Design and Analysis of Algorithm – Final Exam (**Fall-2022)**

Solution

Question # 1 **[0.5*10 = 5 marks] [CLO 1]**

Answer the following questions. You must explain in only 3-4 lines.

(a) Write any two real life applications of the set cover problem

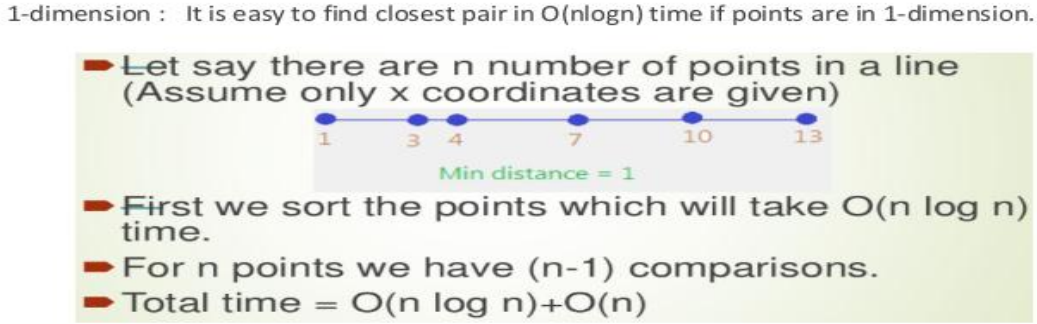
- Selecting small number of sentences to tune all features in speech recognition
- Selecting small number of telescope snapshots to capture light from all galaxies in the night sky

(b) If a problem does not have overlapping sub-problems, then dynamic programming approach works better than divide and conquer. True or False? Also briefly explain.

- False. Dynamic programming works better when there are overlapping subproblems.

(c) Suppose we have five different points in one dimension. How can we find closest pair of points in *O (nlogn)*?

(d)



(e) Discuss any two real life applications of the closest pair problem.

- Collision avoidance
- Clustering

(f) Suppose you want to lay pipelines in a building such that minimum pipe and other materials are consumed covering every desired location. Which technique from algorithms course will you employ and why?

- Minimum spanning tree

(g) What is the advantage of Bellman ford algorithm over Dijkstra algorithm?

- Bellman always works for negative edges but Dijkstra may or may not work..

(h) What is negative weight cycle in a graph and how will you identify it through algorithm?

- When sum of cost all edges in cycle is negative then it is negative weight cycle. We can apply bellman ford one more time after V-1 iterations and if cost of edges change then there is negative weight cycle.

(i) What do you mean by small-o (o) and small-omega (w) in asymptotic notations?

- Small-o :
  If f(n) = o(g(n)) then f(n)<cg(n) for c>0 and n>=n_0

- Small-omega :
  If f(n) = w(g(n)) then f(n)>cg(n) for c>0 and n>=n_0

(j) For a sparse graph, adjacency matrix is better than adjacency list. True or False. Briefly explain

- False. If graph is sparse then most of the entries of adjacency matrix would be empty.

(k) Can a problem be both NP and P ? Why or why not ?

Yes because P is a subset of NP.

Question # 2 **[0.25 * 12 = 3 marks] [CLO 2]**

Write time complexities of below mentioned algorithms.

Algorithm	Worst case time	Best case time
Insertion Sort	$O(n^2)$	O(n)
Radix Sort	O(nk)	O(nk)
Quick Sort	$O(n^2)$	O(nlogn)
Jump Search	$O(n^{1/2})$	O(1)
Brute force string matching	O(nm)	O(m)
Boyer Moore String Matching	O(nm)	O(m)

Question # 3 **[1*4 = 4 marks] [CLO 2]**

For each of the following questions, indicate whether it is T (True) or F (False), justifying by using some examples e.g. assuming a function

1. For all positive $g(n)$, $o\left(g(n)\right) \cap \omega\left(g(n)\right) = emptyset$
2. For all positive $g(n)$, If $g(n) + \omega\left(g(n)\right) =$ o $(g(n))$
3. f and g be two positive functions such that $f(n) = O\left(g(n)\right)$ then $f(n)^2 = O\left(g(n)^2\right)$
4. $2^{n+1} = \Theta(2^n)$

Solutions:

1. For all positive $g(n)$, $o\left(g(n)\right) \cap \omega\left(g(n)\right) = emptyset$

Let $g(n) = n^2$; $o(n^2) = \{n, \log n, \ldots\}$ and $\omega(n^2) = \{n^3, n^4, n^5, n^6, \ldots\}$
$\{n, \log n, \ldots\} \cap \{n^3, n^4, n^5, n^6, \ldots\} = \emptyset$

True.

2. For all positive $g(n)$, If $g(n) + \omega\left(g(n)\right) =$ o $(g(n))$

Let $g(n) = n^2$; $\omega(n^2) = \{n^3, n^4, n^5, n^6, \ldots\}$ and $o(n^2) = \{n, \log n, \ldots\}$

$n^2 + \{n^3, n^4, n^5, n^6, \ldots\} \neq o(n^2)$ **False**

3. f and g be two positive functions such that $f(n) = O\left(g(n)\right)$ then $f(n)^2 = O\left(g(n)^2\right)$

Let $f(n) = n^2$ then g(n) = $\{n^2, n^3, \ldots,\}$
$(n^2)^2$ = O $((n^2, n^3, \ldots,)^2)$

True

4. $2^{n+1} = \Theta(2^n)$. **True**

Is $2^{n+1} = O(2^n)$? Yes, for large n $f(n) \leq c \cdot g(n)$
The key observation is that $2^{n+1} = 2 \cdot 2^n$, so $2 \cdot 2^n \leq c \cdot 2^n$ for any $c \geq 2$.

Is $2^{n+1} = \Omega(2^n)$? Yes, for large n $f(n) \geq c \cdot g(n)$.
$2 \cdot 2^n \geq c \cdot 2^n$, This would be satisfied for any $0 < c \leq 2$. This imply $2^{n+1} = \Theta(2^n)$

Question # 4 **[1 + 0.5 + 0.5 + 2 + 1 = 5 marks] [CLO 2]**

1 Define P, NP, NP-Hard and NP-Complete Problems.

2 Assuming P != NP, then why NP-complete ∩ P = ∅ is correct?

Reason: The answer is B (no NP-Complete problem can be solved in polynomial time). Because, if one NP-Complete problem can be solved in polynomial time, then all NP problems can be solved in polynomial time. If that is the case, then NP and P set become same which contradicts the given condition.

3 Which of the following is true about NP-Complete and NP-Hard problems.

a. If we want to prove that a problem X is NP-Hard, we take a known NP-Hard problem Y and reduce Y to X
b. The first problem that was proved as NP-complete was the circuit satisfiability problem.
c. NP-complete is a subset of NP Hard
d. All of the above
e. None of the above

4 Let S be an NP-complete problem and Q and R be two other problems not known to be in NP. Q is polynomial time reducible to S and S is polynomial-time reducible to R. Then why R cannot be NP-Complete.

Ans: R is not in NP. A NP Complete problem has to be in both NP and NP-hard.

5 Identify which of the following belongs to P, NP, NP-Hard and NP-Complete

I. Integer Factorization
II. Vertex Cover
III. Graph Isomorphism
IV. 2-SAT

Ans: NP, NP-Complete, NP and P

Question # 5 **[2 + 3 = 5 marks] [CLO 4]**

Consider the following Greedy Algorithm for set-cover problem Greedy Algorithm (SET-COVER):

(i) pick the set that covers the most points.

(ii) Throw out all the points covered.

(iii) Repeat (i) and (ii) until all points are covered

(a) Is it always possible to get optimal solution from above algorithm? Show with an example [2 marks]

Ans: No. For example, consider 5 points and following sets: {S1: P1, P2}, {S2: P1, P3}, {S3: P1, P2, P5}, {S4: P3, P4}, {P1, P5}. Optimal set is {S3, S4} but {S3, S1, S2, S4} is a possibility

(b) Proof that if the optimal solution uses k sets, the greedy algorithm finds a solution with at most $k \ln n$ sets. [2 marks]

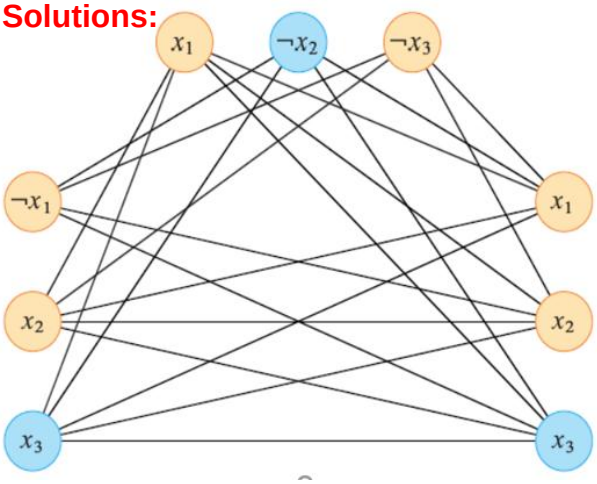
Proof: Since the optimal solution uses k sets, there must some set that covers at least a $1/k$ fraction of the points. The algorithm chooses the set that covers the most points, so it covers at least that many. Therefore, after the first iteration of the algorithm, there are at most $n(1-1/k)$ points left. Again, since the optimal solution uses k sets, there must some set that covers at least a $1/k$fraction of the remainder (if we got lucky we might have chosen one of the sets used by the optimal solution and so there are actually $k-1$ sets covering the remainder, but we can't count on that necessarily happening). So, again, since we choose the set that covers the most points remaining, after the second iteration, there are at most $n(1-1/k)^2$ points left. More generally, after t rounds, there are at most $n(1-1/k)^t$ points left. After $t = k \ln n$ rounds, there are at most $n(1-1/k)^{k \ln n} < n(1/e)^{\ln n} = 1$ points left, which means we must be done. ■

Question # 6 **[2.5 + 2.5 + 1 = 6 marks] [CLO 4]**

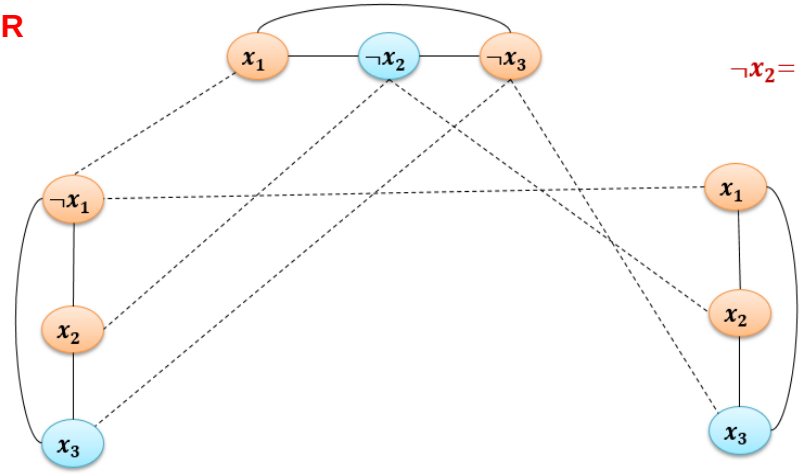
1. Prove that the 3-SAT problem reduces to the vertex cover problem, by constructing graph G of φ and show that φ is satisfiable iff G contains a vertex cover of size k and V-K = $|\varphi|$.

$$\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

Solutions:



OR



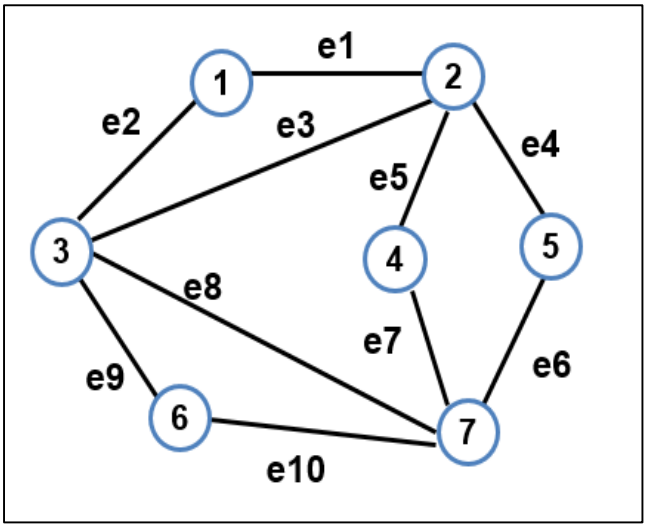
Vertex Cover (S) = Orange Box (x_1, $\neg x_3, \neg x_1$, x_2, x_2, x_3)

3-SAT literals = V – S → $(\neg x_2, \neg x_3, \neg x_1) = \neg x_2, \neg x_3, \neg x_1 = 1$

2. Prove that the vertex cover problem reduces to the set cover problem. Use the graph given below to show that the SET-COVER instance has set cover of size k iff G has a vertex cover of size k.

Solution:

X = E, {e1, e2, e3, e4, e5, e6, e7, e8, e9 , e10}
F1 = {e1, e2}
F2 = {e1, e3, e4, e5}
F3 = {e2, e3, e8, e9}
F4 = {e5, e7}
F5 = {e4, e6}
F6 = {e9, e10}
F7 = {e6, e7, e8, e10}



Set Cover k = 3 i.e. (2, 3, 7)

Vertex Cover k = 3 i.e. (F2, F3, F7)

3. Prove that the **3-SAT** $\in$ ***NP***.

Solution:

Clearly 3-Satisfiability is in NP, since we can verify in polynomial time that a proposed truth assignment satisfies the given set of clauses.

Question # 7 **[2+2 = 4 marks] [CLO 4]**

Consider the following algorithm. Give answer of any two subparts (**from a to d)**.

Input : String S of "n" characters and String "P" of "m" characters, where m <= n

Output: index "i" where P begins as a substring of S

```
1:      for i = 0 to n-m
2:          hits = 0
3:          while (hits < m)
4:              if P[hits] != S[hits + i]
5:                  break
6:              hits = hits+1
7:          if (hits==m)
8:              print(i)
9:              break
```

a) Give an example of worst case input for this algorithm. Also tell exactly how many character comparisons are made (line 4) on your worst case input as a function of n and m?

Solution:

S = BBBBBBBBBC and P = BBC

Total comparisons as a function of n and m = (m) x (n-m+1) = 3 x (10-3+1) = 24

b) For a fixed n, what value for m would maximize the cost of the worst case? Justify your answer through example input

Solution:

For a fixed n, m should be around half the size of n to maximize cost of worst case. For example : S = AAAAAAAAAH and P = AAAAH. Here n=10 and m=5:
Then total comparisons = (m) x (n-m+1) = 5 (10-5+1) = 30. So there is no way you get more than 30 comparisons for this input

c) Suppose that all characters in String "P" are different from each other. How would you accelerate the above algorithm to run in time *O (n)* on an "n" character string "S". Answer briefly in three to four lines.

Solution:

We know that one occurrence of P in S cannot overlap with another, so we don't need to double-check the way the naive algorithm does. If we find an occurrence of Pk in S followed by a nonmatch, we can increment by k instead of 1.

d) If we remove line 9 in above code then what would be output of above code if S = ABAAABABAABABAA and P = BAA

Solution:

Multiple indexes will be printed when match is found i-e (1, 7 and 12)

Question # 8 **[4 marks] [CLO 4]**

Points given below are sorted by polar angle. Show all steps to make convex hull on these points in *O (n)* time. (See appendix for formula of counter clockwise turn)

Sorted points [(0, 0), (7, 0), (3, 1), (5, 2), (9, 6), (1, 4)]

.

Solution:

Sorted points [(0,0), (7,0), (3,1), (5,2), (9,6), (1,4)]

Insert first 3 points in Stack.

(3,1)
(7,0)
(0,0)

1. Check Condition for point 4 (5, 2) to insert in stack

Calcluate CCW c=(5,2), b=(3,1), a=(7,0)

$(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)$

$(2 - 0)(3 - 7) - (5 - 7)(1 - 0) = -8 - 2$ clockwise

Clockwise so remove (3,1) from stack

~~(3,1)~~
(7,0)
(0,0)

Next Calcluate CCW c = (5,2), b = (7,0), a = (0,0)

(2 - 0) (7 – 0) – (5 - 0) (0 - 0) = 14 CCW

(5,2)
(7,0)
(0,0)

2. Check Condition for point 5 (9, 6) to insert in stack

Calcluate CCW c= (9,6) b=(5,2), a=(7,0)

(6 - 0) (5 – 7) – (9 - 7) (2 - 0) = -16 clockwise

clockwise so remove (5,2) from stack

Calcluate CCW c= (9,6) b=(7,0), a=(0,0)

(6 - 0) (7 – 0) – (9 - 0) (0 - 0) = 42 CCW

~~(5,2)~~
(7,0)
(0,0)

(9,6)
(7,0)
(0,0)

Insert (9,6) in stack

3. Check Condition for point 6 (1, 4) to insert in stack

Calcluate CCW c= (1,4), b= (9,6) a=(7,0)

(4- 0) (9 - 7) – (1 – 7) (6 – 0) = 8 – 6 = 2 CCW

Insert (1,4) in stack

Since point (1,4) is the last point in the list, the algorithm terminates here. The points in the stack are the convex

(1,4)
(9,6)
(7,0)
(0,0)

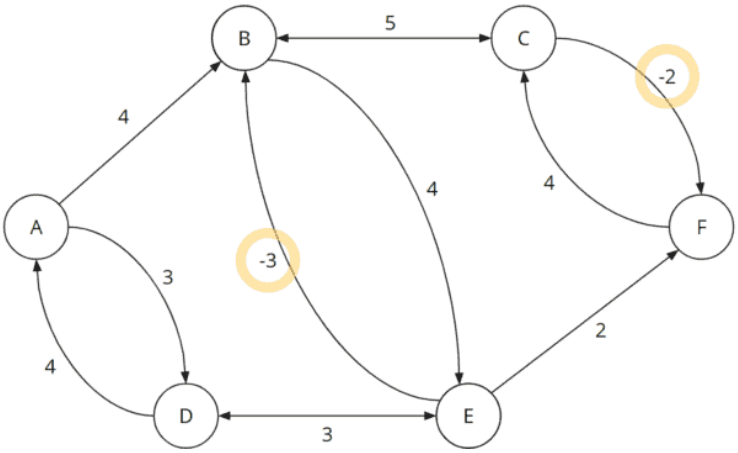
Question # 9 **[5 marks] [CLO 3]**

Apply the Bellman ford algorithm on the provided directed graph G to compute the shortest path from node "A" to all other nodes. Show all step to find the optimal cost and proof it optimal. In last, discuss the time complexity it take.

G =

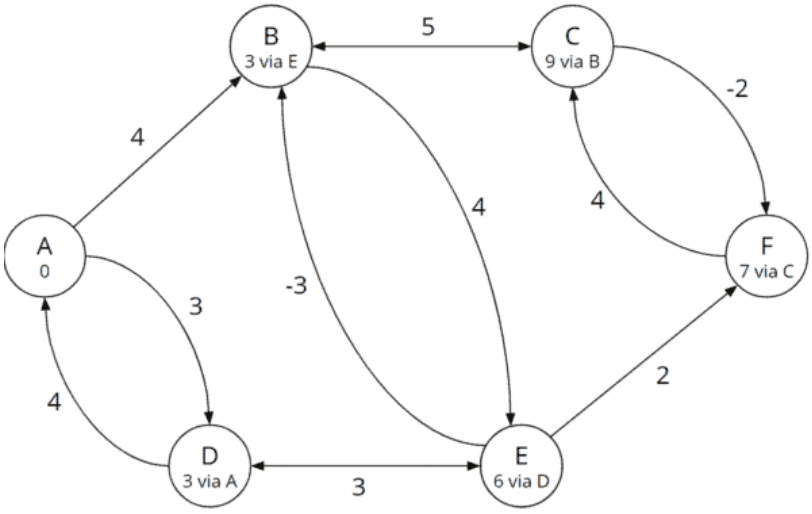
	A	B	C	D	E	F
A	0	4	-	3	-	-
B	-	0	5	-	4	-
C	-	5	0	-	-	-2
D	4	-	-	0	3	-
E	-	-3	-	3	0	2
F	-	-	4	-	-	0

Solution



Node	Predecessor	Total cost from the start
A	–	0
B	–	∞
C	–	∞
D	–	∞
E	–	∞
F	–	∞

First Iteration

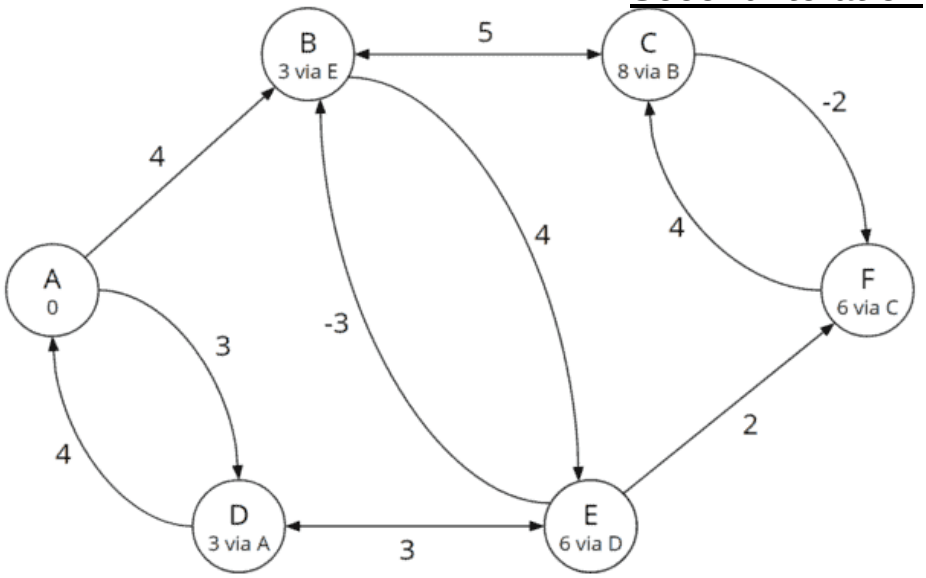


Total costs and predecessors at the end of iteration 1

Node	Predecessor	Total cost from the start
A	–	0
B	E	3
C	B	9
D	A	3
E	D	6
F	C	7

The graph currently looks like this:

Second Iteration



Total costs and predecessors at the end of iteration 2

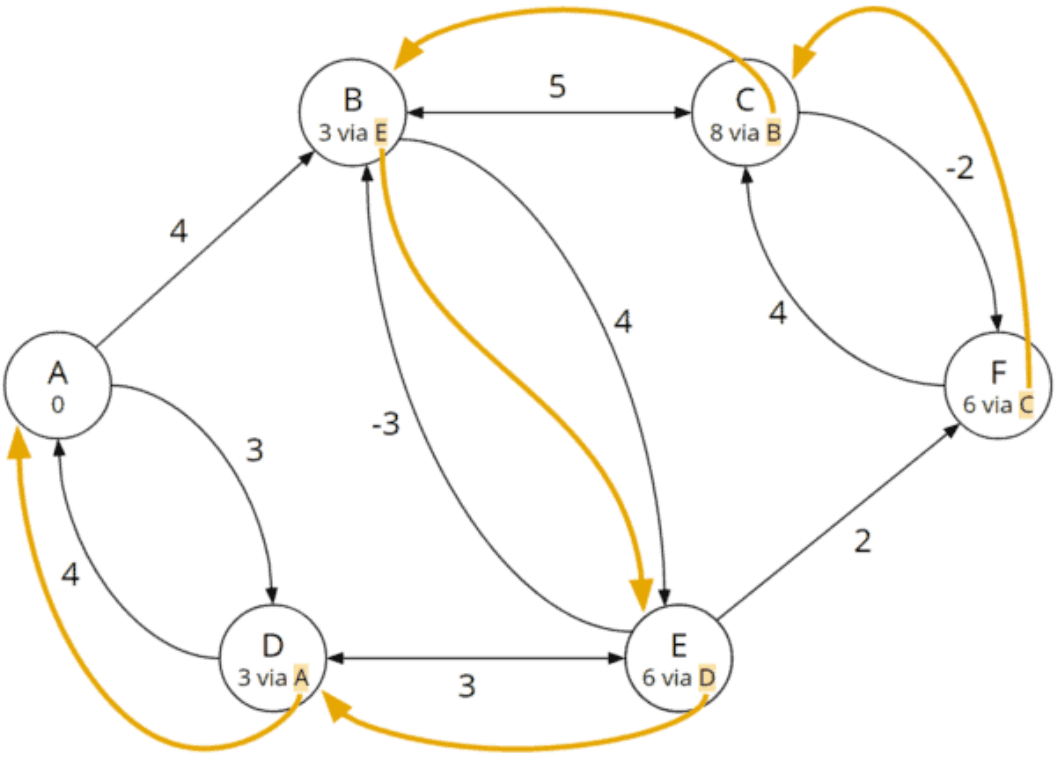
The table currently looks like this:

Node	Predecessor	Total cost from the start
A	–	0
B	E	3
C	B	8
D	A	3
E	D	6
F	C	6

And once again, the total costs and predecessors in the graph:

Third Iteration

After the third check of all edges, the algorithm will not have detected any further cost reductions.



Backtrace for determining the complete path

Complete path is: A→D→E→B→C→F

Question # 10 **[5 marks] [CLO 3]**

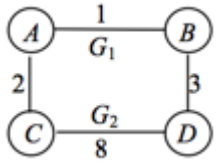
Professor Toole proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows.

- Given a graph G = (V, E), partition the set V of vertices into two sets V1 and V2 such that |V1| and |V2| differ by at most 1.
- Let E1 be the set of edges that are incident only on vertices in V1, and let E2 be the set of edges that are incident only on vertices in V2.

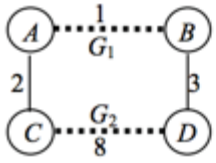
Solve a minimum-spanning-tree problem on each of the two subgraphs G1 = (V1, E1) and G2 = (V2, E2) recursively. Finally, select the minimum-weight edge in E that crosses the cut (V1, V2), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.

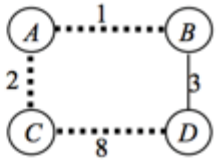
Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.
Note: The minimum-weight edge that crosses the cut means, the minimum-weight edge that will combines V1 and V2.

Solution:
We argue that the algorithm fails. Consider the graph G below. We partition G into V1 and V2 as follows: V1 = {A, B}, V2 = {C, D}. E1 = {(A, B)}. E2 = {(C, D)}. The set of edges that cross the cut is Ec = {(A, C), (B, D)}.



Now, we must recursively find the minimum spanning trees of G1 and G2. We can see that in this case, MST(G1) = G1 and MST(G2) = G2. The minimum spanning trees of G1 and G2 are shown below on the left.
The minimum weighted edge of the two edges across the cut is edge (A, C). So (A, C) is used to connect G1 and G2. This is the minimum spanning tree returned by Professor Borden's algorithm. It is shown below and to the right.





We can see that the minimum-spanning tree returned by Professor Toole algorithm is not the minimum spanning tree of G, therefore, this algorithm fails.

Question # 11 **[2 + 2 = 4 marks] [CLO 2]**

Compute the time complexity for algorithms one by one. Show all steps.

void func(int n)
{
int i = 1;
while (i < n) {
int j = n;
while (j > 0) {
j = j / 2;
}
i = i * 2;
}
}
	void func(int n)
	{
	int count = 0;
	for (int i=0; i<n; i++)
	for (int j=i; j< i*i; j++)
	if (j%i == 0){
	for (int k=0; k<j; k++)
	printf("*");
	}
	}

1. Time complexity = $O(\log^2 n)$.

In each iteration, i become twice (time complexity O(logn)) and j become half (time complexity=O(logn)). So, time complexity will become $O(\log^2 n)$.

2. Time complexity = $O(n^5)$