Problem 1 (Augmenting a matching):-

(a) We have matching M & P is the augmenting path for M.
When we augment M with P, we get new matching $\hat{M} = M \oplus P$

$$M \oplus P = (M - P) \cup (P - M)$$

$M \oplus P$ is symmetric difference which means we include all the edges that are not in M but in P and all edges that are in M but not in P.

To prove the augmentation $\hat{M}$ is also a matching we know the all edges that are in $\hat{M}$ comes from;

(i) All the edges that are not in M but are present in augmenting path P. We know by the definition of an augmenting path that it starts and end with edge that is not matched by matching M & it has all alternate edges that are in M & not in M. it means $\hat{M} = E - M$ & is also odd length.
Since we have eleminated the edges that are in M & P we are left with all the edges which are disjoint means no two edges touch a comman vertex. This is guaranteed by not considering the edges that are in M, thus discarding all the edges that were touching a comman vertex.

(ii) All the edges that are in M but not in P, by definition of matching we know that all the edges that must be in M must be disjoint. & Since we are considering edges that are only in M but not in P, they are also be disjoint because no edge will have the Comman vertex in M.

from both (i) & (ii) we are left with edges that are always disjoint meaning no edges have common vertex.

By matching's definition, we already know that matching is an edge subset such that all edges in M touch a disjoint vertex. So, the augmented $\tilde{M}$ given by $M \oplus P$ is also a matching.

(b) $|\tilde{M}| = |M| + 1$?

$\tilde{M}$ is generated by augmenting P ~~with~~ to M using symmetric difference.

We know, by definition of augmenting path that length of P is always odd. & alternate path start & end that is not in matching & then they alternate.

When we augment matching with the path we notice that an alternating paths we fill the state, we add one to augmentation & take the next one out of augmentation so for every successive edge one is going in $\tilde{M}$ while the next one is

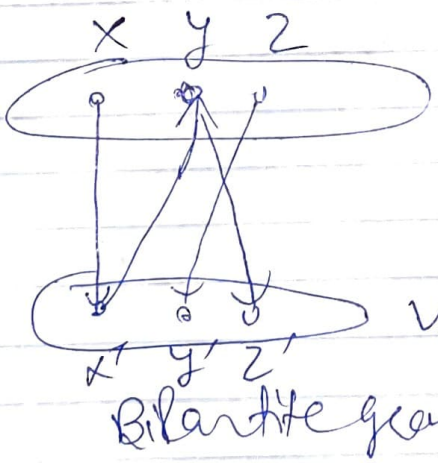
coming out of $\hat{m}$ leaving us with no net gain until the last one because we know the length is always odd, the last one added is not in matching leaving us with an additional edge.

So, if we have a matching M & an augmenting path P by augmenting our matching with Path P effectively along the path we flip the state of every edge meaning affording the edge not in M but in P & what was already in matching M & also in P is taken out. Doing this results in a new matching $\hat{m}$ which has length $|\hat{M}| = |M| + 1$. or cardinality.

Problem 2 (Path and cycle cover) :-

We are given a directed graph $G = (V, E)$
We need to find Path/cycle cover of G.
Such that the path or cycle cover consists of vertex disjoint directed paths/cycles such that every vertex will have both indegree & outdegree atmost 1.

We first need to create a bipartite graph G' from the given directed graph $G = (V, E)$. This new created bipartite will help us to find paths or cycle which will have the maximum weight matching. This bipartite graph G' which has a vertex set containing two copies of vertex v from the given directed graph G. We can say the vertex copies as V_1 & V_2.
For each edge in E we add an edge (u_1, u_2) to the bipartite graph G'.



Bipartite graph G'

To find the path which has the maximum weight matching, we can use the algorithm for finding the maximum weight matching in bipartite graph which is taught in the class.
We now use this maximum weight matching to get a path by including all the edges & vertexes that are included in the maximum weight matching.
We take these edge and vertex & append it our solution we now can remove the vertex that were present in the maximum weight matching we also remove the edges that were associated with these ~~edges~~ vertices. We add this path with maximum weight to collection s/n. By doing this, we left with paths which will not have any common vertex. So, we end up with the cycle cover having only disjoint vertex.

Now, we use the same maximum weight matching in ~~then~~ ~~get~~ bipartite graph. to get another path which is vertex disjoint.
We keep on doing this until we cannot find a new matching in a bipartite graph.

We can then just return all the edges from the above method. This will result in a cycle cover with disjoint vertex directed paths & cycles with each of them

having maximum weight so we have cycle cover of maximum total weight.

Running Time :-

(1) We are using Dijkstra's algorithm to find the shortest path which gives us maximum weight matching & thus takes $O(m + n \log n)$ time.

(2) We find matching for new Bipartite graph which takes $O(n)$ times.

Thus total time taken by algorithm $O(mn + n^2 \log n)$.