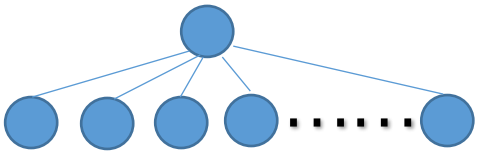
Problem 6.1

相当于从 n-1 个边中寻找权值最小的边，那么只需要遍历一次，比较 n-2 次即可。



Problem 6.2

1. 初始时只有一个顶点，那么有$n-1$条备选边，进行$n-2$次边的权重比较之后向生成树中添加第i个顶点，此时这个顶点开始比较$(n-i)$次，执行updateFringe，然后在剩余边比较$(n-i-1)$次再次得到权值 min。

   故总共进行了$n-2+\sum_{i=2}^{n-1}\big((n-i-1)+(n-i)\big)=(n-1)(n-2)$次比较

2. $n-1$

Problem 6.4

当图稀疏的时候，应用 Kruskal 算法，当图稠密的时候，应用 Prim 算法。

Kruskal 算法困难点在于判断加入此边后是否会有环，在此使用并查集，通过判断边的两个顶点是否在一个并查集中，如果不在则此边加入最小生成树。对于一个图，它需要先以 O(eloge)的代价对所有边进行排序，然后从头到尾判断一遍即可，维护一组并查集(数组)。

Prim 算法困难点在于每次选择权值最小的边，它需要维护一组优先级队列，并且每次加入一个点后需要修改优先级队列中的值，其时间复杂度主要在于每一次修改值，以及取出 min 后维护优先队列。

Prim 算法的时间复杂度为 O(n2)。与图中边数无关，该算法适合于稠密图。

Kruskal 算法需要对图的边进行访问，所以 Kruskal 算法的时间复杂度只和边又关系，可以证明其时间复杂度为 O（eloge）。

所以对于一个图，点个数一定，因为 Kruskal 算法时间复杂度与边数有关，当图稀疏时，Kruskal 算法时间复杂度优于 Prim 算法，当图稠密时，Prim 算法时间复杂度优于 Kruskal 算法。

Problem 6.5

1. 可以使用改版的 Kruskal 算法或者 Prim 算法，只需要把选择权值最小的边条件改为选择权值最大的边即可。算法正确因为可以取图中权值最大的边 MAX_EDGE，将所有边的权值改为 MAX_EDGE 减去之前权值，然后使用 Kruskal 算法或者 Prim 算法生成最小生成树。故算法正确。
2. 先用第一小题算法生成最大权重生成树(边集为 U)，然后 F = E(G) – U。

Problem 6.6

不可能。因为对于点 s，取与它相连的权值最小的边<s, v>，那么这条边必定为 s 到 v 的最短路径，即<s, v>在最短路径树中。对于 MST，假设<s, v>不在 MST 中，那么将边<s, v>加入到 MST 中必定会构成一个环，并且环中的一条边<s, w>的权值大于<s, v>，这与 MST 性质矛盾，故得证。

Problem 6.7

a) 由于$e \notin E'$且$\dot{w}(e) > w(e)$，根据 MST 性质，当 e 加入E'后生成的环， e 必定是环中权值最大的边，那么在权值由$w(e)$变为$\dot{w}(e)$后，MST 性质依旧成立，故最小生成树不变。

b) 由于$e \notin E'$且$\dot{w}(e) < w(e)$，根据 MST 性质，当 e 加入E'后生成的环， e 必定是环中权值最大的边，那么在权值由$w(e)$变为$\dot{w}(e)$后，MST 性质不一定成立，这时候我们只需要遍历一遍环中每个边的权值，得到权值最大边为 MAX_EDGE，如果此边为 e，则最小生成树不变，否则 T 的$E' = E' - \text{MAX_EDGE} + e$。

c) 由于$e \in E'$且$\dot{w}(e) < w(e)$，在修改之前，MST 性质成立，在修改之后，对于任意边，其值依旧为环中权值最大的边，故最小生成树不变。

d) 由于$e \in E'$且$\dot{w}(e) > w(e)$，那么更改后 MST 性质可能不成立，这时候我们取两个点集 E1 和 E2，分别为 T 去除边 e 后的两个点集(当然存在空集的可能)，此时我们遍历边的两个 vertices 分别位于 E1 和 E2 的所有边，取其中权值最小的为 MIN_EDGE，如果 MIN_EDGE = e，那么最小生成树不变，否则 T 的$E' = E' + \text{MIN_EDGE} - e$。

Problem 6.8

```
1   Graph light_spanning_tree(G, U, V, E){
2       Vectices Union temp_V = U - V;
3       temp_E = E - {all edges contains vectices of U};
4       Tree T = prim(temp_V,temp_E);
5       if(T == NULL)
6         return NULL;
7       for all vectices in U{
8         vertex v = get_vertex(U);
9         if(there is no edge between v and temp_V)
10          return NULL;
11        else
12          Look for the edge e that is the least weighted between v and temp_V;
13          T = T + e;
14          U = U - v;
15      }
16      return T;
17  }
```

Problem 6.9

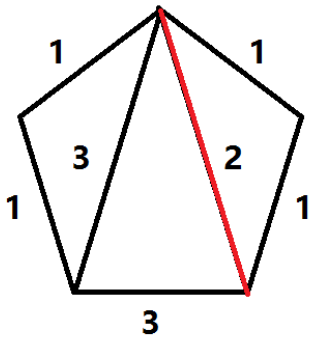
应用改版的 Kruskal 算法，只需要在初始化最小生成树 T 时将 E' = S，并查集初始化每一个顶点各自为一个集合后，执行下面的 Kruskal_S()函数

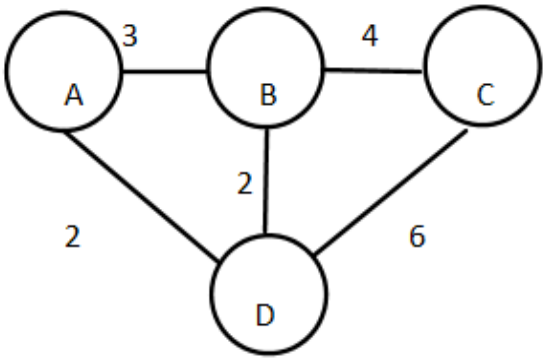
```
1  Kruskal_S(…){
2  For each vertex v of S, in some order
3      if (color[v]==white)
4          DFS(v) in S, and unite them;
5  }
```

之后再对E − E′的所有边进行权值排序，然后按照从小到大的顺序判断此边的两个 vertices 是否在同一个并查集中即可，如果不在就在 E′中添加此边，并 unite 这两个并查集，执行此循环直到 unite 到只剩一个并查集。

Problem 6.10

1. 错误, 当这一条唯一的最重边的一个 vertices 是叶节点时, 所有的生成树必定包含此边。
2. 正确，由于最小生成树具有 MST 性质，假设最重边 e 属于一个最小生成树，那么连结在环中的 e 的邻边，必定存在一个会在最小生成树中形成一个包含 e 的环。由于此邻边的权值在环中不是最重边(因为 e 比他大)，所以不满足 MST 性质，矛盾。
3. 错误，对于一个权值均相等的图，它的每一条边都是最轻边，但是有的边不属于某个最小生成树(边数大于n − 1的连通图)。
4. 正确，假设其不是最轻边，那么在最小生成树 T 中加入此边必定会形成一个环，而由于它是最轻边，它的权值不可能是环中最大值，故不满足 MST 性质，与 T 为最小生成树矛盾，得证。另外在 Kruskal 算法中，由于它的权值最小，它必定作为第一条边加入最小生成树中。





5. 错误，如上图所示。
6. 错误，如右图所示, 可见 A 到 C 的最短路径是 A-B-C, 但是最小生成树中只包含边<A,D>, <B,D>， <B,C>，<A,B>不属于最小生成树的一部分。

7. 正确，证明 prim 算法时，其过程对于负权重仍然成立。

Problem 6.11

命题正确。

充分性：已知 T 是 G 的最小生成树，故 T 满足 MST 性质，那么对于 T′，对于任意的非 T′边 e，加入 T′后必定构成一个环，由于在 T 中 e 为权值最大的边，在 T′中平方后 e 为权值最大的边依然成立，故 T′为 G′的最小生成树。

必要性：已知 T′是 G′的最小生成树，故 T′满足 MST 性质，那么对于 T，对于任意的非 T 边 e，加入 T 后必定构成一个环，由于在 T′中 e 为权值最大的边，在 T 中开方后 e 为权值最大的边依然成立，故 T 为 G 的最小生成树。

Problem 6.12

1. 已知最小生成树可以通过一条权值相同的边来两两转化，由于所有边的权重各不相同，所以最小生成树唯一。

   引理 1：一个环的顶点集合任意划分成两个非空子集，则至少有两条边的顶点分别属于这两个子集。

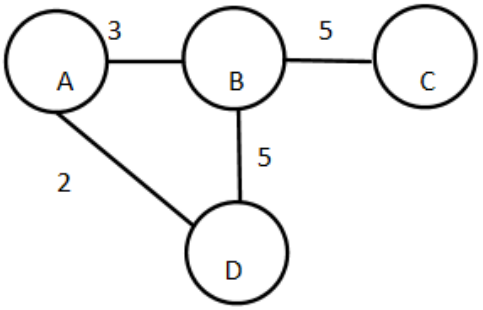
证明：若不然，则情况 1：两个子集之间无边相连，该环不连通，矛盾；情况 2：两个子集之间只有一条边相连，设为 UiUj，则这条边是 Ui 通向 Uj 的唯一路径，与其在环中矛盾。

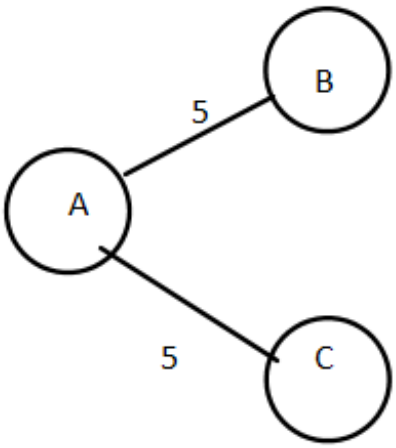
引理 2：一个每条边权重不同的连通图中的任意一个环中的最长边不会存在于该图的任何一棵最小生成树中。

证明 :设每条边权重不同的连通图(U, V) 中存在环 C, 这个环为的顶点集为{u1,u2,...,uk}, 其中的最长边为 ui-uj，假设这条边存在于某最小生成树 Y 中。在 Y 中去掉边 ui-uj，则该最小生成树被分成两个不连通的子树，子树各自是连通的（该证明结论很直观，说明过程冗长，略过），子树 1 包含环 C 的顶点的子集 C1，子树 2 包含 C\C1，两个集合均非空（一个至少包含 ui，另一个至少包含 uj，否则在 Y 中去掉边 ui-uj 而不影响连通性，与 Y 是最小生成树矛盾）。由引理 1 知存在环 C 中的另一条边 ul-um 可以连接两个子树，且 ul-um 的权重小于 ui-uj，这样得到的生成树 Y1 总权重小于 Y，与 Y 是最小生成树矛盾。

最后是该命题的证明：

设存在两个不同的生成树 Y1,Y2，Y1 不等于 Y2，则必然存在 e∈Y1 且 e 不属于 Y2，否者 Y1 包含于 Y2，Y2 又是最小生成树，两个树相等，矛盾。将 e 加入 Y2 中，形成一个包含 e 的环 C，由引理 2，C 中存在边 f，使得 f 的权重小于 e 的权重，将 f 去掉不影响连通性，且得到的树的总权重小于 Y2，与 Y2 是最小生成树矛盾。





2. 详见上图。
3. 错误。对于右图(只有三个 vertices 的)不满足(a)，因为{A}与{B,C}中间的最小权重不唯一。
4. 先用 Kruskal 算法或者 Prim 算法生成最小生成树，接下来遍历剩下的每一条边，将这条边加入最小生成树，并找出形成的环。如果这条加入的边的权值比这个环中其他的任意一条边都大，那么这个图拥有一棵唯一的最小生成树。

Problem 6.13

1. 由于有用边不存在于任何圈中，假设边 e 为有用边，则 e 必为割边，所以 e 必在生成树中，因此 e 必属于最小生成树中。
2. 设 e 为危险边，假设 e 在最小生成树中，那么根据性质在危险边所在的某个圈中必有一条不在最小生成树中的边 e’ ，那么最小生成树中加入 e’ 后必定会形成一个包含 e 的环，由于 e 是危险边，所以 e 的权值大于 e’ ，那么将最小生成树去除边 e，加入边 e’ ，得到的新的树权值小于初始的最小生成树，与其之前为最小生成树矛盾，故得证。
3.

```
1  Graph anti_Kruskal(){
2      init T = G;
```

```
3       Sort edges in descending order of weight;
4       visit all edges of the G{
5           int e = get_one_edge(G);
6           int v , w = two_vertices_of_edge(e);
7           Graph temp_G = G - e;
8           int max_weight = DFS(temp_G, v, w);
9           if((max_weight != -1)&&(max_weight < weight(e)))
10              T = T - e;
11      }
12      return T;
```

第三行 O(mlogm)、第四行 O(m)、第八行 O(n+(m-1))

故时间复杂度为 O(mlogm + m*(n+(m-1)))。

Problem 6.14

不能。

如右图，将 V_1={A, C}, V_2={B, D}

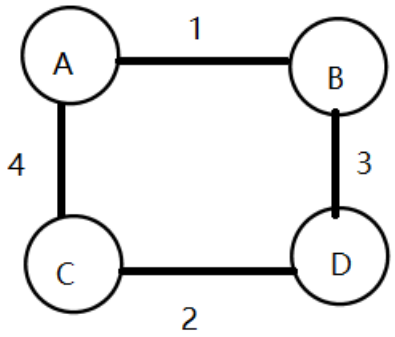
那么生成的最小生成树为

<A, C>, <A, B>, <B, D>

而正确的最小生成树应该为

<C, D>, <A, B>, <B, D>

故算法错误

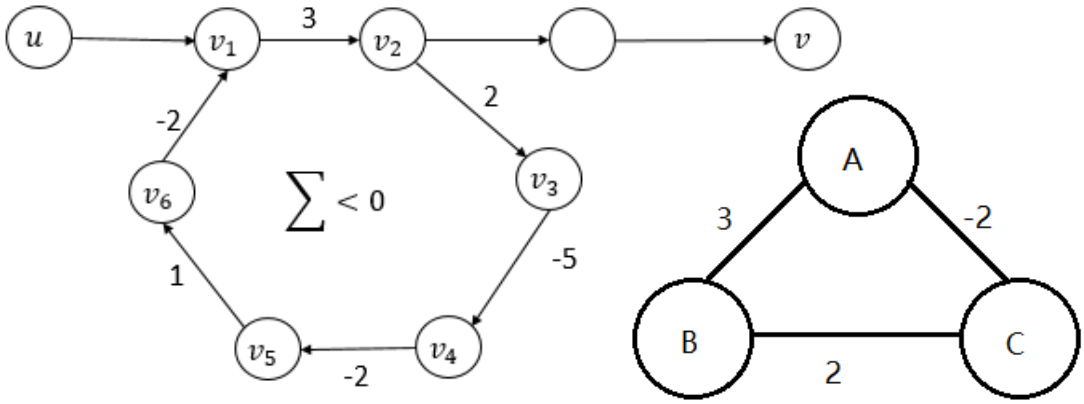


Problem 6.15

```
1   int The_second_minimum_spanning_tree(G, V, E){
2       T<temp_V, temp_E> = Prim(G);
3       int BST_weight = sum of G<V, E>;
4       int max[v][v]; //max[i][j]=the max weight form i to j
5       init(max) //init max[i][j] = -1;
6       call BFS() to update max[][];
7       int second_MST = INF;
8       for all V<u, v> not in T{
9           int temp_BST_weight = BST_weight - max[u][v] +
    weight(u, v);
10          if(second_MST > temp_BST_weight)
11              second_MST = temp_BST_weight;
12      }
13      return second_MST;
14  }
```

Problem 6.16

因为当权值可以为负时，可能在图中会存在负权回路，最短路径只要无限次地走这个负权回路，便可以无限制地减少它的最短路径权值δ(s, v)，这就变相地说明最短路径不存在，Dijkstra 算法无法终止。下图说明从 u 到 v 的最短路径是不存在的。



另外由于 Dijkstra 算法是贪心的，在右图示例中，
以 B 为起点，他会认为边<B, C>为 B 到 C 的最短路径。

Problem 6.17

因为 Dijkstra 算法每次更新路径的时候都是从新加入顶点集的顶点考虑更新的（因为他适用的有向无环图的特性保证了这样找一定是最短路径）因此在这个过程中可能会忽略到权值较小的边，而对于最小生成树的算法，拿 prim 算法为例，他是从顶点集中所有的顶点对外部顶点的对小边来考虑的，所反可以保证生成树最小，但是因为 DijKstra 算法每次只考虑新加入的顶点为整个顶点集的更新出口，所以应该有丢失最小权值边的可能。

反例：
V：1，2，3，4
E：(1,2)=3，(2,4)=4，(1,3)=1，(3,4)=5
从 1 出发得到最短路径生成树是{(1,2)，(1,3)，(3,4)}
但最小生成树是{(1,2)，(1,3)，(2,4)}

Problem 6.19

1. 不会发生变化，设最小生成树为 T，权值加一后为 T'，那么对于 T'中的任一条边 e'，加入 T'中必会形成一个环，由于 T 具有 MST 性质，e 在 T 中形成的环与 e'在 T'中形成的一样，并且都满足 e/e'是这个环中的最大权值边，故 T'满足 MST 性质，得证。

2. 会发生变化
   V：1，2，3，4
   E：(1,2)=1，(2,3)=1，(3,4)=1，(1,4)=4
   从 1 出发到 4 的最短路径是$1-2-3-4$，代价为 3。
   但是在所有边权重加一后，最短路径变为$1-4$，代价为 5。

Problem 6.21

适用。对于图 G 中任意顶点 u(u != s)，其 s 到 u 的最短单源路径有且仅有一条权值为负的边。不妨为图 G 中所有的负权值边加上一个正数 R 使其均大于 0，由于只且必须经过一次，相当于 s 到达所有点的最短路径加 R，路径不变。此时 Dijkstra 算法以 s 为起点成立，并且生成的最短路径与图 G 中以 s 为起点的相同，故 Dijkstra 算法适用。

Problem 6.23

1. 以 s 为起点，进行 DFS 遍历，如果遍历途中遇到权值le > L，则终止向下递归。如果 s 可以遍历到 t 点，则说明存在一条可行路径。
2. 可以使用经过变形的 Dijkstra 算法，只需要将 newDist = myDist + weight 更改为 newDist = max{myDist , weight}。最后返回的最短路径值即为油箱容量。

Problem 6.28

```
1  Find_shortest_path_via_v0(G, V, E, v0, src , dest){
2      Graph REV_G = {G 的所有边反向};
3      int a1[n][n] = Dijkstra(G, V, E, v0);
4      int a2[n][n] = Dijkstra(REV_G, V, REV_E, v0);
5      return a1[v0][dest] + a2[v0][src];
6  }
```