Homework 5 : Q1.

Q1) Given Algo

① choose any point u_1

② for $i = 2$ to k :
choose a point u_i such that it is far away from all selected points $(u_1 \ldots u_{i-1})$
Add it to selected points

③ These $u_1 \ldots u_k$ will be cluster centers & assign all points in X to ~~the~~ ~~center~~ ~~such that~~ ~~that~~ closest centers.

The above algorithm although doesn't provide optimal solution, it provides us with a approximate solution with a factor of 2.

ie. $\text{Cost}(T) \leq 2\,\text{cost}(T^*)$

where T is the soln using above algo & T^* is the optimal solution

Proof:

Let $r = \max_{x \in S} \rho(x, T)$ be the cost of T. & x_0 be the point at which this max is achieved. It means r is the distance of

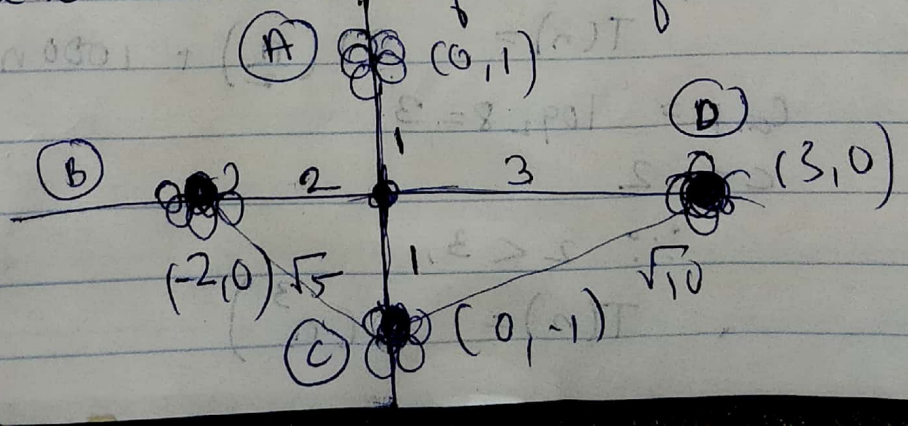
point x_0 from its closest cluster which will $\leq$ distance of all other points from each other in their cluster. This is true since if the dist bet 2 points was greater than r, it must have been already included as the cluster center as we use greedy here.

∴ Now, T is the set of all k cluster centers which are all $> r$ apart

∴ $T \cup \{x_0\}$ consists of $k+1$ points that are $\geq r$ apart.

But T^* has k points. ∴ It means 2 points in T must have same closest representative in T^* since $|T^*| = k$. which means 2 points at a dist $\geq r$ are assigned same representative in T^*. This means $\text{cost}(T^*) = r/2$.

We can see this from fall example:

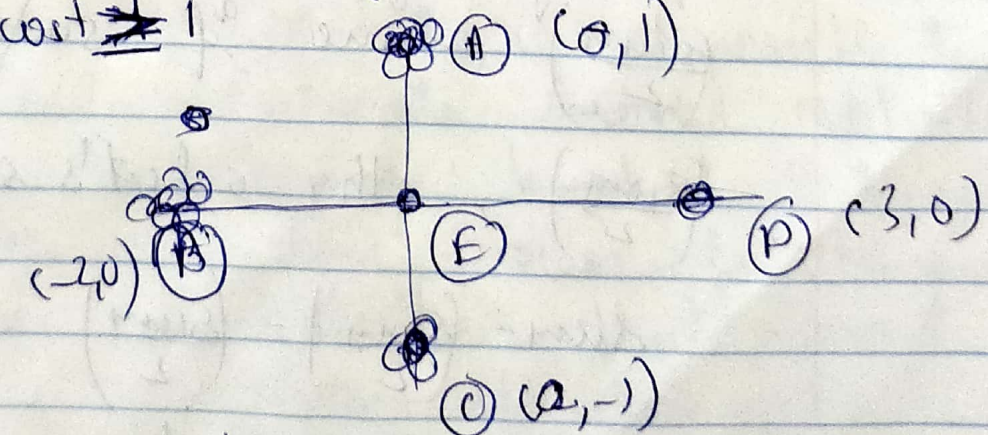


In the above figure we see that, The algo will choose B, C & D as clusters since they are far away from each other (5, $\sqrt{5}$, $\sqrt{10}$) which are all > 2.

~~Go to~~ & choosing those as clusters ~~will~~ is not optimal since points near (A) ~~with~~ are far from all clusters Optimal sol^n will be to choose origin as 3rd cluster instead of (C).

Say, choosing (C) gives a cost of 2 since $d((C), (A)) = 2$ (since points in (A) are closest to (C), ~~#~~ (C) will Represent those)

But, choosing Origin as 3rd cluster will give cost ~~>~~ 1



~~will~~ If we shift cluster ~~for~~ center from (C) to (E), cost of ~~+~~ will be 1. which is 1/2 of ~~2~~ what we got when cluster center was (C). Also it can anything greater than 1. eg:- When

Our cluster centre is at $(0, -\frac{1}{2})$ the cost will be $\frac{3}{2}$ which will be greater than 1. ie $\geq \frac{r}{2}$ where $r = 2$ which is the cost when c was the centre.

∴ we see it gives upper bound with a factor of 2.

& The above example proves that it doesn't give optimal sol^n but an approximate solution.

Homework 5: Q 2

(Q4) In ASP, every activity has a start time & end time associated with it.
Here, we add a reward for each activity.
We aim to find a set of activities that gives max rewards & is feasible i.e. that can be scheduled without overlap.
We use Dynamic programming to solve this.
First we sort the activities based on finish times (since that gives max no. of activities, like we did in greedy approach).
Here, we iterate on all activities & find max rewards ~~for~~ from $a_1 \ldots a_k$ for k^{th} iteration.
$\therefore$ for $k = n$, we get our answer for all activities

Algo:

Sort all activities in increasing order of finish time. (arr is array of activities)
table[0] = arr[0].profit
(arr has all activities in sorted order)

~~table(i) stores~~

Q.8)

```
for i := 1 to n-1, do
    inclprofit = jobs[i].profit
    nonconflict = find ~~job that~~ latest
                  job that is not conflicting
                  with i^th job.
if (nonconflict != -1)   // found
    inclProfit += table[nonconflict]

if(inclProfit > table[i-1]):
    table[i] = inclProfit
    inc[i] = 1
    backIndex[i] = nonconflict.

else:
    table[i] = table[i-1]
    inc[i] = 0
    backIndex[i] = i-1

result = table[n-1]
return result.
```

~~findsetofjobs():~~

~~while(1):~~

~~initialise i = n-1~~

~~while(i):~~

printSetofJobs():

```
i = n-1
while (1):
    if (inc[i] != 0)
        print job[i]
    if (backIndex[i] == -1)
        break;
    i = backIndex[i];
```

Intuition behind the algo:

Here, ~~we~~ find table[k] stores optimal profit gained by including jobs from 0 to k. ∴ table[n-1] gives optimal or maximum profit by including all jobs from 0 to n-1
One addⁿ here is that we have to print set of jobs that gives max profit for that I have made use of two arrays: incl(k) → It stores 1 if k^{th} job belongs to the set when maximising ~~jobs~~ profit for job 0 to k.
& backIndex(k) stores the index of previous job that is included in such a set.
So, while printing the set, we iterate over these arrays, & see if incl(i) is 1, which means it is included

in the set & then go to next job included in the set using backIndex[i]. This gives us all the jobs that maximises profit. For this we have to start from i = n-1, (since table(n-1) has optimal profit) & iterate till backIndex(i)=-1 since -1 means that is the last job to be included.

The printing of such jobs take O(n) ~~maxim~~ in worst case & O(1) in best case

∴ Total complexity - ~~O(n+n) = O(n)~~

O(nlogn+n) = O(nlogn)

logn → ~~since~~ If we use binary search to get latest nonconflicting job with ~~job~~ a given job.

* Using linear search gives $O(n^2)$.

Homework 5: Q3

Q3) Find Optimal TSP Tour.

Given -

	0	1	2	3	4
0	0	2	3	4	5
1	3	0	10	4	3
2	6	10	0	9	2
3	8	5	6	0	10
4	2	3	9	6	0

Here, we can see that $d_{ij} \neq d_{ji}$ which means its a directed graph.

Although ~~TSP~~ DP approach doesn't give a very good complexity, it reduces the complexity from $O(n!)$ to $O(n^2 2^n)$.

Here we use Held Karp algo, to solve the TSP problem using Dynamic Programming. This is also called a TSP (asymmetric TSP) where $d_{ij} \neq d_{ji}$.

Here, we compute costs for all subsets which gives the 2^n term in complexity.

~~The~~ Let's say, we start our tour from vertex 0.

∴ The possible subsets are - (excluding 0)

length = 1 {1}, {2}, {3}, {4} ($^4C_1 = 4$)

length = 2 {1,2} {1,3} {1,4} {2,3} {2,4} {3,4} (${}^4C_2 = 6$)

length = 3 {1,2,3} {1,2,4} {2,3,4} {1,3,4} (${}^4C_3 = 4$)

length = 4 {1,2,3,4} (${}^4C_4 = 1$)

Now, we find
$g(x, S)$ → which is ~~path~~ min cost of going to x from 0 visiting all vertices in S.
This S we construct from above subsets, we start with 1 vertex to go to all vertices.
Lastly we get $g(0, \{1,2,3,4\})$ which is cost of going to 0 from 0 visiting all vertices $\{1,2,3,4\}$.
Let c_{xy} be cost of going from $x \xrightarrow{c_{xy}} y$.

① for length = 0
$g(x, \phi)$ = cost of going to x from 0 visiting ϕ which is just c_{0x} for $x \in \{1,2,3,4\}$
$\therefore g(1, \phi) = c_{01} = 2$
$g(2, \phi) = c_{02} = 3$

$g(3, \phi) = C_{03} = 4$
$g(4, \phi) = C_{04} = 5$

(2) for length = 1

$g(x, s)$ where $|s| = 1$

$\{ s = (a) \mid a \in \{1, 2, 3, 4\} \}$ & $a \neq x$ which gives min cost of going to x from 0 visiting all vertices in s here (1).

$g(2, \{1\}) = C_{01} + C_{12} = g(1, \phi) + C_{12} = 12$
$g(3, \{1\}) = g(1, \phi) + C_{13} = 2 + 4 = 6$
$g(4, \{1\}) = g(1, \phi) + C_{14} = 2 + 3 = 5$

$g(1, \{2\}) = g(2, \phi) + C_{21} = 3 + 10 = 13$
$g(3, \{2\}) = g(2, \phi) + C_{23} = 3 + 9 = 12$
$g(4, \{2\}) = g(2, \phi) + C_{24} = 3 + 2 = 5$

$g(1, \{3\}) = g(3, \phi) + C_{31} = 4 + 5 = 9$
$g(2, \{3\}) = g(3, \phi) + C_{32} = 4 + 6 = 10$
$g(4, \{3\}) = g(3, \phi) + C_{34} = 4 + 10 = 14$

$g(1, \{4\}) = g(4, \phi) + g_{41} = 5 + 3 = 8$
$g(2, \{4\}) = g(4, \phi) + g_{42} = 5 + 9 = 14$
$g(3, \{4\}) = g(4, \phi) + g_{43} = 5 + 6 = 11$

(3) For length = 2

$g(x, S)$ where $|S| = 2$

$S = \{y, z\}$ $\quad y, z \in \{1, 2, 3, 4\}$ such that $x \neq y \neq z$

$g(x, S)$ gives min cost of going to x from 0 visiting all vertices in S (here visiting y & z).

The possibilities are $0 \to y \to z \to x$

$0 \to z \to y \to x$

we take min of these two.

z can be visited before x or y can be visited before x.

$0 \to 2 \to 3 \to 1$ $\qquad$ $0 \to 3 \to 2 \to 1$

$g(1, \{2, 3\}) = \min(c_{31} + g(3, \{2\}),\ c_{21} + g(2, \{3\}))$

$= \min(5 + 12,\ 10 + 10)$

~~This means~~ $\min(17, 20)$

$= 17$ corresponds to $c_{31} + g(3, \{2\})$

$\therefore p(1, \{2, 3\}) = p(x, S) = 3$

Here $p(x, S)$ means the last vertex we visit before x while going from 0 to x eg:- in $0 \to 2 \to 3 \to 1$ parent is 3 since we visit 3 before 1.

$0 \to 3 \to 4 \to 1$ $\qquad$ $0 \to 4 \to 3 \to 1$

$g(1, \{3, 4\}) = \min(c_{41} + g(4, \{3\}),\ c_{31} + g(3, \{4\}))$

$= \min(3 + 14,\ 5 + 11)$

$= \min(17, 16) = 16$

$\therefore p(1,\{3,4\}) = 3$

$g(1,\{3,4\}) = 16.$

$0 \to 2 \to 4 \to 1$

- $g(1,\{2,4\}) = \min(c_{41} + g(4,\{2\}),$
  $c_{21} + g(2,\{4\}))$
  $0 \to 4 \to 2 \to 1$

  $= \min(3+5, 10+14)$

  $= 8$

  $\therefore p(1,\{2,4\}) = 4$

$0 \to 3 \to 1 \to 2$

- $g(2,\{1,3\}) = \min(c_{12} + g(1,\{3\}),$
  $c_{32} + g(3,\{1\}))$
  $0 \to 1 \to 3 \to 2$

  $= \min(10+9, 6+6)$

  $= 12$

  $\therefore g(2,\{1,3\}) = 12$

  $p(2,\{1,3\}) = 3$

$0 \to 4 \to 1 \to 2$

- $g(2,\{1,4\}) = \min(c_{12} + g(1,\{4\}),$
  $c_{42} + g(4,\{1\}))$
  $0 \to 1 \to 4 \to 2$

  $= \min(10+8, 9+5)$

  $= 14$

  $\therefore g(2,\{1,4\}) = 14$

  $p(2,\{1,4\}) = 4$

$0 \to 4 \to 3 \to 2$

$g(2, \{3,4\}) = \min(c_{32} + g(3,\{4\}),$
$c_{42} + g(4,\{3\}))$

$0 \to 3 \to 4 \to 2$

$= \min(6+11, 9+14)$
$= 17$

$\therefore g(2,\{3,4\}) = 17$
& $p(2,\{3,4\}) = 3$

$0 \to 2 \to 1 \to 3$

$g(3,\{1,2\}) = \min(c_{13} + g(1,\{2\}),$
$c_{23} + g(2,\{1\}))$ $0 \to 1 \to 2 \to 3$

$= \min(4+13, 9+12)$
$= 17$

$\therefore g(3,\{1,2\}) = 17$
& $p(3,\{1,2\}) = 1$

$g(3,\{1,4\}) = \min(c_{13} + g(1,\{4\}),$
$c_{43} + g(4,\{1\}))$
$= \min(8, 6+5)$
$= 11$

$\therefore g(3,\{1,4\}) = 11$
$p(3,\{1,4\}) = 4$

$g(3,\{2,4\}) = \min(c_{23} + g(2,\{4\}),$
$c_{43} + g(4,\{2\}))$
$= \min(9+14, 6+5)$
$= 11$

$\therefore g(3, \{2,4\}) = 11$
$p(3, \{2,4\}) = 4$

$g(4, \{1,2\}) = \min(c_{14} + g(1, \{2\}),$
$c_{24} + g(2, \{1\}))$
$= \min(3+13, 2+12)$
$= 14$
$\therefore g(4, \{1,2\}) = 14$
$p(4, \{1,2\}) = 2$

$g(4, \{1,3\}) = \min(c_{14} + g(1, \{3\}),$
$c_{34} + g(3, \{1\}))$
$= \min(3+9, 10+6)$
$= 12$
$g(4, \{1,3\}) = 12$
$p(4, \{1,3\}) = 1$

$g(4, \{2,3\}) = \min(c_{24} + g(2, \{3\}),$
$c_{34} + g(3, \{2\}))$
$= \min(2+10, 10+12)$
$= 12$
$\therefore g(4, \{2,3\}) = 12$
$p(4, \{2,3\}) = 2$

(4) for length = 3

$g(1, \{2,3,4\}) = \min(c_{21} + g(2, \{3,4\}),$
$c_{31} + g(3, \{2,4\}),$
$c_{41} + g(4, \{2,3\}))$

$= \min(10 + 17,$
$5 + 11,$
$3 + 12,)$

$= \min(27, 16, 15)$

$= 15$

$p(1, \{2,3,4\}) = 4$

3

$g(2, \{1,3,4\}) = \min(c_{12} + g(1, \{3,4\})$
$c_{32} + g(3, \{1,4\})$
$c_{42} + g(4, \{1,3\})$

$= \min(10 + 16,$
$6 + 11,$
$9 + 12,)$

$= \min(26, 17, 21)$

$= 17$

$\therefore g(2, \{1,3,4\}) = 17$

$p(2, \{1,3,4\}) = 3$

$g(3, \{1,2,4\}) = \min(c_{13} + g(1, \{2,4\})$
$c_{23} + g(2, \{1,4\})$
$c_{43} + g(4, \{1,2\})$

$c_{14} + g(1,\{2,3\}) = 3 + 17 = 26$

$c_{24} + g(2,\{1,3\}) = 2 + 12 = 14$

$c_{34} + g(3,\{1,2\}) = 10 + 17 = 27$

$= \min(4 + 8,\ 9 + 14,\ 6 + 14)$

$= \min(12, 23, 20)$

$= 12$

$\therefore g(3, \{1,2,4\}) = 12$

$p(3, \{1,2,4\}) = 1$

$g(4, \{1,2,3\}) = \min(c_{14} + g(1,\{2,3\}),\ c_{24} + g(2,\{1,3\}),\ c_{34} + g(3,\{1,2\}))$ min $= (26, 14, 27)$

$= 14$

~~Length = 4~~

$p(4,\{1,2,3\}) = 2$

Here we find $g(0, \{1,2,3,4\})$ which is going from vertex 0 to 0 visiting all vertices $\{1,2,3,4\}$

$\therefore g(0, \{1,2,3,4\}) = \min(c_{10} + g(1,\{2,3,4\}),\ c_{20} + g(2,\{1,3,4\}),\ c_{30} + g(3,\{1,2,4\}),\ c_{40} + g(4,\{2,3,1\}))$

$= \min(3 + 15,\ 6 + 17,\ 8 + 12,\ 2 + 14)$

$= \min(18, 23, 20, 16)$

$= 16$

$\therefore g(0, \{1,2,3,4\}) = 16$ ... (1)

$\therefore p(0, \{1,2,3,4\}) = 4$

Now
we backtrack to get the route
Here is the use of p(x,S)
Now
the total cost of tour = 16 since
g(0,{1,2,3,4}) = 16

Now,
p(0,{1,2,3,4}) = 4
Which means 4 is the node before 0.
0 ← 4

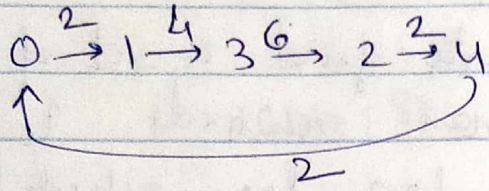
we got this from g(4,{4,3,1})
whose parent is 2
∴ node before 4 is 2
0 ← 4 ← 2

we got this from g(2,{1,3})
whose parent is 3
∴ node before 2 is 3
0 ← 4 ← 2 ← 3

we got this from g(3,{1})
whose parent is 1 since we are visiting 3 in 1

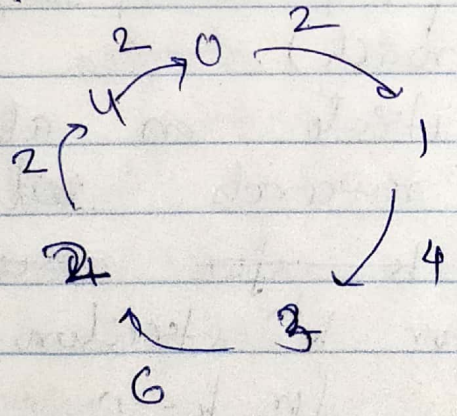
∴ 0 ← 4 ← 2 ← 3 ← 1

∴ The tour is



adding all values
we get 2+4+6+2+2 = 16 ... (2)

∴ This is the shortest Route starting from 0 & the length is 16.



0 → 1 = 2
1 → 3 = 4
3 → 2 = 6
2 → 4 = 2
4 → 0 = 2

∴ Cross checking also gives 16
∴ (1) = (2).

Homework 5 : Q4

Q4) This algo doesn't give optimal solution to TSP, but it tries to approximate TSP with MST. TSP is a NP hard problem. We can use MST here, since MST is ~~the~~ to gives us all the vertices of the graph, such that they are connected & the summation of all edges is minimum. It spans the graph. TSP also spans the graph, but it is a cycle & not a tree, Taking out an edge would give spanning tree.

∴ The solution to TSP can't have weight less than that of Minimum Spanning Tree (since there is an extra edge to return back to start point)

In other words, MST yields a lower bound on the solution to the TSP problem. It can also be used to find upper bound on TSP.

The cond" we assume here is the triangle inequality $w(a,c) \leq w(a,b) + w(b,c)$.

As per given in the algorithm if we visit all vertices in MST & take the same route while

returning, we will visit every vertex twice.

$\therefore$ Cost = $2 \times (MST)$

But taking shortcuts can avoid visiting vertices multiple times.
By triangle inequality the shortcut edges don't increase the distance.
$\therefore$ By taking shortcuts, we obtain a soln to TSP that is at most twice the weight of MST.
Since $w(MST)$ is also a lower bound on TSP, solution we have found is within a factor of 2 optimal. This means our approach is an approximation algorithm for TSP that approximates the solut^n with a factor of 2.

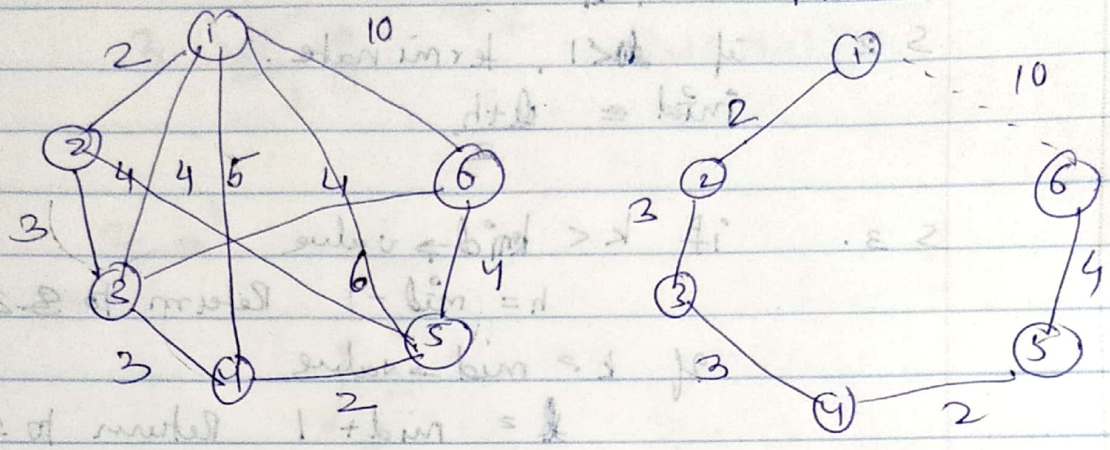
$\therefore w(MST(G)) \leq w(TSP(G)) \leq 2w(MST(G))$

$\therefore$ we can see that the difference bet^n optimal soln & our solution is no greater than the weight of MST

TSP is a NP hard pblm, we can only compute approximate Soln in Polynomial Time.

Hw: Q4

We can see this using a counter example



Now, $w(MST(H)) = 14$

instead of traversing all way back from 6 to 6–5–4–3–2–1 we take a shortcut 6–1

∴ Our Path would be 1 2 3 4 5 6 1

weight of above path = $14 + w(6,1) = 24$

whereas, optimal tour for TSP can be lesser than this.

Let's see the Path 1 2 3 6 5 4 1

Weight of this Path is $2+3+4+4+2+5 = 20$ which is less than above Path

also $w(MST) \leq 20 \leq 2w(MST)$ as $14 \leq 20 \leq 28$

28 gives upper bound for TSP & 14 gives lower bound for TSP

We see that above algo provides a approximate Soln in the form of upper & lower bound but not optimal Soln. Since