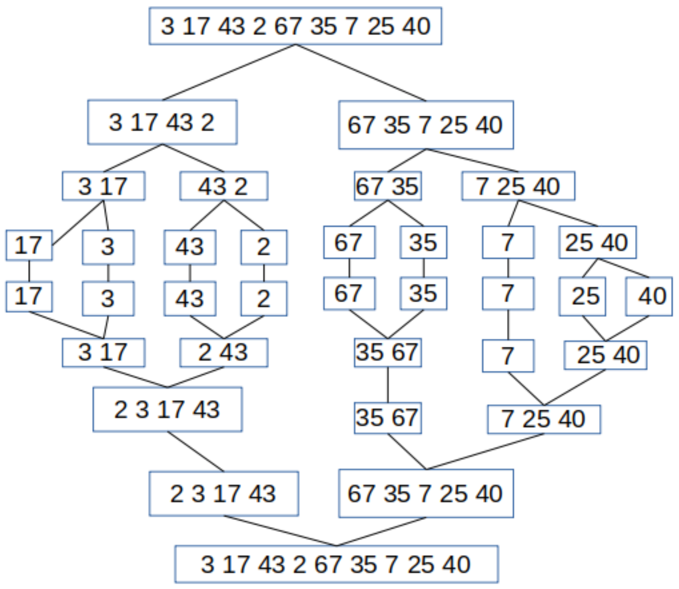
Matt Fletcher CS317-02 Homework 1

1.



2.

Three steps:

a) Move R to left, and L to R until $R < P$ and $L > P$.

b) If during (a) R and L did not cross, swap R and L, and continue (a).

c). If during (a) R and L did cross, swap R and P, freeze P in its new place, split the list in 2 around P, and recursively continue (a) on each new list.

An asterisk next to an element indicates that it is frozen.

P	L							R
43	21	90	8	44	35	6	2	13
P		L						R
43	21	90	8	44	35	6	2	13
P		R						L
43	21	13	8	44	35	6	2	90
P		L						R
43	21	13	8	44	35	6	2	90
P				L			R	
43	21	13	8	44	35	6	2	90
P				L			R	
43	21	13	8	2	35	6	44	90
P						R	L	
43	21	13	8	2	35	6	44	90
R						P	L	
6	21	13	8	2	35	43	44	90
P	L				R		PL	R
6	21	13	8	2	35	43*	44	90
P	L			R			PR	L
6	21	13	8	2	35	43*	44	90

```
P         R         L
6         2         13        8         21        35        43*       44*       90

                    P         L                   R
2         6*        13        8         21        35        43*       44*       90

                    P         R         L
2         6*        13        8         21        35        43*       44*       90

                                        PL        R
2         6*        8         13*       21        35        43*       44*       90

                                        PR        L
2         6*        8         13*       21        35        43*       44*       90


2         6*        8         13*       21*       35        43*       44*       90
```

All elements are either frozen or single element lists, therefore, the list is sorted.

3.

Five applications of graph theory: social media connections, navigation, cellular connections to towers, circuit design, register allocation

---

4. A directed graph has edges which indicate the direction they travel. The direction of travel cannot oppose the node. On the other hand, an undirected graph allows travel both ways along any edge.

5.

a) Edges: ac, cf, ab, be, bd

Nodes:a,b,c,d,e,f

b) Edges: fc, ca, ba, be, bd

Nodes: a,c,f,b,e,d

6.

a) Adjacency List

```
[ a ]-->[ b ]-->[ c ]-->[ d ]-->[ e ]-->[ f ]-->[ g ]
  |       |       |       |       |       |       |
  v       v       v       v       v       v       v
[->b]   [->d]   [->f]   [->b]   [->b]   [->c]   [->e]
  |       |       |       |       |       |       |
  v       v       v       v       v       v       v
[->c]   [->e]   [->a]   [->0]   [->g]   [->g]   [->f]
  |       |       |               |       |       |
  v       v       v               v       v       v
[->0]   [->a]   [->0]           [->0]   [->0]   [->0]
          |
          v
        [->0]
```

a) Adjacency Matrix

	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	1	0	0	1	1	0	0
C	1	0	0	0	0	1	0
D	0	1	0	0	0	0	0
E	0	1	0	0	0	0	1
F	0	0	1	0	0	0	1
G	0	0	0	0	1	1	0

b) Adjacency List

```
[ a ]-->[ b ]-->[ c ]-->[ d ]-->[ e ]-->[ f ]-->[ g ]
  |       |       |       |       |       |       |
  v       v       v       v       v       v       v
[->b]   [->d]   [->a]   [->0]   [->0]   [->c]   [->0]
  |       |       |                       |
  v       v       v                       v
[->g]   [->e]   [->0]                   [->g]
  |       |                               |
  v       v                               v
[->0]   [->0]                           [->0]
```

b) Adjacency Matrix

	A	B	C	D	E	F	G
A	0	1	0	0	0	0	1
B	0	0	0	1	1	0	0
C	1	0	0	0	0	0	0
D	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0
F	0	0	1	0	0	0	1
G	0	0	0	0	0	0	0

---

7. Traversal order:
A 1/16
B 2/9
C 10/15
D 3/4
E 5/8
F 11/14
G 12/13
Stack order:
Top →→→ →Bottom
$\boxed{GFCHEDBA}$

---

8. Traversal order:
A 1
B 2
C 2
D 3
E 3
F 4
G 5
Queue:
Front →→→ →Back
$\boxed{ABCDEFG}$

9.
Efficiency of Matrix: $\Theta(\|V^2\|)$
Efficiency of List: $\Theta(\|V\| + \|E\|)$

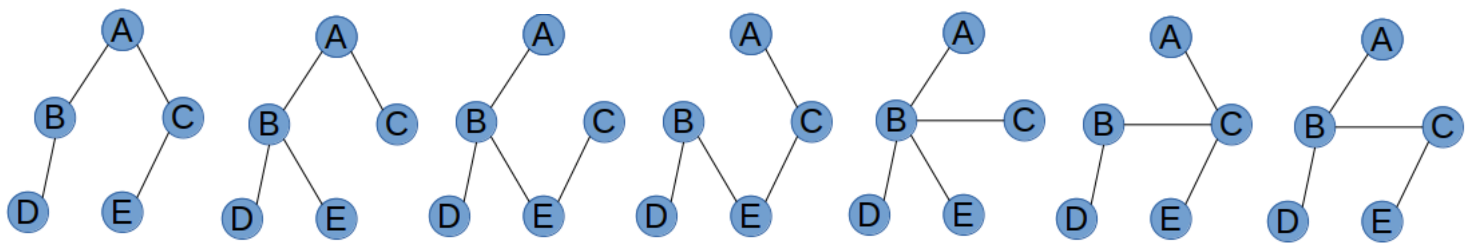
---

10. The greedy technique builds a solution piece by piece by picking the most obvious benefit first, without ever reconsidering a decision. It generates a globally optimal solution from a locally optimized choice.

TODO Greedy algorithm and the greedy choice property.

---

11. A spanning tree is a set of edges that connects every node in the graph without forming loops or cycles.



TODO FIND ALL SPANNING TREES.

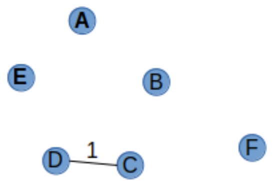
---

12. A minimum spanning tree (MST) is the smallest tree that connects every node in the graph. If the graph is weighted, the smallest tree has the smallest sum of weights. If the graph is unweighted, the MST has the smallest number of edges.
An example of a minimum spanning tree is determining the minimum amount of wire needed to connect multiple nodes.

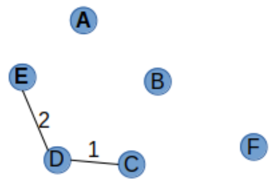
13.
Kruskal's Algorithm:
→1. Create a table with each edge sorted in ascending order.

Edge	CD	DE	AD	AE	BC	AB	BE	BD	AC	BC
Weight	1	2	3	4	5	6	6	8	9	11

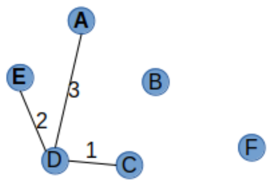
→2. Starting from either node on the smallest line, attach the next smallest line that does not create a loop.
Start by attaching line CD, weight 1.



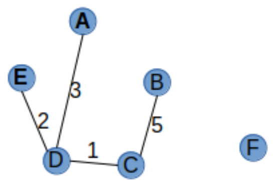
Then, find the smallest weighted edge from any end node that does not create a cycle. This is edge DE.



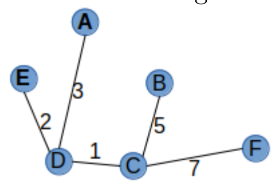
Same as before, add edge AD.



Although AE is the next smallest edge, adding it would create a cycle. So we add BC instead.



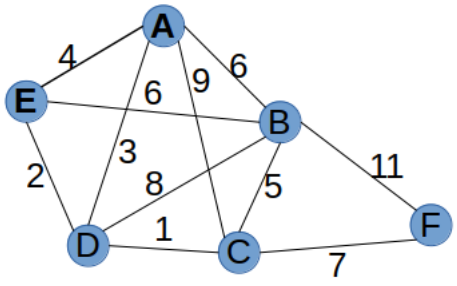
Add the next edge.



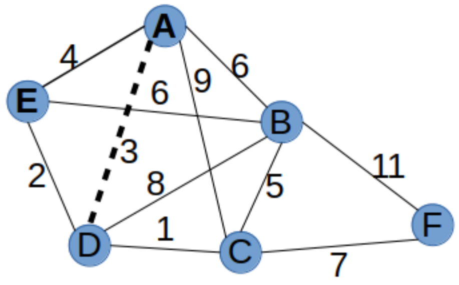
At this point, all the nodes are connected and adding any more lines would result in a cycle being created. Hence, we are done.

Prims algorithm starts from some vertex (Vertex A in this example) and finds the smallest addition that can be added to the current tree in order to not create a cycle.

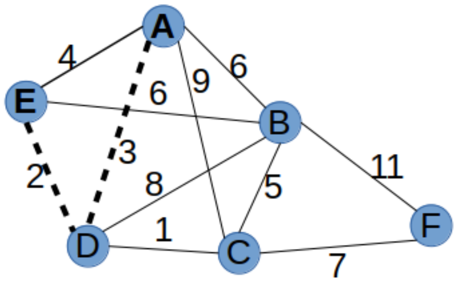
A dashed line indicated that edge has been selected.



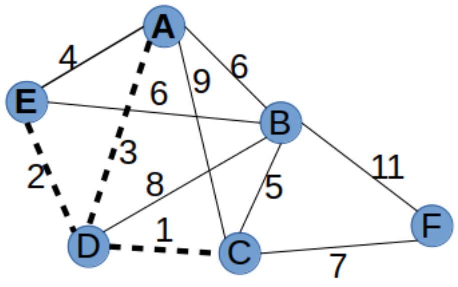
Start at vertex A. There are 4 edges that can be selected: AE, AD, AC, and AB. AE has the lowest weight, so we select that edge.



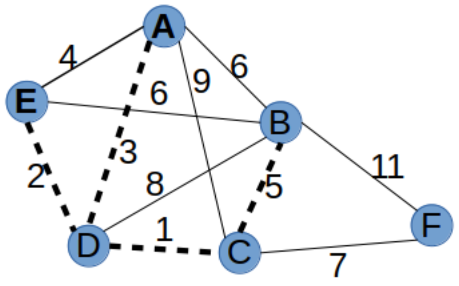
Now, we check all of the edges coming out of both vertex A and E. The lowest weight gets selected, and we choose edge ED.



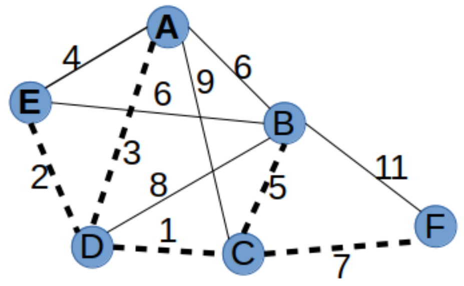
At this point, we check vertices A, E, and D. Although DA has the lowest weight, it would cause a cycle. So, we skip it and check the other edges.



Continuing...



Almost there...



And done!

We are done because all vertices have been reached, and no loops or cycles have been created. The total weight is $3+2+1+5+7=\boxed{18}$.

14.

Start at vertex A. Find the shortest path to all other vertices, and indicate in table. Use ∞ if there is no path.

V	**A**	B	C	D	E	F	G
A	0_A	7_A	3_A	$4_D A$	∞	∞	∞

C is the next unvisited vertex to visit having the smallest tentative distance. Therefore, we visit C and check if there exists a shorter path between A and immediate neighbors of C through C.

V	**A**	B	**C**	D	E	F	G
A	0_A	7_A	3_A	4_D	∞	∞_A	∞_A
C	0_A	7_A	3_A	4_D	12_C	∞_A	∞_A

Now, check B and see if any paths can be optimized or created. A route to F is possible through B, so we change the value from ∞ to 10_B.

V	**A**	**B**	**C**	D	E	F	G
A	0_A	7_A	3_A	4_D	∞	∞_A	∞_A
C	0_A	7_A	3_A	4_D	12_C	∞	∞_A
B	0_A	7_A	3_A	4_D	12_C	10_B	∞_A

D is the next unvisited node. A route through from A to E through D is now shorter than the current route of A to E through D, therefore, we replace the value.

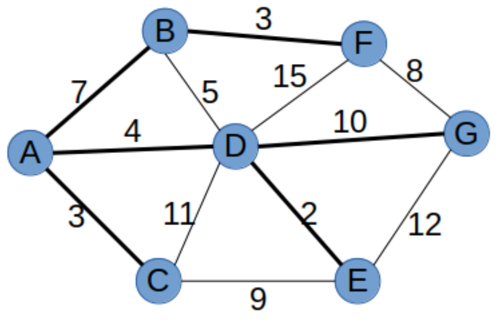
V	**A**	**B**	**C**	**D**	E	F	G
A	0_A	7_A	3_A	4_D	∞	∞_A	∞_A
C	0_A	7_A	3_A	4_D	12_C	∞	∞_A
B	0_A	7_A	3_A	4_D	12_C	10_B	∞_A
D	0_A	7_A	3_A	4_D	6_D	10_B	14_D

Next, we visit E. No routes are optimized or created during this visit.

V	**A**	**B**	**C**	**D**	**E**	F	G
A	0_A	7_A	3_A	4_D	∞	∞_A	∞_A
C	0_A	7_A	3_A	4_D	12_C	∞	∞_A
B	0_A	7_A	3_A	4_D	12_C	10_B	∞_A
D	0_A	7_A	3_A	4_D	6_D	10_B	14_D
E	0_A	7_A	3_A	4_D	6_D	10_B	14_D

Visit F, no routes optimized. Visit G, no routes optimized. Therefore, the table is finished as follows:

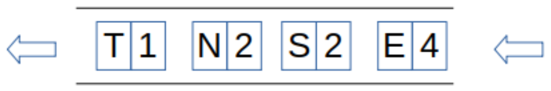
V	**A**	**B**	**C**	**D**	**E**	**F**	**G**
A	0_A	7_A	3_A	4_D	∞	∞_A	∞_A
C	0_A	7_A	3_A	4_D	12_C	∞	∞_A
B	0_A	7_A	3_A	4_D	12_C	10_B	∞_A
D	0_A	7_A	3_A	4_D	6_D	10_B	14_D
E	0_A	7_A	3_A	4_D	6_D	10_B	14_D
F	0_A	7_A	3_A	4_D	6_D	10_B	14_D
G	0_A	7_A	3_A	4_D	6_D	10_B	14_D



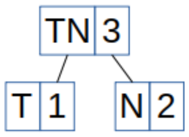
15.
Create a table of the numbers and their count:

T	N	S	E
1	2	2	4

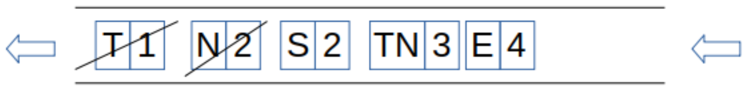
Insert these into a priority queue:



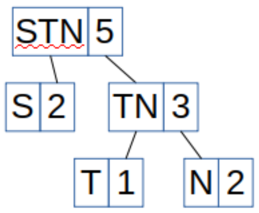
Dequeue the first 2 elements T and N, combine into binary tree:

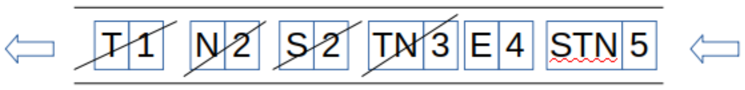


Queue the newly created element into the queue in its proper position. .

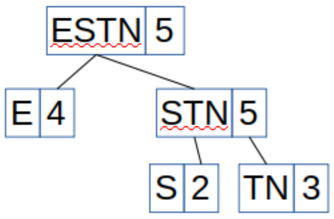


Dequeue the next 2 elements, place into binary tree, and queue the newly created element.

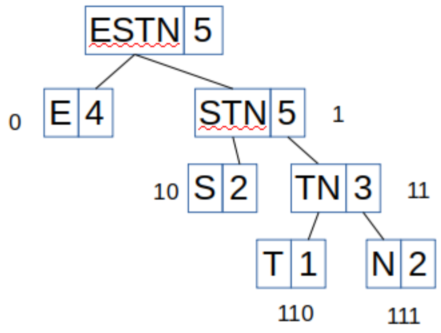




Now, dequeue the 2 elements left in the queue, and insert these into a binary tree.



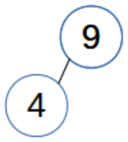
Now, we assign a value to each node. To the left is 0, to the right is 1.



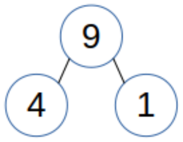
According to this, E gets replaced with a 0, S with a 10, T with a 110, and N with a 111.
Therefore, TENNESSEE gets converted to 110 0 111 111 0 10 10 0 0.
This is 17 bits, compared to the original 9 8 bit ASCII characters equaling 72 bits.

16.
Begin by creating a heap.
Array: 9,4,1,6,3,8,2,10
Take the first element, put it as the root.

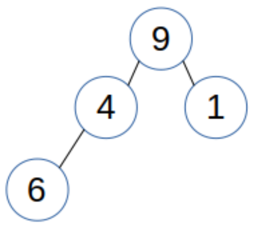
Take the next element, and attach it to the bottom left. No reheaping necessary, because $4 < 9$.



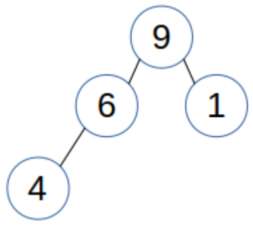
Add the next element to the bottom right. No reheaping necessary.



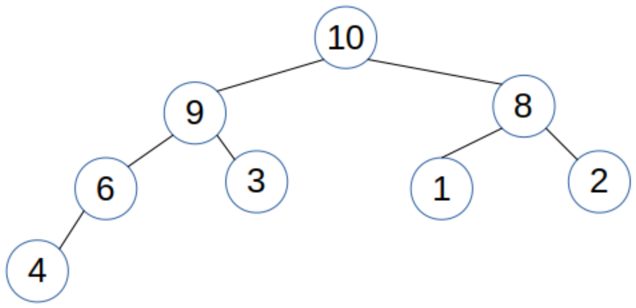
Add the next element to the bottom left of 4. Because $6 > 4$, a reheaping is necessary.



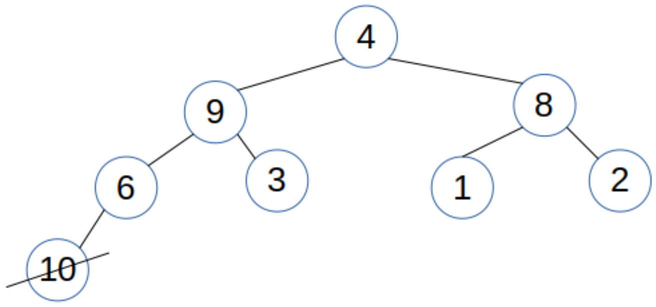
Reheaped.



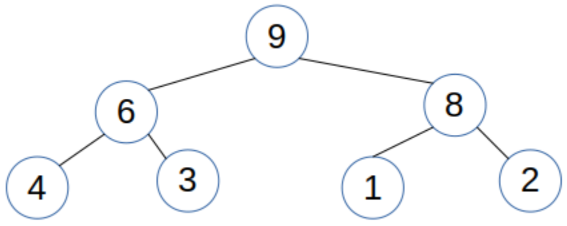
Continue this algorithm until all elements have been added to the list (some steps omitted for clarity.)



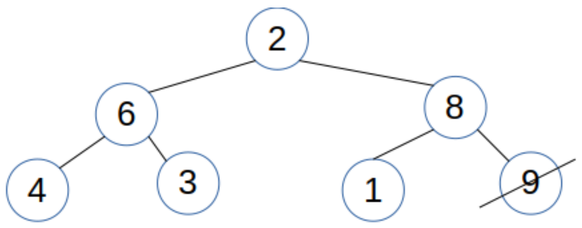
Now, swap the root element (the max, 10) with the bottom right element (2), and delete the max element. This max element is the largest in the list.

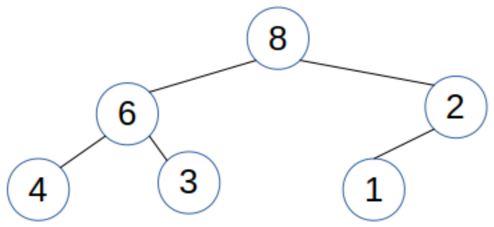


Now, reheap. From the bottom up, check if any elements are greater than their parents.

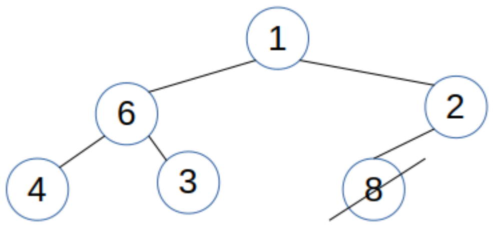


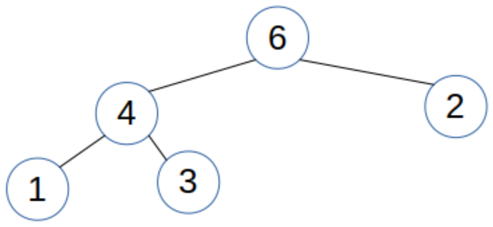
Swap max key with bottom right key, delete max key, reheap. Append this deleted key to the deleted list.



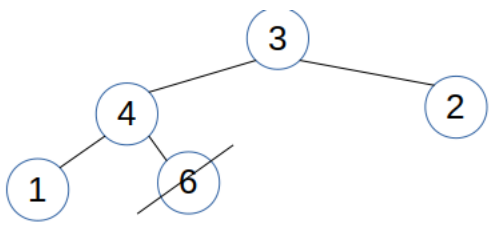


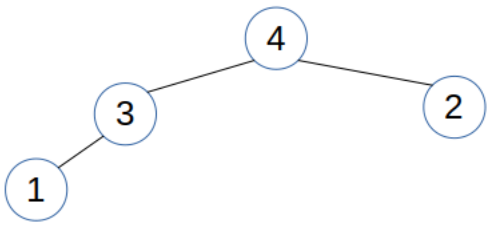
Same as before.



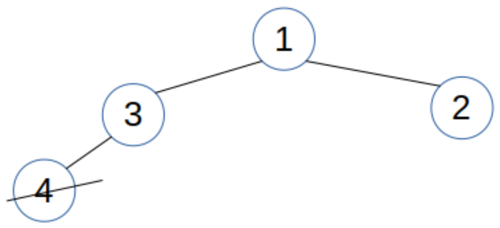


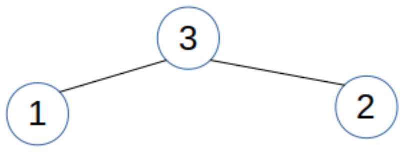
Same again.



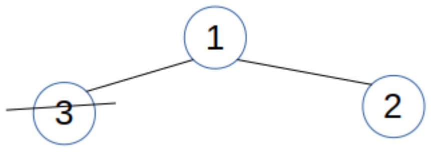


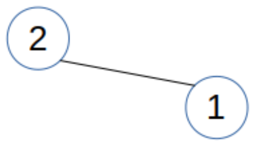
I'm getting tired.



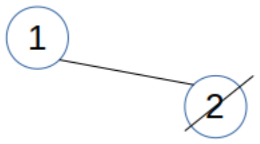


So close to done.





One more....



And we're done.
Sorted list: 1,2,3,4,6,8,9,10

17.

18.