1. Experiments Scheduling

(a) Describe optimal substructure

The optimal solution to this problem will complete steps in order with the the least switching of shifts as possible. So if we are able to schedule a student who can do the most consecutive steps starting from first to last step. This would allow least amount of switching since a student can execute multiple steps in a given sequence.

Suppose a student n can complete steps $(i \ldots\ldots k)$ k steps in increasing order. We can have two subproblems we can have a student do steps $(1, \ldots i-1)$ which is before the i step or a student do $(i \ldots\ldots T)$ T which is the total number of steps. The solution for scheduling students for 1 to T total steps include solution to $(1 \ldots i-1)$

and (i...T) steps a student can do this problem has an optimal substructure

b) The greedy algorithm that could find a optimal way to schedule is one that finds a student that does the most steps in order starting from the first step. Then we will update what steps are remaining and find another student than can do the most steps in order starting from the step after the student before until all steps are completed.

d) Runtime of my greedy algorithm is

O(numSteps * numStudent)

unless each student does only one step.

- my while loop go thru all numsteps for each student

e) Greedy Algorithm Proof

my greedy algorithm always schedule the student that can do the most consecutive steps in increasing order with least amount of switching

1) my ALG: $S_1, S_2 \ldots S_K$
Soln OPT: $T_1, T_2 \ldots T_L$
where $L \geq K$

2) my ALG and OPT are both scheduling least students with most consecutive time in order.

3) Claim $(1 \leq i \leq K)$ where i is the smallest index where $u_i \neq v_i$
e g u l
in order make sure we get the shortest time also to accommodate the wait time will only be a simple condition statement. So it will not have an impact of the overall runtime.

4) $(S_1, S_2, \ldots i\ T_{i+1} \ldots T_L)$ would also be a solution.
$S_1 \ldots S_{i-1}$ same as $T_i \ldots T_{i-1}$

5) So by design of my ALG
S_i the last student to be schedule that will give the least amount of switching

6) So $S_i \leq T_L$ (finish time)
$(S_1, S_2, \ldots i\ T_{i+1} \ldots T_L)$ is a solution because T_{i+1} is the next student to be schedule for least amount of switching.

7) We can keep doing this using same claim.

ALG $S_1 \ldots S_2 \ldots S_k$
OPT $T_1 \ldots \ldots \ldots T_k\ T_{k+1} \ldots T_L$

by the design of my algorithm it will schedule students with the least amount of switching to complete the experiment.

8) This is a contradiction that a solution can schedule less switching student that my greedy algorithm

So my greedy algorithm will give an optimal solution

Public Public Transit

a) An algorithm solution to this problem would be Dijkstra Shortest path algorithm. The reason we are tring find the Shortest path from the Source to the destination. For the problem the best approach would to use Dijkstra to find the Shortest path and also implement it on the condition of waiting for the arrival of the train.

b) The complexity of my proposed is $O(Station^2)$ which stations is Ch cked a aint 1 Stations (ith Step) [n Student * k Steps]

I belive it should be
O(numSteps(numStudents * numSteps)

O(numsteps * numStudents + numsteps^2)

c) Implement Dijkstra algorithm in ShortestTime.

d) I use existing code to help me the next min station from the source that hasn't been checked. Also use boolean table to update what was checked

e) The current complexity of Shortest time is $O(V^2)$ where V is the vertices or stations. This can be made faster using a priority queue which will result in $O(V + E \log V)$
It takes O(V) to build the queue
Then to keep track of the min

for each would cost $O(\log V)$
So together takes $O(V \log V)$
Then for each each edge or weight we check each edge and update the distance for each neighbor vertex in the priority queue that takes $O(E \log V)$. So total runtime $O((V+E) \log V)$

g) implemented in code.

Extra work

* used geekforgeek for dikjstra
* used notes

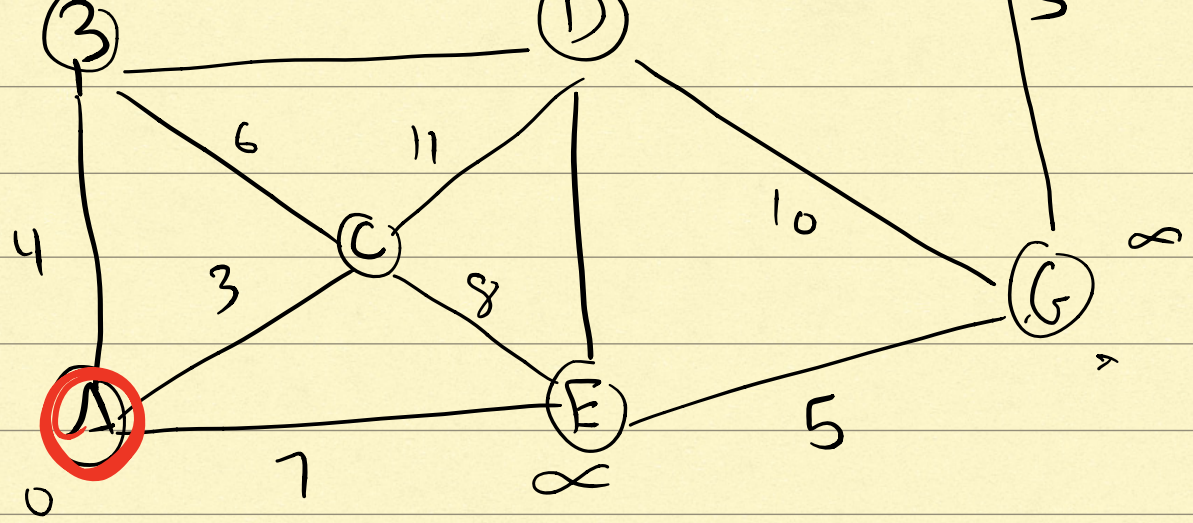
Init path
goal path

- Init = 0 all other = ∞
- keep set visit nodes
  Start with init

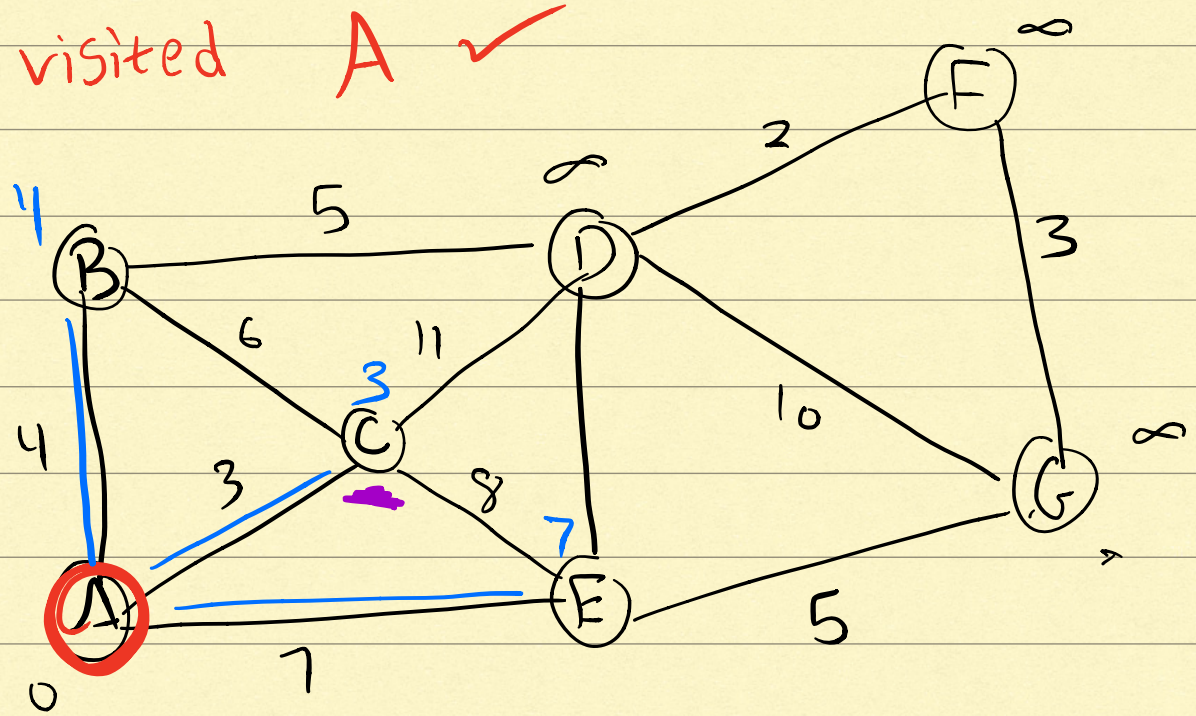
* Current node consider all unvisited neighbors
  - Calculate distance
    = distance to curr node +
      distance from curr to neighbor
    if less replace

- When finish consider all neighbor of curr node
  - mark visit and remove from unvisit
  - If dest vist ✓ finish

- set unvisit node mark small distone as curr and go back to *

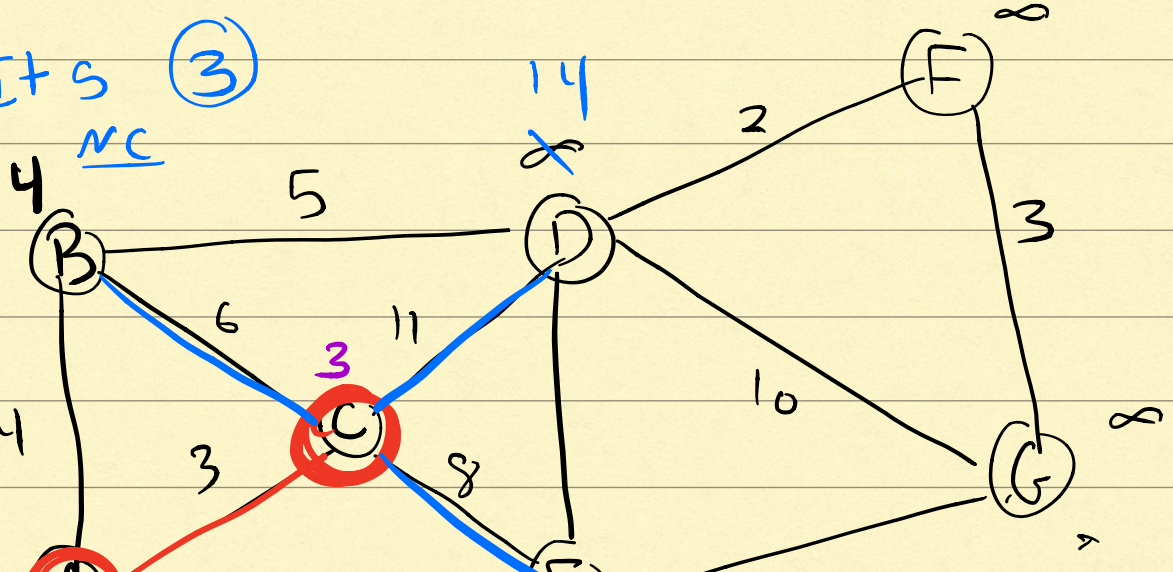


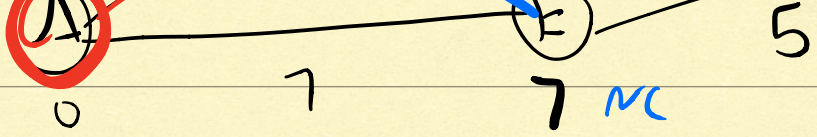
visited A ✓



Pick next node Smallest dist

It's (3) NC





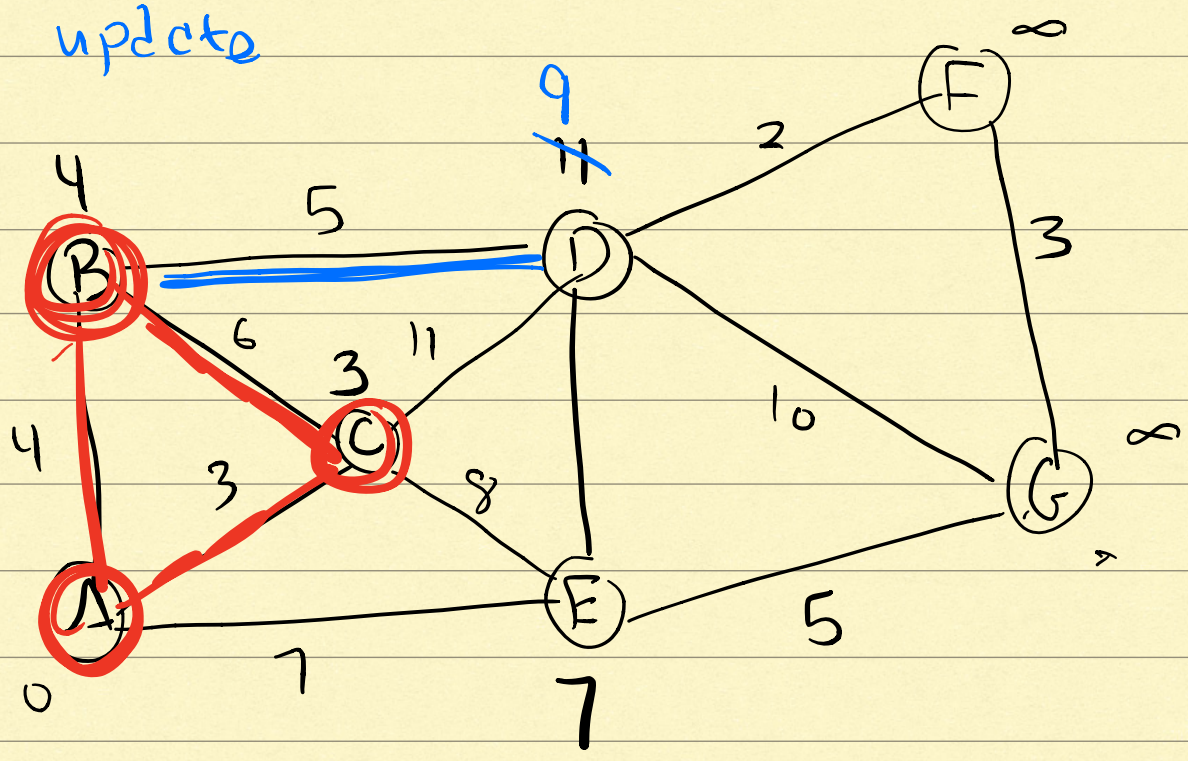
NC

no change

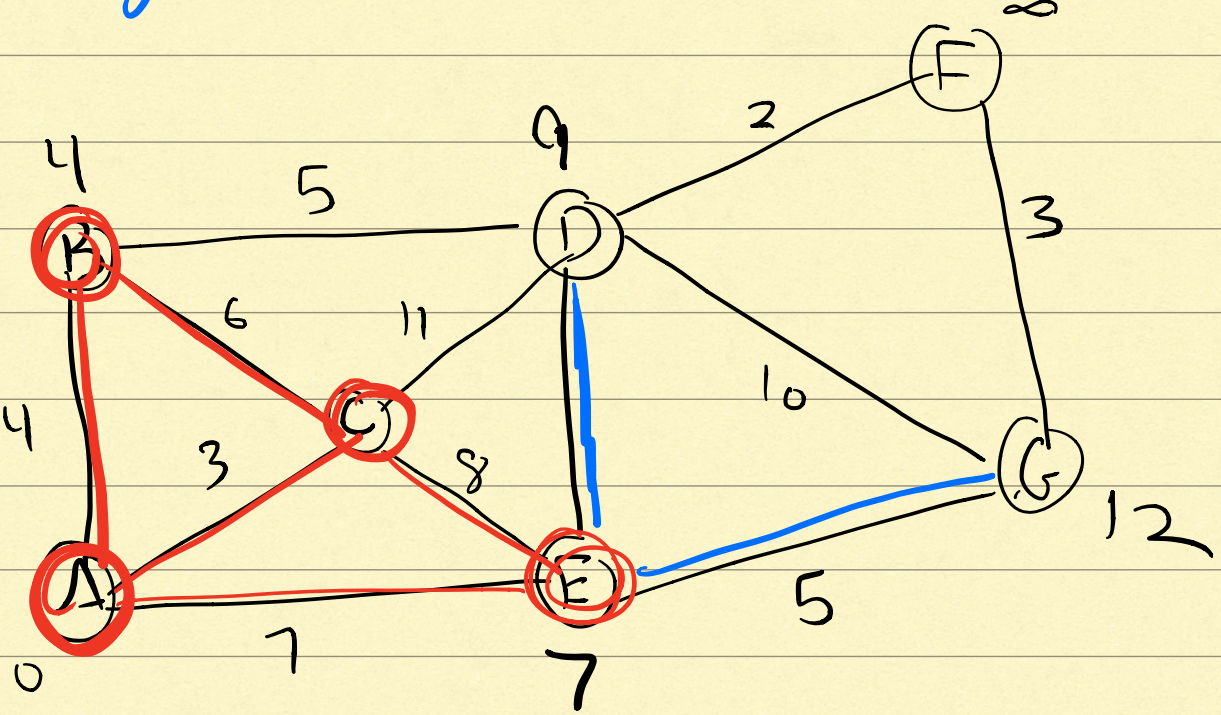
C C

Visited A and now C

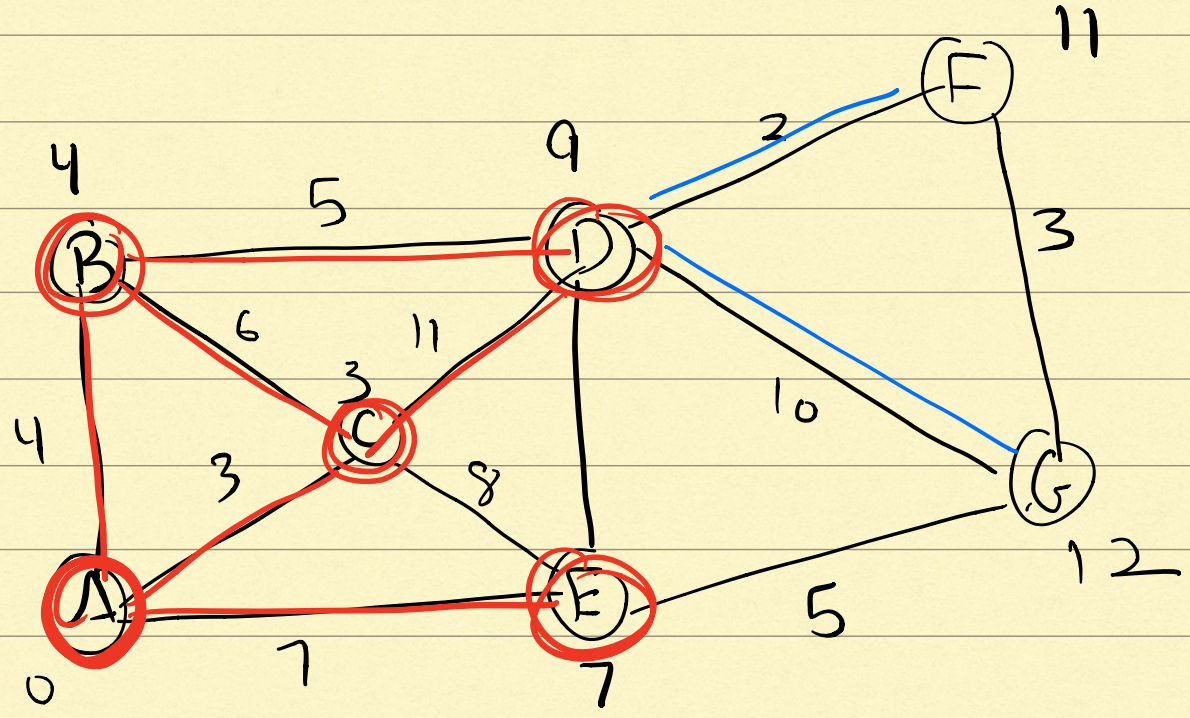
If new calculated route smaller update



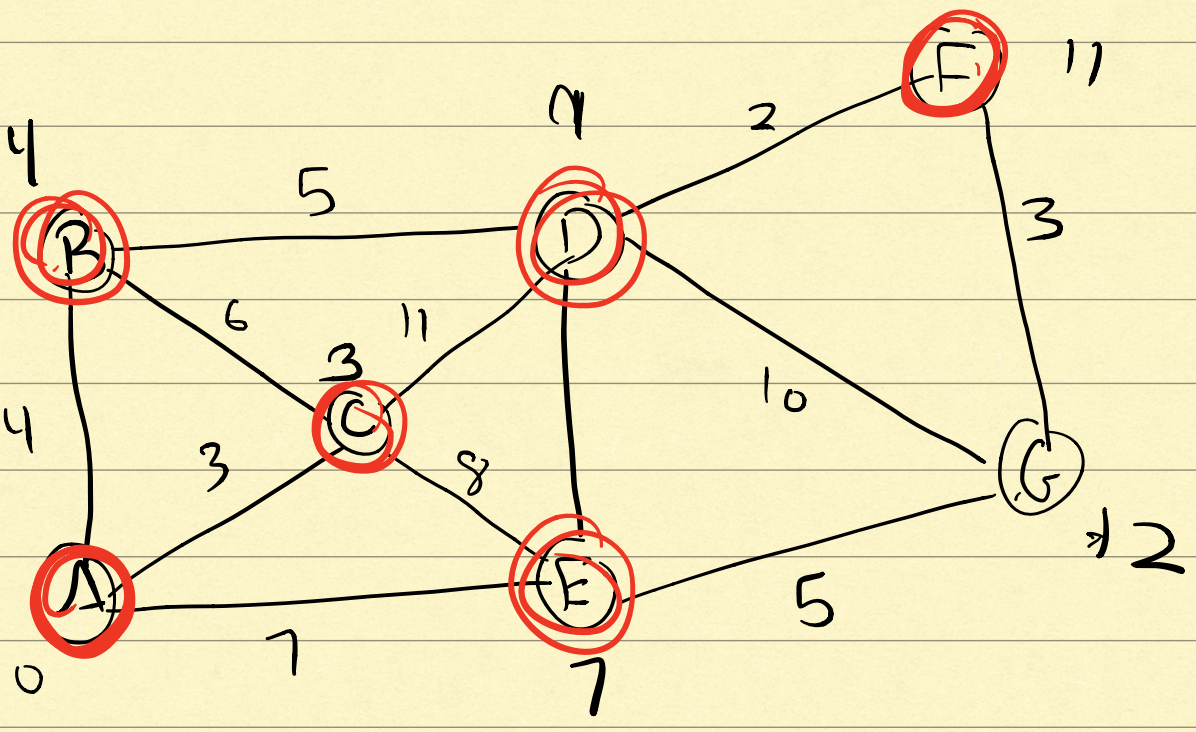
- Check unvisit node
- only update if new dist > old dis



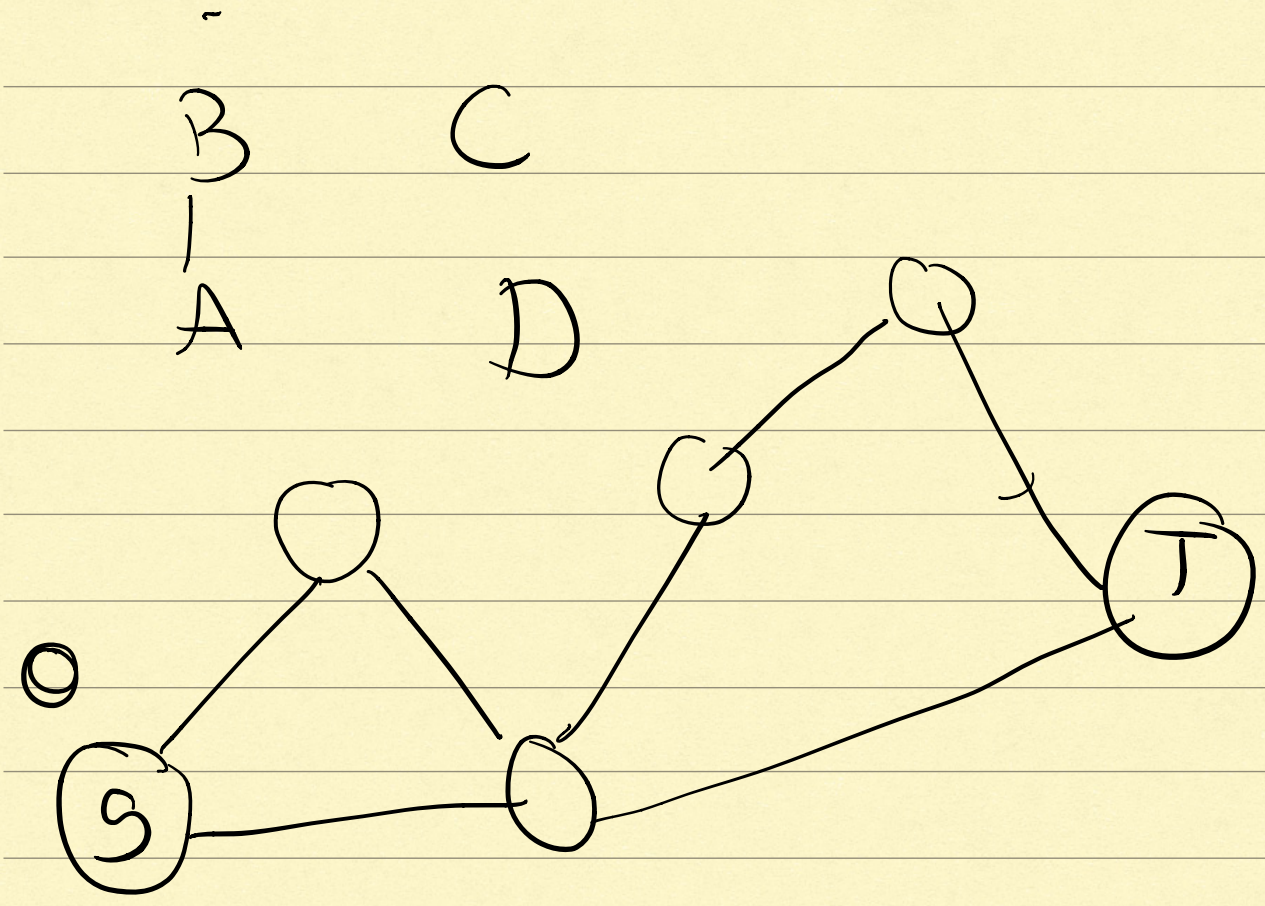
- Check unvisited node gh
- Only update if new dist < old dis



Check unvisited node



- F is the final
- retrive value and end.



S → init T- final

Assume start = 0

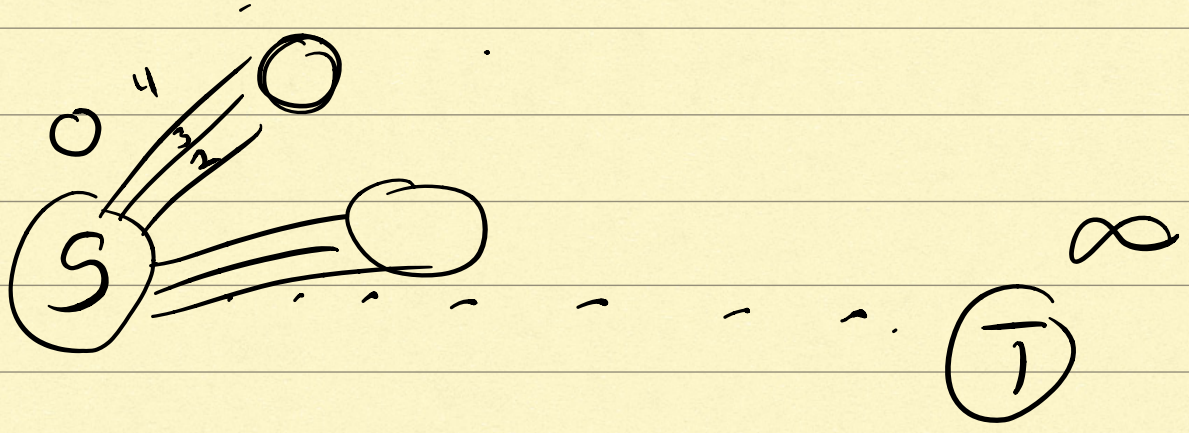
sv EV 1 n th is wei t

first train means
0 freq.

Freq every min at

- length Station = V

- visitList = false for all
- each station = ∞



for all stations

- find notVisited
- mark as visited
- waiting depend if train is available

for current visited check each edge and update

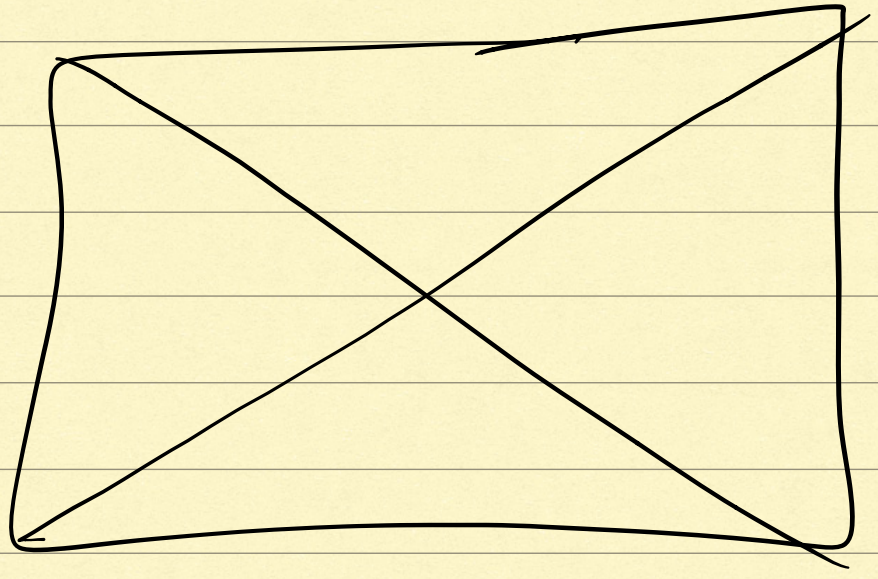
- if (time at station > first tra
  only one pat)

  wait = station time +
  length to station

else

else L

wait = station time
+ length each station
+ length freq trains
-

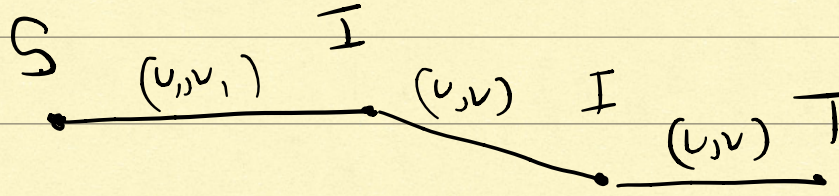


	1	2	3	4
1	0	2	3	4
2	2	0	5	6
3	3	5	0	7
4	4	6	7	0

(1,1) (2,2) (3,3) (4,4) -0

Solving problem.

- Start at Oak city
- map connected graph with edge (U,V)



- length(e) = value of edges in min
  first(e) = first train traveling
  freq(e) > 0
- time arrive = time train arrive ✓ ok to board
- remember arrival x < y must wait till x = y.

myShortestTravelTime
(Start Vertex, end Vertex, start time in min, length(U,V), first train, freq on to V from U.

- length(U.V).length Should give me # verti
- I want store the Shortest time So int[]