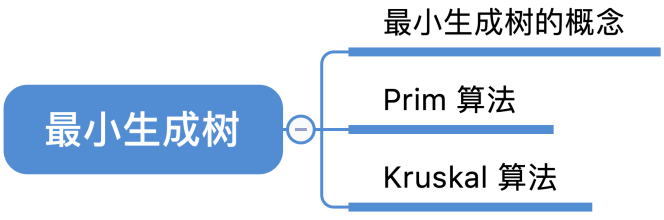
本节内容

最小生成树

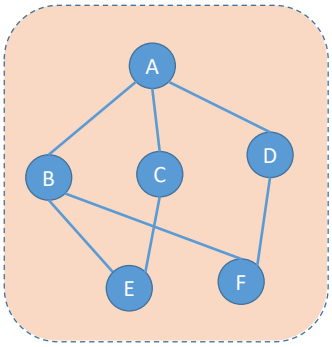
知识总览



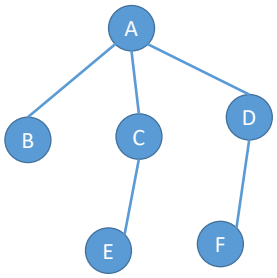
生成树

边尽可能的少，但要保持连通

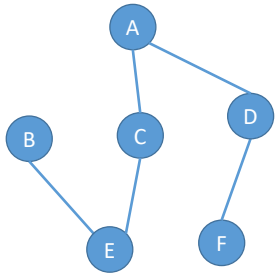
连通图的生成树是包含图中全部顶点的一个极小连通子图。
若图中顶点数为n，则它的生成树含有 n-1 条边。对生成树而言，若砍去它的一条边，则会变成非连通图，若加上一条边则会形成一个回路。



无向图G

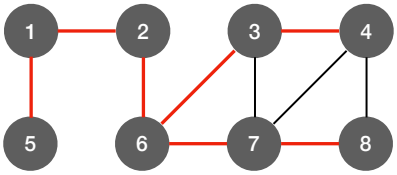


G的生成树1



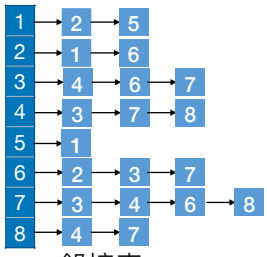
G的生成树2 .....

广度优先生成树

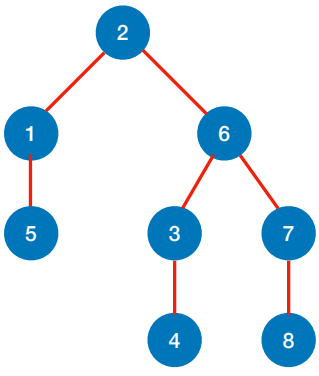


	1	2	3	4	5	6	7	8
1	0	1	0	0	1	0	0	0
2	1	0	0	0	0	1	0	0
3	0	0	0	1	0	1	1	0
4	0	0	1	0	0	0	1	1
5	1	0	0	0	0	0	0	0
6	0	1	1	0	0	0	1	0
7	0	0	1	1	0	1	0	1
8	0	0	0	1	0	0	1	0

邻接矩阵

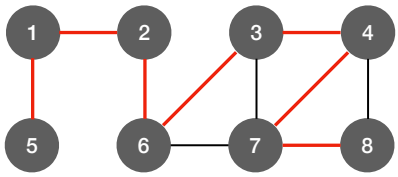


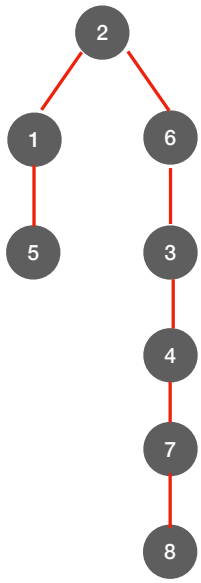
邻接表



广度优先生成树

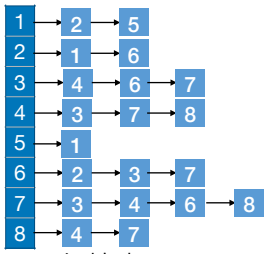
深度优先生成树





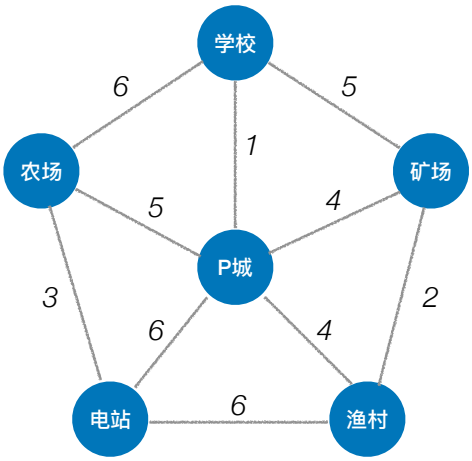
	1	2	3	4	5	6	7	8
1	0	1	0	0	1	0	0	0
2	1	0	0	0	0	1	0	0
3	0	0	0	1	0	1	1	0
4	0	0	1	0	0	0	1	1
5	1	0	0	0	0	0	0	0
6	0	1	1	0	0	0	1	0
7	0	0	1	1	0	1	0	1
8	0	0	0	1	0	0	1	0

邻接矩阵



邻接表

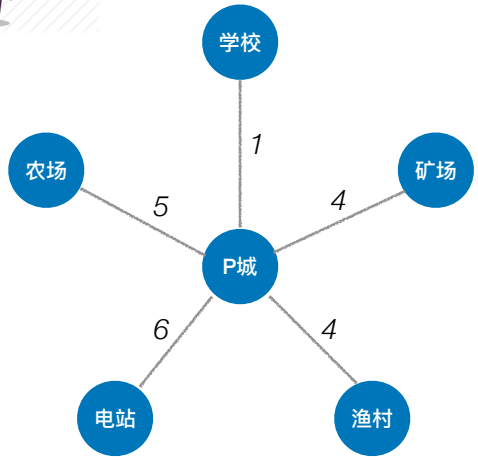
最小生成树（最小代价树）



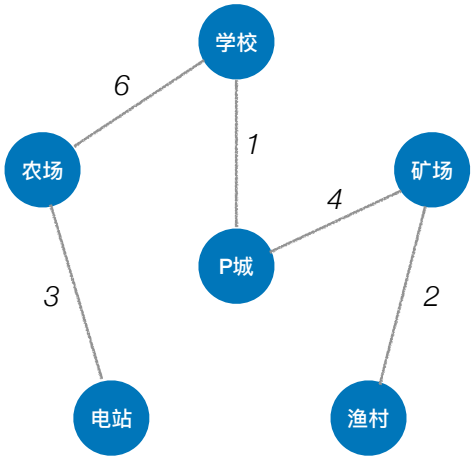
道路规划要求：所有地方都连通，且成本尽可能的低

还有没有更便宜的修路方案...?

最小生成树（最小代价树）



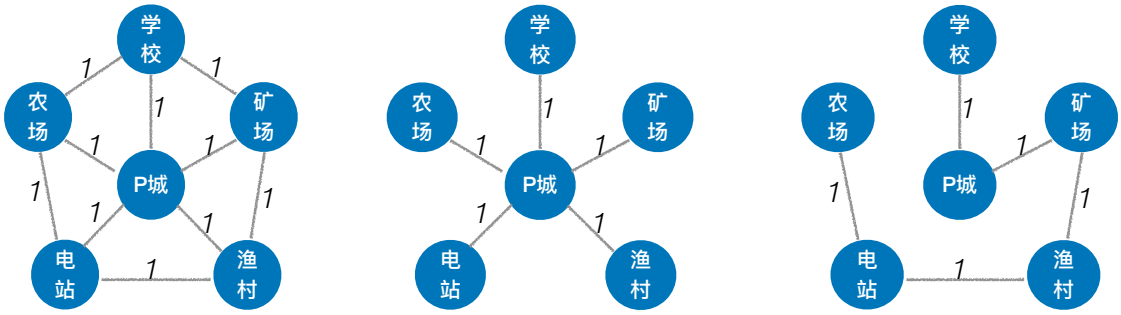
代价：19



代价：16

最小生成树（最小代价树）

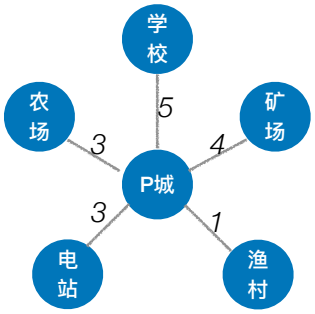
对于一个带权连通无向图$G = (V, E)$，生成树不同，每棵树的权（即树中所有边上的权值之和）也可能不同。设R为G的所有生成树的集合，若T为R中边的权值之和最小的生成树，则T称为G的最小生成树（*Minimum-Spanning-Tree, MST*）。



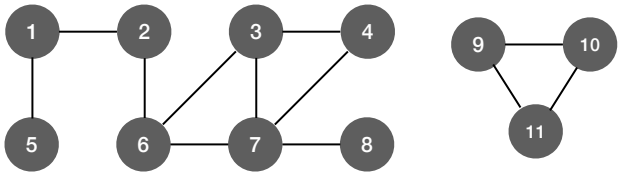
- 最小生成树可能有多个，但边的权值之和总是唯一且最小的
- 最小生成树的边数 = 顶点数 – 1。砍掉一条则不连通，增加一条边则会出现回路

最小生成树（最小代价树）

对于一个带权连通无向图$G=(V,E)$，生成树不同，每棵树的权（即树中所有边上的权值之和）也可能不同。设R为G的所有生成树的集合，若T为R中边的权值之和最小的生成树，则T称为G的最小生成树（*Minimum-Spanning-Tree, MST*）。



- 如果一个连通图本身就是一棵树，则其最小生成树就是它本身
- 只有连通图才有生成树，非连通图只有生成森林



最小生成树（最小代价树）

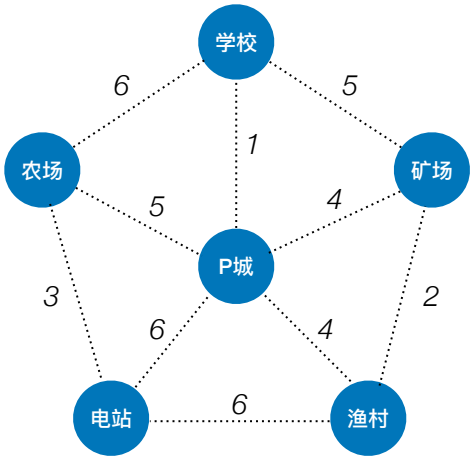
对于一个带权连通无向图$G=(V,E)$，生成树不同，每棵树的权（即树中所有边上的权值之和）也可能不同。设R为G的所有生成树的集合，若T为R中边的权值之和最小的生成树，则T称为G的最小生成树（*Minimum-Spanning-Tree, MST*）。

- 最小生成树可能有多个，但边的权值之和总是唯一且最小的
- 最小生成树的边数 = 顶点数 − 1。砍掉一条则不连通，增加一条边则会出现回路
- 如果一个连通图本身就是一棵树，则其最小生成树就是它本身
- 只有连通图才有生成树，非连通图只有生成森林

求最小生成树 ⊖ Prim 算法 / Kruskal 算法

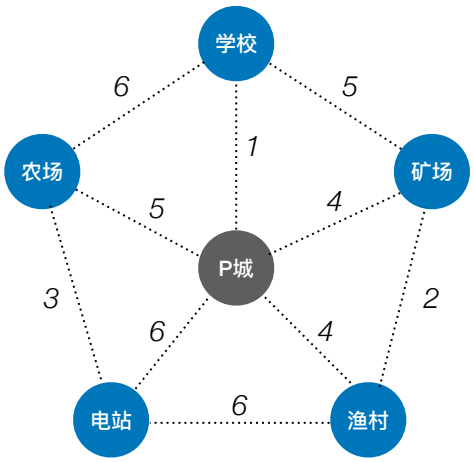
王道考研/CSKAOYAN.COM

Prim 算法（普里姆）



Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

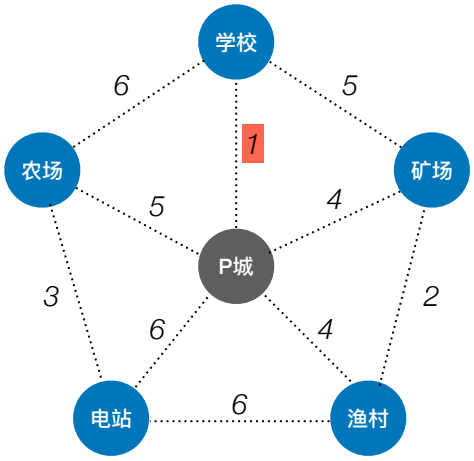
Prim 算法（普里姆）



Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

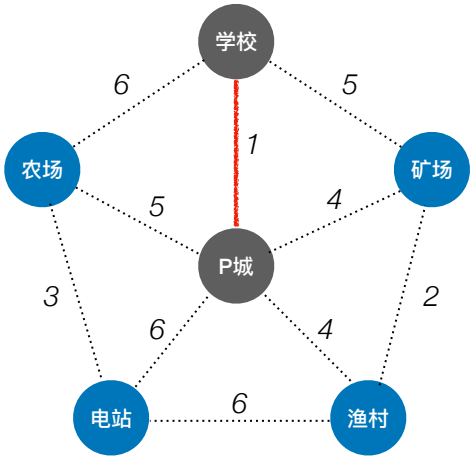
王道考研/CSKAOYAN.COM

Prim 算法（普里姆）



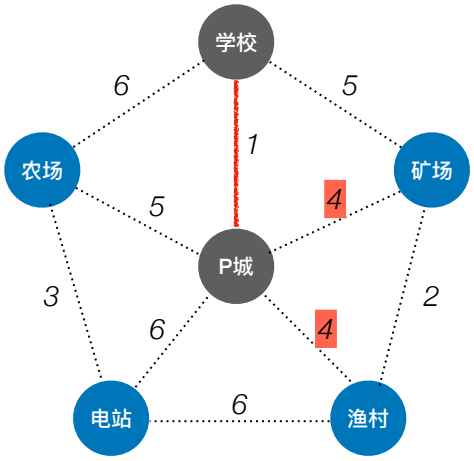
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



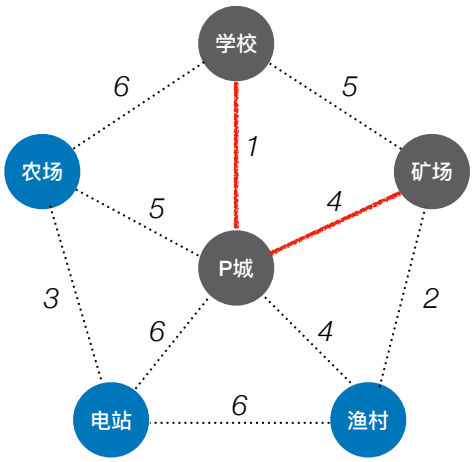
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



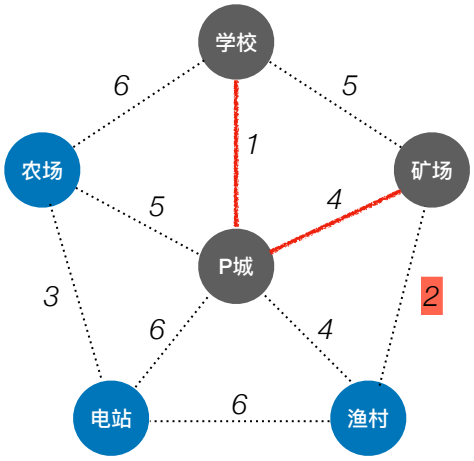
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



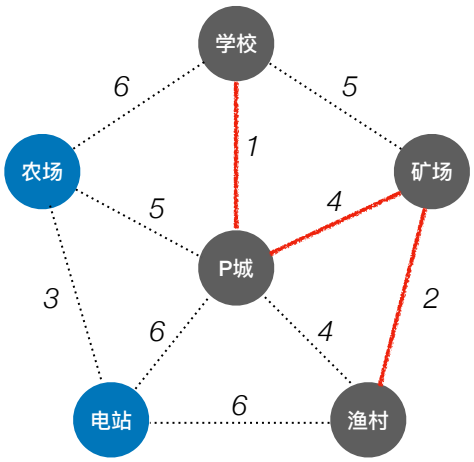
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



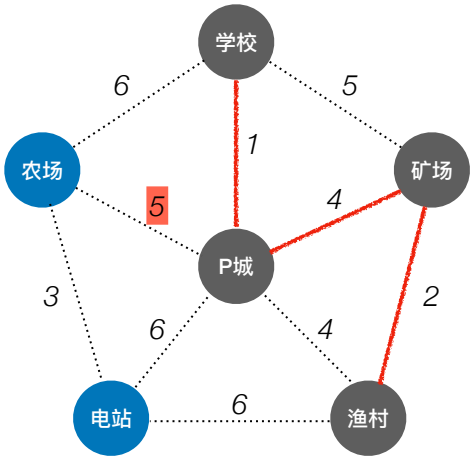
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



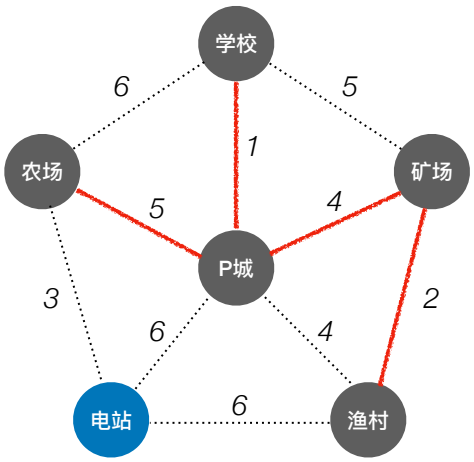
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

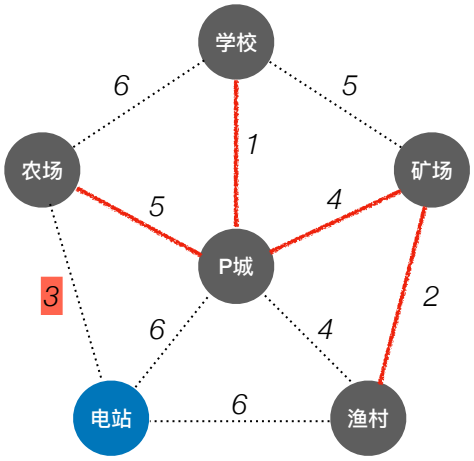
Prim 算法（普里姆）



Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

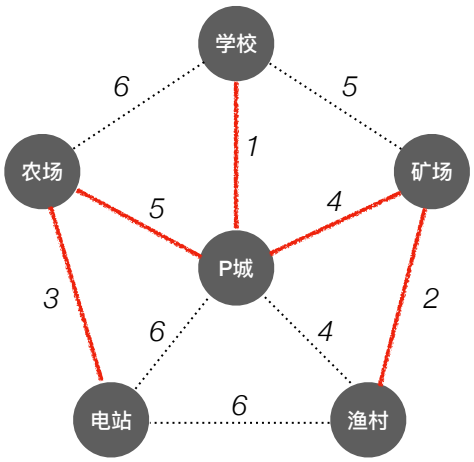
王道考研/CSKAOYAN.COM

Prim 算法（普里姆）



Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

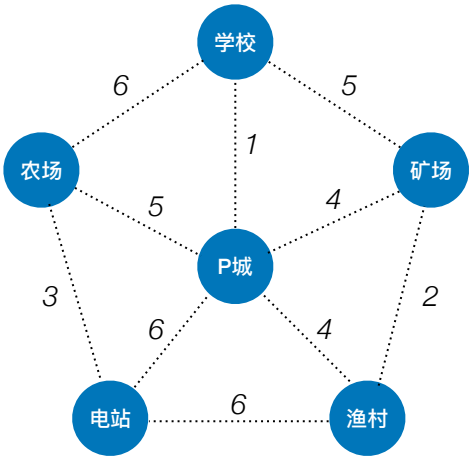
Prim 算法（普里姆）

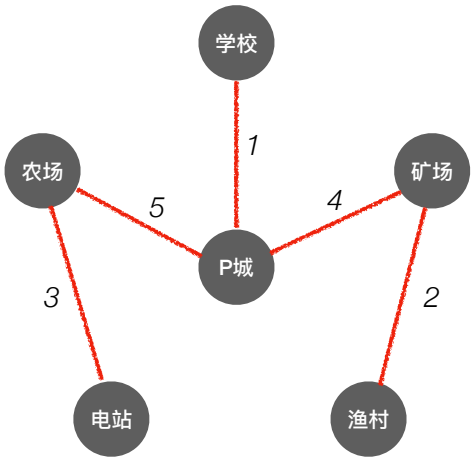


Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

王道考研/CSKAOYAN.COM

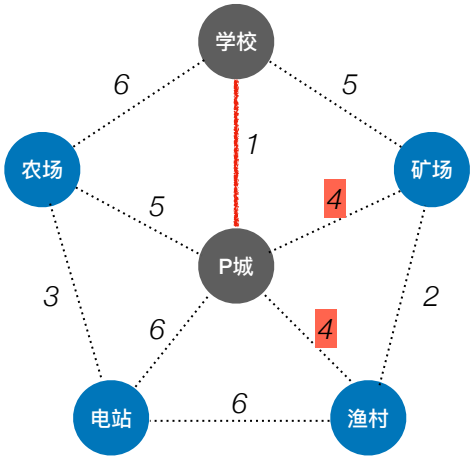
Prim 算法（普里姆）





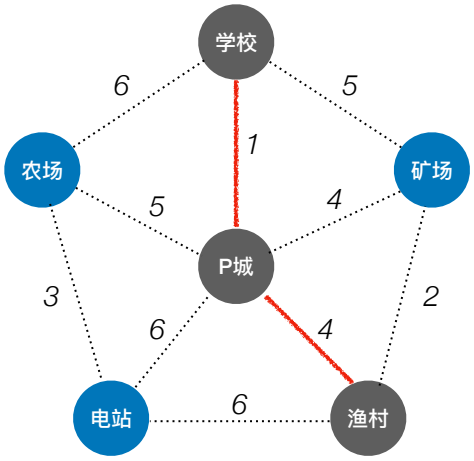
最小代价：15

Prim 算法（普里姆）



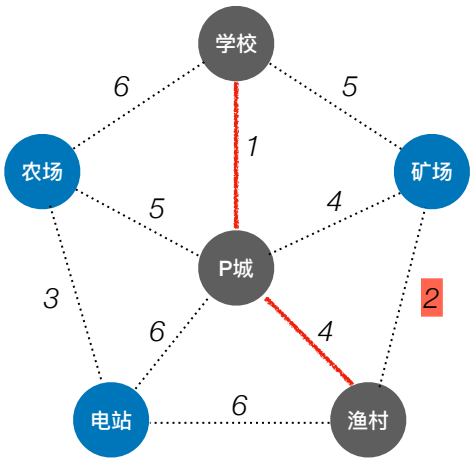
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



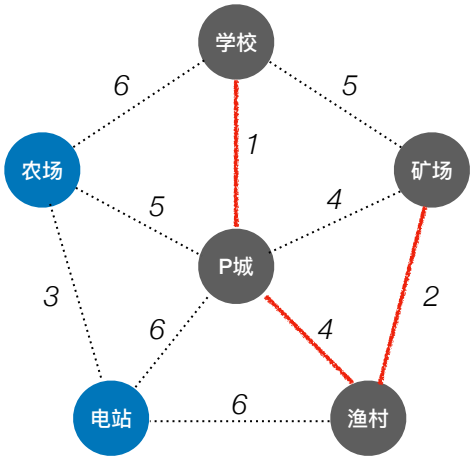
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



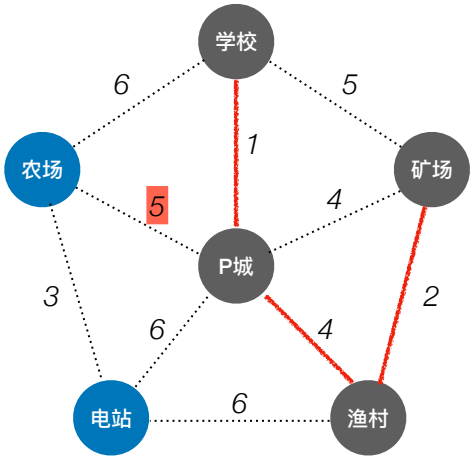
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



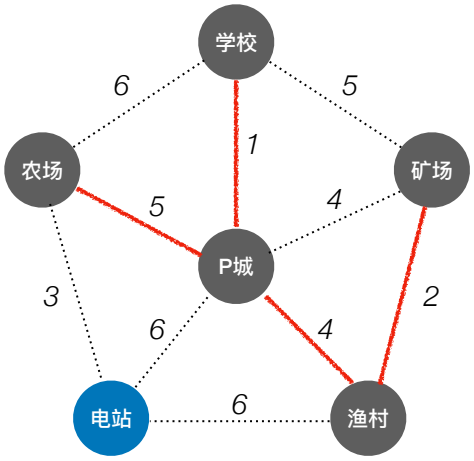
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



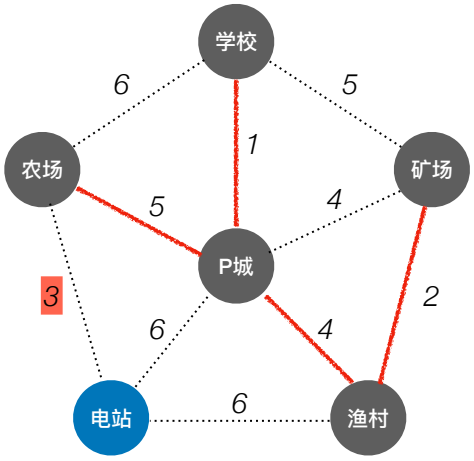
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



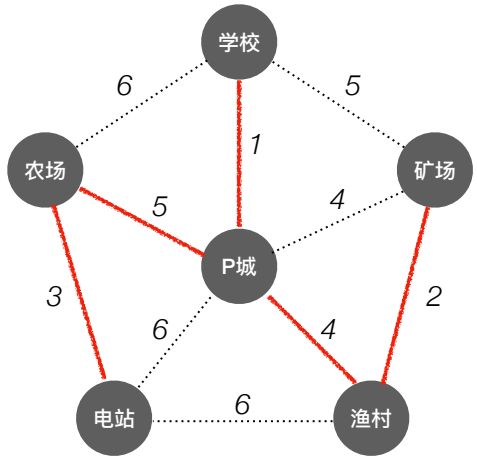
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



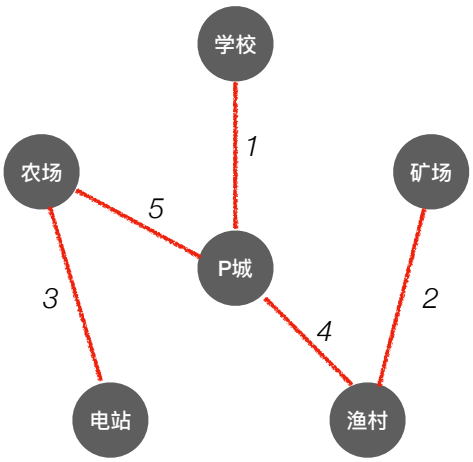
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）

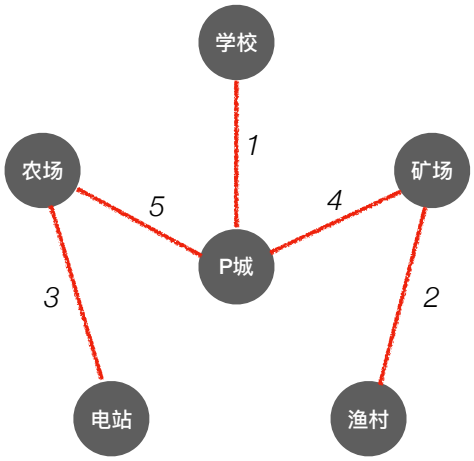


Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）

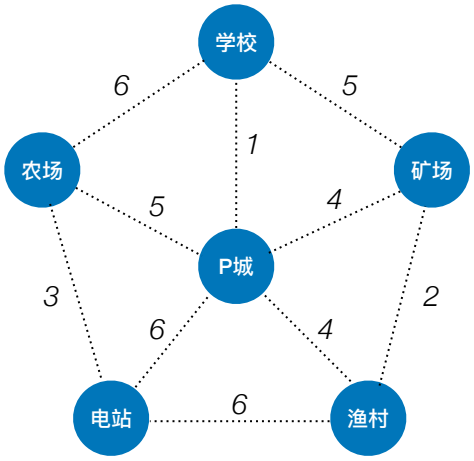


最小代价：*15*



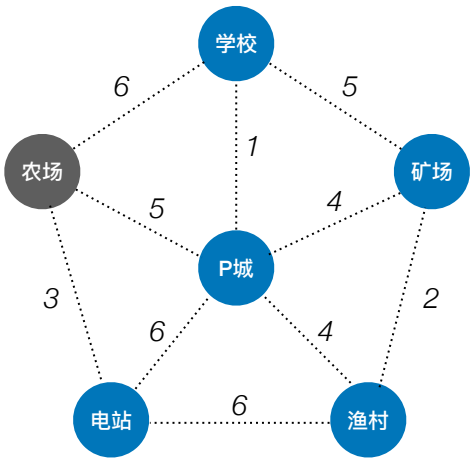
最小代价：*15*

Prim 算法（普里姆）



Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

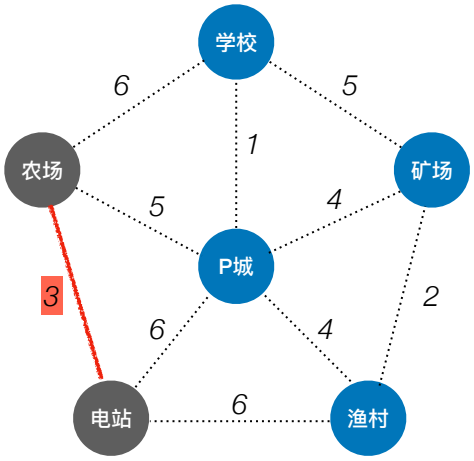
Prim 算法（普里姆）



Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

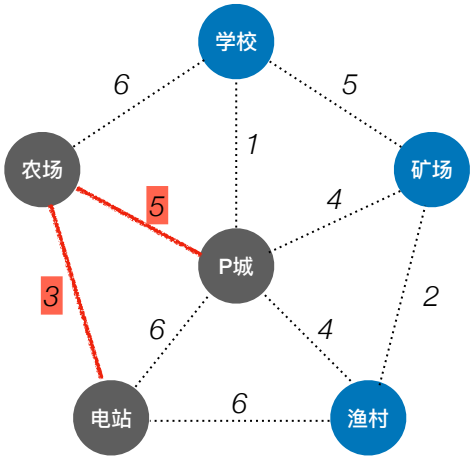
王道考研/CSKAOYAN.COM

Prim 算法（普里姆）



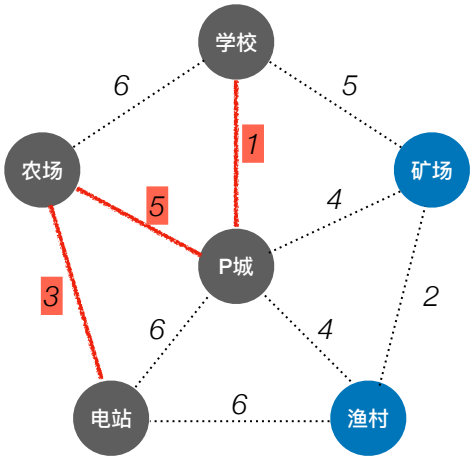
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



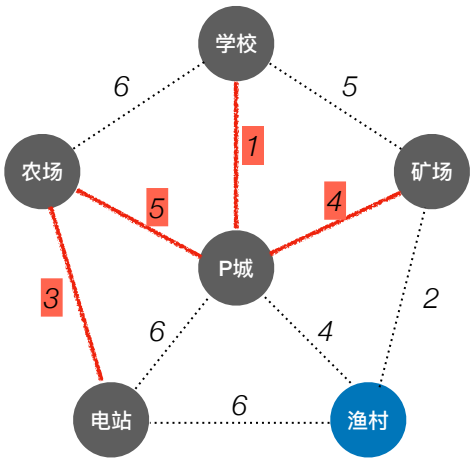
Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Prim 算法（普里姆）



Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

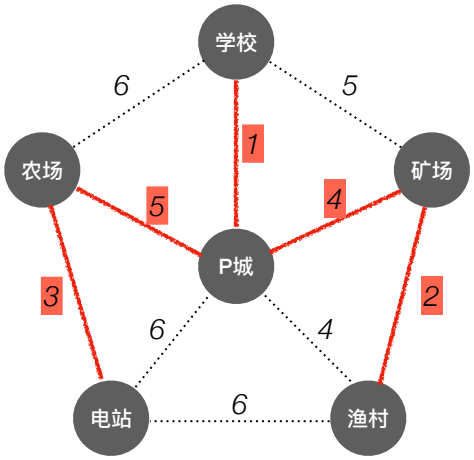
Prim 算法（普里姆）



Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

王道考研/CSKAOYAN.COM

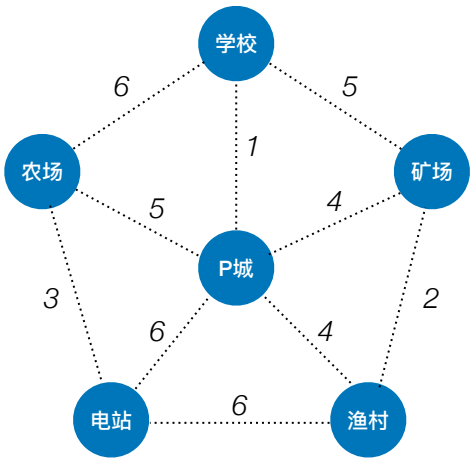
Prim 算法（普里姆）



最小代价：15

Prim 算法（普里姆）：
从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

Kruskal 算法（克鲁斯卡尔）

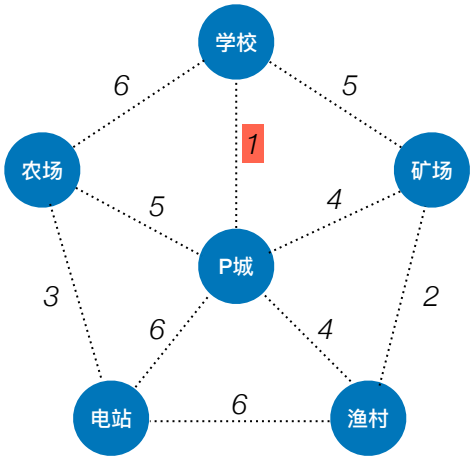


Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

Kruskal 算法（克鲁斯卡尔）

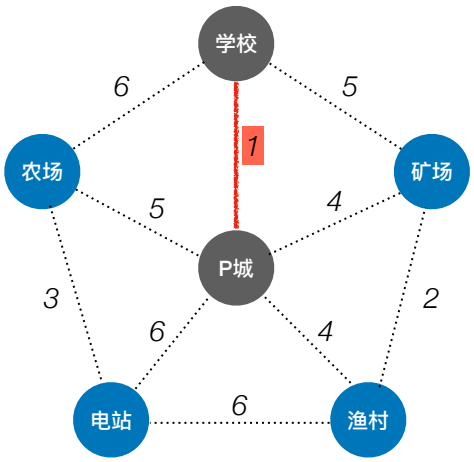


Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

Kruskal 算法（克鲁斯卡尔）



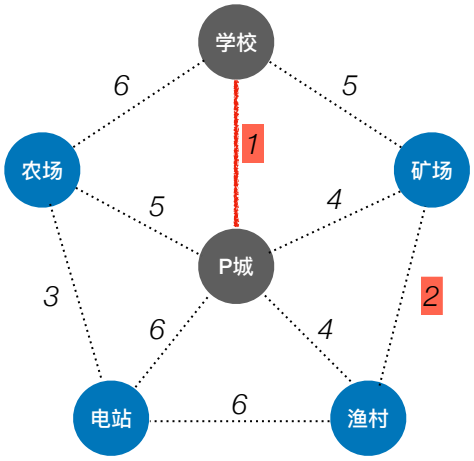
Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

王道考研/CSKAOYAN.COM

Kruskal 算法（克鲁斯卡尔）

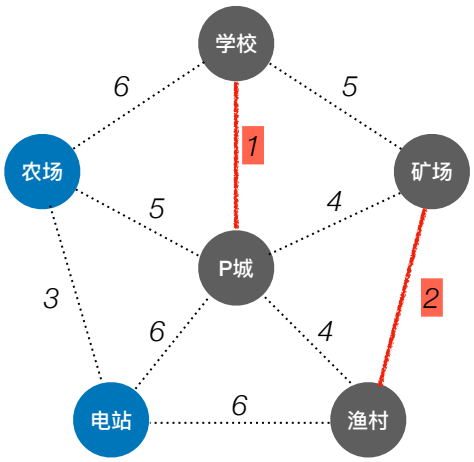


Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

Kruskal 算法（克鲁斯卡尔）



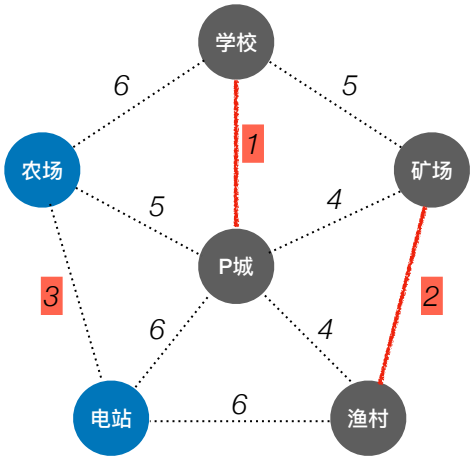
Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

王道考研/CSKAOYAN.COM

Kruskal 算法（克鲁斯卡尔）

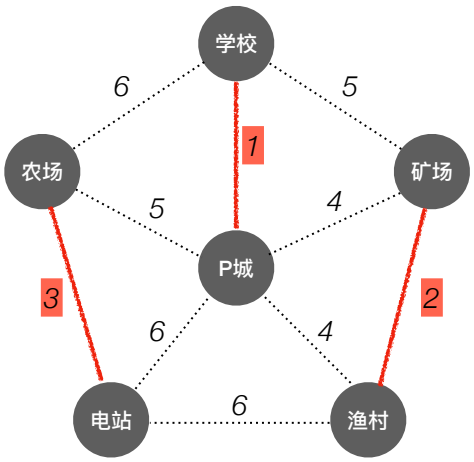


Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

Kruskal 算法（克鲁斯卡尔）



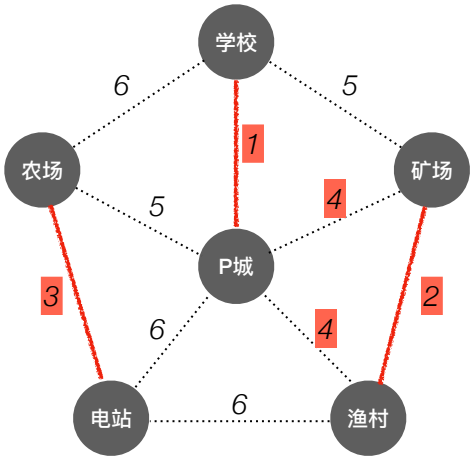
Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

王道考研/CSKAOYAN.COM

Kruskal 算法（克鲁斯卡尔）

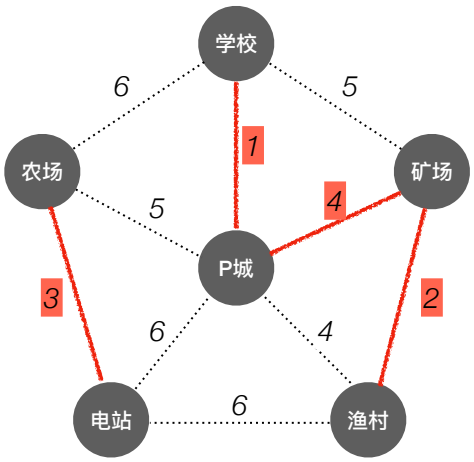


Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

Kruskal 算法（克鲁斯卡尔）



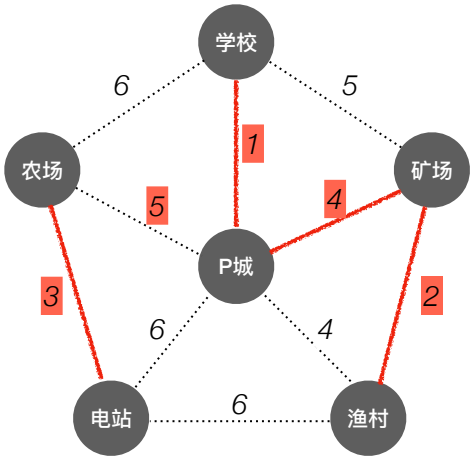
Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

王道考研/CSKAOYAN.COM

Kruskal 算法（克鲁斯卡尔）

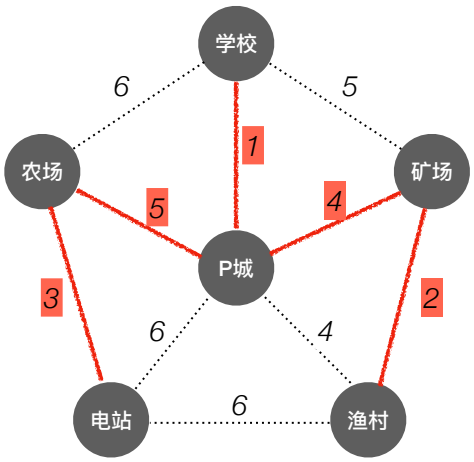


Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

Kruskal 算法（克鲁斯卡尔）



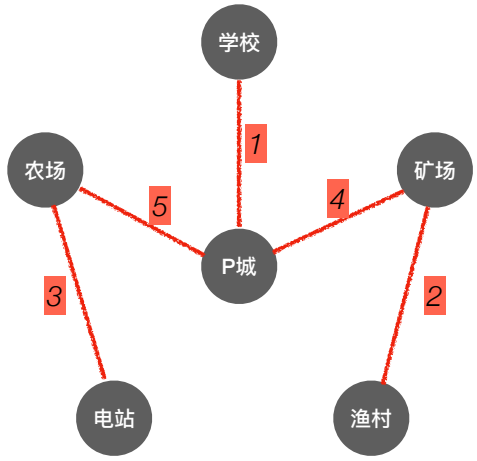
Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

王道考研/CSKAOYAN.COM

Kruskal 算法（克鲁斯卡尔）



最小代价：15

Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）

直到所有结点都连通

Prim 算法 v.s. Kruskal 算法

Prim 算法（普里姆）：

从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

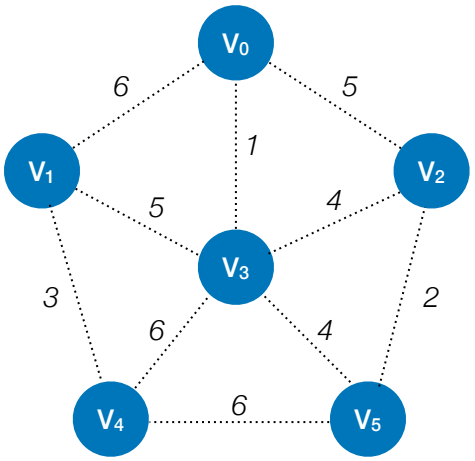
时间复杂度：$O(|V|^2)$
适合用于边稠密图

Kruskal 算法（克鲁斯卡尔）：

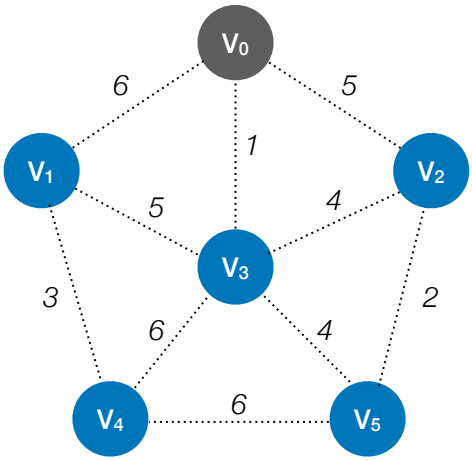
每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）
直到所有结点都连通

时间复杂度：$O(|E|\log_2|E|)$
适合用于边稀疏图

Prim 算法的实现思想



Prim 算法的实现思想



初始：从V_0开始

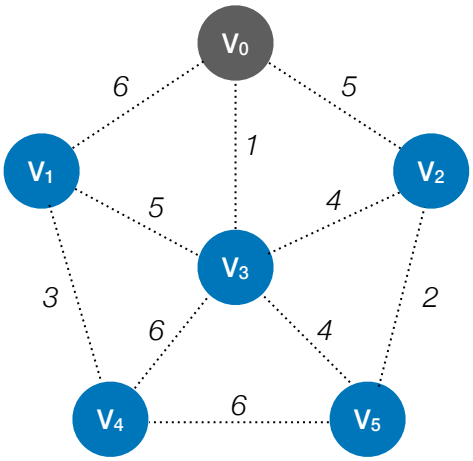
标记各节点是否已加入树

	V0	V1	V2	V3	V4	V5
isJoin[6]	✓	✗	✗	✗	✗	✗

lowCost[6]	0	6	5	1	∞	∞

各节点加入树的最低代价

Prim 算法的实现思想



第1轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

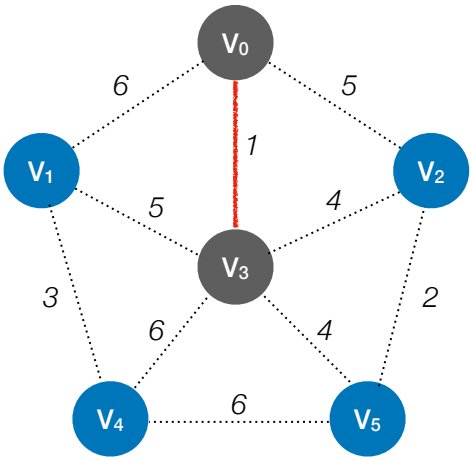
V0	V1	V2	V3	V4	V5
✓	✗	✗	✗	✗	✗

lowCost[6]

0	6	5	1	∞	∞

各节点加入树的最低代价

Prim 算法的实现思想



第1轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

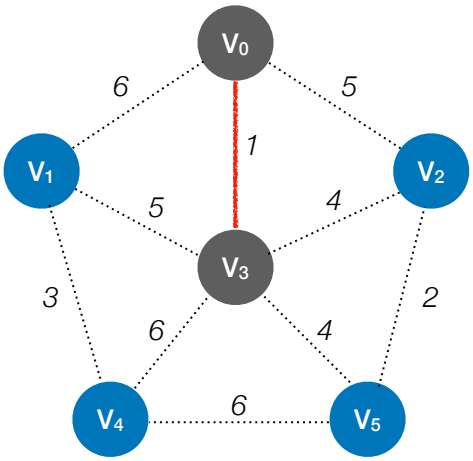
V0	V1	V2	V3	V4	V5
✓	✗	✗	✓	✗	✗

lowCost[6]

0	6	5	1	∞	∞

各节点加入树的最低代价

Prim 算法的实现思想



第1轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

V0	V1	V2	V3	V4	V5
✓	✗	✗	✓	✗	✗

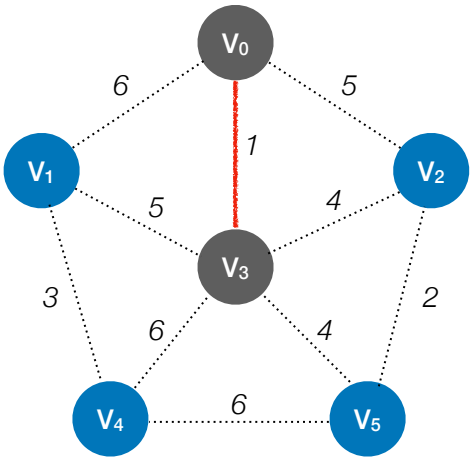
lowCost[6]

0	6	5	1	∞	∞

各节点加入树的最低代价

再次循环遍历，更新还没加入的各个顶点的lowCost值

Prim 算法的实现思想



第1轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

V0	V1	V2	V3	V4	V5
✓	✗	✗	✓	✗	✗

lowCost[6]

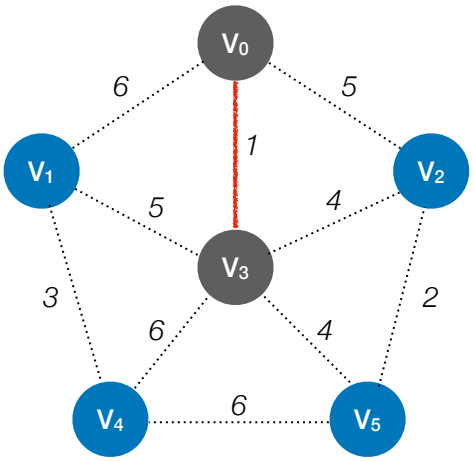
0	5	4	1	6	4

各节点加入树的最低代价

再次循环遍历，更新还没加入的各个顶点的lowCost值

王道考研/CSKAOYAN.COM

Prim 算法的实现思想



第2轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

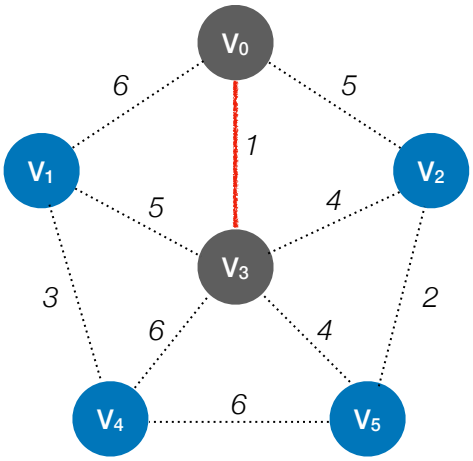
V0	V1	V2	V3	V4	V5
✓	✗	✗	✓	✗	✗

lowCost[6]

0	5	4	1	6	4

各节点加入树的最低代价

Prim 算法的实现思想



第2轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

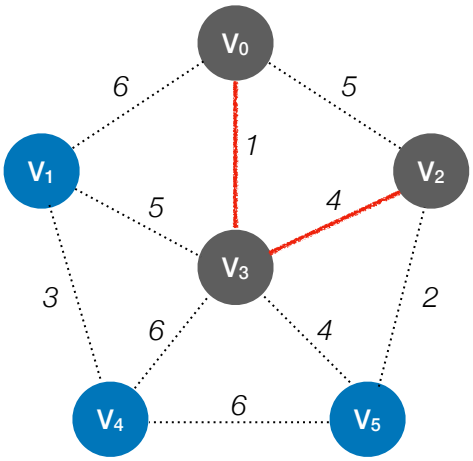
V0	V1	V2	V3	V4	V5
✓	✗	✗	✓	✗	✗

lowCost[6]

0	5	4	1	6	4

各节点加入树的最低代价

Prim 算法的实现思想



第2轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

V0	V1	V2	V3	V4	V5
✓	✗	✓	✓	✗	✗

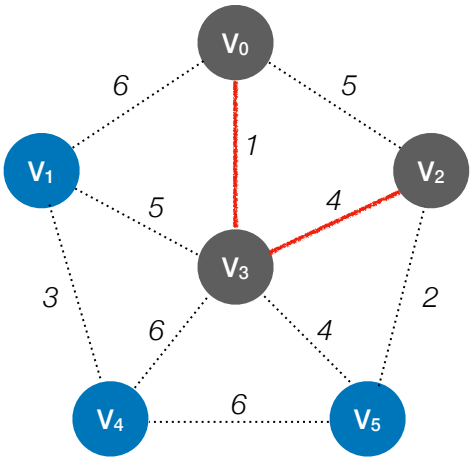
lowCost[6]

0	5	4	1	6	4

各节点加入树的最低代价

再次循环遍历，更新还没加入的各个顶点的lowCost值

Prim 算法的实现思想



第2轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

V0	V1	V2	V3	V4	V5
✓	✗	✓	✓	✗	✗

lowCost[6]

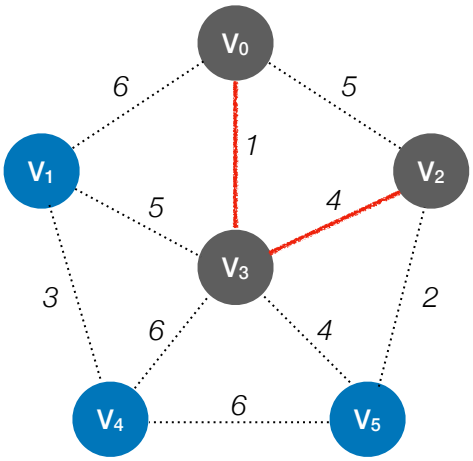
0	5	4	1	6	2

各节点加入树的最低代价

再次循环遍历，更新还没加入的各个顶点的lowCost值

王道考研/CSKAOYAN.COM

Prim 算法的实现思想



第3轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

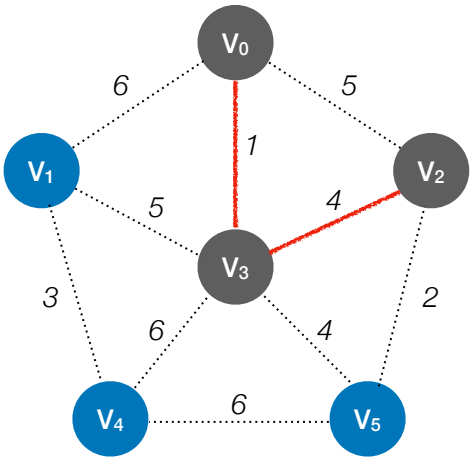
V0	V1	V2	V3	V4	V5
✓	✗	✓	✓	✗	✗

lowCost[6]

0	5	4	1	6	2

各节点加入树的最低代价

Prim 算法的实现思想



第3轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

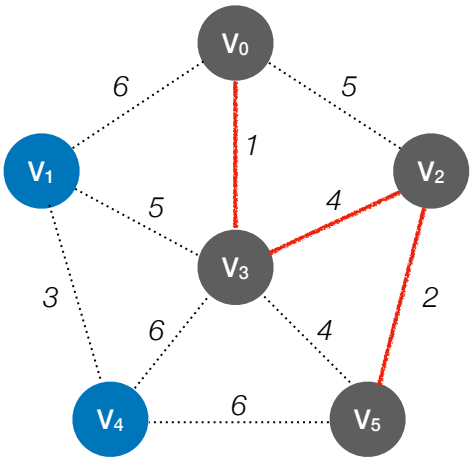
V0	V1	V2	V3	V4	V5
✓	✗	✓	✓	✗	✗

lowCost[6]

0	5	4	1	6	2

各节点加入树的最低代价

Prim 算法的实现思想



第3轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

V0	V1	V2	V3	V4	V5
✓	✗	✓	✓	✗	✓

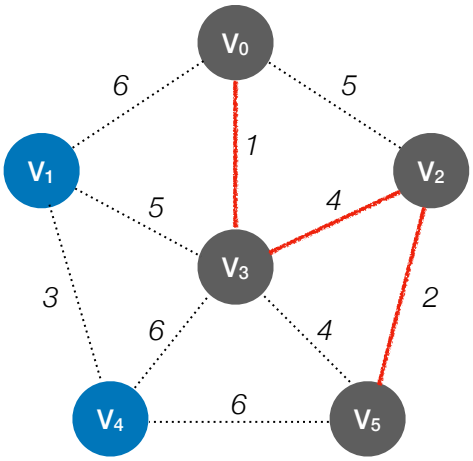
lowCost[6]

0	5	4	1	6	2

各节点加入树的最低代价

再次循环遍历，更新还没加入的各个顶点的lowCost值

Prim 算法的实现思想



第4轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

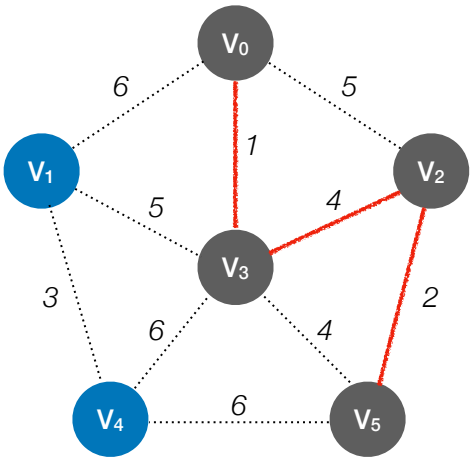
V0	V1	V2	V3	V4	V5
✓	✗	✓	✓	✗	✓

lowCost[6]

0	5	4	1	6	2

各节点加入树的最低代价

Prim 算法的实现思想



第4轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

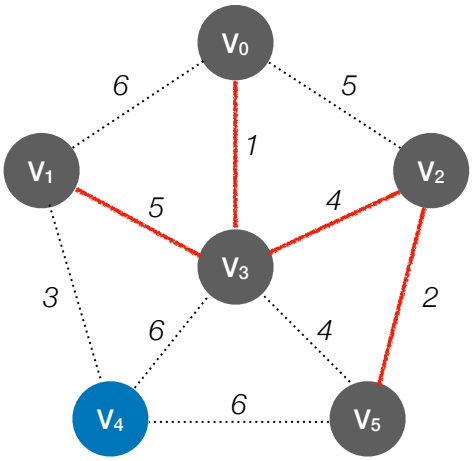
V0	V1	V2	V3	V4	V5
✓	✗	✓	✓	✗	✓

lowCost[6]

0	5	4	1	6	2

各节点加入树的最低代价

Prim 算法的实现思想



第4轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

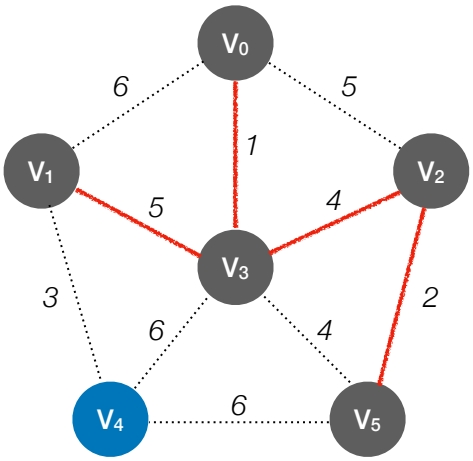
V0	V1	V2	V3	V4	V5
✓	✓	✓	✓	✗	✓

lowCost[6]

0	5	4	1	6	2

各节点加入树的最低代价

Prim 算法的实现思想



第4轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

V0	V1	V2	V3	V4	V5
✓	✓	✓	✓	✗	✓

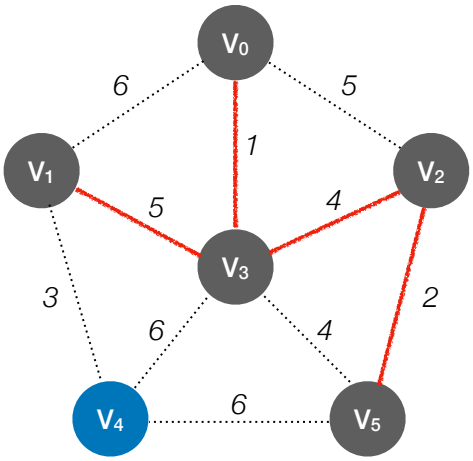
lowCost[6]

0	5	4	1	6	2

各节点加入树的最低代价

再次循环遍历，更新还没加入的各个顶点的lowCost值

Prim 算法的实现思想



第4轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

V0	V1	V2	V3	V4	V5
✓	✓	✓	✓	✗	✓

lowCost[6]

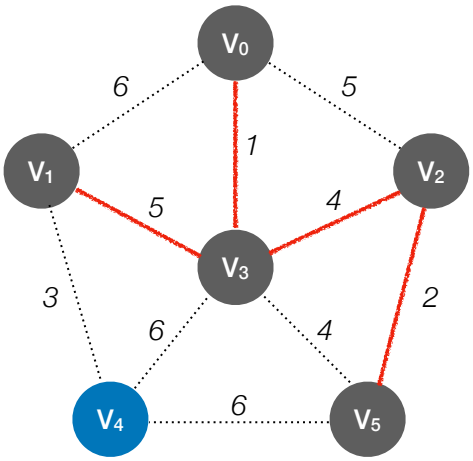
0	5	4	1	3	2

各节点加入树的最低代价

再次循环遍历，更新还没加入的各个顶点的lowCost值

王道考研/CSKAOYAN.COM

Prim 算法的实现思想



第5轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

V0	V1	V2	V3	V4	V5
✓	✓	✓	✓	✗	✓

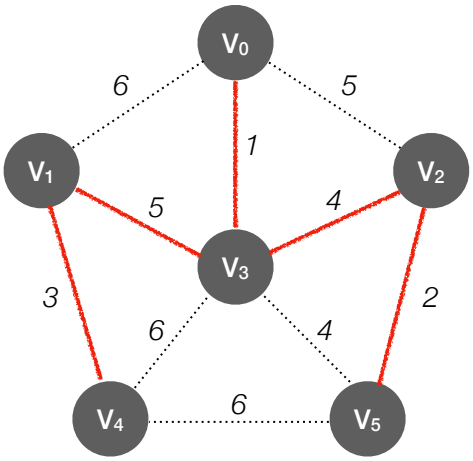
lowCost[6]

0	5	4	1	3	2

各节点加入树的最低代价

再次循环遍历，更新还没加入的各个顶点的lowCost值

Prim 算法的实现思想



第5轮：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点

标记各节点是否已加入树

isJoin[6]

V0	V1	V2	V3	V4	V5
✓	✓	✓	✓	✓	✓

lowCost[6]

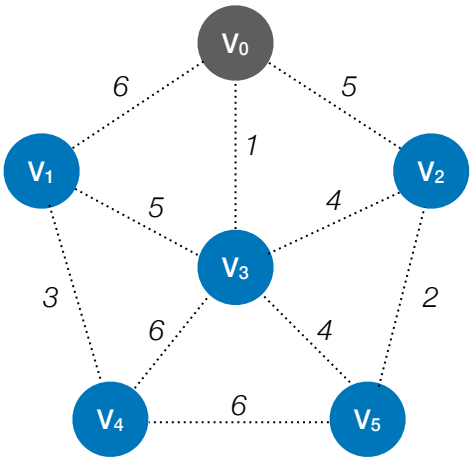
0	5	4	1	3	2

各节点加入树的最低代价

再次循环遍历，更新还没加入的各个顶点的lowCost值

Prim 算法的实现思想

总时间复杂度 $O(n^2)$，即$O(|V|^2)$



从V_0开始，总共需要 n-1 轮处理

每一轮处理：循环遍历所有个结点，找到lowCost最低的，且还没加入树的顶点。

每一轮时间复杂度O(2n)

再次循环遍历，更新还没加入的各个顶点的lowCost值

	V0	V1	V2	V3	V4	V5
isJoin[6]	✓	✗	✗	✗	✗	✗
lowCost[6]	0	6	5	1	∞	∞

Prim 算法 v.s. Kruskal 算法

Prim 算法（普里姆）：

从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

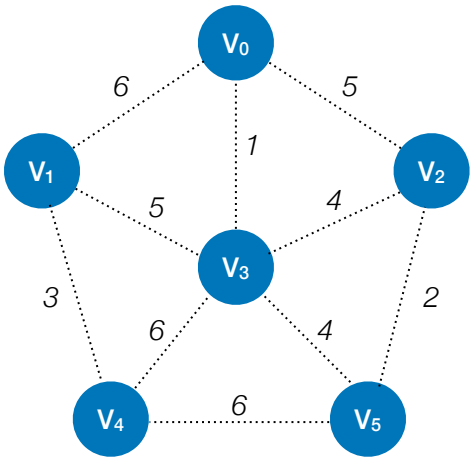
时间复杂度：$O(|V|^2)$
适合用于边稠密图

Kruskal 算法（克鲁斯卡尔）：

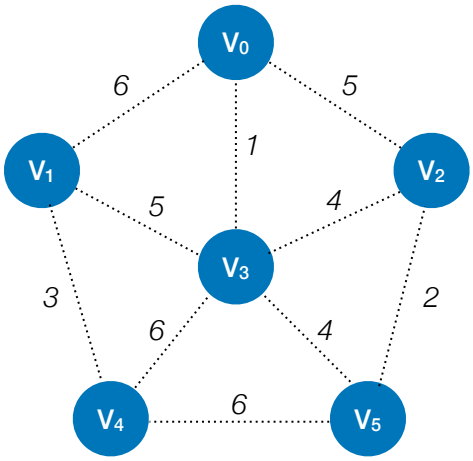
每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）
直到所有结点都连通

时间复杂度：$O(|E|\log_2|E|)$
适合用于边稀疏图

Kruskal 算法的实现思想



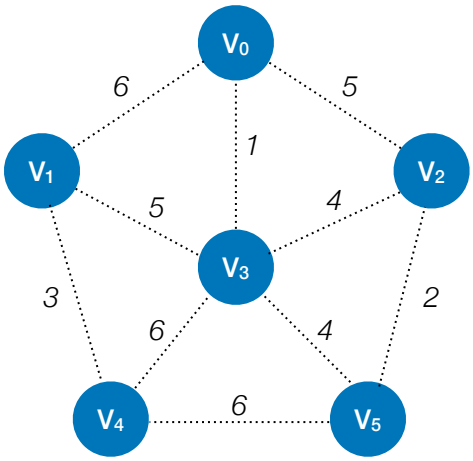
Kruskal 算法的实现思想



初始：将各条边按权值排序

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

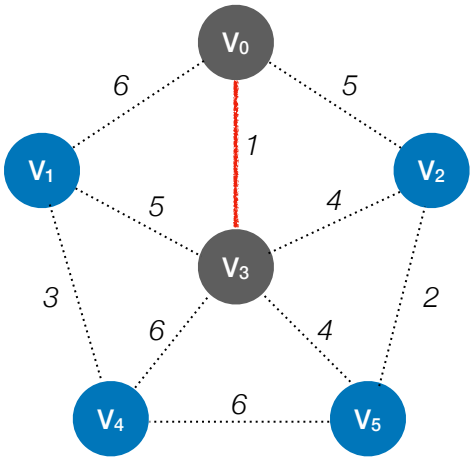
Kruskal 算法的实现思想



第1轮：检查第1条边的两个顶点是否连通（是否属于同一个集合）

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

Kruskal 算法的实现思想

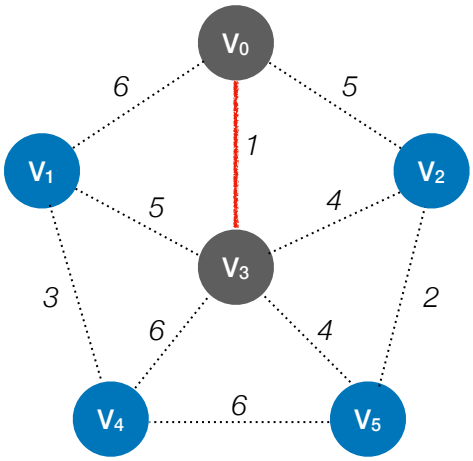


第1轮：检查第1条边的两个顶点是否连通（是否属于同一个集合）

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

不连通，连起来！

Kruskal 算法的实现思想

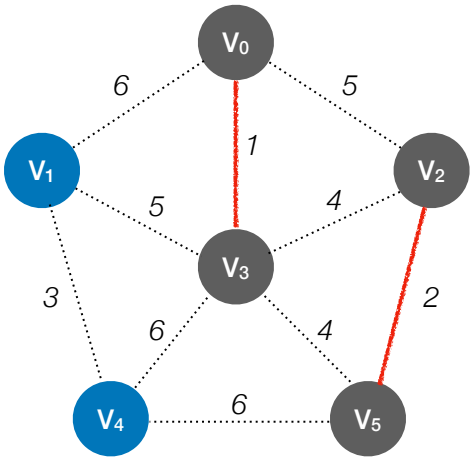


第2轮：检查第2条边的两个顶点是否连通（是否属于同一个集合）

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

王道考研/CSKAOYAN.COM

Kruskal 算法的实现思想



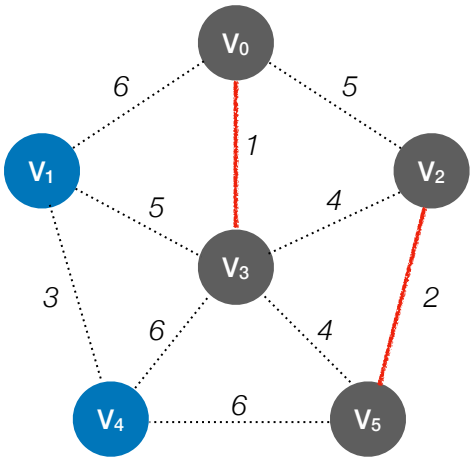
第2轮：检查第2条边的两个顶点是否连通（是否属于同一个集合）

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

不连通，连起来！

王道考研/CSKAOYAN.COM

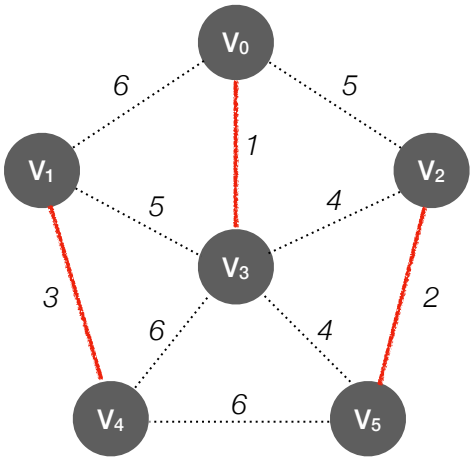
Kruskal 算法的实现思想



第3轮：检查第3条边的两个顶点是否连通（是否属于同一个集合）

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

Kruskal 算法的实现思想

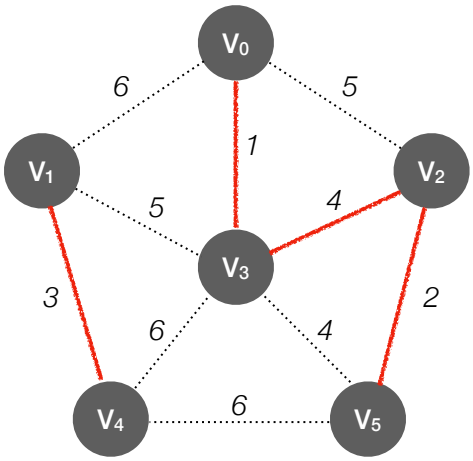


第3轮：检查第3条边的两个顶点是否连通（是否属于同一个集合）

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

不连通，连起来！

Kruskal 算法的实现思想

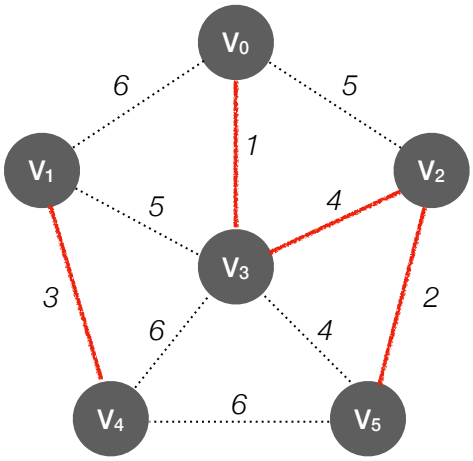


第4轮：检查第4条边的两个顶点是否连通（是否属于同一个集合）

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

不连通，连起来！

Kruskal 算法的实现思想

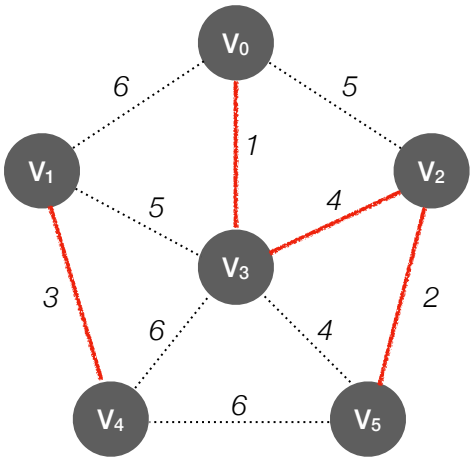


第5轮：检查第5条边的两个顶点是否连通（是否属于同一个集合）

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

已连通，跳过！

Kruskal 算法的实现思想

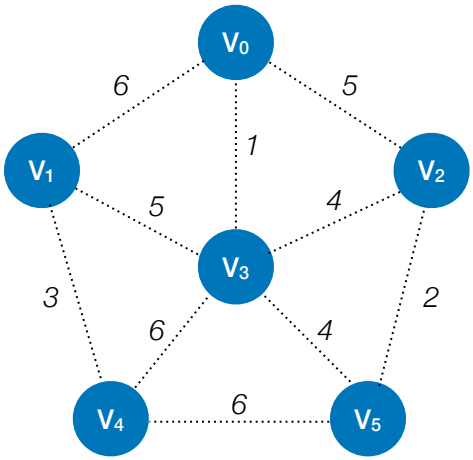


第6轮：检查第6条边的两个顶点是否连通（是否属于同一个集合）

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

王道考研/CSKAOYAN.COM

Kruskal 算法的实现思想



初始：将各条边按权值排序

weight	Vertex1	Vertex2
1	V_0	V_3
2	V_2	V_5
3	V_1	V_4
4	V_2	V_3
4	V_3	V_5
5	V_0	V_2
5	V_1	V_3
6	V_0	V_1
6	V_3	V_4
6	V_4	V_5

共执行 e 轮，每轮判断两个顶点是否属于同一集合，需要 $O(\log_2 e)$

总时间复杂度 $O(e\log_2 e)$

王道考研/CSKAOYAN.COM

知识回顾与重要考点

对于一个带权连通无向图$G=(V,E)$，生成树不同，每棵树的权（即树中所有边上的权值之和）也可能不同。设R为G的所有生成树的集合，若T为R中边的权值之和最小的生成树，则T称为G的最小生成树（*Minimum-Spanning-Tree, MST*）。

Prim 算法（普里姆）：

从某一个顶点开始构建生成树；
每次将代价最小的新顶点纳入生成树，直到所有顶点都纳入为止。

时间复杂度：$O(|V|^2)$
适合用于边稠密图

Kruskal 算法（克鲁斯卡尔）：

每次选择一条权值最小的边，使这条边的两头连通（原本已经连通的就不选）
直到所有结点都连通

时间复杂度：$O(|E|\log_2|E|)$
适合用于边稀疏图