3.13 (a) Use BFS to find the distance between x, y. Following is the pseudocode.

```
BFS(G,x,y){
   initialize Q;
   // x as start; gray means to be visited
   x.height = 0; x.status = gray;
   // white means not visited; black means visited
   other vertexes are all white;
   Q.push(x);
   while(Q is not empty)
      u = Q.pop();
      for (v in neighbors of u)
         if (v.status = white)
            v.status = gray;
            v.height = u.height + 1;
            Q.push(v);
            if(v == y)
               return v.height; // v's height is distance(x,y).
      u.status = blace;
}
```

(b) The necessary and sufficient condition of that a graph is a bipartite graph is the length of the cycles in graph must be even.Use DFS to find the length of a cycle.

```
DFS(G,s){
   initialize Q; // Q is a stack
   Q.push(s); s.status = gray; // s is the start
   all other nodes are white;
   while(s is not empty)
      u = Q.pop();
      for(v in neighbors of u)
         if (v.status == white)
            v.status = gray;
            u.height = v.height + 1;
            Q.push(u);
         else // i.e. a cycle exists
            len_cycle = u.height - v.height + 1; // len_cycle = d(u,v) + 1
      if(len_cycle == odd)
         return G is not bipartite;
      u.status = black;
   return G is bipartite;
}
```

A cycle is containing a path from u to v and the edge uv. The distance between u, v is $u.height - v.height$. So $LengthofCycle = u.height - v.height + 1$.

3.14 We can separate spanning trees into 5 categories according to the number of deleting edges among the middle 4 edges as in Figure 1. Then calculate number of spanning trees respectively.

In category a, to form a tree, we need to remove one edge in the three small cycle.

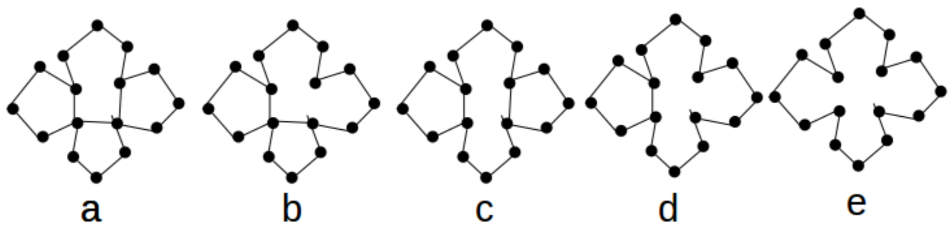
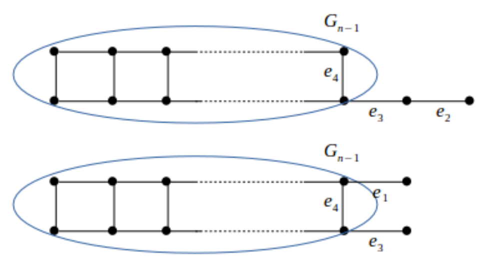


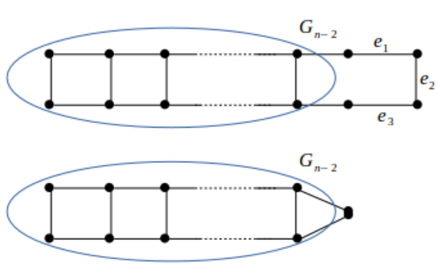
Figure 1: Five categories

What's more, the remaining 7 edges form a large cycle. We also need to remove one edge among the 4 edges because the other three middle edges can not be removed in category a. We can choose one from the four middle edges. So $\tau(a) = \binom{4}{1}^5$ Similarly, $\tau(b) = \binom{4}{1}^3\binom{8}{1}$, $\tau(c) = 2\binom{4}{1}^2\binom{8}{1}$, $\tau(d) = 4 \times 12 \times 4$, $\tau(e) = 16$. Therefore, $\tau(G) = \tau(a) + \tau(b) + \tau(c) + \tau(d) + \tau(e) = 2000$.

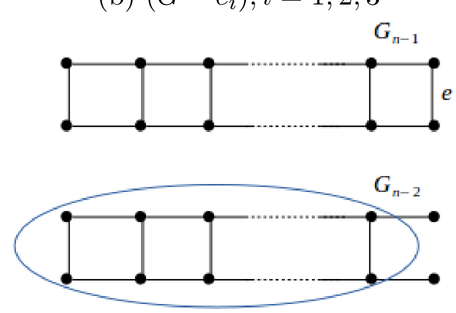
(a) G_n



(b) $(G - e_i), i = 1, 2, 3$



(c) $(G - e_4)$ and its equivalent graph G'



(d) Derive of Contraction of $(G - e_4)$, G_{n-1}, G''

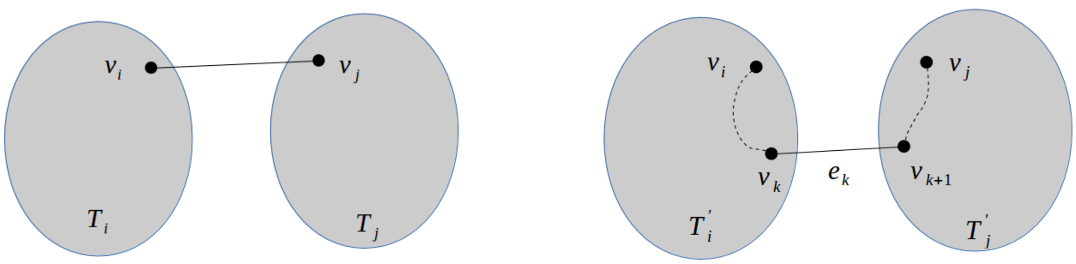
Figure 2: G_n and its subgraphs

3.16 *Proof.* When $n = 1$, $\tau(G_1) = 1$. When $n = 2$, $\tau(G_2) = 4$.
When $n \geqslant 3$, in Figure 2a, $\tau(G_n) = \tau(G_n - e_1) + \tau(G_n - e_2) + \tau(G_n - e_3) + \tau(G_n - e_4)$.
According to Figure 2b, it is obvious that $\tau(G_n - e_1) = \tau(G_n - e_2) = \tau(G_n - e_3) = \tau(G_{n-1})$.
From Figure 2c, $\tau(G_n - e_4) = \tau(G')$. And with contract, from Figure 2d, we can derive that $\tau(G') = \tau(G_{n-1}) - \tau(G'')$. And it is obvious that $\tau(G'') = \tau(G_{n-2})$. So

$\tau(G_n - e_4) = \tau(G_{n-1}) - \tau(G_{n-2})$.
Therefore, $\tau(G_n) = 4\tau(G_{n-1}) - \tau(G_{n-2})$. □

3.19 Lemma: An edge connecting two trees forms another tree.
The lemma is obviously true because the formed graph is connected and has no cycles.

Proof. T, T' are two spanning trees of graph G. We can group the vertexes of graph G as Figure 3a, T_i, T_j are two trees rooted at v_i, v_j respectively. In Figure 3b, vertexes of groups is the same with T, but not rooted at v_i, v_j.
Then, $\forall e : v_iv_j \in E(T)\backslash E(T')$, there exists a path $P : v_i - v_j$ in T'. And $\exists e' = e_k$ such that $v_k \in T_i', v_{k+1} \in T_j'$, i.e. $v_k \in T_i, v_{k+1} \in T_j$. We are easy to prove $e_k \notin T$. Because if $e_k \in T$, a cycle will be formed in T.
Therefore, according to lemma, $T - e + e', T' + e - e'$ are also spanning trees.



(a) Spanning Tree T (b) Spanning Tree T'

Figure 3: Spanning Trees of G

□

3.21 *Proof.* Proof contains two parts.

(a) Prove T is a spanning tree
According to the algorithm, a node is added to S one by one. At start, S contains one node. It is a tree, named T. And with a node added to T, because the node is also a tree, T adding the node will also be a tree by a edge connecting the node with T. This procedure continues until $S = V(G)$. T is always a tree.

(b) Prove T is a minimum spanning tree
Let T^* is a minimum spanning tree(MST) which shares most common edges with T. If $T = T^*$, T will be a MST.
If not, $\exists e \in E(T)\backslash E(T^*), e = uv$. Then a path P exists between u, v. And $\exists e' \in P$ as in Figure 4 such that one endpoint is in S, the other in $\overline{S}$. What's more, $e' \in E(T^*)\backslash E(T)$, (if not, cycle will exist in T). Then according to **Property 3.23**, $T' = T^* - e' + e$ is another spanning tree. And since Prim's algorithm always find the least weighted edge between S and $\overline{S}$, $w(e) \leq w(e')$. Then $w(T') \leq w(T^*)$. It means T' is also a MST. But T' shares one more common edge with T, which is a contradiction to assumption of T^*.
Therefore, T is a MST.

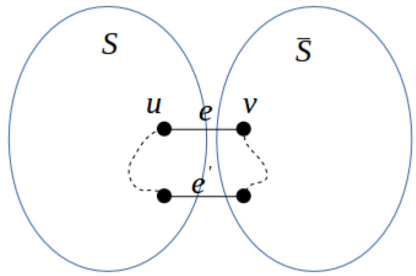


Figure 4: Spanning Tree

□