Assignment 1: Design and Analysis of Algorithms

06 February 2019 10:15

Name: Animesh Patni

A#: A20403240

Problem 1:→

A → Input array
n → number of elements.

Pseudo Code:→

A → Array
x → Indices of array.
y → Indices of array.
i → location of the i^{th} element.

```
1.  SELECT(A, x, y, i):
2.      IF (x<y):
3.          m = BLACK-BOX(A, x, y)
4.          Q = PARTITION(m)
5.          Z = Q-x+1
6.          IF (i<Z):
7.              RETURN SELECT(A,x,Z-1,i)
8.          ELSE IF (i>Z):
9.              RETURN SELECT(A,Z+1,y,i)
10.         ELSE IF (i==Z):
11.             RETURN A[Z].
12.     ELSE - IF (X==Y):
13.             RETURN A[X].
```

Black box takes in the array with the indices x, y & returns the value of median element of A[x....y]. The above

algorithm uses partition algorithm that takes the median as an input & partition or divide the array A around it. As m is the median of array A[x.... y] each of the subarray from the partition will have either A[x.... q-1] or A[q+1 ... y] containing half number of elements of A[x.... y].

The recurrence equation will be something as follows :→

Median → $O(n)$

Partition → $O(n)$

If conditions → $O(1)$

Number of element → $T(\lfloor \frac{n}{2} \rfloor)$

So, the time complexity will mostly be dependent on Median, partition and elements ie

$$T(n) = O(n) + O(n) + T\left(\frac{n}{2}\right)$$

$$= O(n) + T\left(\frac{n}{2}\right)$$

$$= O(n) + O\left(\frac{n}{2}\right) + O\left(\frac{n}{4}\right) \cdots O(1)$$

$$= \underline{O(n)} \text{ in worst case manner.}$$

Problem 2:→

a.) We need to show that the given two condition are satisfied. ie

$$\sum_{x_i < x_k} w_i < \frac{1}{2} \quad \text{and} \quad \sum_{x_i > x_k} w_i \leq \frac{1}{2}.$$

As given weight assigned to each element $w_i = \frac{1}{n}$.

Case 1:→ n is odd

This is fairly simple case as all the elements from 1 ---- n has the same weight ie $\frac{1}{n}$, then x_k will be the median as well as the weighted median for the element $x_1, x_2 \cdots x_n$.

Suppose we have 5 elements x_1, x_2, x_3, x_4, x_5 with a weight of $\frac{1}{5}$ each as $n = 5$. So, x_3 will be the median. ~~assuming that they are placed in sorted ascending order~~. So, for computing the weighted median, we sum all the elements on the left of x_3

& right of x_3 & if the sum of weights of each side crosses $1/2$ after adding x_3, this proves that x_3 is the weighted median as well.

$$W[L] = x_1 + x_2 + x_3 = 0.6$$

$$W[R] = x_3 + x_4 + x_5 = 0.6$$

If, we remove x_3 from both the weights our condition is satisfied.

case 2: n is even:→

For this case we take in the left bracket $\left\lfloor \frac{n+1}{2} \right\rfloor - 1$ elements & on the right side $n - \left\lfloor \frac{n+1}{2} \right\rfloor$.

So $\sum_{x_i < x_k} w_i = \frac{1}{n}\left(\left\lfloor \frac{n+1}{2} \right\rfloor - 1\right)$ &

for the right side $\sum_{x_i > x_k} w_i = \frac{1}{n}\left(n - \left\lfloor \frac{n+1}{2} \right\rfloor\right)$

which by solving comes to $\sum_{x_i < x_k} w_i < \frac{1}{2}$ &

$\sum_{x_i > x_k} w_i \leq \frac{1}{2}$. For example suppose we have 6 elements $x_1, x_2, x_3, x_4, x_5, x_6$ with a weight of $\frac{1}{6}$ each as $n = 6$.

So, the median for the above case would be x_3. So for $x_i < x_3$ we have

weight $\rightarrow w[L] = w[x_1] + w[x_2]$

$= \frac{1}{6} + \frac{1}{6} \Rightarrow 0.33$

if we add x_3 into the above it becomes exactly 0.5. Thus without x_3 the weight is less than $\frac{1}{2}$ thus satisfying the condition $\sum_{x_i < x_k} w_i < \frac{1}{2}$, now similarly for the right side

$w[R] = w[x_4] + w[x_5] + w[x_6]$

$= 1/6 + 1/6 + 1/6 = 0.5$

if we adding $w[x_3]$ to the above

.

it satisfies the condition $\sum_{x_k > x_i} w_i \leq \frac{1}{2}$

prooving that if the weight of all the elements are the same, then the weighted median is similar to the median of the given elements.

b.) As we first need to sort the array A & then compute the median & weighted median, the cost of sorting the array in the worst case manner using heap sort, merge sort is $O(n \lg n)$ Now for computing the weighted median we need to scan for the element that makes the sum of weight to cross $\frac{1}{2}$ for the element $x_i < x$ as well as $x_i > x$, so this scanning through the element will take $O(n)$ time. So, the total running time should be $O(n \log n)$, containing sorting as well as scanning in the worst case.

c.) Let's do this by SELECT Algorithm mentioned in question 1. Let x be the medians of medians. Then compute both the sides $\Sigma x_i < x$ & $\Sigma x_i > x$ w_i & check for the weighted median condition. If they are not large than $\frac{1}{2}$ then this is our final solution otherwise recursively call the select algorithm, with the partitioned sub-array smaller or larger depending on where the weighted median lies. The SELECT algorithm runs in $O(n)$ times which is what we are looking for. Pseudo-code would be similar to that of question 1

```
1. SELECT_WEIGTHED_MEDIAN (X):
2. If there is only one element in X return.
3. else
4. find the median of x.
5. Partition around the median
6. compute weight for left & right.
7. If w[left] as well as w[right] < 1/2
8.     return the median x[m].
```

9. else if $w[right] > 1/2$.

10. add weight of left to the median

11. new array $X_1 = x\left[n - \left\lfloor \frac{n+1}{2} \right\rfloor\right]$

12. recursursively call with X_1.

13. else if $w[left] > \frac{1}{2}$

14. add weight of right to the median

15. new array $x_2 = x\left[\left\lfloor \frac{n+1}{2} \right\rfloor - 1\right]$

16. recursively call with x_2.

So, the worst case recursive call for the above modified select algorithm will take $O(n)$.

d) So, the weighted median can be the best solution for the post-man problem, if the number of elements on the left side of p_k (weighted median) & the right side of p_k are the same. As in this case the post office sits in between & have equal distance from point a as well as b. This

case only works if points are odd in number. If the points ie n is even, then eventually there would be an inbalance as the distance from left side will be less than the distance of the right side.

e.) When it comes to 2-dimensional problem, we need to find cost of each axis separately & then decide on the location

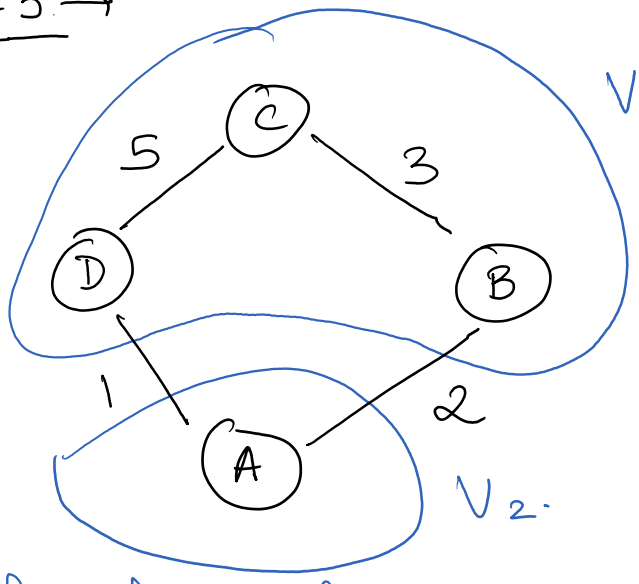
$$cost_i = \sum_{i=1}^{n} w_i (|x - x_i|) + (|y - y_i|).$$

So, we need to find a point p that minimizes the cost, so find the minimum of both the x-axis & y-axis would give us the best solution, so by calculating the weighted median of the x-axis ie x_k & y-axis ie y_k, we get a point (x_k, y_k) which should be the best

solution for the 2-dimensional problem
The final equation being:

$$\min cost(x,y) = \min_x g(x) + \min_y g'(y).$$

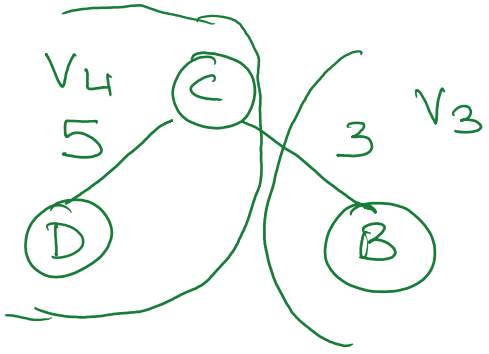
Problem-3:→



Taking the above partition we have

$V_1 = \{B, C, D\}$ $V_2 = \{A\}$

Now taking V_1 & again partitioning it randomly



$V_3 = \{B\}$

$V_4 = \{D, C\}$

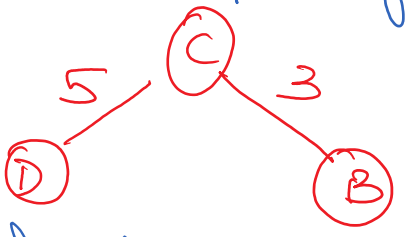
Now taking V_4 & again partitioning it

$V_5 = \{C\}$

$V_6 = \{D\}$

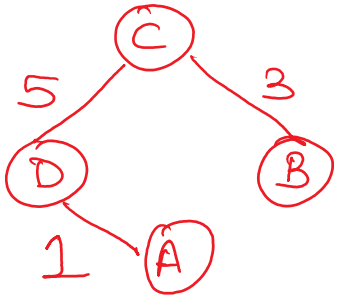
Now recursively adding edges back to form a spanning tree, as the last edge divided was $D \to C$ we add it to the spanning tree

Going one level up, we add next edge ie $C \to B$ into the spanning tree.



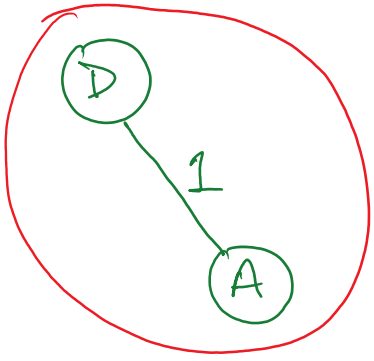
Now, uniting the final set into one by choosing the minimum edge. So, the MST if the partition is done over A is:→

Fig-1

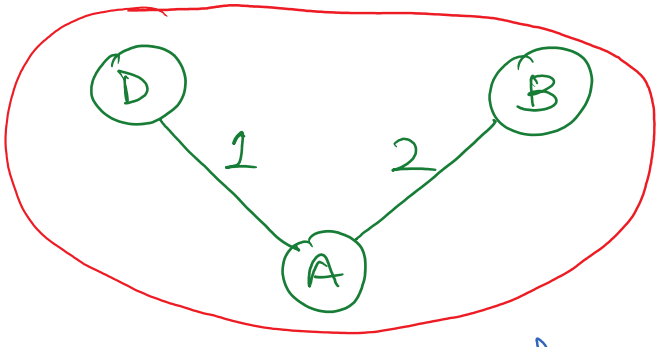


COST = 9

Now, computing MST without partitioning it into 2, for the same graph G. Taking minimum edge weight into the set ie A → D



Now from the remaining vertex we search for the minimum edge that would be A → B & will put it into a set



Now again looking for the minimum edge that is not in set which is B → C & putting it into the set, as there are no more vertexes left

D → A → B → C is the MST for the whole graph G_1 with a cost of 6 shown below.

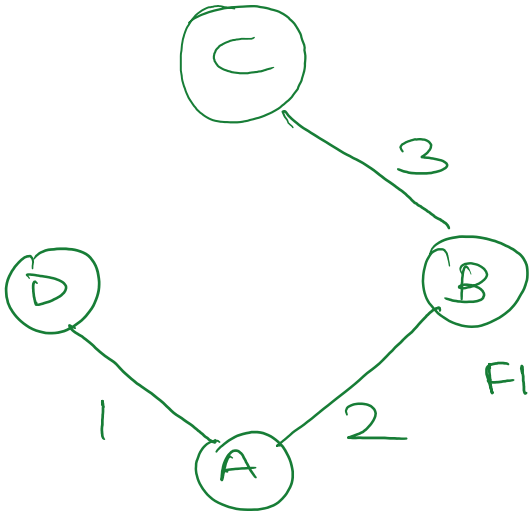


FIG 2.

Figure 1 computes MST with a cost of 8 while figure 2 does it in 6. This clearly shows that the algorithm fails to produce a minimum spanning tree if the partition is done in worst case manner. The algorithm of partitioning is failing in this case because, we are just randomnly picking the vertex out of G_1, which makes the scenario fail to produce a optimum MST.

Second part :→ 3-2.

We would go ahead and this through divide & conquer approach.

(1) Divide the graph into two sets of vertices V_1 & V_2. V_2 having one 1 vertex

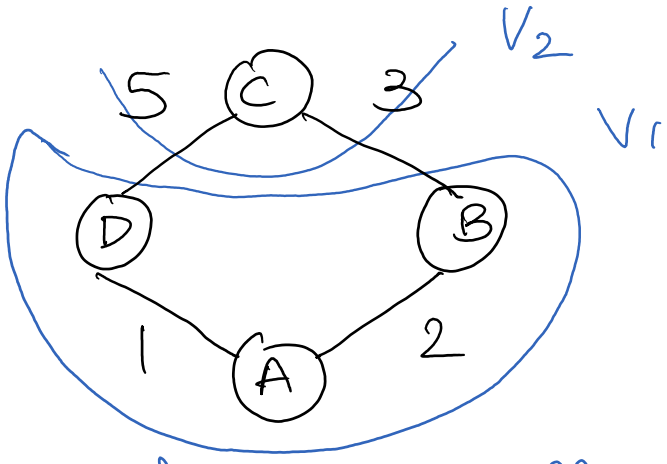
(2) Part on the nodes recursively unless V_1 is partitioned in such a way that all the sets have 1 vertex in it.

(3) Now, select the last partitioned vertex which would have the least cost & connect it to the vertex.

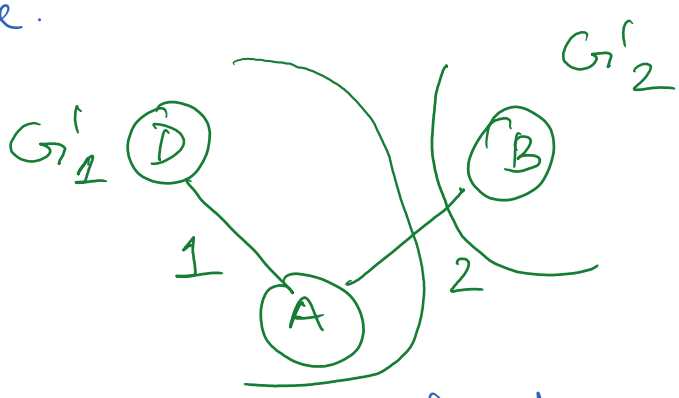
(4). Recursively keep on doing step 3 till we cover all the vertices & that will be our final MST.

For the best case part we will pick the best possible cut in Graph G.

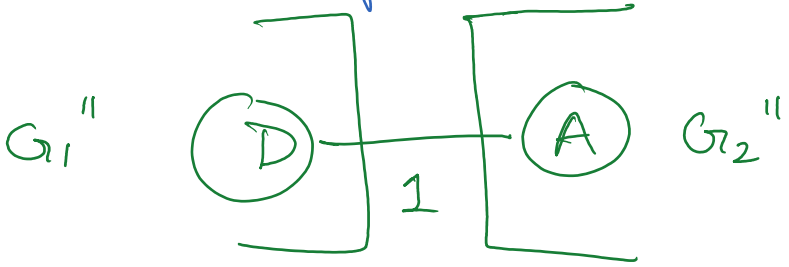
V_2



Now, moving ahead we will again partition G_1 into G'_1 & G'_2 in which G'_2 have one vertex.



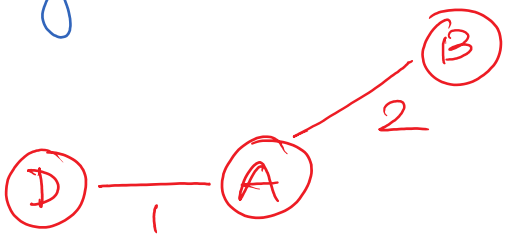
Now, we will again partion G_1' to G_1'' & G_2'' & see that there is one single element left in all the graphs.



After partitioning is done we will recursively take the edges that have the

smallest weights. ie taking the smallest edge to unite the spanning tree.

Now after connecting the above 2 we look at how to connect G_1 & G_2 with the minimum edge ie



After this we connect G_1 & G_2 with the minimum edge & unite the two spanning tree to form a MST.
So, the minimum edge from B → C & D → C is B → C with a weight of 3.

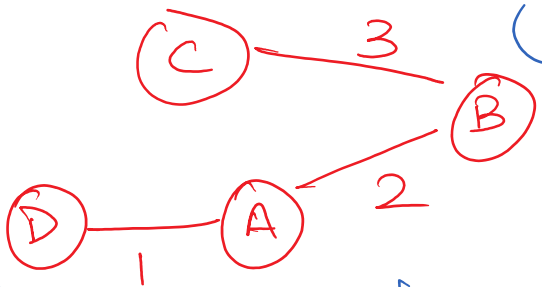


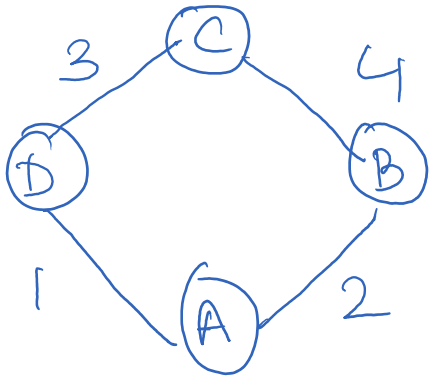
Fig 3.

Thus forming the MST with a weight of 6. So, this algorithm produces a

MST cause we make sure that the last two vertices left in the MSTs has minimum edges & starting partition is done on the vertex that has max. edge weight. Then from there we recursively add edges to tree choosing each edge greedily & uniting all partitions into one. As comparing weights of Fig 1 & Fig 2 we clearly see that if the cut is done in the best case manner it produces a MST.

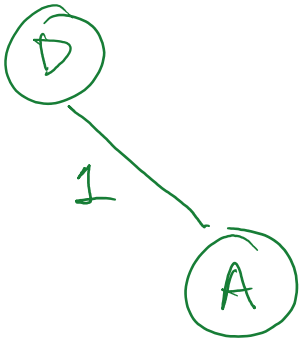
Problem 4:→

Part 1:→

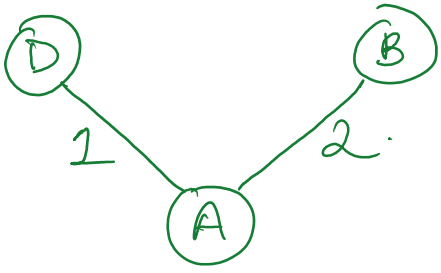


So for calculating the minimum spanning tree, we use Kruskal's &

take the minimum edge A → D & put it in the set.



Now, we take the next minimum edge that is not in set A → D, so we take in A → B & put it in the set.



From the remaining edges we pick D → C and put it in the set. so the final MST with a cost of 1 + 2 + 3 = 6.

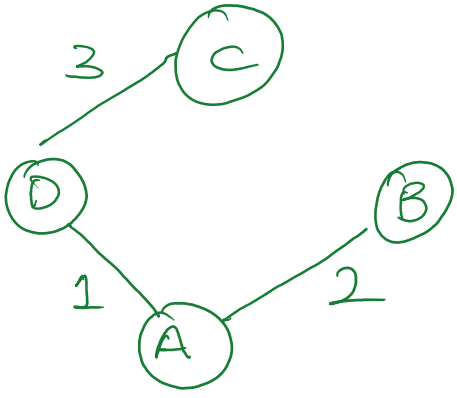
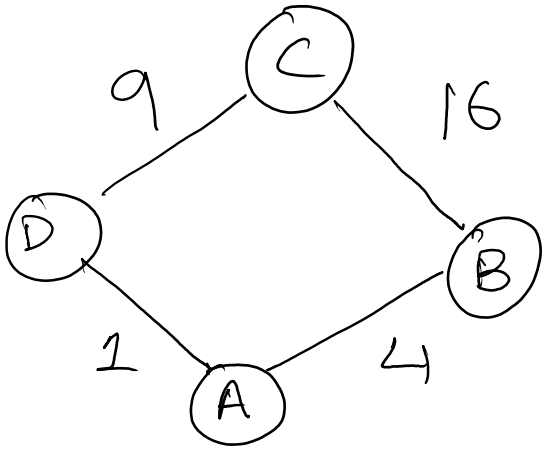
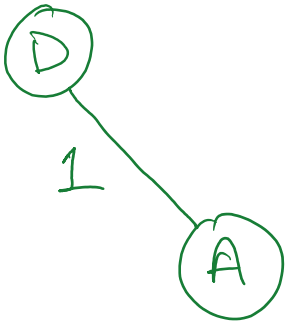


Fig 1.

Now, if we square the cost of each edge of the graph, then G' will become :→

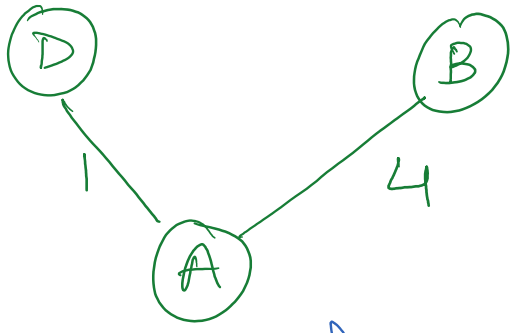


Computing MST for the above graph, taking A → D as it is the minimum edge



now, taking A → B as it minimum edge

present which are not in set & pushing it into the set



The minimum edge that is not in set is D → C, adding it to the set which eventually becomes our MST.

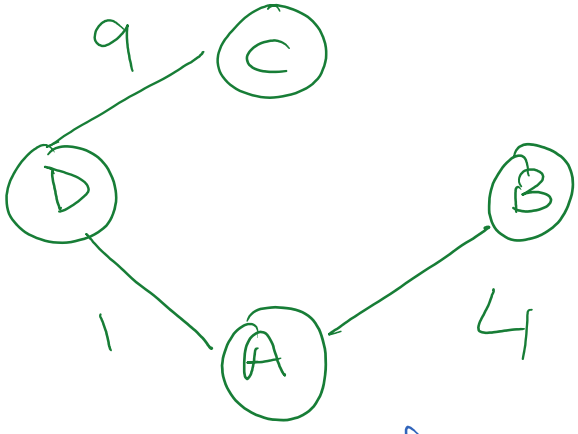


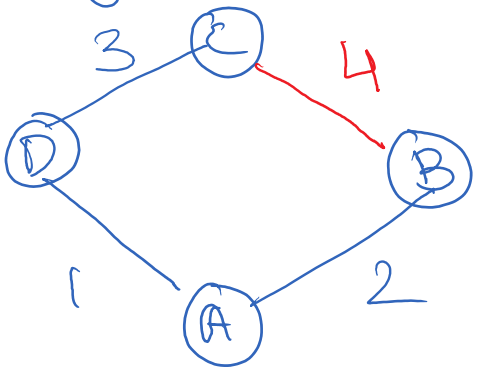
Fig 2.

with a cost of 14, thus proving the MST will remain the same (fig 1 & fig 2), while the cost would increase from 6 to 14.

PART 2: →

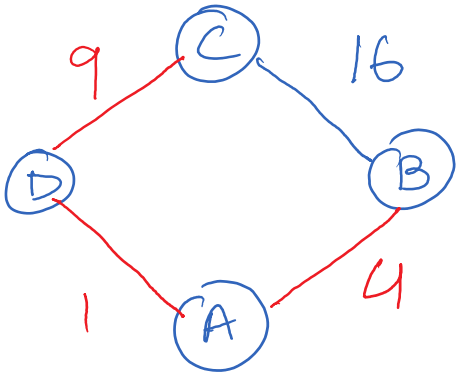
Suppose, the shortest path from B → C is 4 in the original graph, so when the weight changes from $c \rightarrow c^2$ the shortest path would change to B → A → D → C, whose sum will be 14. as the weight from B → C would go to 16.

Original Graph



4 < 6

New Graph



14 < 16

So, this disproves that the shortest path will remain the same.