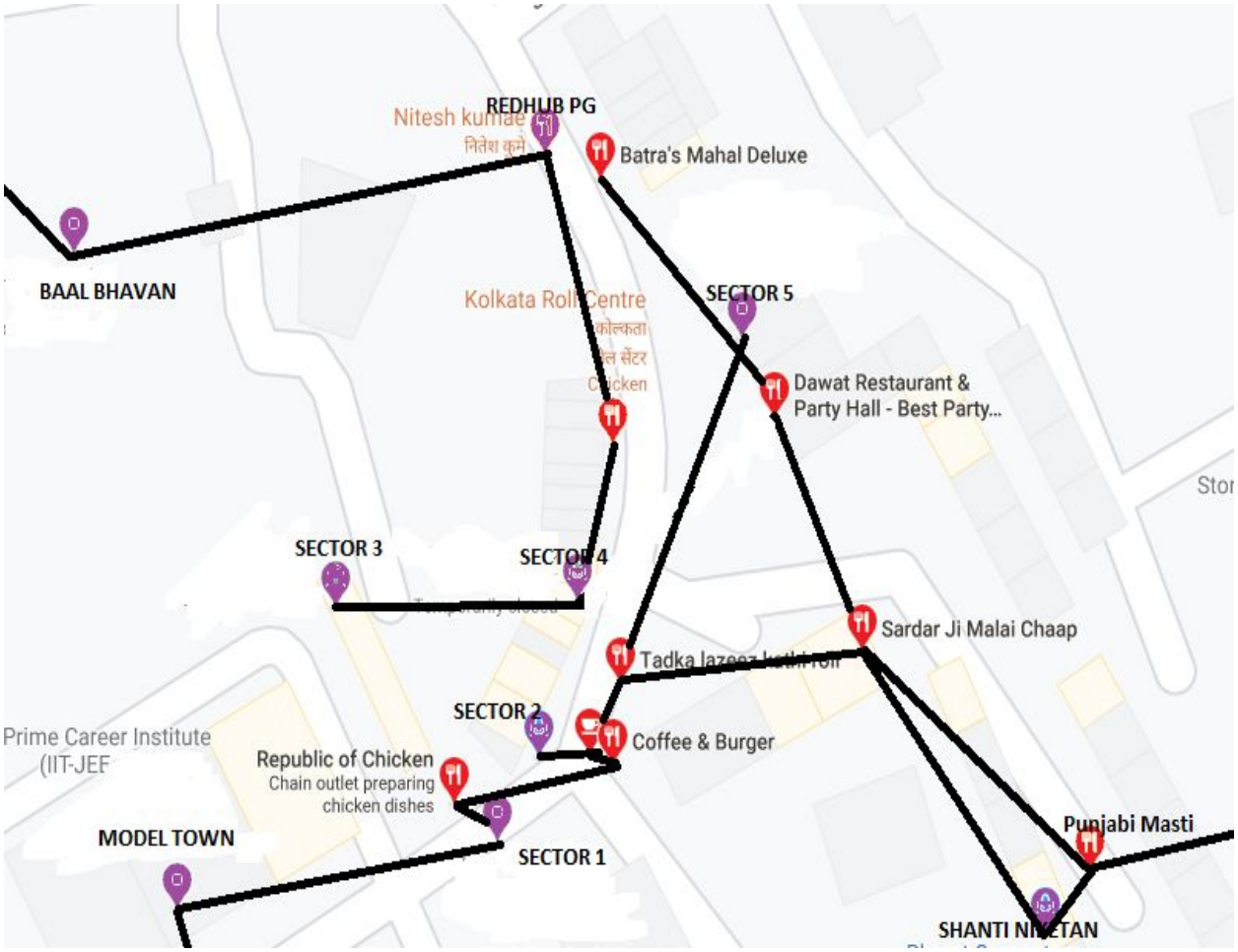
An example is given below:

Names of Places (abbreviation used in below matrix) - Location number

Baal Bhavan (B B) - 1
Red Hub PG (R H PG) - 2
Kolkata Roll Center (K R C) - 3
Sector 4 (S 4) - 4
Sector 3 (S 3) - 5
Batra's Mahal Deluxe (B M) - 6
Sector 5 (S 5) - 7
Dawat Restaurant & Party Hall (D R) - 8
Sardar Ji Malai Chaap (Sji) - 9
Tadka Lazeez (T L) - 10

Coffee (C) - 11
Burger (B) - 12
Sector 2 (S 2) - 13
Republic of Chicken (R C) - 14
Sector 1 (S 1) - 15
Model Town (M T) - 16
Shanti Niketan (S N) - 17
Punjabi Masti (P M) - 18

All the locations joined with black line are direct roads where robots can travel. We are assuming that in congested areas robots can't walk.



Adjacency Matrix of Time :

	B B 1	R H PG 2	K R C 3	S 4 4	S 3 5	B M 6	S 5 7	D R 8	Sji 9	T L 10	C 11	B 12	S 2 13	R C 14	S 1 15	M T 16	S N 17	P M 18
B B 1	0	**50**	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
R H PG 2	**50**	0	**45**	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
K R C 3	∞	**45**	0	**30**	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
S 4 4	∞	∞	**30**	0	**35**	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
S 3 5	∞	∞	∞	**35**	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
B M 6	∞	∞	∞	∞	∞	0	**40**	**47**	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
S 5 7	∞	∞	∞	∞	∞	**40**	0	**7**	∞	**60**	∞	∞	∞	∞	∞	∞	∞	∞
D R 8	∞	∞	∞	∞	∞	**47**	**7**	0	**25**	∞	∞	∞	∞	∞	∞	∞	∞	∞
Sji 9	∞	∞	∞	∞	∞	∞	∞	**25**	0	**15**	∞	∞	∞	∞	∞	∞	**40**	**30**
T L 10	∞	∞	∞	∞	∞	∞	**60**	∞	**15**	0	**10**	∞	∞	∞	∞	∞	∞	∞
C 11	∞	∞	∞	∞	∞	∞	∞	∞	∞	**10**	0	**5**	**6**	∞	∞	∞	∞	∞
B 12	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	**5**	0	∞	**15**	∞	∞	∞	∞

S 2 13	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	**6**	∞	0	∞	∞	∞	∞	∞
R C 14	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	**15**	∞	0	**7**	∞	∞	∞
S 1 15	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	**7**	0	**30**	∞	∞
M T 16	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	**30**	0	∞	∞
S N 17	∞	∞	∞	∞	∞	∞	∞	∞	**40**	∞	∞	∞	∞	∞	∞	∞	0	**3**
P M 18	∞	∞	∞	∞	∞	∞	∞	∞	**30**	∞	∞	∞	∞	∞	∞	∞	**3**	0

Then we have done robot scheduling by using a dictionary (named, time = { }) in the class Robots. Then we modeled an algorithm for finding the shortest path, for modeling this algorithm we have used the Dijkstra concept and greedy strategy.

For every order we have done a survey for the company, if the company wants to start a new branch, then we have suggested some places, where the company can earn more profits. For implementing this we have used **disjoint sets and connected components concepts of DFS**. We have found the places where currently robots are unreachable but the company is getting orders from these places.

The technical problem can be stated as:

You are having an undirected graph, G(V,E) where V are possible locations in a city from where orders can be placed or delivered, E is the set of edges (roads from one location to another). Each edge is having a cost which is the time to reach from

one vertex to another, i.e. (u,v) is in E then T : Time to reach from u to v or vice versa.

There can be at most N orders pending at a time. There are M total numbers of robots at work.

You have to assign the orders to the robots and feed the information to them about the path which they have to follow for delivering that particular order. Use interval scheduling for utilising the robots at the most. Assigning more than one order to the robots if this optimizes the solution, means it takes less time to deliver orders.

Assumptions that we had made:

1. Robot is present at the hotel location.
2. Total number of orders at a particular time can not be more than 20,0000 (i.e N<=20,000).
3. Total number of robots at work can not be more than 100 (i.e. M<=100).
4. Total number of possible locations can not be more than 50,000.
5. You can even have the same location as your hotel location and your address.
6. All the edges of the graph are positive as time can not be negative.
7. Traffic is considered to be constant at a particular road.
8. It will not take much time for a robot to deliver the parcel after reaching the customer's location, taken as negligible.
9. Orders will be delivered by robots on a first come first serve basis. But if we have free robots then client's who have taken membership will be given more priority.
10. Customers can cancel the order within 15 minutes of order. After that there will be no refund for cancellations of orders.
11. After delivery each robot will come back to the hotel for next delivery.

12. Robots can take orders from the hotel only if it is going to deliver means while coming back they are not supposed to go to the hotel and deliver an order.
13. In case the customer has not specified the hotel then we have a default hotel location.
14. By default the company has two robots.

The technical part that we have addressed

For the problem stated above, the technical parts that we have addressed in our implementation are:

- First technical problem is to find the shortest path. In real life it is possible that the path is not unique so from all shortest paths we need to find the best optimal path. Here, the best optimal path would mean that I choose the path delivering the maximum number of orders. You can see this in a way that, path with the least time is chosen. In case of ambiguity, a path with maximum number of undelivered vertices will be chosen.

- Second technical problem is to schedule the robots properly so that minimum time is taken for every order. By robot scheduling we mean that, say we have 5 orders and 3 robots at work, then it is not necessary that orders will be given to robots like 1-1, 2-2, 3-3, 4-1, 5-2 ((order number)-(robot number)). But this depends on the order by which robots are getting free. Say robot 2 gets free before robot 1 then orders will be assigned like 1-1, 2-2, 3-3, 4-2, 5-1.
- Third technical problem is to find the next optimal branch location for the company. This is based on the observation that all the hotels and customer locations are not reachable from current company location.

Solution to the problem (algorithm)

To tackle the first technical problem i.e. to find the best optimal shortest path. We have used the concept of dynamic programming, Dijkstra's algorithm.

Algorithm 1: Finding the best optimal shortest path

```
def initialising_robot():
        path=[]
        count_placed_together=0
        best={}
```

```
        visited={}
        for i in vertices:
                dist[i]=-1
                count[i]=0
                possible[i]=False
                best[i]=i
                visited[i]=0


def path_taken(c,v):
        visited[c] =1
        if c == v:
                dist[c]=0
                count[c]=0
                possible[c]=True
                return

        minimum_dist = 1000000
        minv = c
        maxcount = 0
        flap = False

        for i in adjacent_lst[c]:
                        if dist[i]==-1:
                            if visited[i]==0:
                                path_taken(weight,adjacent_lst,i,v)

                        if possible[i]==True:
                                flap=True
                                if dist[i]+weight[(c,i)]<minimum_dist:
                                        minimum_dist=dist[i]+weight[(c,i)]
                                        minv=i
                                        if order_status[i]==False:
                                                maxcount=self.count[i]+1
                                        else:
                                                maxcount=self.count[i]

                                if dist[i]+weight[(c,i)]==minimum_dist:
```

```
                                        if order_status[i]==False:
                                                if count[i]+1>maxcount:
                                                        minv=i
                                                        maxcount=self.count[i]+1
                                        else:
                                                if count[i]>maxcount:
                                                        minv=i
                                                        maxcount=self.count[i]

        if flap:
                dist[c]=minimum_dist
                count[c]=maxcount
                best[c]=minv
                possible[c]=True

        if not flap:
                dist[c]=-100

        Return
```

def find_path(self,c,v):

```
        if c==v:
                self.path.append(c)
                return
       path.append(c)
       find_path(self.best[c],v)
```

Complexity Analysis:

It works for cyclic graphs also. Extra space complexity other than storing the graph is analysed below.

Time Complexity:

O(V) for initialising

O(E) for recursive function calculating the shortest path
O(V) to print the path
Hence total time complexity is O(E+V) for each order
For k orders total time complexity is: O(kV+kE)

Space Complexity:

O(V) for each best,count,dist and possible. Hence 4*V
Hence total space complexity is O(V)

```
@agarwal:~/Desktop/Algo_EndSem/Final
yashi@agarwal:~/Desktop/Algo_EndSem/Final$ python3 main3.py
================================================================================
Select the location for your company from [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18].
 Enter : 6
================================================================================
Welcome to our company
Place your order here for safest and fastest delivery.
Would you like to place an order?....Press[Y/N]y
Please enter your location 17
Please enter your hotel location18
Have you availed the membership 0 or 11
We have successfully placed your order
-------------------------------------------
In the company do you want that our robot move now? n

Do you want to explore some additionals for the company
n
Welcome to our company
Place your order here for safest and fastest delivery.
Would you like to place an order?....Press[Y/N]y
Please enter your location 8
Please enter your hotel location9
Have you availed the membership 0 or 10
We have successfully placed your order
-------------------------------------------
In the company do you want that our robot move now? y

initialising the robot
Path till hotel [6, 8, 9, 18]
The path taken by the robot is
[6, 8, 9, 18, 17]
The total order it has placed together is 2
The total time taken is  105
orders still remaining are:
[]
we have sent the robot with Id 1
---------

Do you want to explore some additionals for the company
```

Fig1: Screenshot depicting the path taken by robot based on our shortest path algorithm

To tackle the second technical problem i.e. to give interval scheduling. We have used the concept learnt from the course **Discrete structures.**

Algorithm 2: Scheduling the robots

```
def robot_scheduing_initialise(self):
        time={}
        for i in range(1,max_robots+1):
                time[i]=0

def robot_scheduling(self,c):
        time[self.robot_to_be_sent]+=dist[c]
        print("we have sent the robot with Id",robot_to_be_sent)
        min=∞
        minv="nil"
        for i in range(1,max_robots+1):
                if min>time[i]:
                        min=time[i]
                        minv=i
        robot_to_be_sent=minv
```

Complexity Analysis:

Time Complexity:

O(K) for initialising, where the k is maximum number of robots present

O(K) for finding the minimum

Hence total time complexity is O(K)

Space Complexity:

O(K) for storing the total time taken by robot

Hence total space complexity is O(K)

```
@agarwal:~/Desktop/Algo_EndSem/Final
Would you like to place an order?....Press[Y/N]y
Please enter your location 12
Please enter your hotel location16
Have you availed the membership 0 or 11
We have successfully placed your order
-----------------------------------------------
In the company do you want that our robot move now? y

initialising the robot
Path till hotel [6, 8, 9, 10, 11, 12, 14, 15, 16]
The path taken by the robot is
[6, 8, 9, 10, 11, 12, 14, 15, 16]
The total order it has placed together is 1
The total time taken is  154
orders still remaining are:
[1, 2]
we have sent the robot with Id 1
---------

initialising the robot
Path till hotel [6, 8, 9, 10, 11, 13]
The path taken by the robot is
[6, 8, 9, 10, 11, 13]
The total order it has placed together is 1
The total time taken is  103
orders still remaining are:
[2]
we have sent the robot with Id 2
---------

initialising the robot
Path till hotel [6, 8, 9, 18, 17]
The path taken by the robot is
[6, 8, 9, 18, 17]
The total order it has placed together is 1
The total time taken is  105
orders still remaining are:
[]
we have sent the robot with Id 2
---------

Do you want to explore some additionals for the company
```

Fig2 : Screenshot of input illustrating robot scheduling

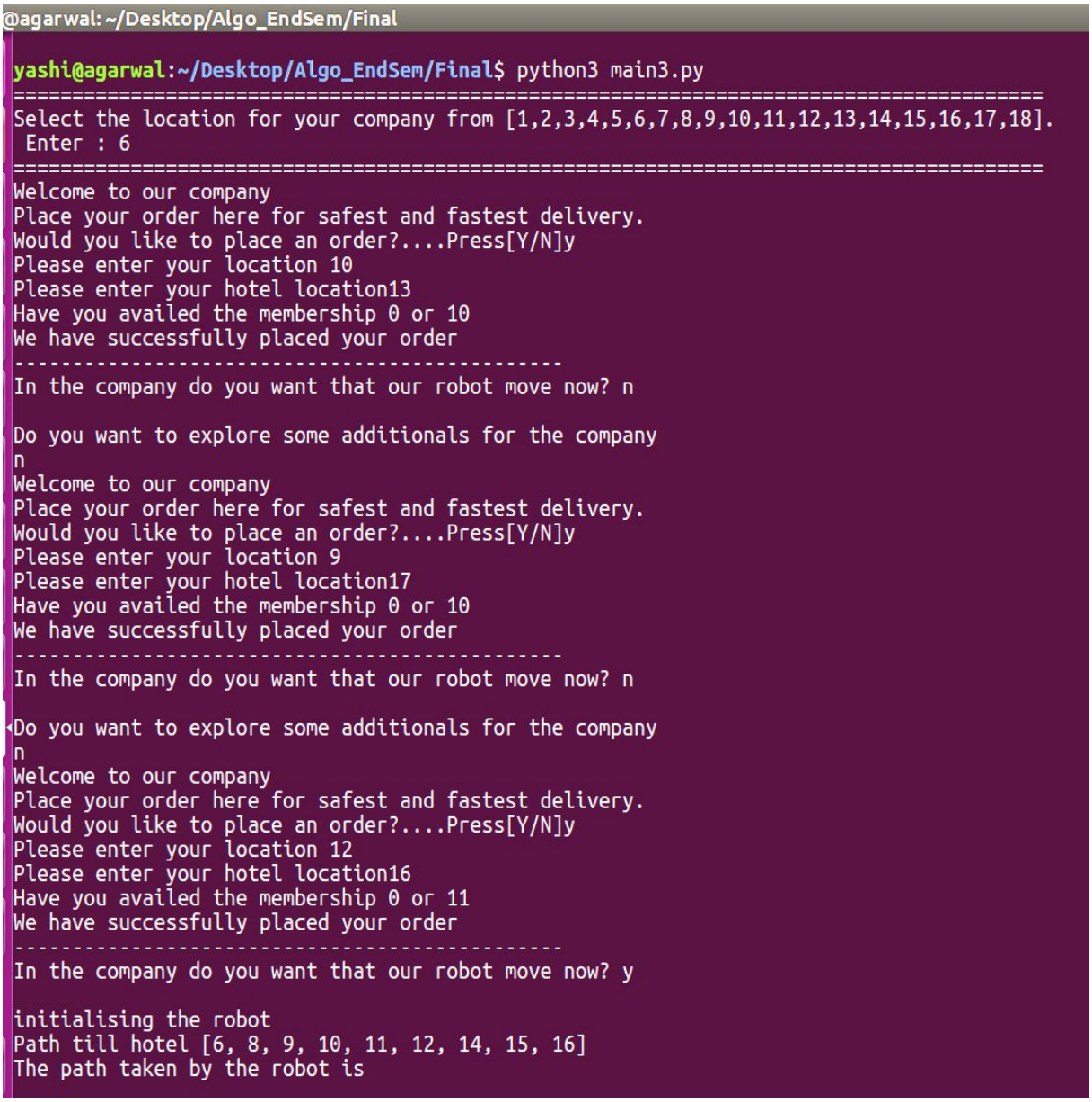


Fig3 : Screenshot illustrating robot scheduling

To tackle the third technical problem i.e. to give interval scheduling. We have used the concept learnt from **disjoint sets, dfs and dijkstra.** Here we have taken two cases:

1. If the graph is not connected then we will set up our branch in the unconnected component such that it has the maximum number of vertices covered. So to tackle this we have used the concept of disjoint sets and dfs.

2. If the graph is connected then we will add the branch to the vertex which is at maximum distance from our branch. So to tackle this problem we used the concept of dijkstra.

Algorithm 3: finding the best optimal branch

Case1: Graph is not connected:

```
for i in vertices:
        order_possible[i]=0
    explore(location,1)
    cc=2

    for i in vertices:
        if order_possible[i]==0:
            explore(i,cc)
            cc=cc+1
    if cc!=2:
        count={i:0 for i in range(1,cc+1)}
        for i in vertices:
            count[order_possible[i]]+=1
        maxc=0
        maxi="nil"
        for i in range(2,cc+1):
            if maxc<count[i]:
                maxc=count[i]
                maxi=i
        lst=[]
```

```
            for i in self.vertices:
                if order_possible[i]==maxi:
                    lst.append(i)
        print("your best headquarter location would be at one of the following vertices")
        print(lst)
```

def explore(c,cc):

```
        order_possible[c]=cc
        for i in adjacent_lst[c]:
            if order_possible[i]==0:
                explore(i,cc)
```

Case 2: Graph is connected

def dijkstra(self):

```
        dist={}
        flap={}
        mark=vertices
        for i in self.vertices:
            self.dist[i]=∞
            flap[i]=False

        dist[location]=0

        while(len(mark)!=0):
            minv=mark[0]
            for i in mark:
                if dist[minv]>dist[i]:
                    minv=i
            print("dist=",dist)
            for i in adjacent_lst[minv]:
                if flap[i]==False:
```

```
                    if dist[i]>dist[minv]+weight[(minv,i)]:
                        dist[i]=dist[minv]+weight[(minv,i)]
            mark.remove(minv)
            flap[minv]=True
```

Def main()

```
lst=[]
            dijkstra()
            max=0
            for i in vertices:
                if max<dist[i]:
                    lst=[i]
                    max=dist[i]
                elif max==dist[i]:
                    lst.append(i)

    print("your best headquarter location would be at one of the following vertices")
    print(lst)
```

Complexity Analysis:

Time Complexity:

For case 1 :

O(V) for initialising
O(E) for explore each component
O(N) to find the minimum of N is the number of disjoint components. N can be at most V

Hence total time complexity is O(V+E)

For case 2:

O((V+E)logV for dijkstra (to find the shortest path)
O(V) to find the maximum distance from company
Hence the total time complexity is O((V+E)logV)

Space Complexity:

O(K) for storing the total time taken by robot
Hence total space complexity is O(K)

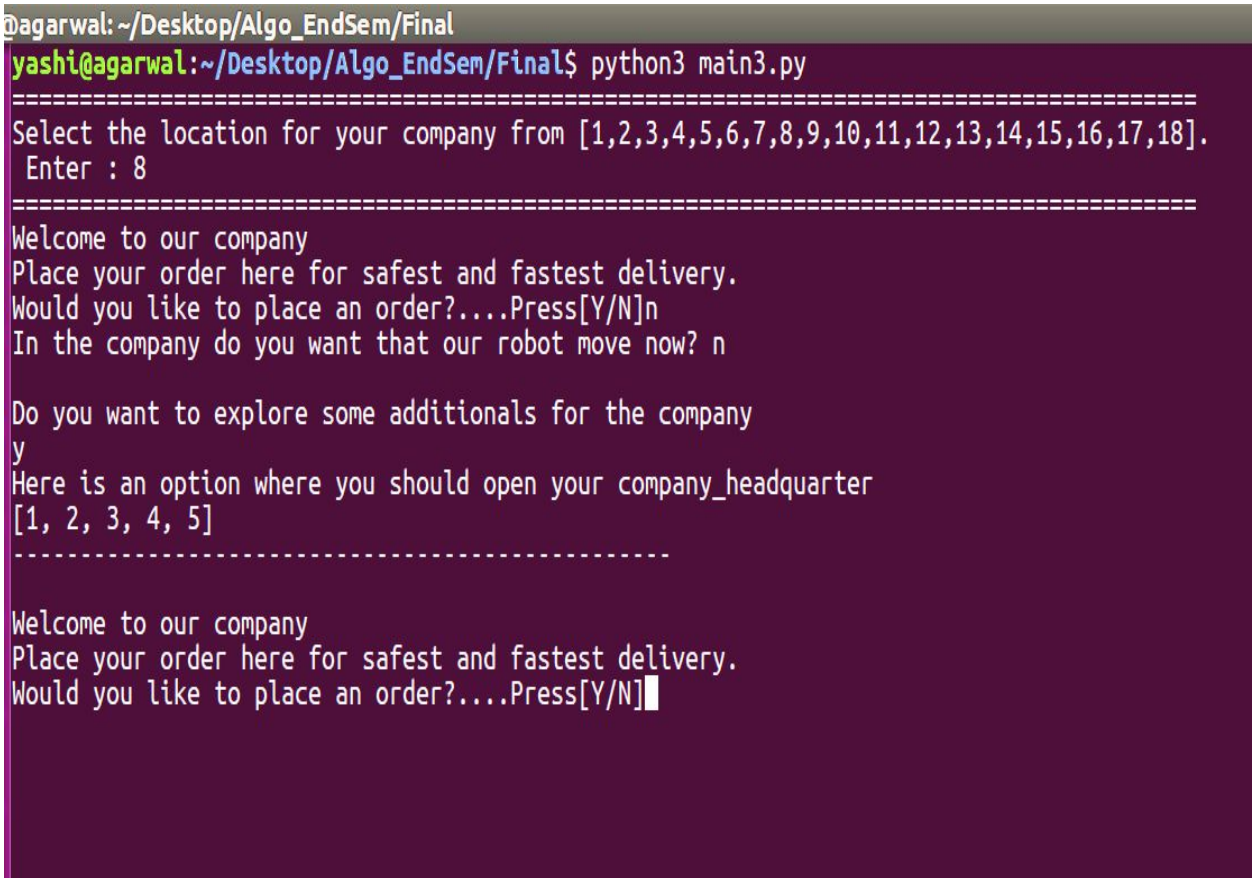


Fig4 : Screenshot illustrating company headquarter

Scope, Limitations, Future works

Scope

1. This model is greatly similar to a real world model.
2. This model is extremely flexible as it allows you to change company location.
3. It gives further scope for the company by telling the next best location to place your headquarters.
4. Can be used in pandemic situations like the current COVID 19 situation.
5. Needs low maintenance and labour work if a person really thinks about opening such a company.
6. This model also increases scope in robotics. As we need very efficient robots, with high precision.

Limitation

1. We had considered only one path to deliver. Using more than one path to deliver the parcels could gradually decrease the delivery time for some other customer.

2. Subsets of path. Since we have given priority order on the basis of first come first serve and membership basis but choosing on the basis of the path containing maximum number of deliveries would further optimise the solution.

3. All customers could not be happy sometimes. If you don't have membership and other customers have membership then he will be served before you even if you are the first one to place the order. This situation is worsened if

the customers are large and you live in a remote area. As in this case your order cannot be served in between.

4. In our implementation, at a particular instant, only one customer is placing an order while in real scenario many customers are placing orders.

5. Though it resembles the real world problem, as the complexity of graphs increases, changes in the program are not that easy to tackle as we need to hard code them.

6. If there is a sudden change in graph then we need to hard code that change. For example if there is a construction on a road for a certain period of time.

7. We have taken a scenario in which robots are used. But in the real world purchasing and maintaining such robots will be very costly.

8. We have taken here the traffic constant. But in real world traffic hugely varies with time and place.

9. If we get an order, and at that instant all the robots are at work, and say this new order has the same customer's address and hotel location which the previous one had, then two robots will be sent for these two orders but only one robot would have delivered both the orders. This is because the first robot was instructed to deliver the order, and just after that new order comes with the same addresses.

   Example : Let us say we have 3 robots and we get orders in this way-

   Order 1
   Customer's address = Model Town
   Hotel = Coffee
   Robot 1 is going to deliver this parcel.

Order 2
Customer's address = Model Town
Hotel = Coffee
Robot 1 is going to deliver this parcel also.

Order 3
Customer's address = Sector 5
Hotel = Sardarji Malai Chaap
Robot 2 is going to deliver this parcel.

Order 4
Customer's address = Shanti Niketan
Hotel = Dawat Restaurant & Party hall
Robot 3 is going to deliver this parcel.

Order 5
Customer's address = Shanti Niketan
Hotel = Dawat Restaurant & Party hall

Now this parcel could be delivered by Robot 3, but Robot 3 was asked to delivered order4 just a few seconds ago. Now this order (i.e.order 5) will be delivered by the robot who gets free firstly.

This is not the best assigning of orders.

Future works:

1. As a company manager it is a tedious task to keep a record of all the roads (graph) and traffic all the time. So we can implement using gps.

2. Here we have chosen a list and dictionary for the implementation but using different data structures like disjoint sets and heaps could further decrease the time complexity and increase efficiency of the algorithm.

3. Now to tackle this problem we gave an algorithm but it can be further optimised. Today many researches are going on globally and people are coming up with new algorithmic solutions. So there is a lot of scope in improving this model.

4. We can use bug algorithms for designing robots and can take input from google maps.

5. Building cheap and easily maintained robots with good sensors that can come on the market.

Additional reading

Disjoint sets

A disjoint-set data structure (also called a union–find data **structure** or **merge–find set**) is a data structure that tracks a set of elements partitioned into a number of disjoint (non-overlapping) subsets. It provides near-constant-time operations to add new sets, to merge existing sets, and to determine whether elements are in the same set. In addition to many other uses, disjoint-sets play a key role in Kruskal's algorithm for finding the minimum spanning tree of a graph.

Representation

A disjoint-set forest consists of a number of elements each of which stores an id, a parent pointer, and in efficient algorithms, either a size or a "rank" value.

The parent pointers of elements are arranged to form one or more trees, each representing a set. If an element's parent pointer points to no other element, then the element is the root of a tree and is the representative member of its set. A set may consist of only a single element.

Operations

MakeSet

The *MakeSet* operation makes a new set by creating a new element with a unique id, a rank of 0, and a parent pointer to itself.

```
function MakeSet(x) is
   if x is not already present then
      add x to the disjoint-set tree
      x.parent := x
      x.rank   := 0
      x.size   := 1
```

Find

Find(x) follows the chain of parent pointers from x up the tree until it reaches a root element, whose parent is itself. This root element is the representative member of the set to which *x* belongs, and may be *x* itself.

```
function Find(x)
   while x.parent ≠ x
      x:= x.parent
   return x
```

Find function with Path compression

Path compression

Path compression flattens the structure of the tree by making every node point to the root whenever *Find* is used on it. This is valid, since each element visited on the way to a root is part of the same set.

function *Find*(x)
  if x.parent ≠ *x*
    x.parent := *Find*(x.parent)
  return x.parent

Union by rank based on trees

Union_sets(a,b) merges the two specified sets (i.e. a, b). For optimising we have used rank of a and b sets. If both of them are having the same rank then b.parent is set as a, and rank of a is increased by one. If rank of b is greater than a.parent is set as b and rank of b still is the same.

```
Function union_sets(a, b)
   a = Find(a)
   b = Find(b)
   if (a != b)
      if (a.rank < b.rank)
                swap(a, b)
     b.parent = a
      if (a.rank == b.rank)
                a.rank++
```

References

- http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf
    1. Dijkstra
    2. Greedy Strategy
    3. Depth First Search
    4. Connected components

- https://en.wikipedia.org/wiki/Disjoint-set_data_structure