

数据结构与算法

DATA STRUCTURE

第二十七讲 最小生成树

胡浩栋

信息管理与工程学院

2017 - 2018 第一学期

课堂内容

- 最小生成树

最小生成树

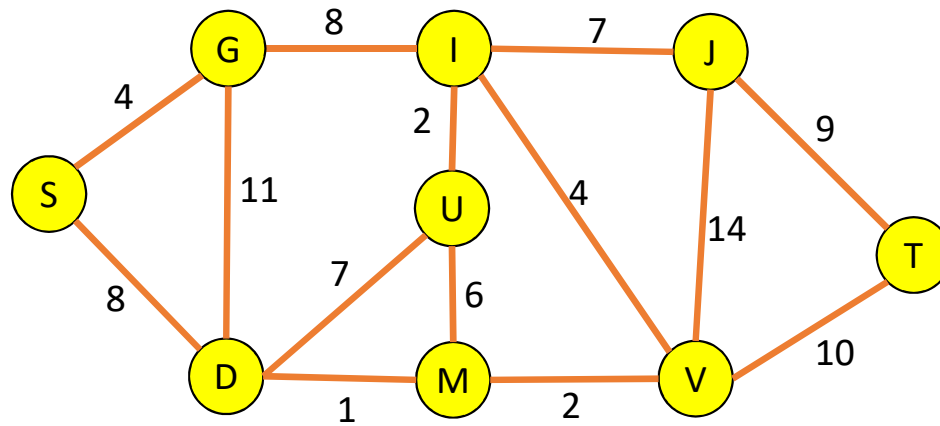
Minimum Spanning Tree

生成树

- 树是一个无向，无环的连通图
- 一个连通无向图上的生成树，是这个图的子图
 - 子图是树结构
 - 包含全部顶点集

生成树

- 定义生成树的cost就是树的所有边的权重和
- 我们要找一个cost最小的生成树，也就是最小生成树MST



生成树

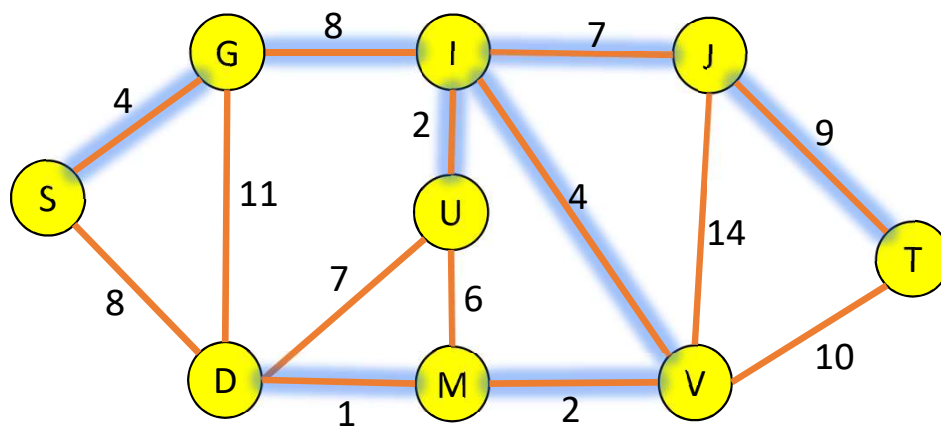
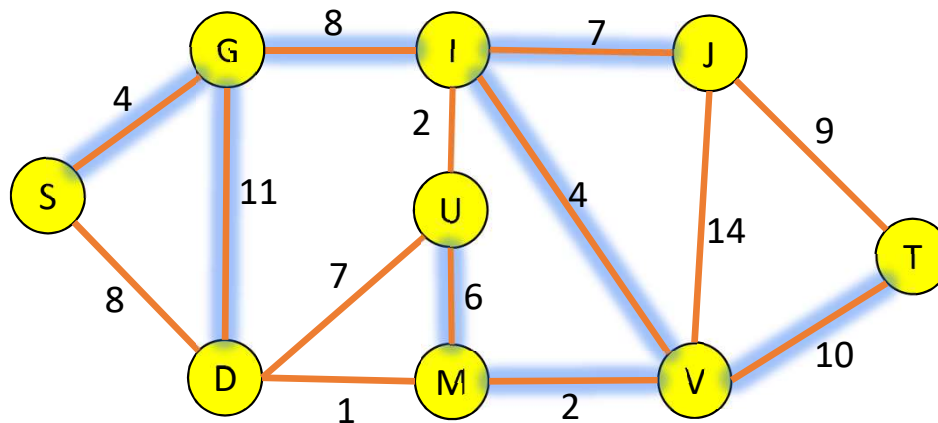
• Cost是

$$4+8+7+4+10+2+6+11=52$$

• Cost是

$$4+8+7+4+9+2+2+1=37$$

这个是MST

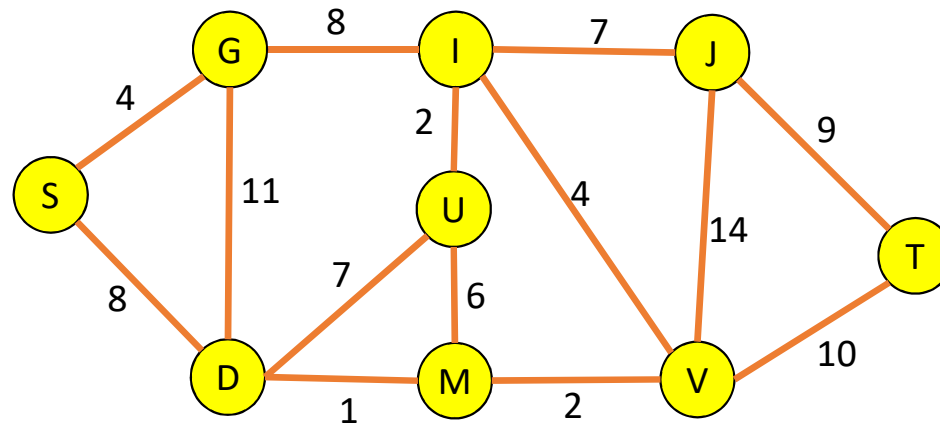


为什么要找MST

- 城市道路，公共设施设计
- 物流算法，TSP路径问题
- 别的图形算法

如何在算法里找MST

- Prim算法和Kruskal算法
- 贪婪算法
 - 如何每次选择一条最“好”的边
 - 使得之前的选择都包含在某个MST里面



Prim算法

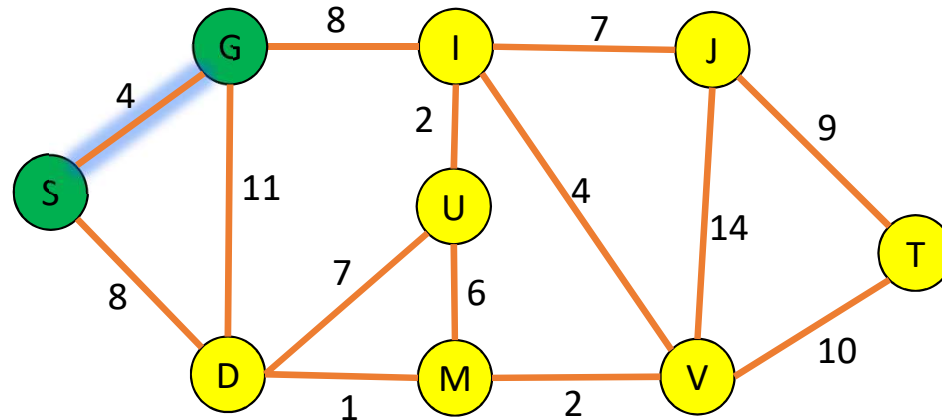
一个贪婪算法

Prim算法思路

- 第一步选一条最短的边
- 从这条边出发，维护一棵树，
 - 逐个添加边使得它成长为生长树
 - 添加的边必须是一个顶点在树里，一个顶点不在
 - 把符合条件的边里选一条最短的

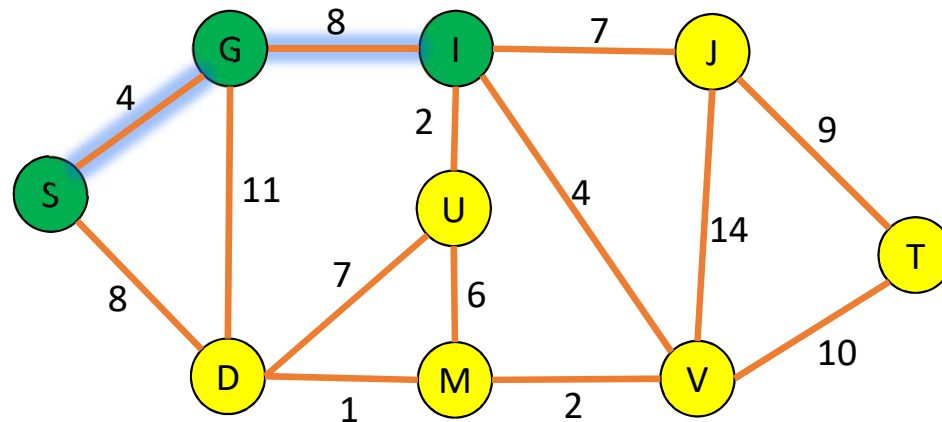
Prim算法例子

- 第一步选一条最短的边
- 从这条边出发，维护一棵树，
 - 逐个添加边使得它成长为生长树
 - 添加的边必须是一个顶点在树里，一个顶点不在
 - 把符合条件的边里选一条最短的



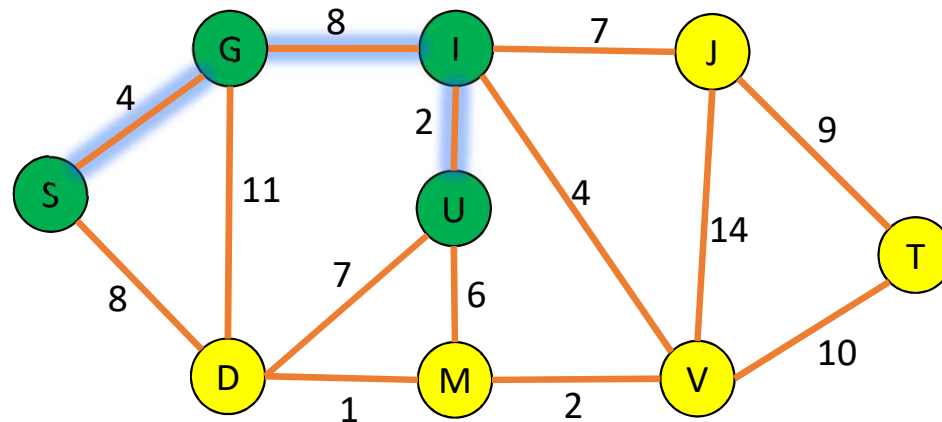
Prim算法例子

- 第一步选一条最短的边
- 从这条边出发，维护一棵树，
 - 逐个添加边使得它成长为生长树
 - 添加的边必须是一个顶点在树里，一个顶点不在
 - 把符合条件的边里选一条最短的



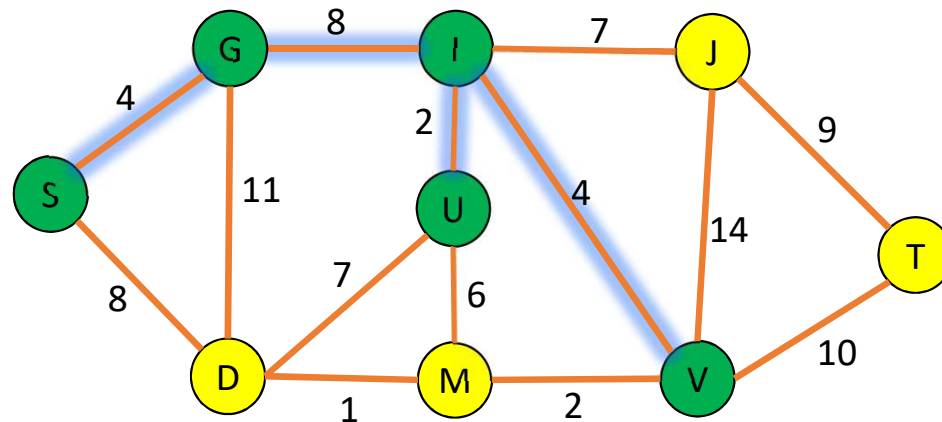
Prim算法例子

- 第一步选一条最短的边
- 从这条边出发，维护一棵树，
 - 逐个添加边使得它成长为生长树
 - 添加的边必须是一个顶点在树里，一个顶点不在
 - 把符合条件的边里选一条最短的



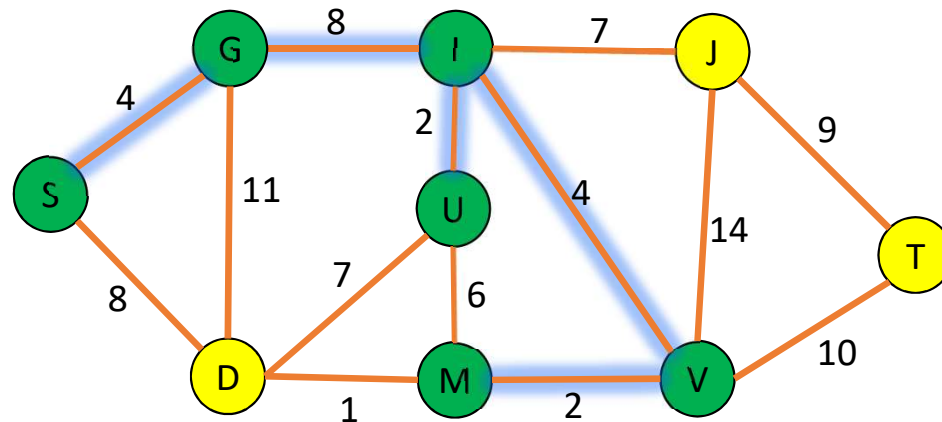
Prim算法例子

- 第一步选一条最短的边
- 从这条边出发，维护一棵树，
 - 逐个添加边使得它成长为生长树
 - 添加的边必须是一个顶点在树里，一个顶点不在
 - 把符合条件的边里选一条最短的



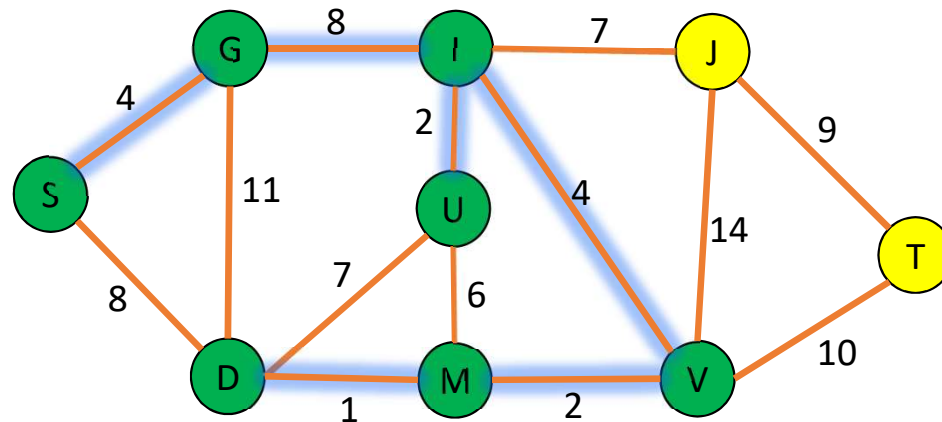
Prim算法例子

- 第一步选一条最短的边
- 从这条边出发，维护一棵树，
 - 逐个添加边使得它成长为生长树
 - 添加的边必须是一个顶点在树里，一个顶点不在
 - 把符合条件的边里选一条最短的



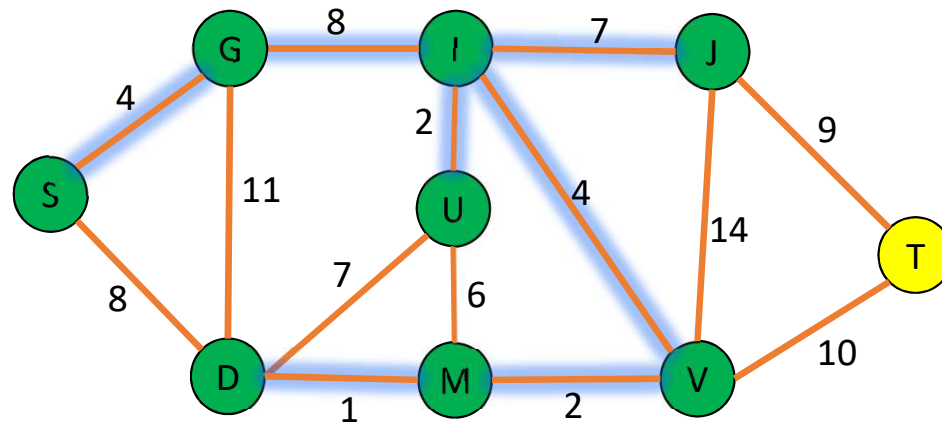
Prim算法例子

- 第一步选一条最短的边
- 从这条边出发，维护一棵树，
 - 逐个添加边使得它成长为生长树
 - 添加的边必须是一个顶点在树里，一个顶点不在
 - 把符合条件的边里选一条最短的



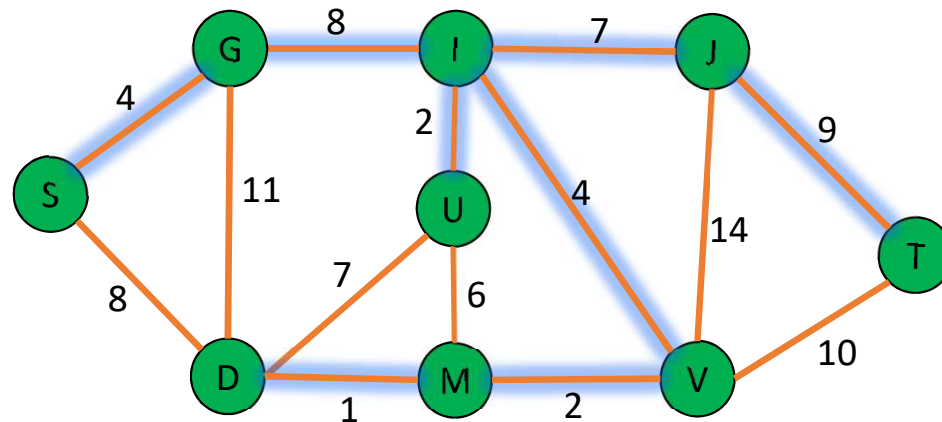
Prim算法例子

- 第一步选一条最短的边
- 从这条边出发，维护一棵树，
 - 逐个添加边使得它成长为生长树
 - 添加的边必须是一个顶点在树里，一个顶点不在
 - 把符合条件的边里选一条最短的



Prim算法例子

- 第一步选一条最短的边
- 从这条边出发，维护一棵树，
 - 逐个添加边使得它成长为生长树
 - 添加的边必须是一个顶点在树里，一个顶点不在
 - 把符合条件的边里选一条最短的



伪代码

- 复杂度是 $O(nm)$ ，如果不优化的话
 - 每次加一个顶点，需要遍历边来找到符合条件的边

$MST = \{(s, u)\}$, (s, u) 是最短的边

VisitedVertices = $\{s, u\}$

while $|\text{VisitedVertices}| < |V|$:

find the lightest edge (x, v) in E ,

x in VisitedVertices and v not

$MST \leftarrow (x, v)$

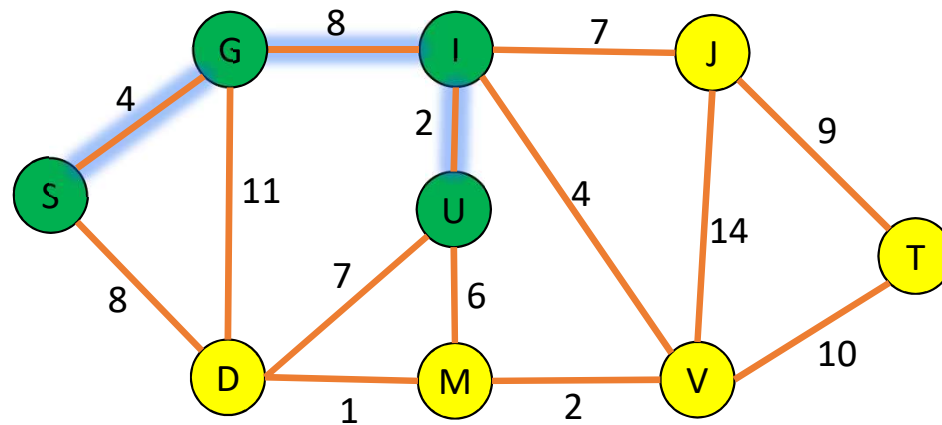
VisitedVertices $\leftarrow v$

两个问题

- 一个是怎么证明找到的生成树是MST？
 - 即存在一个MST，使得每个步骤前找到的边都在MST里
- 如何实现，使得找最“好”边的过程优化？

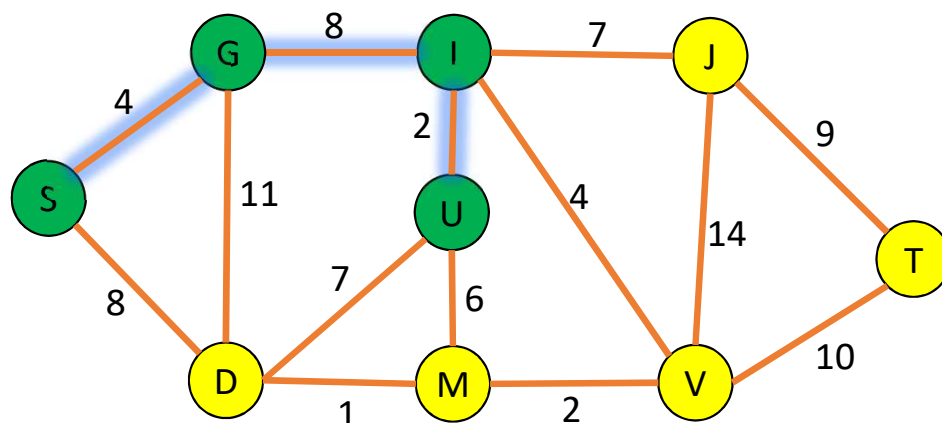
如何实现

- 假如我们已经有SG和GI, IU两条边了
- 如何选择下一条最“好”边
- 其实是选择一个黄点, 使得其到绿点集合的距离最短



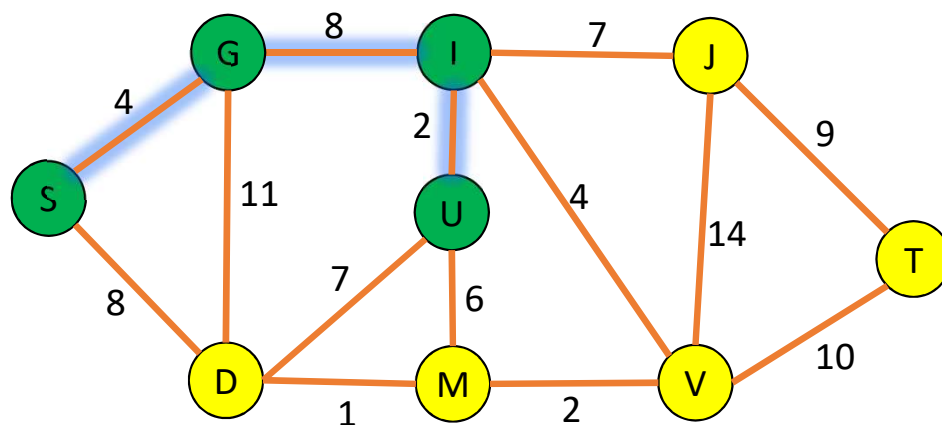
如何实现

- 其实是选择一个黄点，使得其到绿点集合的距离最短
- 回忆dijkstra算法，也是逐个添加顶点，需要找距离源点最近的黄点，通过更新预估值 $d(v) = \min\{d(v), d(u) + w(u, v)\}$
- 这里做法也类似，不过预估值改为 $d(v) = \min\{d(v), w(u, v)\}$



如何实现

- 对每个顶点添加一个预估值 $d(v)$ 和其父节点指针
- 不过预估值改为 $d(v) = \min\{d(u), w(u, v)\}$



伪代码

- 初始化
 - 未完成 ●
 - 完成 ●
- 最短边寻找过程

For each vertex u

$d(u) = \infty$; $u.status = \text{●}$

$d(s) = 0$, s 是最短边的一个顶点

For $i = 1, \dots, n$:

Find u with status ●, such that $d(u)$ is min

For each neighbor v with status ●:

$d(v) \leftarrow \min\{\textcolor{red}{d(v)}, \textcolor{blue}{w(u, v)}\}$

$p(v) \leftarrow \textcolor{blue}{u}$ if $d(v)$ is updated

$u.status = \text{●}$

算法复杂度

- 和Dijkstra算法一样
 - $O((n + m) \log(n))$ ，如果优先队列是红黑树，最小堆
 - $O(n \log(n) + m)$ ，如果优先队列是Fibonacci堆

为什么正确

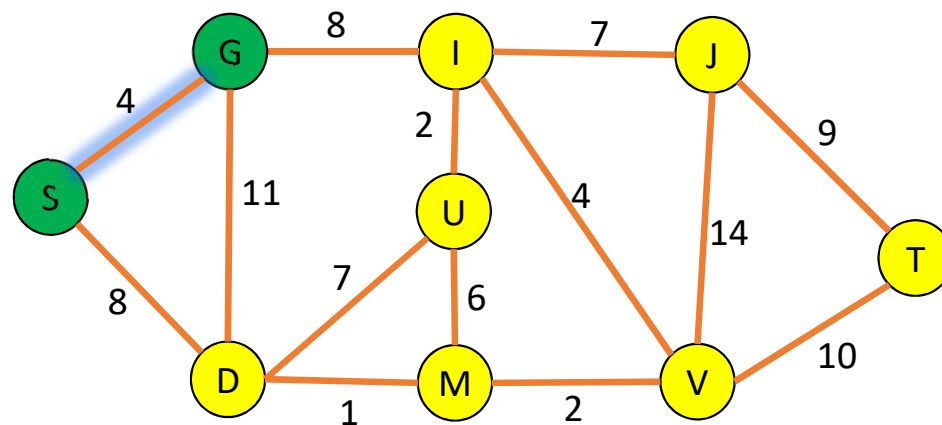
- 我们每一步根据当前状态选择一条“最好”边
- 要保证最后选出的生长树是最小（之一）
 - 需要证明子问题的最优解还原成原问题的解还是最优的
 - 或者证明每一步选择后，之前的子树都在某一个MST里

分解原问题

- 子问题：在 k 个顶点集上，起始点是 s ，找一个最小的生成树
- 怎么归纳？
 - 从起始点出发的边上找一条最短边，把这两个点塌陷成点 s ,
 - 就变成了 $k-1$ 个顶点集中，起始点是 s ，找一个最小生成树
- 需要证明，从子问题的MST，再添加塌陷的那条边，结果还是 k 个顶点集上的ST，而且是最小的

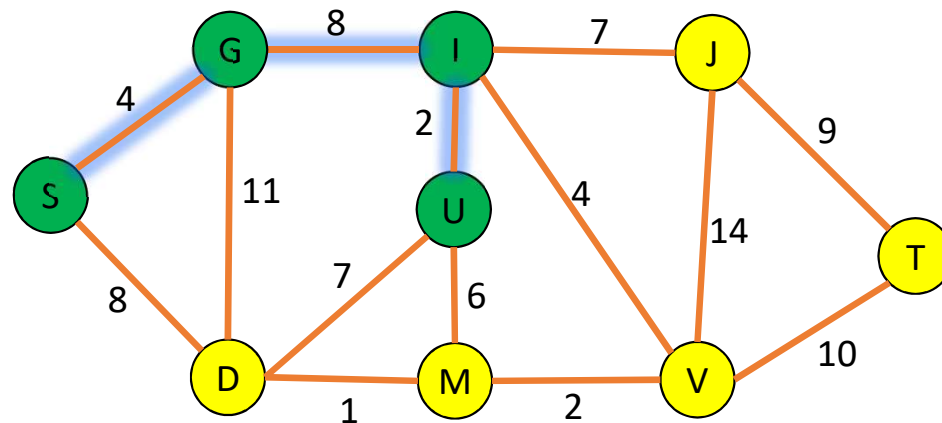
初始状态

- 选了一条最短的边SG，显然存在MST使得SG是它的一条边
- 把S、G塌陷成一个大的节点，原问题就变成了 $n-1$ 个点上的MST问题



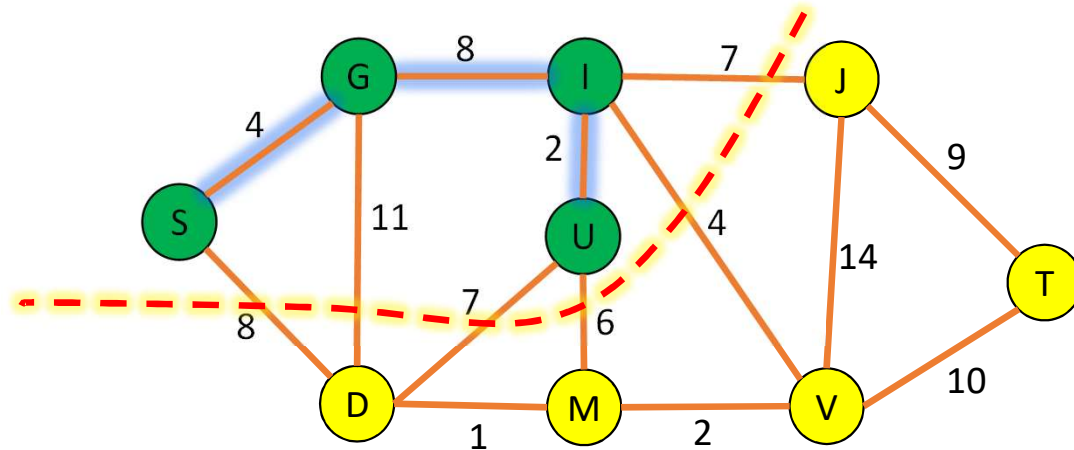
一般情形

- 假设中间状态，SG，GI，IU塌陷之后，变成更小的子问题
- 如果子问题的MST可解，考虑如何证明还原成原问题的解也是MST



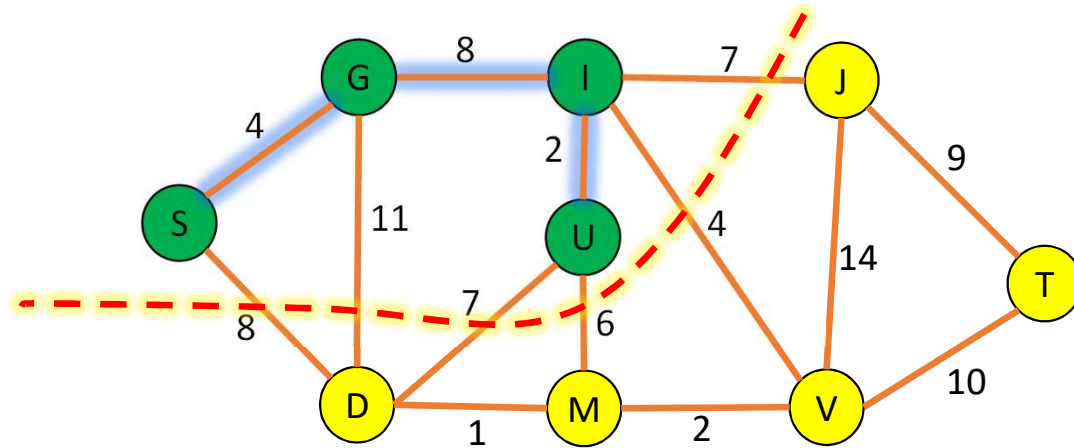
一般情形

- 实际，我们不用真的塌陷，而是用一条曲线切割图，一边是绿点，一边是黄点。
- 为什么可以这么做，因为SGIU是树结构，是可以塌陷的
- 如果绿点集合有环，比如GIJVM，把U包在里面，ST在外面，这样是不存在这样的cut，使得一边一种颜色，像bipartite



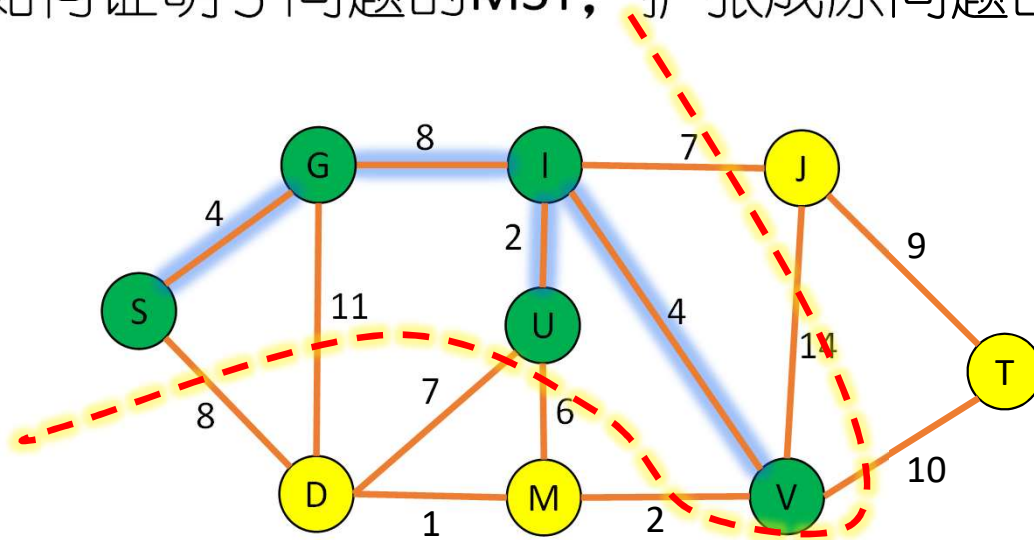
一般情形

- 归纳过程：再做一次选择，让问题变得更小
- 就是在虚线相交的边中选一条最短的边



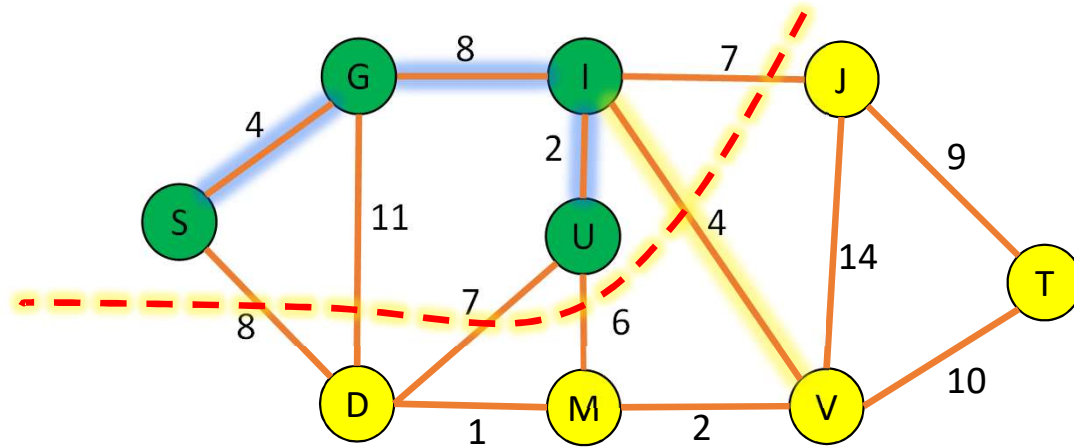
一般情形

- 归纳过程：再做一次选择，让问题变得更小
- 虚线相交的边中选一条最短的边，然后把邻接顶点V变成绿点
- 把新的绿点集合看作一个大节点后，子问题就更小了
- 问题是如何证明子问题的MST，扩张成原问题的ST后，还是MST



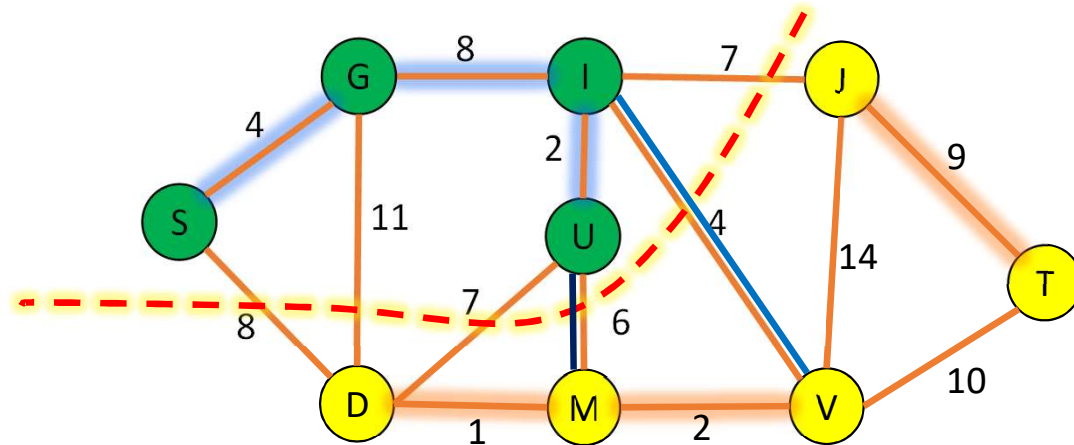
等价于

- 证明之前选的子树就是某一MST的一部分
- 数学归纳法：假设之前的绿点子树是某一MST的一部分
- 证明下一步选择后，还是某一MST的一部分
- 注意下一步算法选择IV



为什么正确

- 因为任意MST里至少有一条边连接绿点集合和黄点
- 反证法：如果这条边不是最短的IV
- 那么我们可以添加最短的IV，这样肯定有一个环，而且IV是环的一条边，那么一定有另外一条边也是一端是绿色，一端是黄色。
- 这条边可以被IV代替，整体cost不增加，所以也是MST

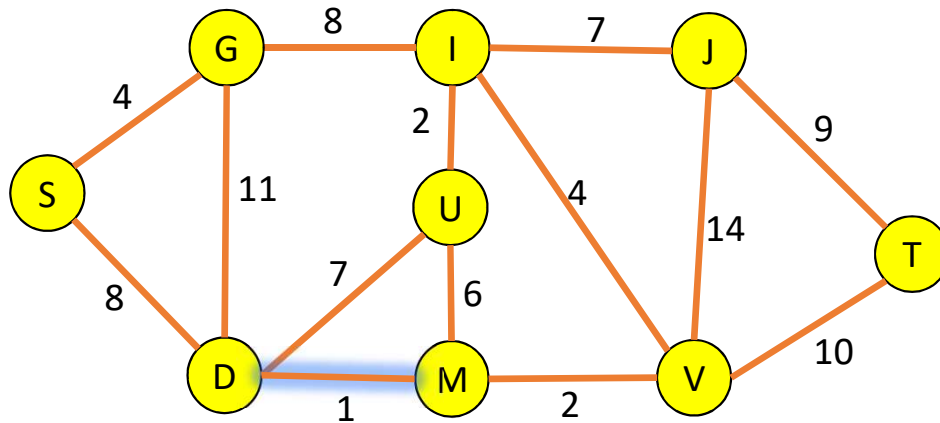


Kruskal算法

另一个贪婪算法

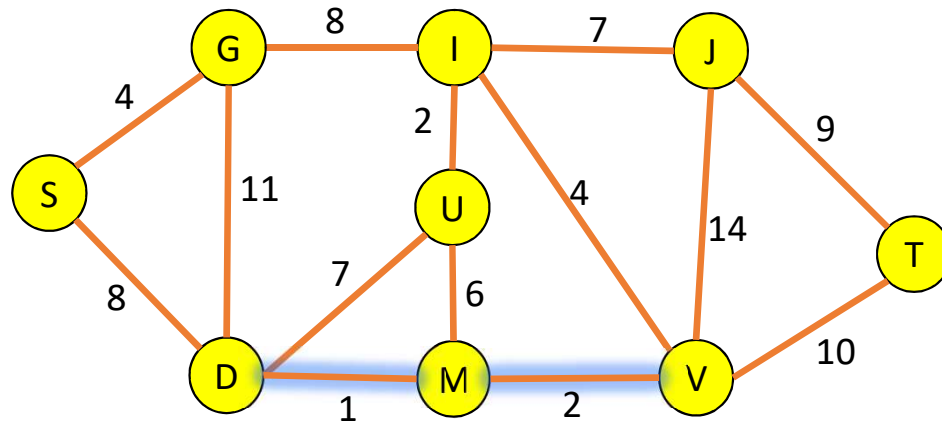
Kruskal

- 每一步都选一条最短的边
- 不管是不是和之前的连通
- 只要新的边不造成环路



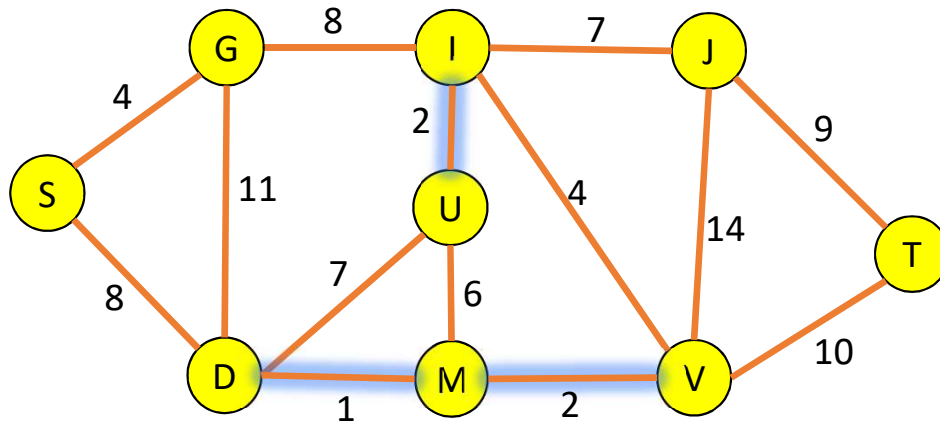
Kruskal

- 每一步都选一条最短的边
- 不管是不是和之前的连通
- 只要新的边不造成环路



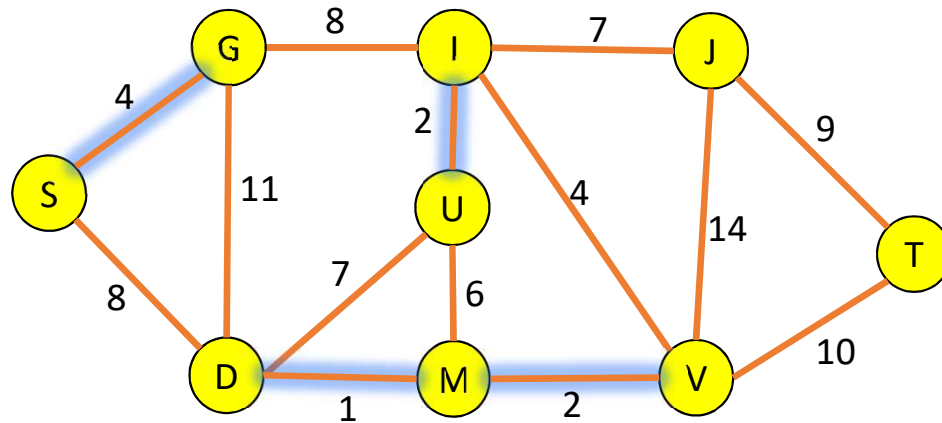
Kruskal

- 每一步都选一条最短的边
- 不管是不是和之前的连通
- 只要新的边不造成环路



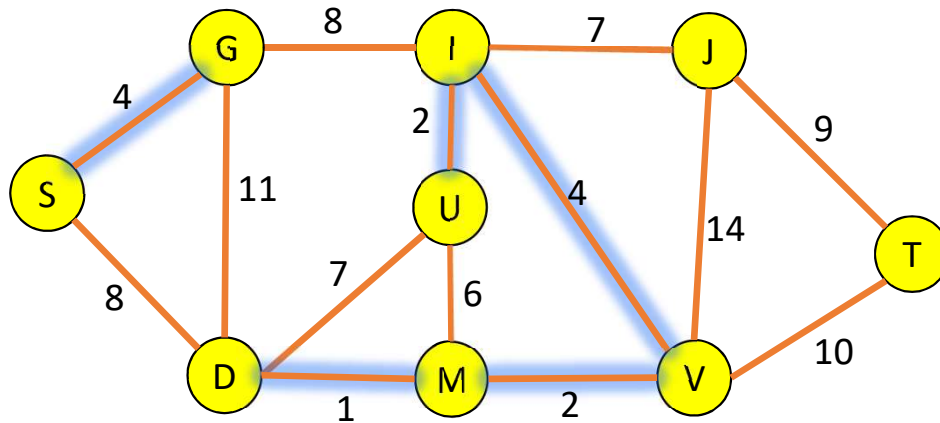
Kruskal

- 每一步都选一条最短的边
- 不管是不是和之前的连通
- 只要新的边不造成环路



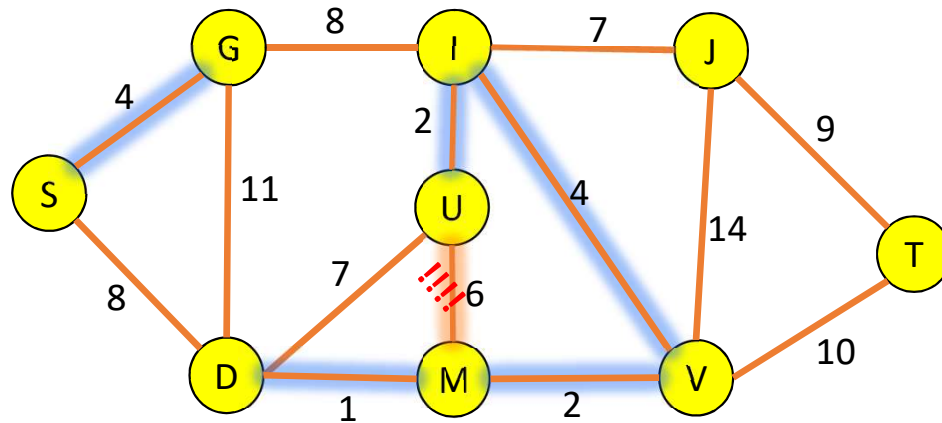
Kruskal

- 每一步都选一条最短的边
- 不管是不是和之前的连通
- 只要新的边不造成环路



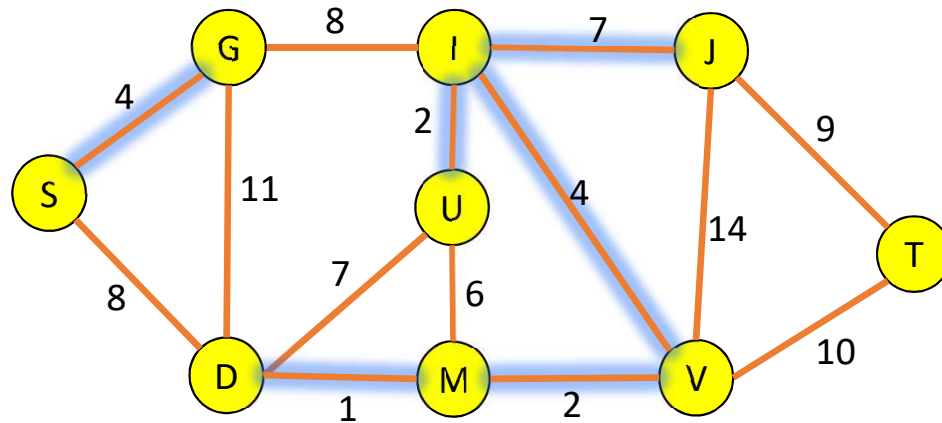
Kruskal

- 每一步都选一条最短的边
- 不管是不是和之前的连通
- 只要新的边不造成环路



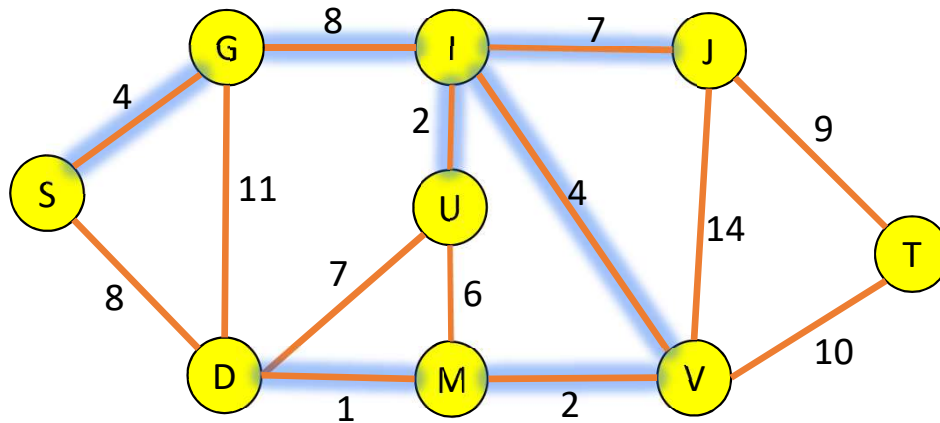
Kruskal

- 每一步都选一条最短的边
- 不管是不是和之前的连通
- 只要新的边不造成环路



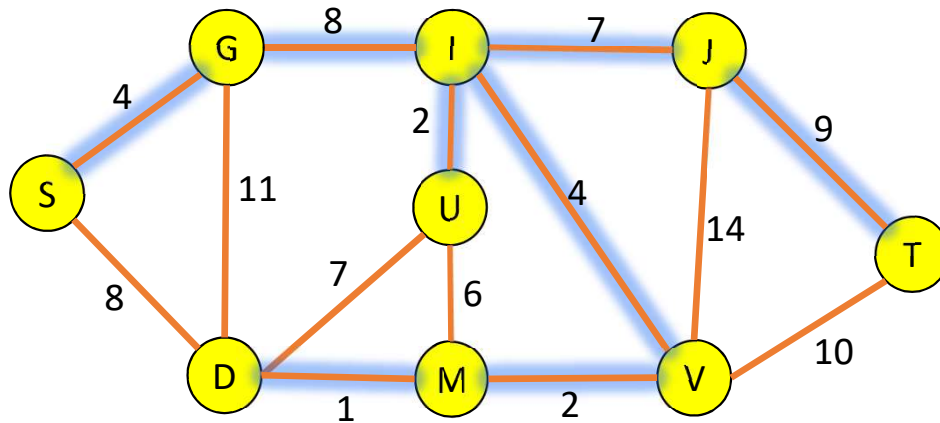
Kruskal

- 每一步都选一条最短的边
- 不管是不是和之前的连通
- 只要新的边不造成环路



Kruskal

- 每一步都选一条最短的边
- 不管是不是和之前的连通
- 只要新的边不造成环路



伪代码

- 初始化

Sort all edges by weight
 $MST = \{\}$

- 最短边寻找
过程

For each edge e in sorted order:
 If add e to MST won't **cause a cycle**
 add e to MST

两个问题

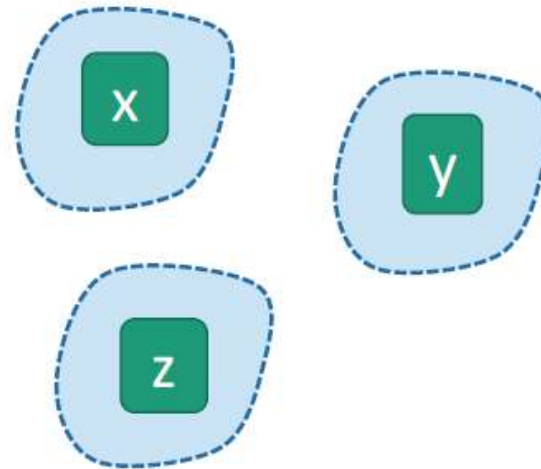
- 一个是怎么证明找到的生成树是MST？
 - 即存在一个MST，使得每个步骤中找到的边都在这个MST里
- 如何实现？
 - Dfs算法可以检查cycle
 - 使用Union-Find数据结构检查cycle更好（ $\log n$ 复杂度）

Union-find结构

- **makeSet(u)**: 建立集合{u}
- **find(u)**: 返回u所在集合的代表元素（根节点）
- **union(u,v)**: 合并u所在集合和v所在集合

```
makeSet(x)  
makeSet(y)  
makeSet(z)
```

```
union(x,y)
```

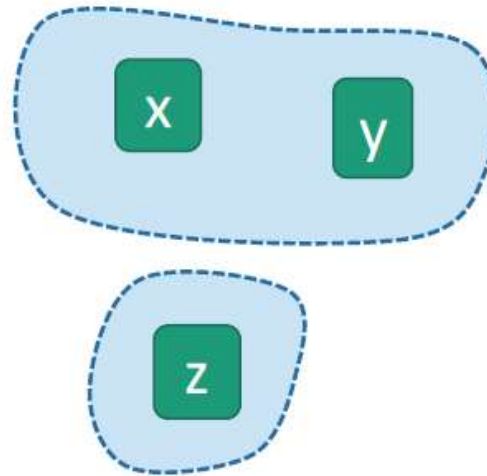


Union-find结构

- **makeSet(u)**: 建立集合{u}
- **find(u)**: 返回u所在集合的代表元素（根节点）
- **union(u,v)**: 合并u所在集合和v所在集合

```
makeSet(x)  
makeSet(y)  
makeSet(z)
```

```
union(x,y)
```



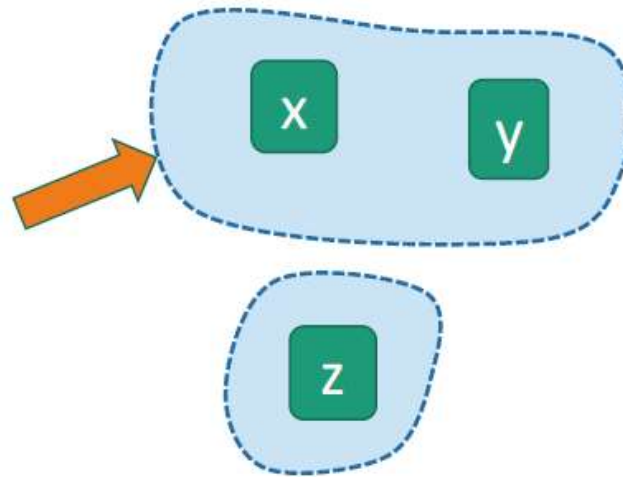
Union-find结构

- **makeSet(u)**: 建立集合{u}
- **find(u)**: 返回u所在集合的代表元素（根节点）
- **union(u,v)**: 合并u所在集合和v所在集合

```
makeSet(x)  
makeSet(y)  
makeSet(z)
```

```
union(x,y)
```

```
find(x)
```

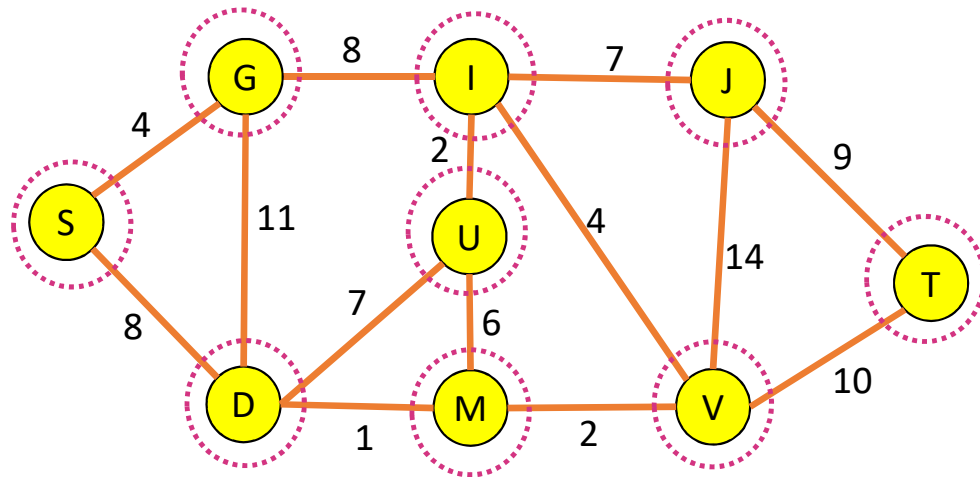


伪代码

- **kruskal**($G = (V, E)$):
 - **Sort** E by weight in non-decreasing order
 - $MST = \{\}$ // initialize an empty tree
 - **for** v in V :
 - **makeSet**(v) // put each vertex in its own tree in the forest
 - **for** (u, v) in E : // go through the edges in sorted order
 - **if** **find**(u) \neq **find**(v): // if u and v are not in the same tree
 - add (u, v) to MST
 - **union**(u, v) // merge u 's tree with v 's tree
 - **return** MST

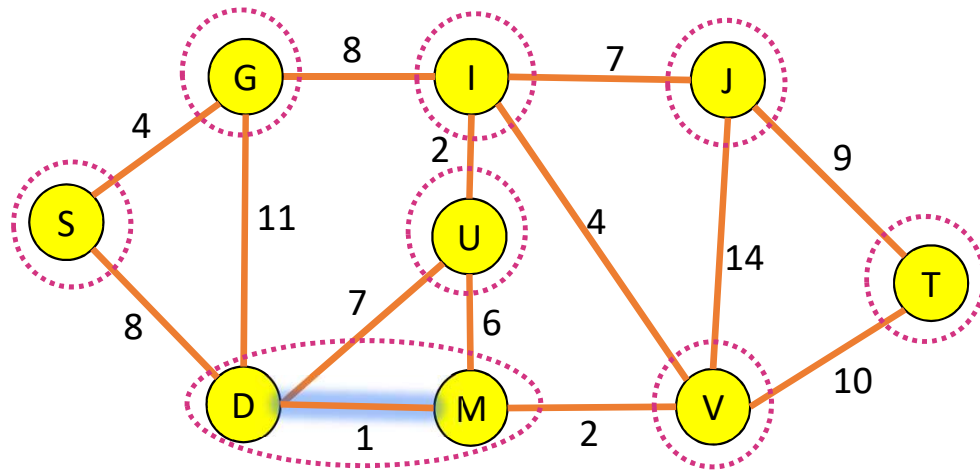
实现

- 起始状态



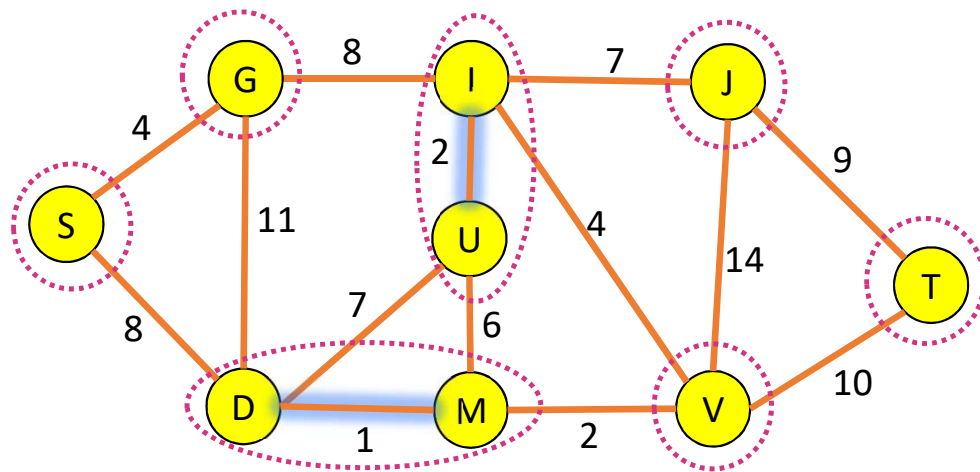
实现

- 下一步骤



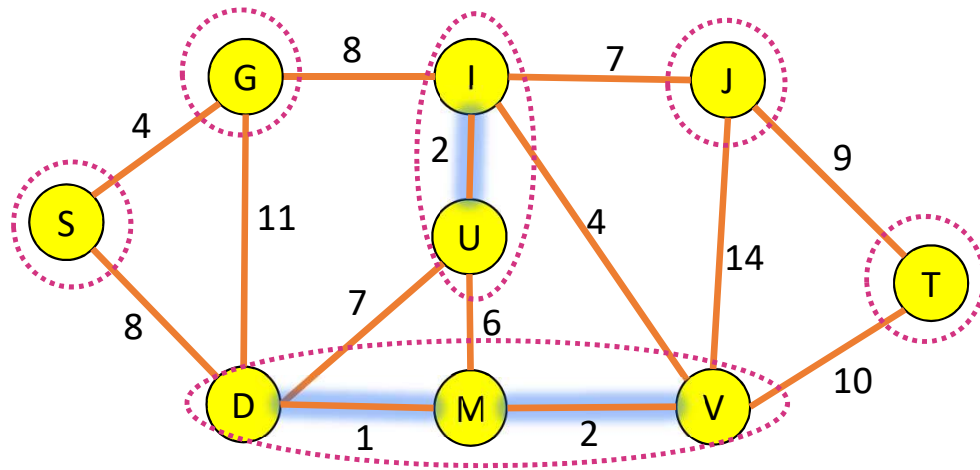
实现

- 下一步骤



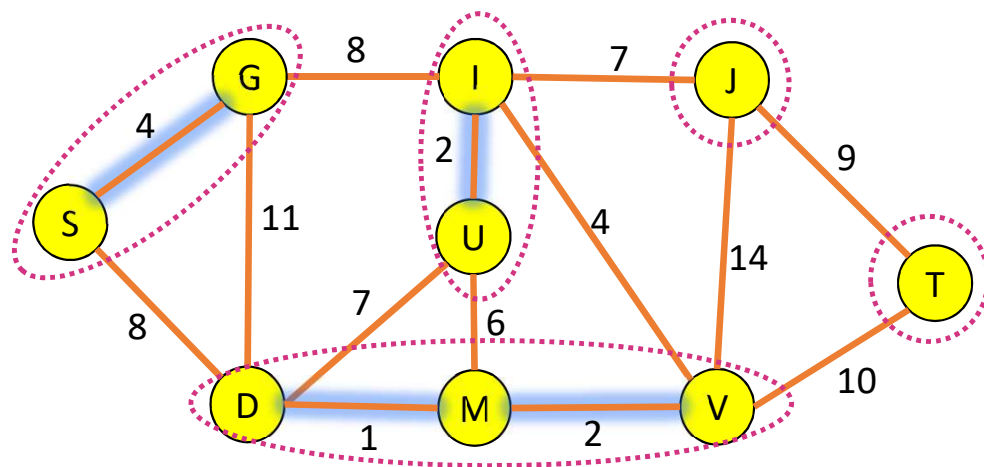
实现

- 下一步骤



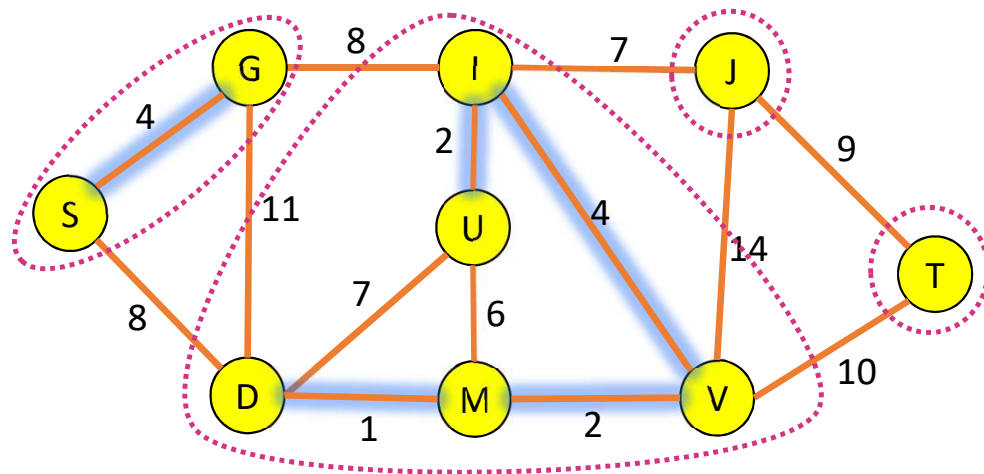
实现

- 下一步骤



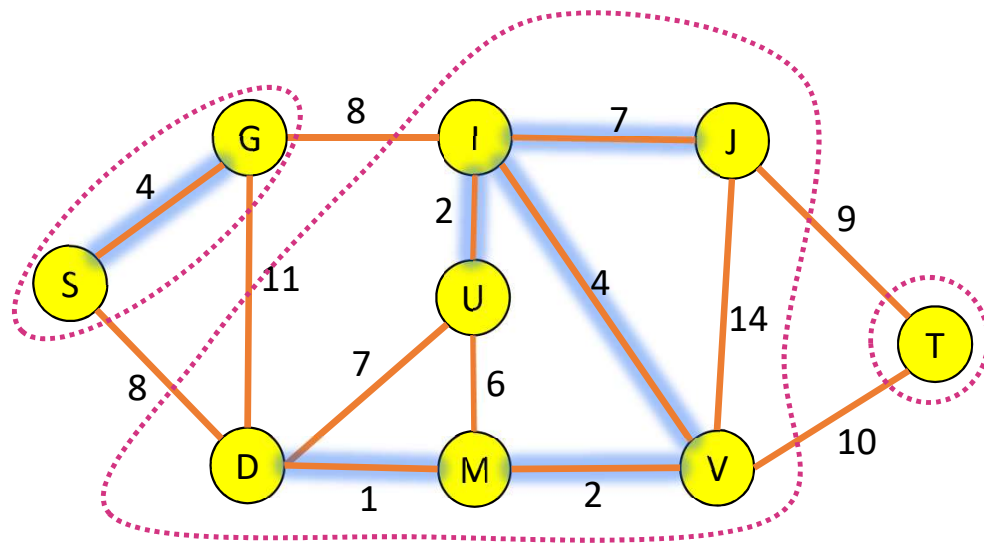
实现

- 下一步骤



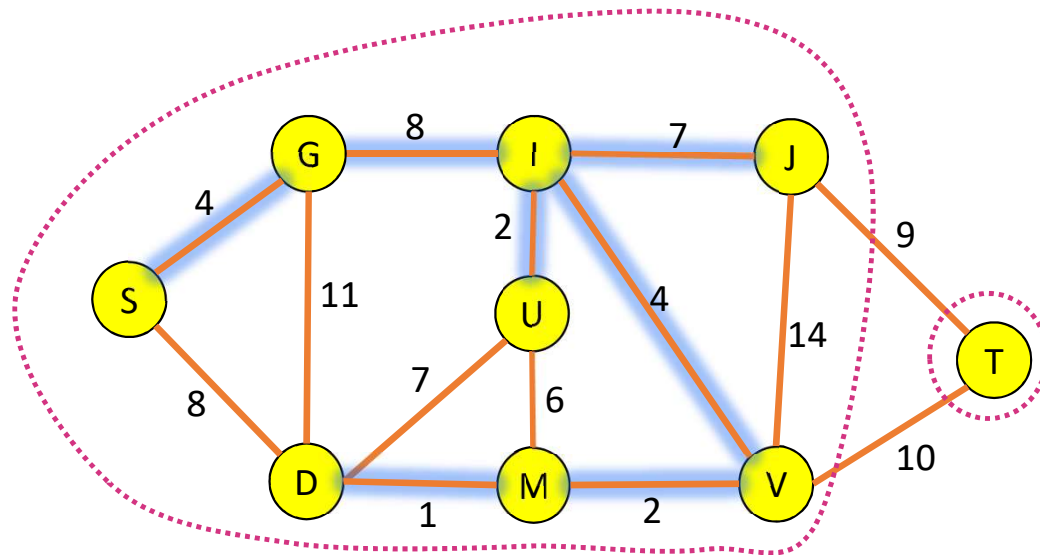
实现

- 下一步骤



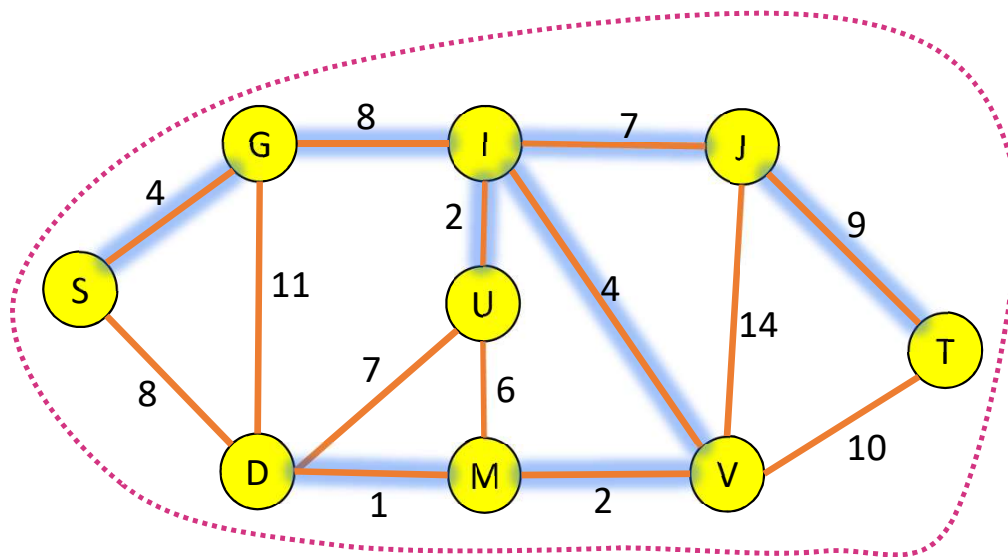
实现

- 下一步骤



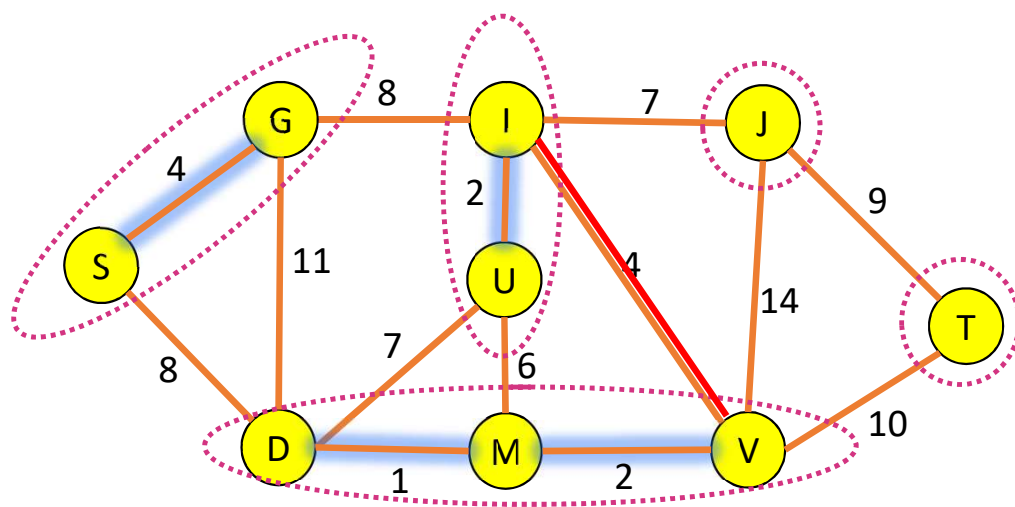
实现

- 下一步骤



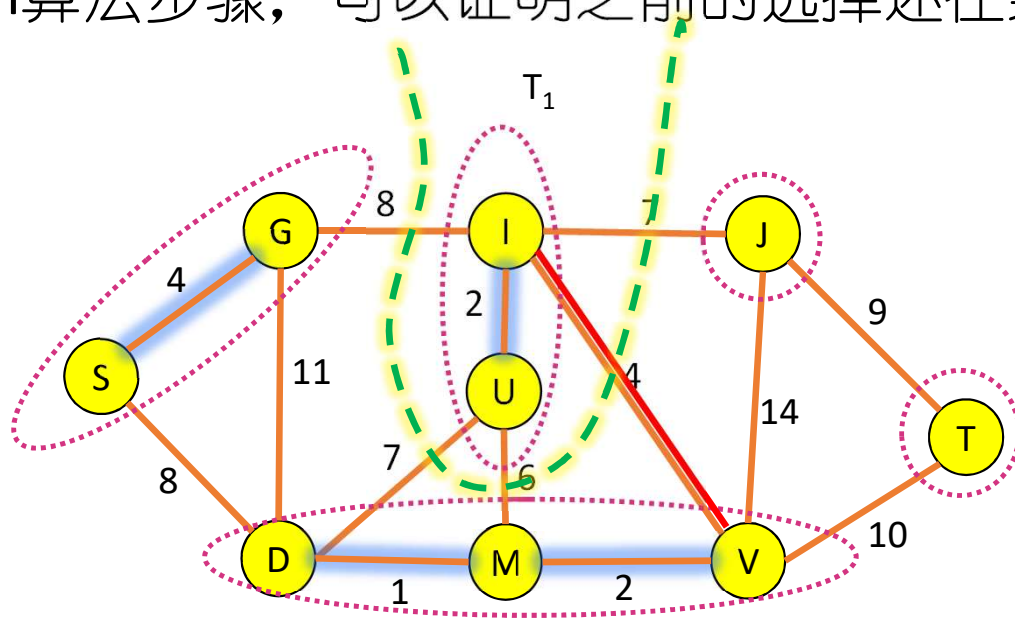
归纳

- 假设第k步时，之前选的边都在某个MST里
- 下一步，按算法，会选择IV
- 需要证明，选择IV后，还在某个MST里



归纳

- 需要证明，选择“IV”后，还在某个MST里
- 考虑 $cut\{T_1, V - T_1\}$
- 类似prim算法步骤，可以证明之前的选择还在某个MST里面



对比

- Prim's
 - 维护一颗树，慢慢成长到MST
 - 时间复杂度 $O(m + n \log(n))$ ，用Fibonacci堆实现
- Kruskal's
 - 维护森林，成长到MST
 - 时间复杂度 $O(m \log(n))$ ，用union-find结构实现

	Description	Runtime
Prim's	Grows a tree	$O(E \log(V))$ with red-black tree $O(E + V \log(V))$ with Fibonacci heap
Kruskal's	Grows a forest	$O(E \log(V))$ with union-find $O(E)$ with union-find and radix sort

总结

- MST的两个算法
 - Prim's
 - Kruskal's
- 都是用贪婪算法
 - 每次都做出一个选择
 - 选择后，只依赖于一个子问题
 - 每次选择都没有淘汰最优解
 - 即存在某个最优解使得之前的选择是其一部分

后记

- C++靠谱网站
 - www.cplusplus.com
 - www.learncpp.com
- 数据结构网站
 - <http://www.geeksforgeeks.org/>
 - https://www.tutorialspoint.com/data_structures_algorithms/
- 做题网站
 - <https://leetcode.com/>
- 最后祝大家考试顺利！

Q&A

Thanks!