Minimum Spanning Tree

Greedy Strategy

e.g.，找硬币问题

贪心策略即一步步扩展局部解，每一步选择必须满足

- feasible：必须满足问题本身的限制
- locally optimal：选择是当前局部可选项内最优的
- irrevocable：在之后步骤中不可撤销选择

但是贪心问题一般得不到正确的解

Minimum Spanning Tree

> 图 G 的生成树 T 是其子图，满足
>
> - T 包含 G 中所有顶点
> - T 是连通无环图，即树
>
> 定义一颗生成树的权为其所有边的权，即可定义 G 的最小生成树
>
> **定义 10.1** 最小生成树的直接定义
>
> 如果 T 是 G 的生成树，且图中不存在任何其他权比 T 小的生成树，则称 T 为图 G 的最小生成树

最小生成树不能通过简单的图遍历完成

Prim Algorithm

Strategy

构建一颗生成树：从某个结点出发，不断发现新节点，过程类似图遍历

保证生成树权值最小：对当前候选节点，选择边权值最小的一条，将与其相连的节点加入 MST

即与当前已生成局部最小生成树相连的节点（fringe）中**贪心**选择权最小的

Correctness

Prim 算法的执行过程可以分为 n 个阶段，初始情况生成树中只有一个节点，记为 $T^{(1)}$ ，之后贪心选择一个节点加入 MST，记为 $T^{(2)}$ ，以此类推

最终局部生成树中有 n 个节点，即是全图的生成树。证明 Prim 算法的正确性只需对上述过程进行归纳，归纳不变量：**局部生成树是局部最小生成树**

为此，需要先将最小生成树的定义**"算法化"**，将全局最小权生成树的性质等价转换为一种更局部的性质。这一思想类似之前课程中对于割点和桥的直接定义 $\rightarrow$ 基于路径的定义 $\rightarrow$ 基于 DFS 的定义。

基于**最小生成树性质（MST property）**得到最小生成树的间接定义

> **定义 10.2** 最小生成树的间接定义
>
> 给定图 G 的生成树 T，定义 T 是图 G 的最小生成树，如果它满足"最小生成树性质"：对任意不在 T 中的边 e，$T \cup \{e\}$ 含有一个环，且 e 是环中最大权值的边（可能不唯一）

MST 直接定义和间接定义等价性不是显然的，需要证明

> **引理 10.1** 所有满足 MST property 的生成树 T 有相同的权值
>
> 证明：基于数学归纳法证明，设生成树 T_1, T_2 均满足 MST 属性，对 T_1 和 T_2 之间**边的差异数目** k 归纳，即 T_1, T_2 均有 $n-1$ 条边，其中有 k 条边不相同
>
> Basis. $k = 0$ 时显然 T_1, T_2 有相同的权值
>
> I.H. 对任意 $0 \leqslant j < k$ ，如果 T_1 和 T_2 边差异数为 j ，则 T_1, T_2 权值相同
>
> Ind.Step. 考虑 T_1, T_2 中不同的边中权值最小的 uv ，不失一般性，设其在 T_2 中，则 T_1 中在 u 与 v 之间存在一条路径，且路径上至少有一条边不在 T_2 中（否则 T_2 中会有环），设其为 $w_i w_{i+1}$ ，则可得
>
> - $uv \leqslant w_i w_{i+1}$ ：这两条边都是 T_1, T_2 之间不同的边，根据定义，uv 是其中最小的边
> - $uv \geqslant w_i w_{i+1}$ ：$T_1 \cup \{uv\}$ 形成环，且 $w_i w_{i+1}$ 在环上，由于 T_1 的 MST property，uv 是环上最大权边
>
> 由此，有 $uv = w_i w_{i+1}$ ，将 T_1 中的 $w_i w_{i+1}$ 删去，加入 uv 得到另一颗生成树 T_3 ，显然 $T_1 = T_3$ ，又易得 T_2, T_3 之间不同的边只有 $k-1$ 条，根据 I.H.，$T_2 = T_3 = T_1$

基于引理 10.1，可证明最小生成树的直接定义与间接定义等价

> **定理 10.1** T 是最小生成树 $\iff$ T 具有 MST property

> $(\Rightarrow)$：设一颗最小生成树 T 不满足 MST property，则存在一条边 $e \notin T$ 且 $T \cup \{e\}$ 存在环，环中存在边 e' 满足 $e' > e$，则将 e' 删去，加入 e，得到一颗权值更小的生成树，与 T 是最小生成树矛盾
>
> $(\Leftarrow)$：设生成树 T 满足 MST property，假设 T_{min} 是一颗 MST，根据 $(\Rightarrow)$ 的证明，T_{min} 具有 MST property，而根据引理 10.1 ，T, T_{min} 有相同的权值，T 也是最小生成树

基于最小生成树的性质，可以归纳证明 Prim 算法的正确性

> **定理 10.2** Prim 算法总能得到图 G 的 MST
>
> 证明：对 Prim 算法的执行阶段 k 归纳
>
> Basis. $T^{(1)}$ 显然为最小生成树
>
> I.H. 对任意 $0 \leqslant j < k$，$T^{(j)}$ 总是最小生成树
>
> Ind.Step. 需要证明 $T^{(k)}$ 具有 MST property，显然 $T^{(k)}$ 由 $T^{(k-1)}$ 新增一个顶点和一条边得到，记新增顶点为 v，新增边为 $u_1 v$。现在考虑在 $T^{(k)}$ 中加一条边形成环，若加的边两个顶点均在 $T^{(k-1)}$ 中，则由 I.H.，$T^{(k-1)}$ 满足 MST property，新加的边是环上权值最大的边。只需考虑新加入的边一个顶点为 v，一个顶点为 $T^{(k-1)}$ 中某顶点 u_i。
>
> 根据 Prim 算法的贪心策略，在所有使 v 和 $T^{(k-1)}$ 相连的边中，$u_1 v$ 是权值最小的边。考虑将边 $u_i v$ 加入 $T^{(k)}$ 形成环，假设 $u_i v$ 不是环上权值最大的边，则环中一定存在至少一条严格大于 $u_i v$ 的边，设从 v 出发，沿顺时针方向遇到第一条权值大于 $u_i v$ 的边记为 $w_a w_{a+1}$，沿逆时针方向遇到第一条权值大于 $u_i v$ 的边记为 $w_{b-1} w_b$（仅有一条边权值严格大于 $u_i v$ 时证明类似）。不失一般性，假设 w_a 先被 Prim 算法选中，则根据 Prim 算法的贪心原则，v 一定在 w_b 之前被选中， 因为从 w_a 到 w_b 沿环逆时针（逆时针路径上边权包括 $u_1 v, u_i v$ 均小于 $w_a w_{a+1}$，在这些边加入 MST 之前都不会选中 $w_a w_{a+1}$）一定要经过边 $w_{b-1} w_b$，而 $w_{b-1} w_b$ 大于逆时针路径上所有其他的边，这将使 v 先于 w_b 被选中，与之前的假设矛盾。

Implementation

Prim 算法基于 BFS 的过程构建图的生成树。算法每次从一个调度器取出一节点处理，并将其所有不在调度器中的邻居加入调度器，这一过程保证了算法遍历整张图，并得到一颗生成树

Prim 算法的关键在于贪心选择的过程，在算法运行过程中，图中未被选入当前局部 MST 且与当前局部 MST 有边相连的节点称为 **Fringe** 节点，贪心选择的候选即是 Fringe 节点，可以通过将 Fringe 节点维护成一个 **优先级队列** 实现。这个优先级队列就是 Prim 算法执行的调度器。

每次从优先队列取出节点后，其邻居可分为两类：

- 不在 fringe 的，加入 fringe
- 已在 fringe 的，检查其优先级是否需要更新

算法具体框架如下

```
Initialize all nodes in G as UNSEEN
Initialize the priority queue queNode as empty
Initialize edge set MST as empty
Select s to start
s.candidateEdge := NULL
queNode.INSERT(s, -INF)
while queNode != empty do
    v := queNode.EXTRACT-MIN()
    MST:= MST + v.candidiateEdge
    UPDATE-FRINGE(queNode, v)
```

其中的 UPDATE-FRINGE 为

```
foreach neighbor w of v do
    newWeight := vw.weight
    if w is UNSEEN then
        w.candidateEdge := vw
        queNode.INSERT(w, newWeight)
    else
        if newWeight < w.priorpty then
            w.candidateEdge := vw
            queNode.DECREASE-KEY(w, newWeight)
```

Analysis

设图 G 中有 n 个顶点，m 条边，在 Prim 算法的执行过程中

- 从节点的角度看，每个节点都要进入优先队列再离开优先队列，故 INSERT 和 EXTRACT-MIN 要执行 n 次
- 从边的角度看，最坏情况要执行 m 次 DECREASE-KEY

故 Prim 算法的代价完全取决于 priority queue 的实现，可以基于堆和数组实现

操作	数组实现	堆实现
INSERT	O(1)	O(log n)
EXTRACT-MIN	O(n)	O(log n)
DECREASE-KEY	O(1)	O(log n)

基于数组实现时时间复杂度为 $O(n^2+m)$，基于堆实现时间复杂度为 $O((m+n)\log n)$

Kruskal

Strategy

Kruskal 算法的贪心思想同样十分简单，将边权按照从小到大排列，每次选择权最小的边，算法始终保持加入的边不成环，直至最终得到最小生成树

key issue 即是如何保证新加入的边不成环。加入边成环即表示加入边的两个端点在此之前已经连通，而顶点间的连通关系是一种等价关系，判断是否成环即是判断边的两个端点是否连通，此处可以使用之前课程讲过的并查集来判断加入后是否成环

Correctness

类比 Prim 算法，将 Kruskal 算法执行的过程按照已加入的边数分为不同的阶段

在执行过程中选中的边形成的子图不一定连通，算法逐步得到的是一个最小生成森林，逐步加边直至所有连通片连通，得到图的生成树。

> **定理 10.3** Kruskal 算法总能得到图 G 的 MST
>
> 证明过程对上述过程归纳，归纳不变量：Kruskal 算法得到的局部生成森林总是被某个生成树 T 包含
>
> Basis. 显然对于任何 MST T，有 $F^{(0)}=\varnothing\in T$
>
> I.H. $F^{(k-1)}$ 包含在某 MST T 中
>
> Ind.Step. 设 $F^{(k)}=F^{(k-1)}\cup\{e\}$，若 $e\in T$，则 $F^{(k)}\in T$，得证。否则 $e\notin T$，则 $T\cup\{e\}$ 中有环，环上一定有一边 e' 权值等于 e
>
> - T 具有 MST property，易得 $e'\leqslant e$
> - e 由 Kruskal 贪心选出，则 $T\setminus F^{(k-1)}$ 中的边权值均不小于 e，有 $e\leqslant e'$
>
> 故 $e'=e$，可通过调换两条边得到新的 MST $T'=(T\setminus\{e'\})\cup\{e\}$，显然 $T'=T$，则 T' 也是 MST，$F^{(k)}$ 包含于某个 MST，得证

Implementation

算法框架如下

```
Build a priority queue queNode of edges in G
Initialize a disjoint set of nodes in G
Initialize edge set MST as empty
while queNode != empty do
    vw := queNode.EXTRACT-MIN()
    if FIND(v) != FIND(w) then
        MST := MST + vw
        if MST.size == n-1 then
            return
        UNION(v, w)
```

使用堆实现优先队列或直接对边按照权值排序均可

Analysis

排序边/维护优先级队列的代价为 $O(m \log m)$

执行过程中会产生 $O(m)$ 条 UNION/FIND 指令，根据之前课程的结论，采用 WEIGHTED-UNION 和 C-FIND 的代价为 $O(m+n)$

故 Kruskal 算法的代价为 $O(m \log m)$

Prim vs. Kruskal

对于寻找 MST 的算法，由于至少需要检查一遍所有的边，故下界为 $\Omega(m)$

Prim 算法代价为 $O(n^2+m), O((m+n)\log n)$，而 Kruskal 算法代价为 $O(m \log m)$，实际问题中孰优孰劣一般取决于图的稠密程度。总的来说，Prim 更适用于稠密图，Kruskal 更适用于稀疏图