Solitary Assignment

Vinay Ummadi

Solitary Assignment for **Design and Analysis of Algorithms**

March 30, 2022

Problem 1

Solution.

(a) $f(n) = n - 100$ and $g(n) = n - 200$

Since both are functions $f(n)$ and$g(n)$ are linear in nature. We can ignore the constants 100 and 200. Can be written as $f(n) = \Theta(g(n))$

(b) $f(n) = n^{1/2}$ and $g(n) = n^{2/3}$

It is clearly visible that both $f(n)$ and$g(n)$ are polynomial in nature. But $g(n)$ is dominating $f(n)$. Can be written as $f(n) = O(g(n))$

(c) $f(n) = 100 * n + log(n)$ and $g(n) = n + (log(n))^2$

As per asymptotic rules any polynomial dominates logarithms so $100n$ dominates $logn$ and n dominates $(log(n))^2$. Since both are polynomial in nature by ignoring multiplicative constant. This can be written as $f(n) = \Theta(g(n))$

(d) $f(n) = nlog(n)$ and $g(n) = 10nlog(n)$

As per asymptotic rules any polynomial dominates logarithms so ignoring $log(n)$ on both sides we will get $f(n) = n$ and$g(n) = 10n$. We can clearly see both are in polynomial in nature. So can be written as $f(n) = \Theta(g(n))$

(e) $f(n) = n^{1.01}$ and $g(n) = n(log(n))^2$

Both functions are in polynomial nature with different powers. $1.01 > 1$. So can be written as $f(n) = \Omega(g(n))$

(f) $f(n) = n^2/logn$ and $g(n) = n(log(n))^2$

Both functions are in polynomial nature with different power and logarithm multiplications. By comparing powers $2 > 1$, it can be written as $f(n) = \Omega(g(n))$

(g) $f(n) = n^{0.1}$ and $g(n) = (logn)^{10}$

Polynomial with any power grows faster than any logarithms. So can be written as $f(n) = \Omega(g(n))$

(h) $f(n) = (logn)^{logn}$ and $g(n) = n/logn$

From asymptotic rules any polynomial grows faster than any logarithm. So can be written as $f(n) = \Omega(g(n))$

(i) $f(n) = \sqrt{n}$ and $g(n) = (logn)^3$

Polynomial with any power grows faster than any logarithms. So can be written as $f(n) = \Omega(g(n))$

(j) $f(n) = n^{1/2}$ and $g(n) = 5^{log_2 n}$

$g(n)$ can be written as $n^{log_2 5} = n^{2.32}$. By comparing polynomial powers we can conclude that $f(n) = O(g(n))$

(k) $f(n) = n2^n$ and $g(n) = 3^n$

Both are exponential functions with different bases. By comparing bases $3 > 2$, we can write $f(n) = O(g(n))$

(l) $f(n) = 2^n$ and $g(n) = 2^{n+1}$

Both functions are exponential in nature with same base but slightly different powers. We can write it as $f(n) = \Theta(g(n))$

(m) $f(n) = n!$ and $g(n) = 2^n$

In the present case factorial grows faster exponential and will be written as $f(n) = \Omega(g(n))$

(n) $f(n) = logn^{logn}$ and $g(n) = 2^{log_2 n2}$

$f(n)$ can be re-written as $n^{lognlogn}$ and $g(n)$ can be written as $n^{log_2 n}$. $g(n)$ grows faster than $f(n)$. So can be written as $f(n) = \Omega(g(n))$

(o) $f(n) = \sum_{i=1}^{n} i^k$ and $g(n) = n^{k+1}$

Both functions are polynomial in nature different powers. So can be written as $f(n) = \Theta(g(n))$

Problem 2

Solution.

In base b representation the maximum single digit number is $b - 1$ and maximum two digit number is $b^2 - 1$. If we consider there maximum single digit numbers and add them together it should be <= maximum two digit number.

The condition we have formally stated above can be written mathematically as

$$(b - 1) + (b - 1) + (b - 1) <= b^2 - 1$$

By doing the algebraic manipulation we can write it as

$$3(b - 1) <= (b + 1)(b - 1)$$

By cancelling the common terms, it can be written as

$$3 <= b + 1$$

and finally we arrive at a condition.

$$2 <= b$$

The above comparison can be easily verified with base 2 and is also valid for any other bases.

Problem 3

Solution.

(a)

From normal recurrence relation, given recurrence is

$$T(n) = 3((T(\frac{n}{2})) + O(n))$$

If we expand this for one iteration then, it will be

$$T(n) <= 3(3(T(\frac{n}{4}) + c(\frac{n}{2}) + cn$$

Generalizing with variable k

$$T(n) <= 3^k T(\frac{n}{2^k}) + \sum_{i=0}^{log_2 n} 3^i \frac{cn}{2^k}$$

where $k = log_2 n$

Finally T(n) can be written as

$$T(n) = O(n^{log_2 3})$$

(b)

From normal recurrence relation, given recurrence is

$$T(n) = T(n-1) + O(1)$$

If we run this for one iteration then, it will be

$$T(n) \leq T(n-1) + cn$$

If we above for one more iteration then, it will be

$$T(n) \leq T(n-2) + c(n-1) + cn$$

Generalizing for with variable i

$$T(n) = T(1) + c\sum_{i=0}^{n} n - i$$

The above relation will of at max $O(n^2)$

Problem 4

Solution.

(b)

Standard form of masters theorem is given as

$$T(n) = aT(\frac{n}{b}) + n^c$$

By comparing with masters theorem, and finding the variable values

$$a = 5, b = 4, c = 1$$

Since $c < log_b a$

So $T(n) = O(n^{log_b a})$

which is $T(n) = O(n^{log_4 5})$

(e)

Standard form of masters theorem is given as

$$T(n) = aT(\frac{n}{b}) + n^c$$

By comparing with masters theorem, and finding the variable values

$$a = 8, b = 2, c = 3$$

Since $c = log_b a$

So $T(n) = O(n^c log(n))$

which is $T(n) = O(n^3 logn)$

(h)

The given recurrence is

$T(n) = T(n-1) + n^c$ where $c >= 1$ is a constant

$$T(n) = T(n-1) + n^c$$

For n-1 it can be written as

$$T(n-1) = T((n-1)-1) + n^c$$

$$T(n-1) = T(n-2) + n^c$$

Using the result from $T(n-1)$ in $T(n)$

$$T(n) = T(n-2) + n^c + n^c$$

The final result will be

$$T(n) = T(n-2) + 2n^c$$

(i)

The given recurrence is

$T(n) = T(n-1) + n^c$ where $c > 1$ is a constant

$$T(n) = T(n-1) + n^c$$

Writing a generalized equation with variable k

$$T(n) = T(n-k) + kn^c$$

substitute $k = n$

$$T(n) = T(n-n) + nn^c$$

This can be written as

$$T(n) = T(0) + n^{c+1}$$

$$T(n) = 1 + n^{c+1}$$

while ignoring constants, the big-O notation for Tn

$$T(n) = O(n^{c+1})$$

(k)

The given recurrence is

$$T(n) = T(n^{\frac{1}{2}}) + 1$$

The recurrence equation for $T(n^{\frac{1}{2}})$

$$T(n^{\frac{1}{2}}) = T(n^{\frac{1}{4}}) + 1$$

The recurrence equation for $T(n^{\frac{1}{4}})$

$$T(n^{\frac{1}{4}}) = T(n^{\frac{1}{8}}) + 1$$

From the above recurrence relations the T(n) can be written as

$$T(n) = T(n^{\frac{1}{4}}) + 2$$

$$T(n) = T(n^{\frac{1}{8}}) + 3$$

For generalization using k

$$T(n) = T(n^{\frac{1}{2^k}}) + k$$

Big O Notation $\quad T(n) = O(log(log(n)))$

Problem 5

Solution.

$$1 + \omega + \omega^2 + ... + \omega^{n-1} = \frac{\omega^n - 1}{\omega - 1}$$

The sum of nth roots of unity is 0.

To find the product of nth roots of unity

$$= e^{\sum_{k=0}^{n-1} \frac{2ki\pi}{n}}$$

$$= e^{i\pi(n-1)}$$

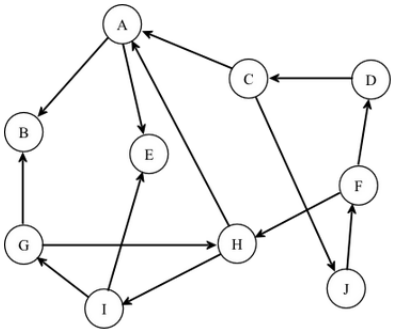
$$= \cos(\pi(n-1) + i\sin(\pi(n-1)$$

The product of the nth roots of unity are -1 and 1 when n is even and odd respectively.

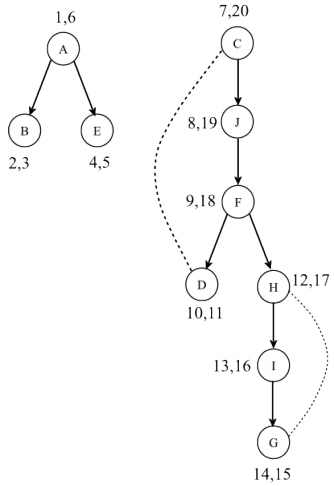
Problem 6

Solution.

Graph G^R after reversing the edges.



After running Depth First Search(DFS) algorithm on the graph G^R will result in the following path with pre numbers and post numbers.



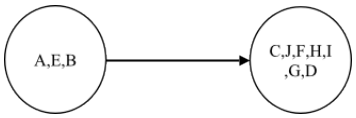
The strongly connected components from the post-numbers in DFS traversal of G^R. Vertex C has the highest post-number and vertex B has the lowest post number.

strongly connected components found in the order: $[C, J, F, H, I, G, D]$, $[A, E, B]$

The connected component $[C, J, F, H, I, G, D]$ will be the source in G^R. That means, $[C, J, F, H, I, G, D]$ will be the sink in G.

The connected component $[A, E, B]$ will be the sink in G^R. That means, $[A, E, B]$ will be the source in G.

Therefore the Meta-Graph with meta-node is shown below.



Problem 7

Solution.

(a) In an undirected graph $\sum_{u \in V} d(u) = 2|E|$. Which means each edge is counted twice, once coming IN and once going outward OUT. Each edge has two vertices and each edge is counted twice.

(b)

Ve is even degree vertices and Vo is odd degree vertices

$$\sum_{u \in V} d(u) = \sum_{u \in V_o} d(u) + \sum_{u \in V_e} d(u)$$

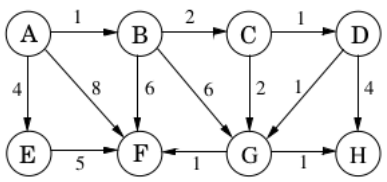
The given condition is

$$\sum_{u \in V} d(u) = 2|E|$$

$$\sum_{u \in V_o} d(u) = 2|E| - \sum_{u \in V_e} d(u)$$

This can be interpreted as sum of odd-degree vertices is even form the above equation.

Problem 8



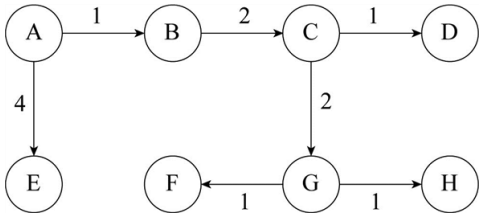
Solution.

(a) Dijkstra can solve single source non-negative edged graphs G(V,E). Both directed and undirected graphs can solved with Dijkstra's algorithm.

The below table shows the updates at every iteration of the Dijkstra's algorithm for the given graph.

Iteration	A	B	C	D	E	F	G	H
0	0	∞	∞	∞	∞	∞	∞	∞
1	0	1	∞	∞	4	8	∞	∞
2	0	1	3	∞	4	7	7	∞
3	0	1	3	6	4	7	5	∞
4	0	1	3	6	4	7	5	∞
5	0	1	3	6	4	6	5	6
6	0	1	3	6	4	6	5	6

(b) The final shortest path is shown below.



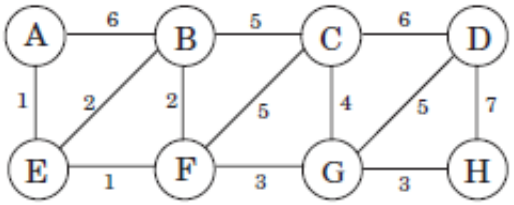
Problem 9

Solution. To find the shortest path between the any given two nodes in the strongly connected undirected graph. Lets assume the two nodes as a and b.

Algorithm :

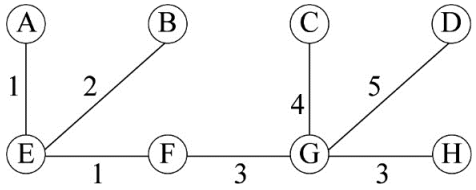
1. First we start with Finding all the shortest paths between vertex a and vertex V_o. V_o is an intermediate node between a and b. 2. Now find all the shortest paths between vertex V_o and vertex b.
3. Use the above paths to find the complete shortest paths between vertex a and vertex b.

Problem 10



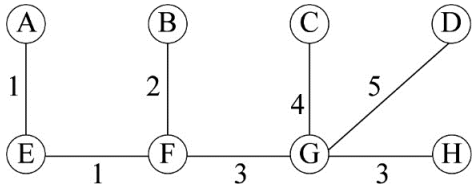
Solution.

(a) The one possible minimum spanning tree (MST) is shown below



The **Total cost of MST** is : $1 + 1 + 2 + 3 + 3 + 4 + 5 = \mathbf{19}$

(b) There are **two** possible minimum spanning trees (MSTs) with same cost of 19. By considering edge $B - F$ instead of edge $B - E$ another MST is formed with cost of **19**. It is shown below



(c) Kruskals MST solution is shown in the part (a) answer of the question.

Order of the edges added is : AE, EF, BE, FG, GH, CG, GD

Problem 11

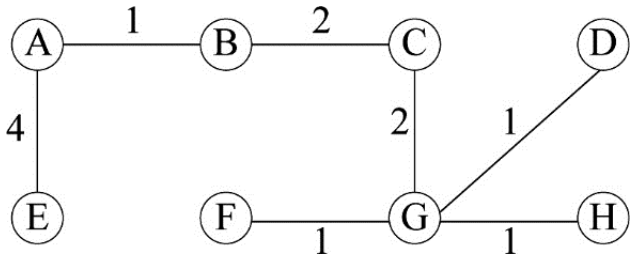
Solution.

Minimum Spanning Tree for the given graph by prims algorithm is given below. Intermediate **cost** and **prev** arrays are tabulated.

Total Minimum Spanning Tree(MST) cost: **12**

Set S	A	B	C	D	E	F	G	H
[ ]	0/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
[A]	0/nil	1/A	2/B	∞/nil	4/A	6/B	6/B	∞/nil
[A, B]	0/nil	1/A	2/B	∞/nil	4/A	6/B	6/B	∞/nil
[A, B, C]	0/nil	1/A	2/B	3/C	4/A	6/B	2/C	∞/nil
[A, B, C, G]	0/nil	1/A	2/B	1/G	4/A	1/G	2/C	1/G
[A, B, C, G, H]	0/nil	1/A	2/B	1/G	4/A	1/G	2/C	1/G
[A, B, C, G, H, D]	0/nil	1/A	2/B	1/G	4/A	1/G	2/C	1/G
[A, B, C, G, H, D, F]	0/nil	1/A	2/B	1/G	4/A	1/G	2/C	1/G
[A, B, C, G, H, D, F, E]	0/nil	1/A	2/B	1/G	4/A	1/G	2/C	1/G

Minimum Spanning Tree(MST) achieved by running Prim's algorithm is shown below .



Problem 12

Solution.

1. For each and every cut, the minimum edge weight is on the minimum spanning tree.
2. If all the edge weights are different in graph, then all the cuts are unique.
3. If all the edge weights are not different in a graph, then result might be ambiguous. 4. Therefore, the minimum spanning tree is unique, if all the edge weights are different in a graph.

Proof with Example : Consider the below undirected graph and we obtain its Minimum Spanning Tree using Kruskal's algorthm. We can observe that the taken graph has distinct edge weights and the MST obtained is also unique. No other MST can be formed from this undirected graph. This proves that graph with distinct edge weights for sure have a unique MST.

Problem 13

Solution. A dynamic programming based algorithm can be developed for this problem. Since we need to find out best route means stoppages at hotels that will result in minimum penalty. In dynamic programming we need to figure out the sub-problems and we start from smallest sub-problem. Here can take sub-problem as finding the minimum penalty hotel stoppages till hotel a_j. Here we can identify the implicit DAG structure, we can only move to hotel a_j only after passing through hotel a_i.

Algorithm 1 DP based algorithm for min penalty route

```
S[0] = 0
for i = 1, 2, 3, ..., n do
    minpenalty = ∞
    for j = 0, 1, 2, ..., i do
        minpenalty = min(S[j] + (200 − (a[i] − a[j]))^2), minpenalty)
        S[i] = minpenalty
    end for
end for
```

Problem 14

Solution. This problem has some similarity to edit distance problem that we saw in textbook example. Here we need to find an maximum scoring alignment. While in edit distance our goal is to find an min cost sequence alignment operations. Here we know the scoring alignment for every possible pair of characters used in the sequences given by δ. we are given condition that, No column contains spaces in both rows. In such case it will length of at most $n + m$. Given $x[1...n]$, $y[1...m]$, $\Sigma = [A, C, G, T]$ and scoring matrix δ of size $|\Sigma| + 1 * |\Sigma| + 1$ that is $5 * 5$

This is similar algorithm we used for edit distance. Both edit distance and sequence alignment are closely related problem the main difference is in edit distance we minimize the score and in alignment we try to maximize the score.

Algorithm 2 DP algorithm for maximum similarity alignment

for i = 0,1,2,3,...,n **do**
    $S(i,0) = i$
end for
for j = 0,1,2,3,...,m **do**
    $S(0,j) = j$
end for
for i = 1,2,3,...,n **do**
    for j = 1,2,3,...,m **do**
        $score = \delta(x(i), y(j))$
        $gap_x = \delta(-, y(j))$
        $gap_y = \delta(x(i), -)$
        $S(i,j) = max\{S(i-1,j-1) + score, S(i-1,j) + gap_y, S(i,j-1) + gap_x\}$
    end for
end for