P5

黄云鹏

2023.5.17

10.3: (MST最小权边问题)

注意题干条件：各边权值各不相同

1）e_1一定在（某一个）最小生成树，可以用反证法证明；当然：

- Kruscal：e_1一定第一个被选中；
- MCE：选中e_1为CE的切，其中e_1一定是MCE，根据定理10.4，其必然属于 某个最小生成树；

2）e_2也一定在最小生成树中（不必考虑重边）

- Kruscal：当e_1被选中后，接下来一定选中e_2；
- MCE：设$e_1 = (u_1, v_1), e_2 = (u_2, v_2)$，则一定存在$x \in \{u_2, v_2\}$，使得$x \notin \{u_1, v_1\}$，则令$V_1 = \{x\}, V_2 = V/V_1$，则$e_2$是$V_1, V_2$的MCE；

3）e_3不一定在最小生成树中

- Kruscal: 选中e_1, e_2之后，e_3的两个顶点可能在一个并查集中，即e_1, e_2, e_3可能构成环；
- MCE：同理，如果e_1, e_2, e_3构成环，则e_3为CE的切中，e_1或e_2也一定是切，因此e_3不再是最小切；

10.6: (Prim和Kruskal算法比较)

所谓稠密，即$m = O(n^2)$；所谓稀疏，即$m = O(n)$；

- Prim时间复杂度:
  - 基于数组实现优先队列：$O(n^2 + m)$；
  - 基于堆实现优先队列：$O((m+n)\log n)$；

- Kruskal时间复杂度:
  - 基于普通并+普通查：$O(mn)$；
  - 基于加权并+普通/路径压缩查：$O(m \log m)$；

- 综上：Prim算法更适合稠密图，Kruskal算法更适合稀疏图；

10.7: (最小生成树的变体)

灵活运用最小生成树算法

1）将原图G中所有边权取负，在新图G'上求得的最小生成树，即原图G上的最大生成树；

2）

- 根据最小生成树的性质，最小反馈边集与最大生成树对应的边集互为补集；
- 试着证明上述结论，再给出相应算法实现；

- 实际上，我们考虑的是无向图的最小反馈边集，感兴趣可以思考一下有向图，会有本质差异；

10.9: (最小生成树的更新)

思考原最小生成树带给我们的信息

- 设增加的新的节点为v，增加的新边集为E'，得到的新图为G'，对应的新MST为T'；

- 直接选择E'中权值最小的边是不一定得到正确的T'的；（想想为什么）

- 但如果直接在G'上运行Kruskal算法，就需要对新边集$E+E'$进行排序，
- 由$|E'|<|E|$，因此时间复杂度为$O(E\log E)$；

- 但是可以发现，在G上运行Kruskal算法得到T的过程中所弃用的边，在G'上运行时也一定会被弃用；
- 想想为什么，并证明之；
- 于是，我们只需要在$\underline{\{T,v,E'\}}$构成的图G''上运行Kruskal算法即可；
- 又$|E'|\le|V|$，因此时间复杂度为$O(|V|\log|V|)$；

10.10: (MST权值更新)

所谓图上的线性时间算法，即时间复杂度不超过$O(m+n)$的算法；

1）$e \notin E$'且$\hat{w}(e) > w(e)$：无需更新；

2）$e \notin E$'且$\hat{w}(e) < w(e)$：在原MST上添加e，形成一个环C，遍历C并删去其中的最大权边；

3）$e \in E$'且$\hat{w}(e) < w(e)$：无需更新；

4）$e \in E$'且$\hat{w}(e) > w(e)$：在原MST上删除e形成两个连通分支，添加连接两个连通分支的权值最小的边；

10.13: (包含特定边集的MST)

在Kruskal算法的基础上进行改进即可

- 初始化时:
  - 将S中的所有边加入MST集合中，并将其中的点加入并查集;
- 迭代时:
  - 在$E-S$上排序并运行Kruskal的更新步骤;

10.14: (确定某边是否在MST中)

注意题干条件：各边权值各不相同

- 记$e=\{u,v\}$；
- 删去权值大于等于$w(e)$的所有边（故e也被删除）；
- 在剩余子图中从u出发搜索v，
  - 如果能搜到，则e一定不在任何一个MST中；（想想为什么）
  - 否则，e一定在某个MST中；（想想为什么）

10.15: (MST相关命题判断) - 1

注意题干条件：未做特殊说明，各边权值可能相同

1）若G有超过n-1条边，且有唯一一条最重边，则这条边必不属于G的任意MST；

- 错，反例：割边；

2）若G中存在一个环，且其上包含了G的唯一最重边e，则e不属于任何最小生成树；

- 对，可以反证；

3）设e是G中的一条权重最小的边，则e必属于某个最小生成树；

- 对，同10.3；

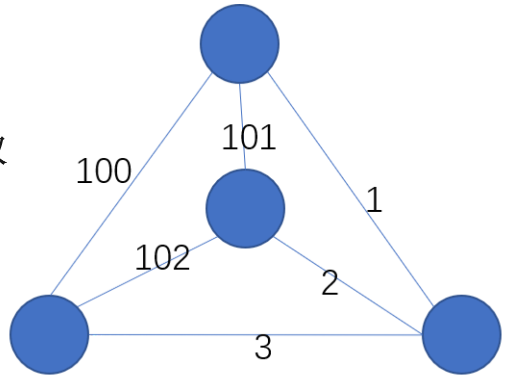
4）如果图中权重最小的边唯一，则该边属于每个最小生成树；

- 对，同10.3；

10.15: (MST相关命题判断) - 2

5）若G中存在一个环，且该环中的最轻边e唯一，则e必属于每个最小生成树；

- 错，反例：如右图，正四面体，一个三角面的三条边权很大，其他的三条边权很小；



6）两个节点间的最短路径必定是某个最小生成树的一部分；

- 错，反例：三角形，边权值分别为2,2,3；

7）当存在负权重的边时，Prim算法仍然有效；

- 对，与Dijkstra不同，Prim只依赖于边权值大小，不依赖于边权值符号；

10.16: (平方权图的MST)

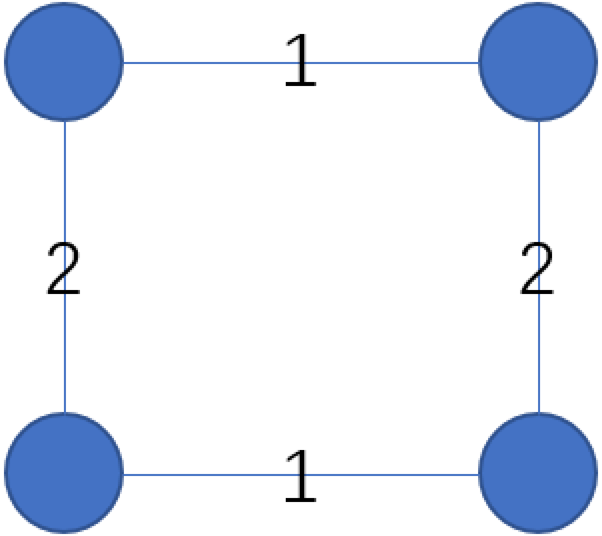
注意题干条件：各边权值为正值且各不相同；同时注意命题形式：当且仅当

- 既然各边权各不相同，则G和G'各自的MST唯一；
- 既然唯一，则用Kruskal算法运行得到的MST就是唯一的MST；
- 由于$f(x) = x^2, x > 0$是单调递增函数，因此G和G'的边权序完全相同，因此Kruskal运行结果完全相同；
- 因此T是G的MST当且仅当T是G'的MST；

10.21: (MST分治算法分析)

关键在于是否具有最优子问题性质

- 不正确，反例如下：



- 当割边均为权值为1的边时，最终得到的树的权重和为5，而MST的权重和显然为4

10.23: (挖井问题)

注意解决实际背景题目的关键，是对问题的建模

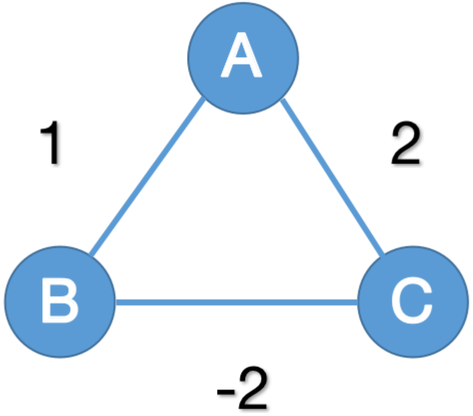
1）求得房子连接图的MST，再在挖井代价最小的房子上挖井；

2）增加一个超级节点，连接所有的房子，边权为房子的挖井代价，然后在新图上求得MST即可；（想想为什么）

10.25: (Dijkstra负权边出错问题)

贪心和动态规划都需要拥有最优子问题的性质才可以

- 反例如下：



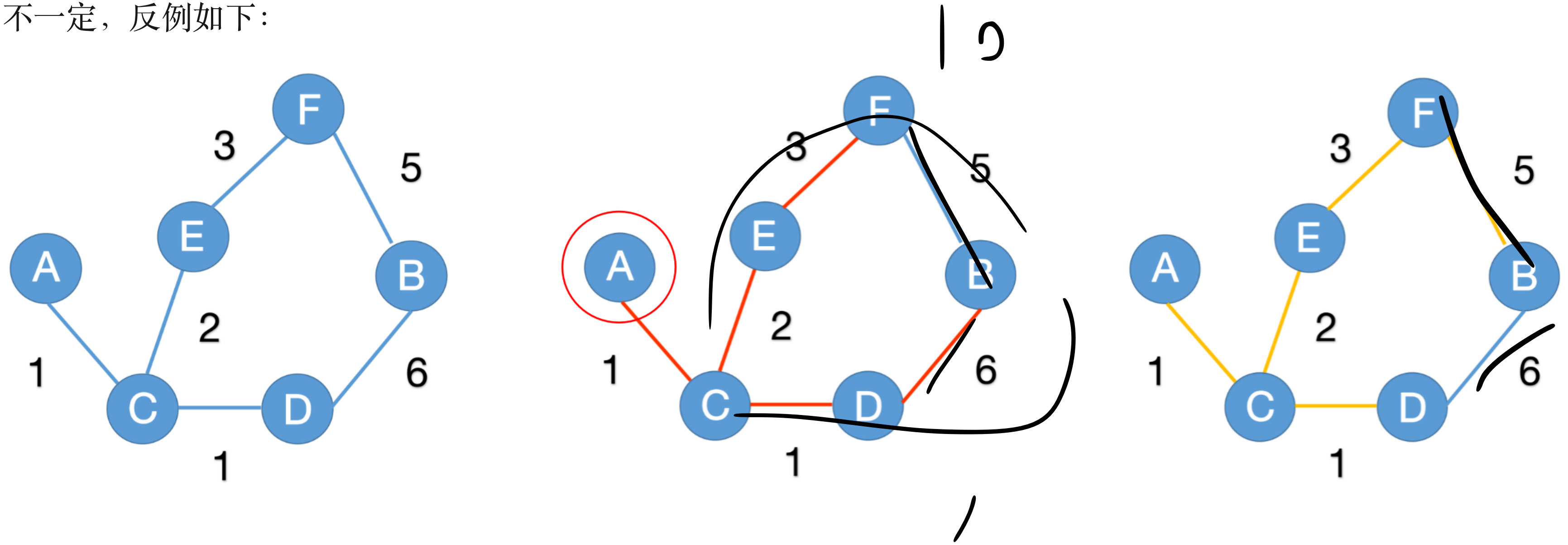
- Dijkstra无法应用负权边的根本原因，在于以下假设：

$$d(s,u^*) < d(s,v) + d(v,u^*), \quad u^* = \arg\min\{d(s,u) | u \in O\}$$

10.26: (最短路径树与最小生成树）

MST和最短路径的本质区别之一，在于前者是对整个图而言，而后者针对于源点和终点

不一定，反例如下：



G　　SPT　　MST

10.27: (特殊权图的SP问题)

在Dijkstra算法的基础上，对特殊边作特殊对待即可

- 设负权边为$e=\{u,v\}$，其权重为$w(e)$，从G中删去e，得到子图G'；

- 在G'上分别以s,u,v为源点，运行Dijkstra算法，得到$\mathrm{dist}_{G'}[s][t],\mathrm{dist}_{G'}[u][t],\mathrm{dist}_{G'}[v][t],\forall t$；

- 于是可以得到在G上的$\mathrm{dist}_G[s][t]=f(\mathrm{dist}_{G'}[s][t],\mathrm{dist}_{G'}[u][t],\mathrm{dist}_{G'}[v][t],w)$；
- 其中f的具体形式为：

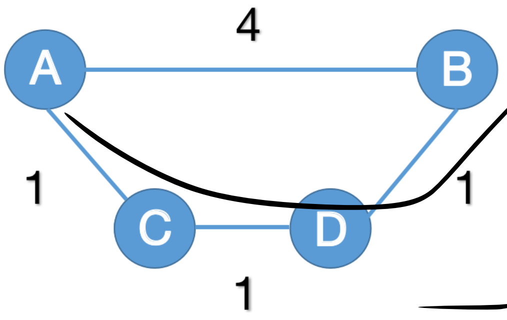
$$f=\min\{\mathrm{dist}_{G'}[s][t],\mathrm{dist}_{G'}[s][u]+w+\mathrm{dist}_{G'}[v][t],\mathrm{dist}_{G'}[s][v]+w+\mathrm{dist}_{G'}[u][t]\}$$

10.31: (边权加1对MST和SP的影响)

同10.26，MST和最短路径的本质区别之一，在于前者是对整个图而言，而后者针对于源点和终点

- MST不会发生变化，因为各个边的偏序关系没有发生变化；（作业中需要更严格的证明）

- 最短路径可能会发生变化，例子如下：



- 思考，若去掉边权非负的条件，上述结论会有变化吗

10.33: (推广的最短路径问题)

对Dijkstra算法进行相应修改即可

- 初始化时：$Cost[s] = c_s$；
- 更新时：$Cost[v] = \min\{Cost[v], Cost[u] + f(c_u, c_v, l_e)\}, e = (u, v)$；
- 其中f的具体形式为：$f = l_e + c_v$；

10.34: (负源点的Dijkstra)

同10.25，抓住Dijkstra无法应用负权边的根本原因

- 对于负权边仅从源点s出发的图，Dijkstra仍然能成立；
- 证明过程可以从根本原因入手进行说明，最好仿照教材上Dijkstra的证明过程进行修改；

10.36: (边数最少的SP问题)

在原Dijkstra算法运行的基础上，增加对边数的维护操作即可

- 初始化时：$\text{best}[s] = 0$；

- 更新时：分为两种情况：
  - 若$\text{dist}_s[v] > \text{dist}_s[u] + l_e, e = (u, v)$，则更新$\text{best}[v] = f(\text{best}[u], \text{best}[v])$；
  - 若$\text{dist}_s[v] = \text{dist}_s[u] + l_e, e = (u, v)$，则更新$\text{best}[v] = g(\text{best}[u], \text{best}[v])$；
  - 其中f, g的具体形式分别为：$f = \text{best}[u] + 1, g = \min\{\text{best}[v], \text{best}[u] + 1\}$；

- 除了边跑Dijkstra边维护best之外，也可运行完Dijkstra之后，再根据$\text{dist}_s[v], path_s[v]$生成的最短路径树上统计best；

10.38: (高速公路加油问题)

首先对实际背景问题进行建模，然后思考模板问题，再在模板问题对应的标准算法上进行修改

1）

- 可行解即途径公路的长度均不超过L，
- 所以只需要在所有$l_e \leq L$的子图$G' =< V, E' >$上进行搜索即可；

2）

- 从s到t的最小油箱容量等于从s到t的合法路径中，最大公路长度最小的那一条，所对应的最大公路长度；
- 只需要修改Dijkstra的更新步骤：$\underline{cap_s[v] = \min\{cap_s[v], f(cap_s[u], l_e)\}, e = (u, v)}$；
- 其中f的具体形式为：$f = \max\{cap_s[u], l_e\}$；

11.1: (硬币兑换问题)

贪心算法一般都是易想难证的方法，不过证明思路最常用的就两种：交换论证法和数学归纳法

- 直觉，优先尽量换面额最大的，换不干净再选面额第二大的，以此类推；
- 最终答案就是硬币金额S的c进制位串中，所有位串数字的和；
- 本贪心算法的证明可以采用交换论证法；

11.3: (公路基站问题)

贪心思路易想，证明技巧同样利用数学归纳法和交换论证

- 算法思路：
  - 从左往右选择基站，第一个基站b_1选择建在x_1右边t距离的位置；
  - 假设b_1能覆盖的房子范围为$\{x_1, x_2, .., x_{k_1}\}$，则第二个基站$b_2$选择建在$x_{k_1+1}$右边$t$距离的位置；
  - 重复上述过程，直到所有房子都能被至少一个基站覆盖；

- 算法正确性证明：
  - 同Prim理，对算法的步骤数k进行归纳
  - 假设对于任意正整数k，存在最优解集包含上述算法前k步选择的基站位置；

12.1: (构建路由表)

Floyd-Warshall算法中嵌入对路由表的维护操作，同时注意后继路由表和前驱路由表的区别

1）

- 初始化时：$GO[i][j] = j$；
- 更新时：$GO[i][j] = GO[i][k]$；

2）

- 初始化时：$FROM[i][j] = i$；
- 更新时：$FROM[i][j] = FROM[k][j]$；

12.2: (输油管道吞吐率)

类似于10.38高速公路加油问题

1）从源点s到终点t的最大吞吐率是s到t所有路径上，最小管道吞吐率最大的那条，所对应的最小管道吞吐率；

- 只需要修改Dijkstra的更新步骤：$cap(s,v)=\max\{cap(s,v),f(cap(s,u),c(u,v))\},e=(u,v)$；
- 其中f的具体形式为：$f=\min\{cap(s,u),c(u,v)\}$；

2）同理，只需要修改Floyd-Warshall算法的更新步骤：

- $D^{(k)}[i][j]=\max\{D^{(k-1)}[i][j],f(D^{(k-1)}[i][k],D^{(k-1)}[k][j])\}$；
- 其中f的具体形式为：$f=\min\{D[i][k],D[k][j]\}$；

12.4: (点集间的SP问题)

添加虚拟/超级节点是常用的简化技巧

- 添加超级节点s，其仅与S中的所有节点相邻，且权值均为0；
- 添加超级节点t，其仅与T中的所有节点相邻，且权值均为0；
- 以s为源点，在增广的图G'上运行Dijkstra，答案就是$\text{dist}_s[t]$；

12.7: (经过特定点的SP问题)

结合Floyd-Warshall的中继节点思想和Dijkstra算法

- 以v_0为源点，在原图G上运行Dijkstra算法，得到$\text{dist}_{v_0}[t]$；
- 以v_0为源点，在转置图G^T上运行Dijkstra算法，得到$\text{dist}_t[v_0]$；
- 因此，任意节点对(u, w)之间的最短路径为：$\text{dist}[u][w] = f(\text{dist}_{v_0}[u], \text{dist}_{v_0}[w], \text{dist}_u[v_0], \text{dist}_w[v_0])$；
- 其中f的具体形式为：$f = \text{dist}_u[v_0] + \text{dist}_{v_0}[w]$；

Thank You