# 最小生成树算法的数据结构

刘恩萌

南京大学计算机科学与技术系
171860013@smail.nju.edu.cn

2018 年 10 月 22 日

# Contents

# Contents

# 最小生成树算法

```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

- 在带权图中选择一棵权值最小的生成树

# 最小生成树算法

```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

- 在带权图中选择一棵权值最小的生成树
- 贪心选择方法

# Contents

# Kruskal 算法

1. 选择策略：在所有连接森林中两棵不同树 的边中，找到权重最小的边
2. 终止：选满 $n-1$ 条边

# Kruskal 算法

1. 选择策略：在所有连接森林中两棵不同树 的边中，找到权重最小的边
2. 终止：选满 $n-1$ 条边
    1. 权重最小 $\Rightarrow$ 排序!

# Kruskal 算法

1. 选择策略：在所有连接森林中两棵不同树 的边中，找到权重最小的边
2. 终止：选满 $n-1$ 条边
   1. 权重最小 $\Rightarrow$ 排序!
   2. 不与已经选择的边构成循环?

# Kruskal 算法

1. 选择策略：在所有连接森林中两棵不同树 的边中，找到权重最小的边
2. 终止：选满 $n-1$ 条边
   1. 权重最小 $\Rightarrow$ 排序！
   2. 不与已经选择的边构成循环?

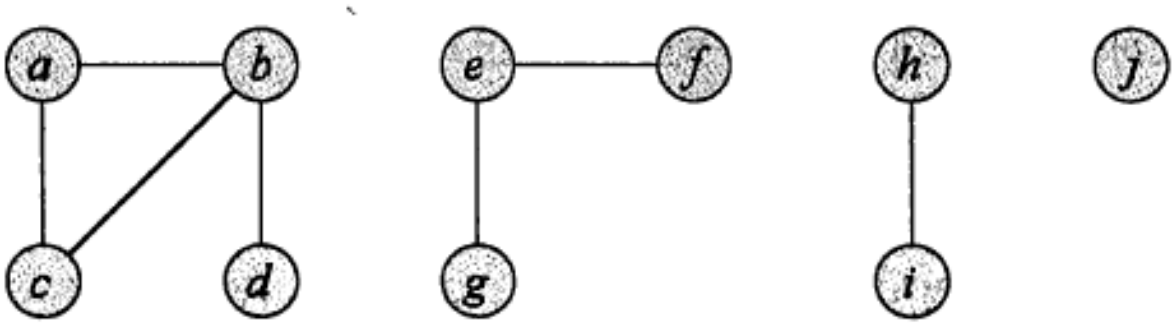
## 问题 B: Happy PA

时间限制: 1 Sec 内存限制: 128 MB
提交: 90 解决: 23
[提交][状态][讨论版][命题人:admin]

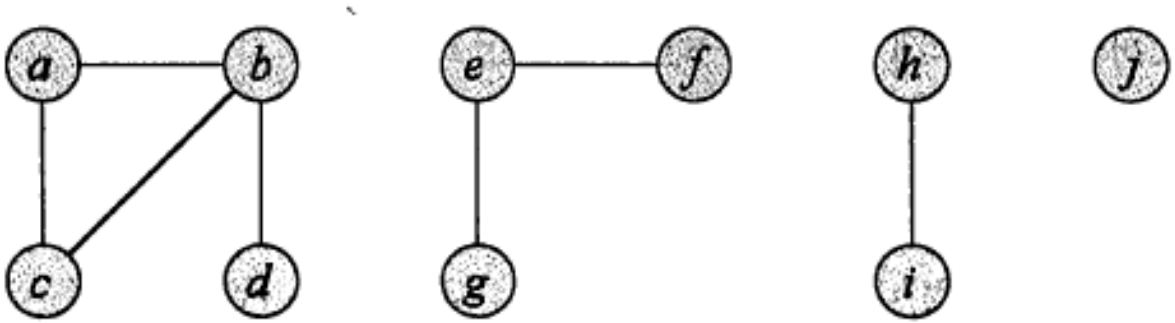
### 题目描述

Happy PAing until tomorrow moring. This night , dalao zty are plan to happy on his PA work . And as a dalao , zty finished the coding task easily . But unfortunately , he sees:

# Kruskal 算法



# Kruskal 算法




**21.1-2** 证明：CONNECTED-COMPONENTS 处理完所有的边后，两个顶点在相同的连通分量中当且仅当它们在同一个集合中。

# 并查集实现 Kruskal 算法

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

# 并查集实现 Kruskal 算法

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

1. 将边按权重从小到大排序

# 并查集实现 Kruskal 算法

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

1. 将边按权重从小到大排序
2. 初始化并查集 (MAKE-SET)

# 并查集实现 Kruskal 算法

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```

1. 将边按权重从小到大排序
2. 初始化并查集 (MAKE-SET)
3. 查询是否已经相连 (FIND-SET)

# 并查集实现 Kruskal 算法

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```

1. 将边按权重从小到大排序
2. 初始化并查集 (MAKE-SET)
3. 查询是否已经相连 (FIND-SET)
4. 不会构成 cycle，则合并 (UNION)

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 按权重大小排序边 | | |

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 按权重大小排序边 | QUICK-SORT | |

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 按权重大小排序边 | QUICK-SORT | $O(E \lg E)$ |
| 初始化并查集 | | |

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 按权重大小排序边 | QUICK-SORT | $O(E \lg E)$ |
| 初始化并查集 | MAKE-SET | |

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 按权重大小排序边 | QUICK-SORT | $O(E \lg E)$ |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | | |

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 按权重大小排序边 | QUICK-SORT | $O(E \lg E)$ |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | |

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 按权重大小排序边 | QUICK-SORT | $O(E \lg E)$ |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $O(2E) \cdot T(\text{FINDESET})$ |
| 合并 | | |

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 按权重大小排序边 | QUICK-SORT | $O(E \lg E)$ |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $O(2E) \cdot T(\text{FINDESET})$ |
| 合并 | UNION | |

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 按权重大小排序边 | QUICK-SORT | $O(E \lg E)$ |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $O(2E) \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

# 时间复杂度分析

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 按权重大小排序边 | QUICK-SORT | $O(E \lg E)$ |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $O(2E) \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

- 取决于并查集的实现方式

# 链表实现并查集

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $O(2E) \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

# 链表实现并查集

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $O(2E) \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

**定理 21.1** 使用不相交集合的链表表示和加权合并启发式策略，一个具有 $m$ 个 MAKE-SET、UNION 和 FIND-SET 操作的序列（其中有 $n$ 个是 MAKE-SET 操作）需要的时间为 $O(m+n\lg n)$。

# 链表实现并查集

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $O(2E) \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

**定理 21.1** 使用不相交集合的链表表示和加权合并启发式策略，一个具有 $m$ 个 MAKE-SET、UNION 和 FIND-SET 操作的序列（其中有 $n$ 个是 MAKE-SET 操作）需要的时间为 $O(m+n\lg n)$。

$$E \geq V-1$$

# 链表实现并查集

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $O(2E) \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

**定理 21.1** 使用不相交集合的链表表示和加权合并启发式策略，一个具有 $m$ 个 MAKE-SET、UNION 和 FIND-SET 操作的序列（其中有 $n$ 个是 MAKE-SET 操作）需要的时间为 $O(m+n\lg n)$。

$$E \geq V-1$$

$$V = O(E)$$

# 链表实现并查集

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $O(2E) \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

**定理 21.1** 使用不相交集合的链表表示和加权合并启发式策略，一个具有 $m$ 个 MAKE-SET、UNION 和 FIND-SET 操作的序列（其中有 $n$ 个是 MAKE-SET 操作）需要的时间为 $O(m+n\lg n)$。

$$E \geq V-1$$

$$V = O(E)$$

$$O(E+V\lg V)$$

# 不相交集合森林实现并查集

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $2E \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

# 不相交集合森林实现并查集

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $2E \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

**不相交集合森林实现并查集的时间复杂度**

当同时使用按秩合并和路径压缩时，最坏情况的运行时间为 $O(m\alpha(n))$。

$$O((V+E)\alpha(V))$$

# 不相交集合森林实现并查集

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $2E \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

**不相交集合森林实现并查集的时间复杂度**

当同时使用按秩合并和路径压缩时，最坏情况的运行时间为 $O(m\alpha(n))$。

$$O((V+E)\alpha(V))$$

$$O(E\alpha(V))$$

# 不相交集合森林实现并查集

| 操作 | 方法 | 时间 |
| --- | --- | --- |
| 初始化并查集 | MAKE-SET | $V \cdot T(\text{MAKE-SET})$ |
| 查询是否已经相连 | FIND-SET | $2E \cdot T(\text{FINDESET})$ |
| 合并 | UNION | $(V-1) \cdot T(\text{UNION})$ |

**不相交集合森林实现并查集的时间复杂度**

当同时使用按秩合并和路径压缩时，最坏情况的运行时间为 $O(m\alpha(n))$。

$$O((V+E)\alpha(V))$$

$$O(E\alpha(V))$$

# 并查集实现 Kruskal 算法的时间复杂度

- 并查集
  - 链表实现 $O(E + V \lg V)$
  - 不相交集合森林实现 $O(E\alpha(V))$

# 并查集实现 Kruskal 算法的时间复杂度

- 并查集
  - 链表实现 $O(E + V \lg V)$
  - 不相交集合森林实现 $O(E\alpha(V))$
- 排序 $O(E \lg E)$

# 并查集实现 Kruskal 算法的时间复杂度

- 并查集
  - 链表实现 $O(E + V \lg V)$
  - 不相交集合森林实现 $O(E\alpha(V))$
- 排序 $O(E \lg E)$

# 并查集实现 Kruskal 算法的时间复杂度

- 并查集
  - 链表实现 $O(E + V \lg V)$
  - 不相交集合森林实现 $O(E\alpha(V))$
- 排序 $O(E \lg E)$


$O(E \lg E)$

# Contents

# Prim 算法

选择策略:

# Prim 算法

选择策略：

- 集合 $A$ 中的边总是构成一棵树
- 从任意一个根结点 $r$ 开始
- 每次选择连接已选择的和未选择的点的、权重最小 的一条边
- 终止：选满 $n-1$ 条边

# Prim 算法

选择策略：

- 集合 $A$ 中的边总是构成一棵树
- 从任意一个根结点 $r$ 开始
- 每次选择连接已选择的和未选择的点的、权重最小 的一条边
- 终止：选满 $n-1$ 条边

- 权重最小 $\Rightarrow$ 排序?

# Prim 算法

选择策略:

- 集合 $A$ 中的边总是构成一棵树
- 从任意一个根结点 $r$ 开始

- 最小优先队列

# Prim 算法

选择策略：

- 集合 $A$ 中的边总是构成一棵树
- 从任意一个根结点 $r$ 开始

- 每次选择未选择的点中 与已选择的点中距离最小 的一个点
- 终止：选满 $n$ 个点

- 最小优先队列

# 优先队列实现 Prim 算法

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

$v.key$: 连接 $v$ 和树中结点的所有边中最小边的权重

# 优先队列实现 Prim 算法

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

$v.key$: 连接 $v$ 和树中结点的所有边中最小边的权重

1. 初始化

# 优先队列实现 Prim 算法

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

$v.key$: 连接 $v$ 和树中结点的所有边中最小边的权重

1. 初始化
2. 从未被连通的点中选点

# 优先队列实现 Prim 算法

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

$v.key$: 连接 $v$ 和树中结点的所有边中最小边的权重

1. 初始化
2. 从未被连通的点中选点
3. 更新未被连通的点的通路信息

# 时间复杂度分析

## [TC:2.6] 二叉最小优先队列

# 时间复杂度分析

## [TC:2.6] 二叉最小优先队列

| | |
|---|---|
| BUILD-MAX-HEAP | $O(n)$ |
| EXTRACT-MIN | $O(\lg n)$ |
| DECREASE-KEY | $O(\lg n)$ |

# 时间复杂度分析

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

# 时间复杂度分析

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

| 操作 | 方法 | 时间 |
|---|---|---|

# 时间复杂度分析

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 初始化 | BUILD-MAX-HEAP | $O(V)$ |

# 时间复杂度分析

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 初始化 | BUILD-MAX-HEAP | $O(V)$ |
| 选点 | EXTRACT-MIN | $V \cdot O(\lg V) = O(V \lg V)$ |

# 时间复杂度分析

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 初始化 | BUILD-MAX-HEAP | $O(V)$ |
| 选点 | EXTRACT-MIN | $V \cdot O(\lg V) = O(V \lg V)$ |
| 更新队列信息 | DECREASE-KEY | $O(E) \cdot O(\lg V) = O(E \lg V)$ |

# 时间复杂度分析

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 初始化 | BUILD-MAX-HEAP | $O(V)$ |
| 选点 | EXTRACT-MIN | $V \cdot O(\lg V) = O(V \lg V)$ |
| 更新队列信息 | DECREASE-KEY | $O(E) \cdot O(\lg V) = O(E \lg V)$ |

$$E \geq V - 1$$

# 时间复杂度分析

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

| 操作 | 方法 | 时间 |
|---|---|---|
| 初始化 | BUILD-MAX-HEAP | $O(V)$ |
| 选点 | EXTRACT-MIN | $V \cdot O(\lg V) = O(V \lg V)$ |
| 更新队列信息 | DECREASE-KEY | $O(E) \cdot O(\lg V) = O(E \lg V)$ |

$$E \geq V - 1$$

$$O(V) + O(V \lg V) + O(E \lg V) = O(E \lg V)$$

# Kruskal v.s. Prim

| Kruskal | Prim |
| --- | --- |
| $O(E \lg E)$ | $O(E \lg V)$ |
| $E$ 较小 | $V$ 较小 |

# Kruskal v.s. Prim

| Kruskal | Prim |
| --- | --- |
| $O(E \lg E)$ | $O(E \lg V)$ |
| $E$ 较小 | $V$ 较小 |

$$E \leq \frac{V(V-1)}{2} \leq V^2$$

$$\lg E \leq 2 \lg V$$

$$O(\lg E) = O(\lg V)$$

| Kruskal | Prim |
| --- | --- |
| $O(E \lg V)$ | $O(E \lg V)$ |

| Kruskal | Prim |
| --- | --- |
| $O(E \lg V)$ | $O(E \lg V)$ |

**两种最小生成树算法的时间复杂度比较**

从渐近意义上来说，Kruskal 算法与 Prim 算法的运行时间相同。


THANK YOU