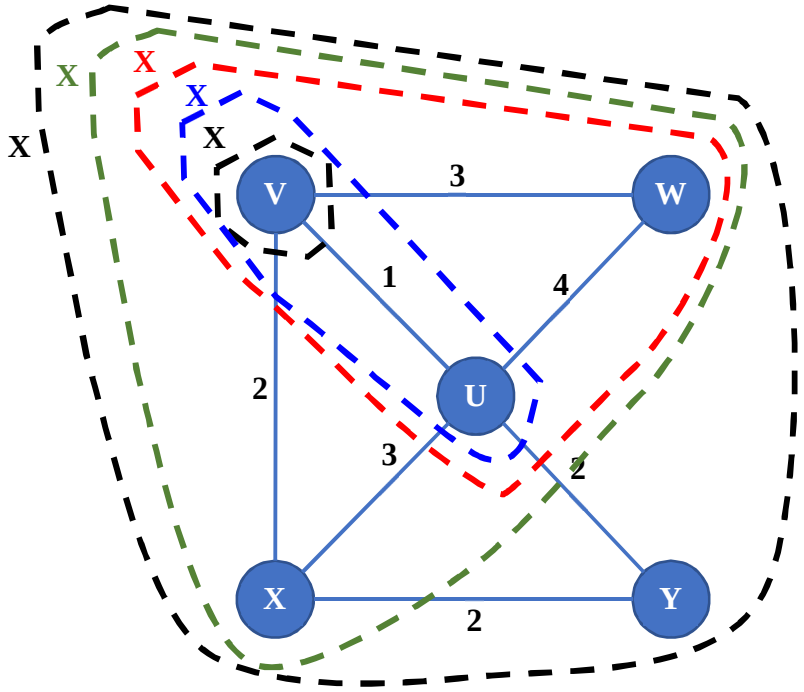
Prof. Emdad Khan

September 2019
Lab#11

Group 1
Group members:
Asad Ali Kanwal
Aser Ahmad Ibrahim Ahmad
Jean Wilbert Volcy
Zayed Hassan

1. Problem 1



Assume R is the set of vertices in the graph, i.e. R = {V, W, U, X, Y}.

Step (1):
X = {} ≠ R → continue.
Pool = {}
A[V] = 0

Step (2):
X = {V} ≠ R → continue.
Pool = {(V, W), (V, U), (V, X)}
A[V] + wt (V, W) = 0 + 3 = 3
A[V] + wt (V, U) = 0 + 1 = 1 ←
A[V] + wt (V, X) = 0 + 2 = 2

∴ A[U] = 1

Step (3):
X = {V, U} ≠ R → continue.
Pool = {(V, W), {V, X), (U, W), (U, X), (U, Y)}
A[V] + wt (V, W) = 0 +3 = 3 ←
A[V] + wt (V, X) = 0 + 2 = 2
A[U] + wt (U, W) = 1 + 4 = 5
A[U] + wt (U, X) = 1 + 3 = 4
A[U] + wt (U, Y) = 1 + 2 = 3

∴ A[W] = 3

Step (4):
X = {V, U, W} ≠ R → continue.
Pool = {(V, X), (U, X), (U, Y)}
A[V] + wt (V, X) = 0 + 2 = 2 ←
A[U] + wt (U, X) = 1 + 3 = 4
A[U] + wt (U, Y) = 1 + 2 = 3

∴ A[X] = 2

Step (5):
X = {V, U, W, X} ≠ R → continue.
Pool = {(X, Y), (U, Y)}
A[X] + wt (X, Y) = 2 + 2 = 4
A[U] + wt (U, Y) = 1 + 2 = 3 ←

∴ A[Y] = 3

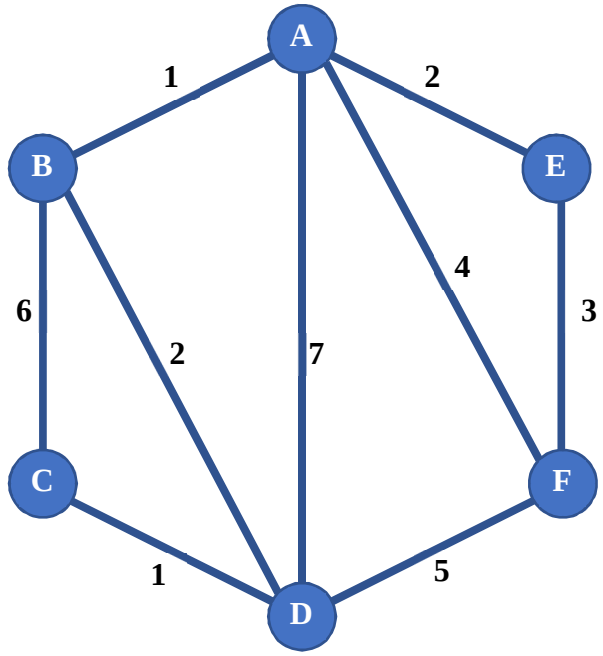
Step (6):
X = {V, U, W, X, Y} = R → stop.

∴ A[V] = 0, A[W] = 3, A[U] = 1, A[X] = 2, A[Y] = 3

---

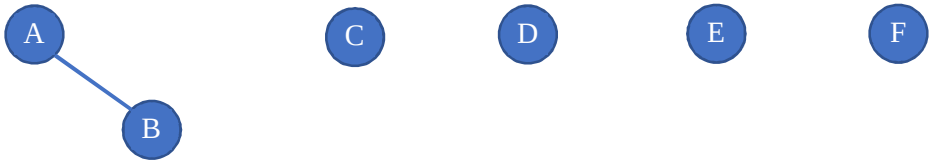
2. Problem 2: proof of slow Dijkstra’s algorithm

3. Problem 3

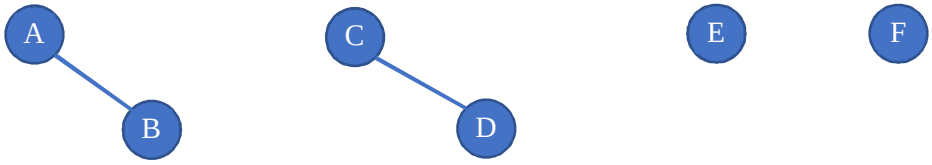


Sorted edges: E_{sorted} = {AB, CD, BD, EA, FE, AF, DF, BC, AD}
No. of vertices = n = 6 → n – 1 = 5.
Initial disjoint subsets:

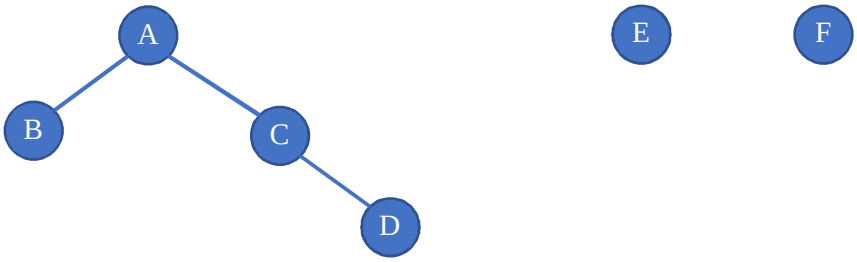
Step (1):
Take edge AB: C(A) ≠ C(B) → Merge C(A) & C(B) and add AB to MST.
MST = {AB}
|MST| = 1 < n – 1 → Continue.



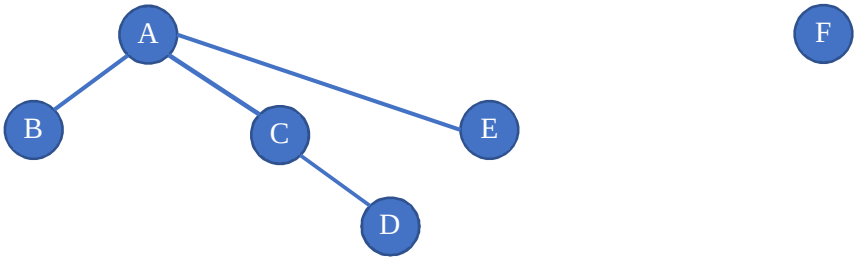
Step (2):
Take edge CD: C(C) ≠ C(D) → Merge C(C) & C(D) and add CD to MST.
MST = {AB, CD}
|MST| = 2 < n – 1 → Continue.



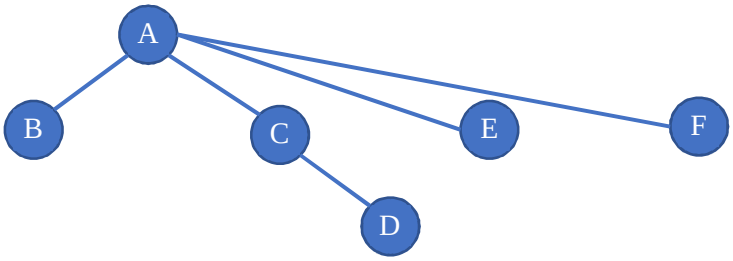
Step (3):
Take edge BD: C(B) ≠ C(D) → Merge C(B) & C(D) and add BD to MST.
MST = {AB, CD, BD}
|MST| = 3 < n – 1 → Continue.



Step (4):
Take edge EA: C(E) ≠ C(A) → Merge C(E) & C(A) and add EA to MST.
MST = {AB, CD, BD, EA}
|MST| = 4 < n – 1 → Continue.



Step (5):
Take edge FE: C(F) ≠ C(E) → Merge C(F) & C(E) and add FE to MST.
MST = {AB, CD, BD, EA, FE}



|MST| = 5 = n – 1 → Stop.
Solution: MST = {AB, CD, BD, EA, FE}.

4. Problem 4

	Operations count
Algorithm IsPrime (n)	
Input: an integer *n*	
Output: true if *n* is prime, false otherwise	
if (n == 1) then return false	*2*
if (n % 2 == 0) then	*2*
return false	
for (i = 3 until i*i ≤ n step 2) do	$1 + \sqrt{(n-2)/2} + 2 * \sqrt{(n-2)/2}$
if (n % i == 0) then	$2 * \sqrt{(n-2)/2}$
return false	*1*
return true	*1*

Running time is proportional to the magnitude of *n* instead of number of inputs (because it is always only one input).

Magnitude of *n* can be represented as a binary number with number of digits *b*:

$b = 1 + \lfloor \log n \rfloor \rightarrow b \; is \; O(\log n) \rightarrow n \; is \; \Theta(2^b)$

Since the running time from the IsPrime algorithm above is $\Theta(\sqrt{n})$.

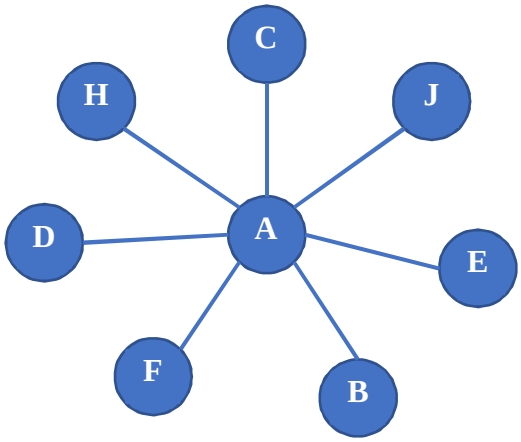
$And \; since \; \sqrt{n} = \sqrt{1 + 2^b} = 1 + (2^b)^{\frac{1}{2}} = 1 + \left(2^{\frac{1}{2}}\right)^b = 1 + \left(\sqrt{2}\right)^b = \Theta(2^b).$

$\therefore$ Asymbtotic running time is $\Theta(2^b)$.

5. Problem 5
   a. If $G(V, E)$ is connected: Yes. Because in a connected graph every edge $e \in E$ will be incident to at least one vertex $v \in V$.
      If $G(V, E)$ is disconnected: if we counted the empty set, then Yes (otherwise No). Because all edges "E" will be connected to a subset of a vertices set $V_1 \subseteq V$, and zero edges "{} – an empty set of edges" will be connected to the set of orphan vertices $V_2 \subseteq V$, where V_1 *and* V_2 are disjoint, and $V_1 \cup V_2 = V$.
      By laws of sets: $E = E \cup \{\}$.

   b. Yes, graphs with a base that is the empty set exist. An example is any discrete graph, like the one shown below:

   c. An example of a graph with 8 vertices and a base of size 1 is shown below:



   d. An example of a graph G having *2n* vertices with the property that every base for G has size of at least *n* is shown below:

      The graph above has number of vertices $= 4 = 2n$
      One smallest base set (except for the empty set) for the above graph is

$$U_{min} = \{B, D\}, |U| = 2 = n$$

      Any other set will have a size $\geq n$.

e. The algorithm for detecting the smallest base set of a graph is as follows:

Algorithm SmallestBase (G)	*Operations count*
Input: a graph G = (V, E) with n vertices and m edges	
Output: the smallest base U of G	
U ← empty set `// the smallest base set`	
S ← A stack of vertices, which are pushed after being sorted by degree and excluding vertices with degree=0	*O (m.log n)*
L ← List (status, Array [2]) `// status = true if this edge is represented by a vertex in the base set. The array will contain the startingVertex at position [0] and endVertex at position [1].`	*O (m)*
while (!S.isEmpty ()) do	*O (n)*
v ← S.pop () `// get vertex with highest degree`	*O (n)*
if (U.find (v) == null) then	*O (n)*
for each element *e* in L do	*O (m)*
if (!e.status) then	*O (m)*
if (e.Array[0] == v or e.Array[1] == v) then	*O (m)*
U.add (v)	*O (m)*
e.status = true	*O (m)*

This is a greedy algorithm. Vertices with highest degree are taken first because taking the vertex with the highest degree covers the maximum number of edges.
Vertices are sorted by their degree and then are pushed into a stack. This guarantees that the vertex with the highest degree is always checked first. Vertices with degree of zero are ignored because their base is the empty set.
Each vertex is popped when checked to guarantee it is not checked again. When the vertex is popped, it is checked for the following:

- Is it already in the smallest base set U? if yes, we ignore it. If no:
- Check all edges (in list L) that are not represented by any vertex in the base set, yet. For each edge: is the current vertex a starting or an end vertex of it? If yes, add the vertex to the smallest base set and mark the edge as represented. Otherwise, ignore the vertex.

The algorithm stops when the stack of vertices is empty.