Problem Set 5 Answer Sheet

151220131 谢旻晖

Problem 6.1

$n-2$

Problem 6.2

1.

第一个点是我们手动设定的，无需比较。

从第二个点开始，每次 $getMin$ 将第 i 个节点加入 $tree$, 要对所有 $fringe$ 的点中比较以得最小权者 ($n-i$ 次 cmp), 同时需要对所有其他的 $fringe$ 点进行 $updateFringe$($n-i$ 次 cmp)。

$$\sum_{i=2}^{n} n-i+n-i=(n-1)(n-2)$$

2.

$n-1$

Problem 6.5

1.

将图所有边权取负，求新图的 MST，再将所有边权恢复，MST 即为最大权重生成树。

2.

求最大权重生成树的补图，其中所有的边形成的集合为最小的反馈边集。

Problem 6.6

是不可能的。下面证明：最短路径树与任何最小生成树都至少共用一条边。

证明：考虑从 s 点开始，我们使用 $Prim$ 算法构造 MST 的过程，第一步，我们从所有与 s 相连的边中选出权最小的那条边 (设为 $s\to v$), 此时我们就保证了 s 到 v 的最短路径是 $s\to v$, 换而言之，$s\to v$ 一定既在 MST 中，又在 SSSPT 中。证毕。

Problem 6.7

case a: 仍然不变

case b: 把修改的边加入原生成树，在生成的环中去掉权值最大的边（破圈）

case c: 仍然不变

case d: 将修改的边删除，原生成树形成两个连通分量，可以用 SCC 算法识别出两个连通分量，接着遍历所有的边，找到其中权值最小的且连接两个连通分量的边，加入。

Problem 6.8

令 $\varPhi=V-U$,$\varPsi=\{(x,y)|x,y\in\varPhi\wedge(x,y)\in E\}$，首先考虑图 $G'=(\varPhi,\varPsi)$ 的 MST，如果 G' 都不连通，则最轻生成树显然不存在，如果连通，我们求出 G' 的最小生成树 T。

接着考虑边集 $\Gamma=\{(x,y)|x\in\varPhi\wedge y\in U\wedge(x,y)\in E\}$, 不断选 Γ 中权值最小的边加入 T，以将 U 中的结点连接至 T, 已连接过的结点不再重复连接。直到将 U 中的结点全部连接完毕，T 为最轻生成树。若 U 中有点与 T 不连通，则不存在最轻生成树。

Problem 6.9

$Krystal$ 改进版本：设 G 的边集为 E，首先将 S 中的所有边选入，然后在 E-S 中不断选不与已选边构

成环且权值最小的边，直到总共选满 $|G.V|-1$ 条边。

Problem 6.11

命题正确。

$\Rightarrow$

采用反证法，如果 T 是 G 的 MST，但 T 不是 G' 的 MST。那么 G' 一定存在一个权值比 T 小的最小生成树 T'。根据平方关系，T' 在 G 中的权也小于 T, 与 T 是 G 的 MST 矛盾，假设不成立。所以 T 是 G 的最小生成树，那 T 一定也是 G' 的最小生成树。

$\Leftarrow$

同理。

Problem 6.12

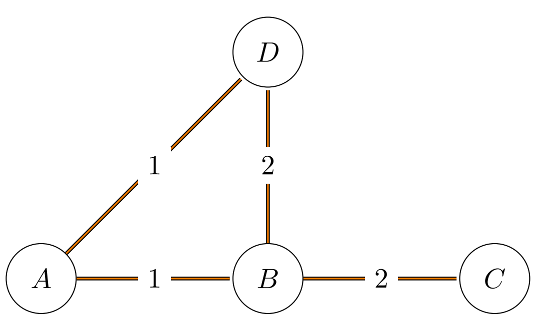
1.

采用反证法证明，若存在两个最小生成树 T_1 和 T_2, 我们找出权值最小的 T_1 和 T_2 中不同的边 e，不失一般性，不妨设他在 T_1 但不在 T_2 中。

根据 MST Property，在 T_2 中加入 e，则构成圈且有 e 为圈中权值最大的边，若圈中其他非 e 的边都是 T_1 中的边，则这些边和 e 就会在 T_1 构成圈，与 T_1 为 MST 矛盾，所以**必定有**某条边 x 在 T_2 但不在 T_1 中。

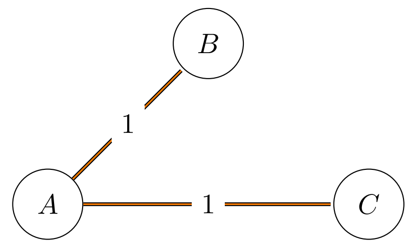
由 MST Property,x 的权 $<e$ 的权，这与 e 是**权值最小**的的 T_1 和 T_2 中不同的边矛盾。所以假设不成立。

2.



3.

给出图显然具有唯一的 MST，但他两个条件均不满足。



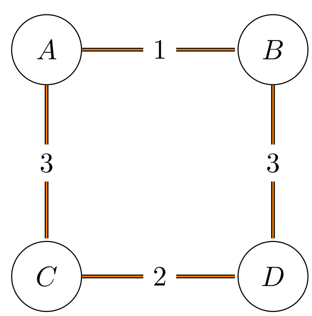
4.

可以将 MST Property 拓宽为 Unique MST Property. 他的定义为: 向 ST 中加入任意一条边 e，在形成的圈中,e 的权最大且没有和他一样大的边。可以很容易的证明一棵树具有 Unique MST Property 性质，那就具有题意的唯一性。

为此，给出算法如下: 在构造出 MST 之后，遍历 E-MST，不断向 MST 中加入边，如果在形成的圈中它的权最大且没有和它一样大的，那么 OK，否则则不唯一。

Problem 6.14

容易举出一个反例



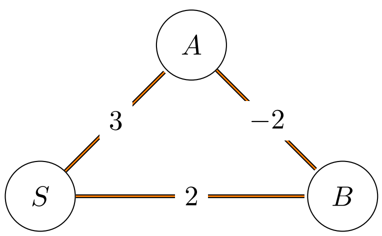
如图所示，令 $V_1=\{A,C\}$,$V_2=\{B,D\}$, 利用题意中的分治算法求出的 MST 为 {AB,AC,BD}, 权为 7，而实际上的最小生成树为 {AB,CD,AC} 或 {AB,CD,BD}, 权为 6。

Problem 6.15

首先，如果一个生成树是第二小的生成树，那么他一定与某棵最小生成树仅有一边之差。然后，先用 *Kruskals* 算法求出原图的 MST，接着我们尝试删去 MST 中的每条边，在此基础上再次运行 *Kruskals* 算法构建生成树（注意不能用刚刚删去的边），计算此树与 MST 的权差，找到其中权差最小但**非 0**（避免找到 MST）的 ST，即为次小生成树。

Problem 6.16

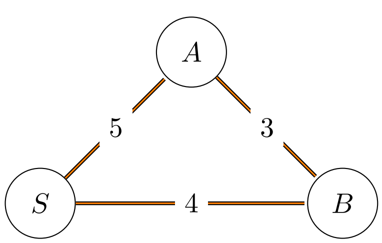
在此图上运行 *Dijkstra* 算法，源点为 S，会错误地得到 S->B 的最短路径为 2 的结论。



Dijkstra 由于是贪心的，每次都找一个距源点最近的点（dmin），然后将该距离定为这个点到源点的最短路径 (d[i]<–dmin); 但如果存在负权边，那就有可能先通过并不是距源点最近的一个次优点，再通过这个负权边生成的路径之和更小。

Problem 6.17

给定源点为 S，在下图中生成的最短路径树为 {SA,SB}, 但 MST 为 {SB,BA}。



Problem 6.18

Algorithm 1 UNIQUE_SSSP

Input: Graph

Output: U[0...|V|-1]

```
status[0...n-1],fringeWgt[0...n-1]
Priority Queue pq=create(n,status,fringeWgt)
pq.insert(s)
U[0...|V|-1]=FALSE
while !pq.isEmpty do
  v=pq.top()
  pq.pop()
  //updateFringe
  myDist=fringeWgt[v]
  for each vertex w in v.Adj do
    newDist=myDist+weight(v− > w)
    if status[w]==unseen then
      pq.insert(w)
    else if status[w]==fringe then
      if newDist<getPriority(pq,w) then
        decreaseKey(pq,w,v,newDist)
      end if
    else
      if newDist==myDist then
        U[w]=TRUE
      end if
    end if
  end for
end whileU[s]=TRUE
```

对 $Dijkstra$ 算法的松弛部分稍作改动即可，在松弛部分，若 $fringe$ 更新后的权与某个 $tree/finished$ 结点 v 的 distance 相同，则 v 的最短路径不唯一，$U[v] = TRUE$。

在 Sara Baase 教材的算法上作改动。其中 Priority Queue 的实现，status 的定义，decreaseKey、insert、pop、top 等函数的实现均与课本相同。

算法伪代码见 **Algorithm 1**.

Problem 6.19

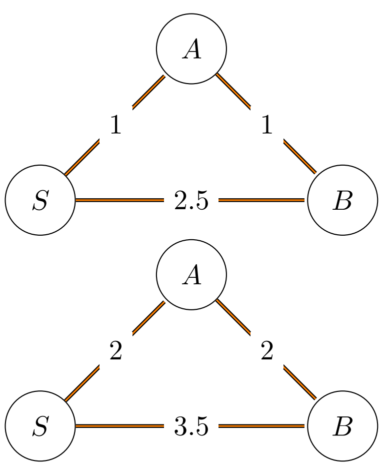
1.

没有变。证明：设原来的最小生成树为 T，加了权之后的生成树为 T'，下面证明 T' 仍然具有 MST Property：向 T' 中加入任意的一条边 e'，形成一个圈，显然 e' 是圈中权值最大的边，因为在 T 中加入与 e' 对应的 e 形成的圈中 e 是权值最大的边，所有边权值加一之后仍然是这样。

因为 T' 具有 MST Property，它仍然是 MST。

2.

会变化。



原来 $S \rightsquigarrow B$ 的最短路径为 $S \to A \to B$, 在所有边权值加一之后，$S \rightsquigarrow B$ 的最短路径为 $S \to B$.

Problem 6.20

Algorithm 2 Extended_SSSP

Input: Graph

Output: U[0...|V|-1]

```
status[0...n-1],fringeWgt[0...n-1]
Priority Queue pq=create(n,status,fringeWgt)
pq.insert(s)
fringeWgt[s]=c_s //初始化自己的权值为 c_s
U[0...|V|-1]=FALSE
while !pq.isEmpty do
  v=pq.top()
  pq.pop()
  //updateFringe
  myDist=fringeWgt[v]
  for each vertex w in v.Adj do
    newDist=myDist+weight(v−>w)+weight(w)//不仅加边的权，还加点的权
    if status[w]==unseen then
      pq.insert(w)
    else if status[w]==fringe then
      if newDist<getPriority(pq,w) then
        decreaseKey(pq,w,v,newDist)
      end if
    else
      if newDist==myDist then
        U[w]=TRUE
      end if
    end if
  end for
end whileU[s]=TRUE
```

在 $Dijkstra$ 算法的基础上稍作改动，初始化时令 Cost[s]=c_s，将框架中的”$newDist = myDist + weightEdge$” 改为”$newDist = myDist + weightEdge + weightVertex$”，其余不变。

算法伪代码见 **Algorithm 2**。

Problem 6.21

适用。

首先原图中我们认为没有负权圈，否则就可以一直转，无意义。

为什么普通的带负权的边 $Dijkstra$ 会失败呢？由于是贪心的，每次都找一个距源点最近的点（dmin)，然后将该距离定为这个点到源点的最短路径 (d[i]<–dmin); 但如果存在负权边，那就有可能先通过并不是距源点最近的一个次优点，再通过这个负权边生成的路径之和更小。

即 $s \rightsquigarrow v_i \to v_j$, 有可能存在 $s \rightsquigarrow v_i \xrightarrow{negative} v_j$，使得 $dist(v_j) < dist(v_i)$。

而在所有负权边都从源点发出的条件下，我们没有了 v_i，也就没有这种问题了。我们无法通过负权边，使得一个已经被添加进最短路径树 (换言之，已经固定下来的)，找到一个比他更短的 path。

Problem 6.23

1.

设油箱容量为 L，在 s 处进行 DFS 遍历，遍历的规则是如果边权大于 L, 就不继续遍历；小于等于 L，才继续往深里走。若遍历到了 t 节点，则存在一条可行路径，否则不存在。

2.

$Dijkstra$ 算法变形，在 s 处求单源结点的最短” 路径”, 只是将框架中的”$newDist = myDist + weight$” 改为”$newDist = max(myDist, weight)$”，其余不变。最后输出 $s \rightsquigarrow t$ 的最短” 路径” 为油箱最小容量。

Problem 6.25

Algorithm 3 FloydWarshallWithGoPath

```
fill dist[ ][ ] with ∞ and GO[ ][ ] with NUL
for each edge (u,v) do
  dist[u][v]=w(u,v) // the weight of the edge (u,v)
  GO[u][v]=v
end for
for k from 1 to |V| do
  for i from 1 to |V| do
    for j from 1 to |V| do
      if dist[i][j] > dist[i][k] + dist[k][j] then
        dist[i][j]=dist[i][k] + dist[k][j]
        GO[i][j]=GO[i][k]
      end if
    end for
  end for
end for
```

1. 算法见 **Algorithm 3**

2. 算法见 **Algorithm 4**

Problem 6.31

利用一个数组 income 记录这个结点接受到的 path 的数量，使用 BFS，每当上层结点遍历到下一层结点时，下一层结点的 income+= 上层结点的 income 值。直到遍历完 v 为止。

更为具体地，由于无向图只有 Tree Edge 和 Cross Edge。初始化时，初始化所有的 income=0, 接着从源点进行 BFS 遍历，若碰到 Tree Edge (x,y), 则 income[y]+=income[x]; 若碰到 Cross Edge，则忽略（因为不是最短路径了）。算法结束后的 income[v] 即为所求值。

Algorithm 4 FloydWarshallWithBackPath

```
fill dist[ ][ ] with ∞ and BACK[ ][ ] with NUL
for each edge (u,v) do
    dist[u][v]=w(u,v) // the weight of the edge (u,v)
end for
for k from 1 to |V| do
    for i from 1 to |V| do
        for j from 1 to |V| do
            if dist[i][j] > dist[i][k] + dist[k][j] then
                dist[i][j]=dist[i][k] + dist[k][j]
                BACK[i][j]=BACK[k][j]
            end if
        end for
    end for
end for
```