Assignment 1 - Solution Sketch

Topics: Ch.1, 2, 3

1. *(25 Points)* **Chapter 1 - Stable Marriage.**

   Answer:

   (a) *(5 Points)* Using the GS algorithm with the men proposing, we get the man-optimal matching (Man #, Woman #): (1,1), (2,2), (3,3), (4,4).

   (b) *(5 Points)* Using the GS algorithm with the women proposing, we get the woman-optimal matching (Man #, Woman #): (1,4), (2,3), (3,2), (4,1)

   (c) *(5 Points)* At the first round, all men propose to the same woman. She dates her favorite. At the second round, all but that favorite one man proposes to the second-best woman. She dates her favorite. The process continues for n rounds until one man is left to propose to the least desirable woman.

   Continuing this manner, we see that the total number of proposals made by the algorithm are: $n + (n-1) + (n-2) + ... + 2 + 1 = n(n+1)/2$. so the worst case is $O(n^2)$

   (d) *(10 Points)* Assume we have three men m_1 to m_3 and three women w_1 to w_3 with preferences as given in the table below. Column w_3 shows true preferences of woman w_3, while in column w_3' she pretends she prefers man m_3 to m_1.

m_1	m_2	m_3	w_1	w_2	w_3	(w_3')
w_3	w_1	w_3	m_1	m_1	m_2	m_2
w_1	w_3	w_1	m_2	m_2	m_1	m_3
w_2	w_2	w_2	m_3	m_3	m_3	m_1

   First let us consider one possible execution of GS algorithm with the true preference list of w_3.

m_1	w_3			w_3
m_2		w_1		w_1
m_3			$[w_3][w_1]w_2$	w_2

   First m_1 proposes to w_3, then m_2 proposes to w_1. Then m_3 proposes to w_3 and w_1 and gets rejected, finally propoes to w_2 and is accepted. This execution forms pairs (m_1, w_3), (m_2, w_1) and (m_3, w_2), thus pairing w_3 with m_1, who is her second choice. Now consider execution of the GS algorithm when w_3 pretends she prefers m_3 to m_1 (see column w_3'). Then the execution might look as follows:

m_1	w_3		$-$	w_1			w_1
m_2		w_1		$-$	w_3		w_3
m_3			w_3		$-$	$[w_1]w_2$	w_2

   Man m_1 proposes to w_3, m_2 to w_1, then m_3 to w_3. She accepts the proposal, leaving m_1 alone. Then m_1 proposes to w_1 which causes w_1 to leave her current partner m_2, who consequently proposes to w_3 (and that is exactly what w_3 wants). Finally, the algorithm pairs up m_3 (recently left by w_3) and w_2. As we see, w_3 ends up with the man m_2, who is her true favorite. Thus we conclude that by falsely switching order of her preferences, a woman may be able to get a more desirable partner in the GS algorithm.

2. *(40 Points)* **Chapter 2 - Running Time Analysis.**

   (a) *(20 Points)* **Growth Rates:**

   Order the functions in ascending order of growth rate.

   Answer: We order the functions as follows:

   In order to compare g_1, g_3, g_5, g_4, let's take logarithms and change variable $z = logn$. Then we have:

$$\log g_1 = z^{\frac{1}{2}}, \quad \log g_3 = z + 3logz, \quad \log g_4 = \frac{4}{3}z, \quad \log g_5 = z^2 \tag{1}$$

   - g_1 comes before g_3 because $z^{0.5} < z < z + 3logz$.
   - g_3 comes before g_4, because $z + 3logz \leq \frac{4}{3}z \Leftrightarrow 3logz \leq \frac{1}{3}z$, which is true: logarithms grow slower than polynomials.
   - g_4 comes before g_5: $\frac{4}{3}z < z^2$ for large n.
   - g_5 comes before g_2.

$$g_5 < g_2 \Leftrightarrow \log g_5 < \log g_2 \Leftrightarrow (\log n)^2 < n \Leftrightarrow \log n < n^{\frac{1}{2}},$$

     which is true: logarithms grow slower than polynomials (for large n).
   - g_2 comes before g_7. Taking logarithms, we are comparing n to n^2. Polynomials of larger degree grow faster.
   - g_7 comes before g_6. Taking logarithms, we are comparing n^2 to 2^n and exponentials grow faster than polynomials.

   (b) *(20 Points)* **Analyzing Running Time:**

   Answer:

   i. *(10 Points)* **Analyze the simple algorithm (step (1) and step (2)) :**

   The upper bound is $O(n^3)$. The outer loop runs for exactly n iterations and the inner loop runs for at most n iterations every time it is executed. Therefore the line of code that adds up entries $A[i]$ through $A[j]$ is executed for at most n^2 times. Adding up entries $A[i]$ through $A[j]$ takes $j - i + 1$ operations, which is $O(n)$; storing the result of the addition in $B[i, j]$ requires constant time $O(1)$. Therefore, the running time of the entire algorithm is at most $n^2 \cdot O(n)$ or $O(n^3)$.

   The lower bound is $\Omega(n^3)$. Consider the times during the execution of the algorithm when $i \leq \frac{n}{4}$ and $j \geq \frac{3n}{4}$. In these cases, $j - i + 1 \geq \frac{3n}{4} - \frac{n}{4} + 1 > \frac{n}{2}$. Therefore, adding up the array entries $A[i]$ through $A[j]$ would require at least $n = 2$ operations, since there are more than $n = 2$ terms to add up. During the execution of the algorithms, we encounter $(\frac{n}{4})^2$ pairs (i, j) s.t. $i \leq \frac{n}{4}$ and $j \geq \frac{3n}{4}$. Therefore, the algorithm must perform at least $\frac{n}{2} \cdot (\frac{n}{4})^2 = \frac{n^3}{32}$ operations, which is indeed $\Omega(n^3)$.

   Alternatively, you could try to compute the exact running time and then derive upper and lower bounds.

   ii. *(10 Points)* **Design a faster algorithm (step (3)):**

   Answer: The inefficiency of the previous algorithm lies in the fact that we are summing up $A[i]$ through $A[j]$ for every $B[i, k]$ with $k \geq j$. This is obviously repeated and is a waste of time. Instead, we could incrementally compute $B[i,k] = B[i,k-1]+A[k]$ doing only one (instead of $k-i+1$) addition. The following improved algorithm implements this idea:

for $i = 1, 2, \cdots n$ **do**
  $B[i, i+1]{:=}A[i] + A[i+1]$
end for
for $k = 2, 3 \cdots n-1$ **do**
  for $i = 1, 2...n-k$ **do**
    $j{:=}i + k$
    $B[i; j]{:=}B[i, j-1] + A[j]$
  end for
end for

The first For loop computes $B[i, i+1]$ for all i, which requires $O(n)$ operations. The second For loop computes all $B[i, j]$ for $j-i = k$ by setting $B[i, j] = B[i, j-1]+A[j]$. Notice that the values $B[i, j-1]$ were already computed in the previous iteration of the outer for loop, when k was $j-1-i < j-i$. For each k, this improved algorithm performs $O(n)$ operations since there are at most n $B[i, j]$'s such that $j-i = k$. $k < n$, so the algorithm has running time $O(n^2)$.

3. *(35 Points)* **Chapter 3 - Graphs.**

(a) *(10 Points)* **DFS vs BFS**
Answer: You can use proof by contradiction approach here. Suppose that $G = (V, E) \neq T$. Then there must be an edge $e \in E$ s.t. $e \notin T$, *i.e.,* e is a "non-edge " on this tree. Because T is a BFS tree, the non-edges are between layers of at distance 0 or 1. Because T is a DFS tree the non-edges are between layers of distance ≥ 2. Contradiction. Therefore there cannot be $e \in E$ but $e \notin T$ and $G = T$.

(b) *(10 Points)* **Connectivity** Consider an undirected graph $G = (V, E)$ with $|V| = n$ nodes. Assume that there are two nodes s and t with distance between them strictly greater than $\frac{n}{2}$.
Answer: We discussed this problem during Discussion 2 - please see slides there for details.

- Prove that there must exist some node v, different from s, t, such that deleting v from G disconnects s from t.
Proof: Run BFS from s and organize nodes in layers, with s being at layer 0. Node t must lie at a layer strictly higher than $\frac{n}{2}$, since its distance from s is strictly greater than $\frac{n}{2}$. There are strictly more than $\frac{n}{2} - 1$ layers between s and t, and excluding s, t. There must be some layer between s and t with exactly one node. Indeed, if all layers had more than one nodes, then there would be strictly more than $2 \cdot (\frac{n}{2} - 1) + 2 = n$ nodes in the graph (counting s, t and the layers in between), which is impossible. Therefore, if you remove the single node in that layer, you disconnect s and t.
- Draw one example of G, s, t, v.
Answer: See figure below. There are $n = 8$ nodes in the graph. The distance between s and t is $5 > 4 = \frac{n}{2}$. There are not one, but two nodes (the first and seconf left from t) that can disconnect s from t.

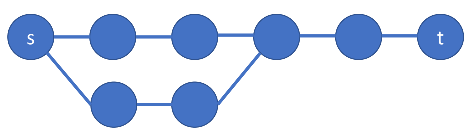


Figure 1: Cut Vertex for $n = 8$

- Provide an algorithm that finds such a node v in $O(m + n)$ time.
Answer: Run BFS and keep a track fo the layers and the number of nodes in each layer. This is possible by adding two more variables per node (layer and counter within layer), and assigning them once (when the nodes is visited). When you find a layer with only node, terminate (before proceeding to next layer). The complexity of this slightly modified BFS is still $O(m + n)$.

(c) *(5 Points)* In the given graph , there are two directed cycles:

i. $2 \rightarrow 3 \rightarrow 4 \rightarrow 2$
ii. $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$

Hence, for all nodes on these two cycles, the answer is yes and for others the answer is no.

(d) *(5 Points)* The second graph is bipartite. The reason is that we can color this graph with two colors using BFS (Fig. 2). The other solution is using the fact that if a graph is bipartite with a cycle, the length of the cycle should be an even number. If you find a cycle with an odd length then the graph is not bipartite (Note that this is for the proof of not being a bipartite graph. So, having cycles with even length does not prove that the graph is bipartite.) The first graph has a cycle with length of 7 which makes it to be not bipartite.

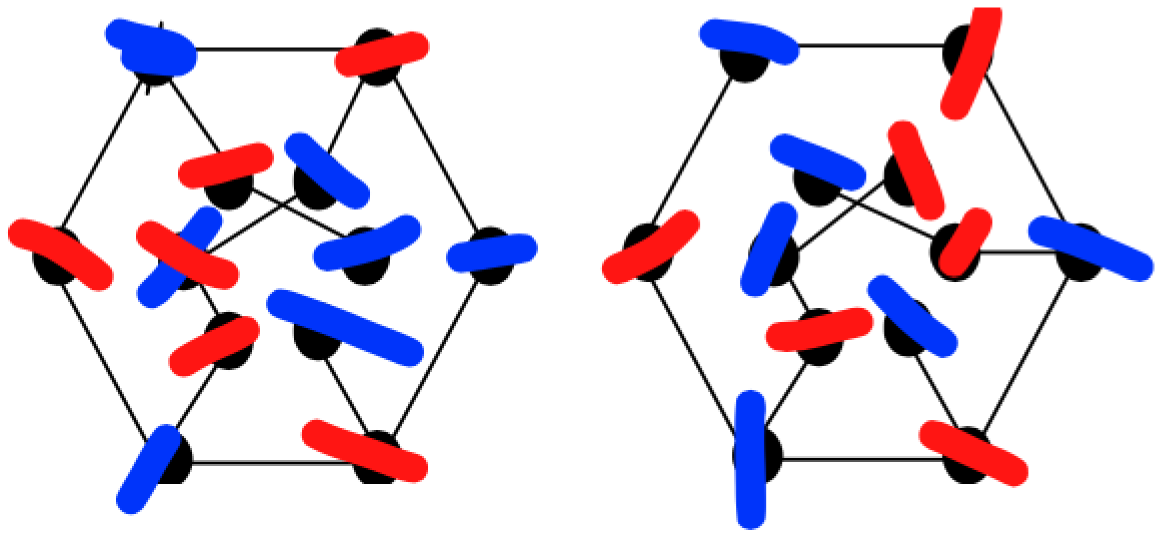
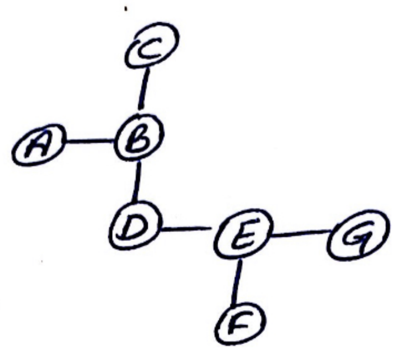


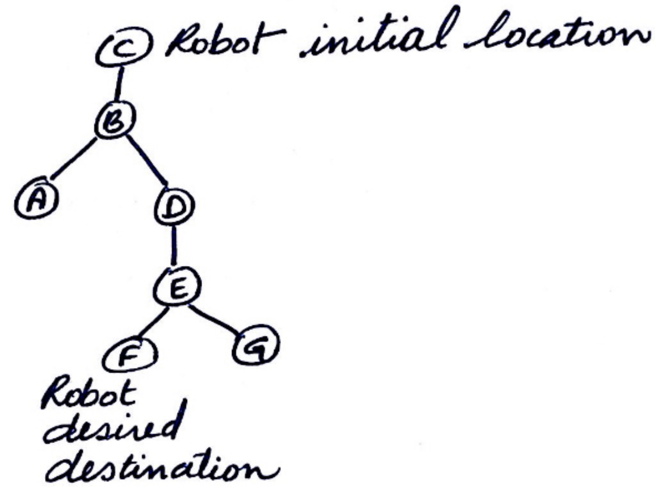
Figure 2: two coloring of the graphs

(e) You can use BFS. Specify the steps.

- Graph



- BFS exploration tree



- Route $C \rightarrow B \rightarrow D \rightarrow E \rightarrow F$

- Go back. No need to run BFS again you can just reverse the arrows

  $F \rightarrow E \rightarrow D \rightarrow B \rightarrow C$

- Distance covered is 4 hops

Figure 3: Robot Motion Planning Problem