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，那么有条备选边，进行次边的权重比较之后向生成树中添加第个顶点，此时这个顶点开始比较updateFringe，然后在剩余边比较再次得到权值min。

故总共进行了次比较

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

* 1. 由于且，根据MST性质，当e加入后生成的环， e必定是环中权值最大的边，那么在权值由变为后，MST性质依旧成立，故最小生成树不变。
  2. 由于且，根据MST性质，当e加入后生成的环， e必定是环中权值最大的边，那么在权值由变为后，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。
  3. 由于且，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。
  4. 由于且，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。

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 | } |

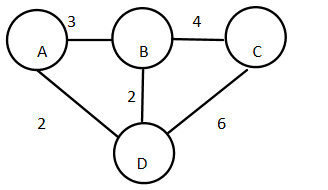
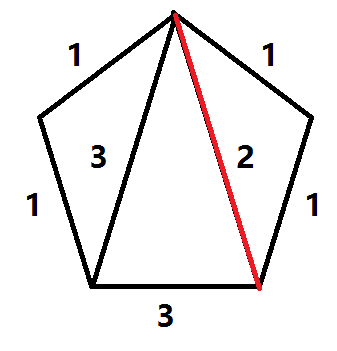
Problem 6.9

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。**

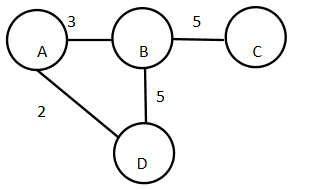
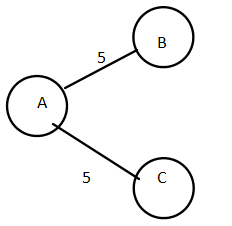
证明：若不然，则情况1：两个子集之间无边相连，该环不连通，矛盾；情况2：两个子集之间只有一条边相连，设为UiUj，则这条边是Ui通向Uj的唯一路径，与其在环中矛盾。

**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。



1. 详见上图。
2. 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。
3. 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

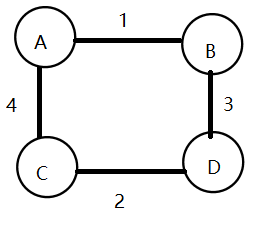
1. 由于有用边不存在于任何圈中，假设边e为有用边，则e必为割边，所以e必在生成树中，因此e必属于最小生成树中。
2. 设e为危险边，假设e在最小生成树中，那么根据性质在危险边所在的某个圈中必有一条不在最小生成树中的边e’，那么最小生成树中加入e’后必定会形成一个包含e的环，由于e是危险边，所以e的权值大于e’，那么将最小生成树去除边e，加入边e’，得到的新的树权值小于初始的最小生成树，与其之前为最小生成树矛盾，故得证。

|  |  |
| --- | --- |
|  | |
| |  |  | | --- | --- | | 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V1={A, C}, V2={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>

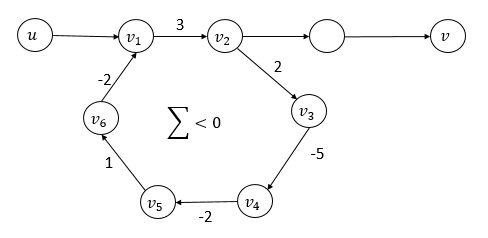
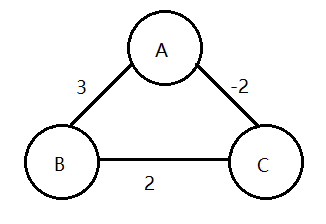
故算法错误

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，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，代价为3。  
但是在所有边权重加一后，最短路径变为，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，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 | } |