堆的可持久化和 k 短路

俞鼎力

Leo.DingliYu@gmail.com

绍兴市第一中学

2014 年 2 月 8 日

目录

Outline

定义

- 这里，k 短路是要求在一张有向带权图 G 中，从起点 s 到终点 t 的可重复经过同一点的不严格递增的第 k 短路的长度。

定义

- 这里，k 短路是要求在一张有向带权图 G 中，从起点 s 到终点 t 的可重复经过同一点的不严格递增的第 k 短路的长度。
- 由于之后的算法的一些限制，我们先默认边权非负。

定义

- 这里，k 短路是要求在一张有向带权图 G 中，从起点 s 到终点 t 的可重复经过同一点的不严格递增的第 k 短路的长度。
- 由于之后的算法的一些限制，我们先默认边权非负。
- 对于一条边 e，定义它的边权（长度）为 $\ell(e)$，定义它的起始点为 $head(e)$、终止点为 $tail(e)$。

定义

- 这里，k 短路是要求在一张有向带权图 G 中，从起点 s 到终点 t 的可重复经过同一点的不严格递增的第 k 短路的长度。
- 由于之后的算法的一些限制，我们先默认边权非负。
- 对于一条边 e，定义它的边权（长度）为 $\ell(e)$，定义它的起始点为 $head(e)$、终止点为 $tail(e)$。
- 用 $d(u, v)$ 表示 u 到 v 的最短距离。

Outline

A* 算法

- 众所周知，解决 k 短路问题的经典算法是 A*，算法核心在于对当前状态的估价。

A* 算法

- 众所周知，解决 k 短路问题的经典算法是 A*，算法核心在于对当前状态的估价。
- 使用 A* 求 k 短路的主要流程是这样的：

A* 算法

- 众所周知，解决 k 短路问题的经典算法是 A*，算法核心在于对当前状态的估价。
- 使用 A* 求 k 短路的主要流程是这样的：
- 将所有边反向并使用最短路算法，得到任意一点 v 到终点 t 的最短路径 $d(v, t)$。

A* 算法

- 众所周知，解决 k 短路问题的经典算法是 A*，算法核心在于对当前状态的估价。
- 使用 A* 求 k 短路的主要流程是这样的：
- 将所有边反向并使用最短路算法，得到任意一点 v 到终点 t 的最短路径 $d(v,t)$。
- 一开始状态位于点 s，假设经过一段时间走到了点 v，那么定义当时的估价值为已经走过的路径长度 $+d(v,t)$。

A* 算法

- 众所周知，解决 k 短路问题的经典算法是 A*，算法核心在于对当前状态的估价。
- 使用 A* 求 k 短路的主要流程是这样的：
- 将所有边反向并使用最短路算法，得到任意一点 v 到终点 t 的最短路径 $d(v,t)$。
- 一开始状态位于点 s，假设经过一段时间走到了点 v，那么定义当时的估价值为已经走过的路径长度 $+d(v,t)$。
- 用一个堆 Q 来维护所有状态的估价值，每次从 Q 中取出估价最小的状态，枚举其下一步走的边，将新的状态及其估价值放入 Q 中。

A* 算法

- 众所周知，解决 k 短路问题的经典算法是 A*，算法核心在于对当前状态的估价。
- 使用 A* 求 k 短路的主要流程是这样的：
- 将所有边反向并使用最短路算法，得到任意一点 v 到终点 t 的最短路径 $d(v, t)$。
- 一开始状态位于点 s，假设经过一段时间走到了点 v，那么定义当时的估价值为已经走过的路径长度 $+d(v, t)$。
- 用一个堆 Q 来维护所有状态的估价值，每次从 Q 中取出估价最小的状态，枚举其下一步走的边，将新的状态及其估价值放入 Q 中。
- 直到找到 k 条 s 到 t 的路径为止。

Outline

数据规模

- 没听懂？

数据规模

- 没听懂？
- 没关系，等下讲的和这个毫无关系。你只要知道如果这张图恰好是一个 n 元环的话，A* 算法的复杂度是 $O(nk)$ 的。

数据规模

- 没听懂？
- 没关系，等下讲的和这个毫无关系。你只要知道如果这张图恰好是一个 n 元环的话，A* 算法的复杂度是 $O(nk)$ 的。
- 有同学想知道 $k = 10000$ 怎么做么？

数据规模

- 没听懂？
- 没关系，等下讲的和这个毫无关系。你只要知道如果这张图恰好是一个 n 元环的话，A* 算法的复杂度是 $O(nk)$ 的。
- 有同学想知道 $k = 10000$ 怎么做么？
- 有同学想知道 $k = 100000$ 怎么做么？

数据规模

- 没听懂？
- 没关系，等下讲的和这个毫无关系。你只要知道如果这张图恰好是一个 n 元环的话，A* 算法的复杂度是 $O(nk)$ 的。
- 有同学想知道 $k = 10000$ 怎么做么？
- 有同学想知道 $k = 100000$ 怎么做么？
- 有同学想知道 $k = 1000000$ 怎么做么？

Outline

k 短路
- 定义
- A* 算法
- 数据规模
- 问题转化
- 求解与优化

堆的可持久化
- 什么样的堆可能可以可持久化
- Brodal Queue
- 二叉堆
- 二项堆

问题转化

- 首先定义 T 为 G 中以 t 为终点的最短路径构成的最短路径树。

$$
\begin{tikzcd}
A \arrow[r, "2"] \arrow[d, "13"] & B \arrow[r, "20"] \arrow[d, "27"] & C \arrow[r, "14"] \arrow[d, "14"] & D \arrow[d, "15"] \\
E \arrow[r, "9"] \arrow[d, "15"] & F \arrow[r, "10"] \arrow[d, "20"] & G \arrow[r, "25"] \arrow[d, "12"] & H \arrow[d, "7"] \\
I \arrow[r, "18"] & J \arrow[r, "8"] & K \arrow[r, "11"] & L
\end{tikzcd}
\qquad
\begin{tikzcd}
55 \arrow[r, dotted, "3"] \arrow[d] & 56 \arrow[r] \arrow[d, dotted, "4"] & 36 \arrow[r] \arrow[d, dotted, "1"] & 22 \arrow[d] \\
42 \arrow[r] \arrow[d, dotted, "10"] & 33 \arrow[r] \arrow[d, dotted, "6"] & 23 \arrow[r, dotted, "9"] \arrow[d] & 7 \arrow[d] \\
37 \arrow[r] & 19 \arrow[r] & 11 \arrow[r] & 0
\end{tikzcd}
$$

问题转化

- 首先定义 T 为 G 中以 t 为终点的最短路径构成的最短路径树。
- 对于一条边 $e \in G$，定义：
  $\delta(e) = \ell(e) + d(tail(e), t) - d(head(e), t)$

$$
\begin{tikzcd}
A \arrow[r, "2"] \arrow[d, "13"] & B \arrow[r, "20"] \arrow[d, "27"] & C \arrow[r, "14"] \arrow[d, "14"] & D \arrow[d, "15"] \\
E \arrow[r, "9"] \arrow[d, "15"] & F \arrow[r, "10"] \arrow[d, "20"] & G \arrow[r, "25"] \arrow[d, "12"] & H \arrow[d, "7"] \\
I \arrow[r, "18"] & J \arrow[r, "8"] & K \arrow[r, "11"] & L
\end{tikzcd}
\qquad
\begin{tikzcd}
55 \arrow[r, dotted, "3"] \arrow[d] & 56 \arrow[r] \arrow[d, dotted, "4"] & 36 \arrow[r] \arrow[d, dotted, "1"] & 22 \arrow[d] \\
42 \arrow[r] \arrow[d, dotted, "10"] & 33 \arrow[r] \arrow[d, dotted, "6"] & 23 \arrow[r, dotted, "9"] \arrow[d] & 7 \arrow[d] \\
37 \arrow[r] & 19 \arrow[r] & 11 \arrow[r] & 0
\end{tikzcd}
$$

问题转化

- 首先定义 T 为 G 中以 t 为终点的最短路径构成的最短路径树。
- 对于一条边 $e \in G$，定义：
  $\delta(e) = \ell(e) + d(tail(e), t) - d(head(e), t)$
- 那么有：$\forall e \in G, \delta(e) \geq 0$、$\forall e \in T, \delta(e) = 0$

$$
\begin{tikzcd}
A \arrow[r, "2"] \arrow[d, "13"] & B \arrow[r, "20"] \arrow[d, "27"] & C \arrow[r, "14"] \arrow[d, "14"] & D \arrow[d, "15"] \\
E \arrow[r, "9"] \arrow[d, "15"] & F \arrow[r, "10"] \arrow[d, "20"] & G \arrow[r, "25"] \arrow[d, "12"] & H \arrow[d, "7"] \\
I \arrow[r, "18"] & J \arrow[r, "8"] & K \arrow[r, "11"] & L
\end{tikzcd}
\qquad
\begin{tikzcd}
55 \arrow[r, dotted, "3"] \arrow[d] & 56 \arrow[r] \arrow[d, dotted, "4"] & 36 \arrow[r] \arrow[d, dotted, "1"] & 22 \arrow[d] \\
42 \arrow[r] \arrow[d, dotted, "10"] & 33 \arrow[r] \arrow[d, dotted, "6"] & 23 \arrow[r, dotted, "9"] \arrow[d] & 7 \arrow[d] \\
37 \arrow[r] & 19 \arrow[r] & 11 \arrow[r] & 0
\end{tikzcd}
$$

问题转化

- 但是最短路径树可能并不一定是一棵树，在此我们要将 T 严格定义为一棵树

问题转化

- 但是最短路径树可能并不一定是一棵树，在此我们要将 T 严格定义为一棵树
- 对于一个节点 $v \neq t$，定义 $next_T(v)$ 为 $v \to t$ 的最短路径（不唯一则任选一条）上 v 的后继节点。

问题转化

- 但是最短路径树可能并不一定是一棵树，在此我们要将 T 严格定义为一棵树
- 对于一个节点 $v \neq t$，定义 $next_T(v)$ 为 $v \to t$ 的最短路径（不唯一则任选一条）上 v 的后继节点。
- 那么 G 中所有的点就以 $next_T$ 形成了一个严格的树结构 T。

问题转化

- 但是最短路径树可能并不一定是一棵树，在此我们要将 T 严格定义为一棵树
- 对于一个节点 $v \neq t$，定义 $next_T(v)$ 为 $v \to t$ 的最短路径（不唯一则任选一条）上 v 的后继节点。
- 那么 G 中所有的点就以 $next_T$ 形成了一个严格的树结构 T。
- 对于一条从 s 到 t 的路径 p，去掉 p 中所有在 T 中的边，将其定义为 $sidetracks(p)$。

问题转化

- 我们有以下三条性质：

$$
\begin{tikzcd}
A \arrow[r, "3"] & B \arrow[d, "4"] & C \arrow[d, "1"] & D \\
E \arrow[d, "10"] & F \arrow[d, "6"] & G \arrow[r, "9"] & H \\
I & J & K & L
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "3"] & B \arrow[d, "4"] & C & D \\
E & F \arrow[r, dotted] & G \arrow[r, "9"] \arrow[d, dotted] & H \arrow[d, dotted] \\
I & J & K \arrow[r, dotted] & L
\end{tikzcd}
$$

问题转化

- 我们有以下三条性质：

1 对于 $sidetracks(p)$ 中任意一条边 e，都有 $e \in G-T$；对于 $sidetracks(p)$ 中任意相邻的两条边 e,f，都满足 $head(f)$ 是 $tail(e)$ 在 T 上的祖先或 $head(f)=tail(e)$。

$$\begin{tikzcd}
A \arrow[r, "3"] & B \arrow[d, "4"] & C \arrow[d, "1"] & D \\
E \arrow[d, "10"] & F \arrow[d, "6"] & G \arrow[r, "9"] & H \\
I & J & K & L
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "3"] & B \arrow[d, "4"] & C & D \\
E & F \arrow[r, dotted] & G \arrow[r, "9"] \arrow[d, dotted] & H \arrow[d, dotted] \\
I & J & K \arrow[r, dotted] & L
\end{tikzcd}$$

问题转化

$$2 \quad l(p) = d(s,t) + \sum_{e \in sidetracks(p)} \delta(e) = d(s,t) + \sum_{e \in p} \delta(e)$$

$$\begin{tikzcd}
55 \arrow[r, dotted, "3"] \arrow[d] & 56 \arrow[r] \arrow[d, dotted, "4"] & 36 \arrow[r] \arrow[d, dotted, "1"] & 22 \arrow[d] \\
42 \arrow[r] \arrow[d, dotted, "10"] & 33 \arrow[r] \arrow[d, dotted, "6"] & 23 \arrow[r, dotted, "9"] \arrow[d] & 7 \arrow[d] \\
37 \arrow[r] & 19 \arrow[r] & 11 \arrow[r] & 0
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "3"] & B \arrow[d, "4"] & C & D \\
E & F \arrow[r, dotted] & G \arrow[r, "9"] \arrow[d, dotted] & H \arrow[d, dotted] \\
I & J & K \arrow[r, dotted] & L
\end{tikzcd}$$

问题转化

2 $l(p) = d(s,t) + \sum_{e \in sidetracks(p)} \delta(e) = d(s,t) + \sum_{e \in p} \delta(e)$

3 对于一个满足性质 1 的边的序列 q，有且仅有一条 s 到 t 的路径 p 满足 $sidetracks(p) = q$。

$$\begin{tikzcd}
55 \arrow[r, dotted, "3"] \arrow[d] & 56 \arrow[r] \arrow[d, dotted, "4"] & 36 \arrow[r] \arrow[d, dotted, "1"] & 22 \arrow[d] \\
42 \arrow[r] \arrow[d, dotted, "10"] & 33 \arrow[r] \arrow[d, dotted, "6"] & 23 \arrow[r, dotted, "9"] \arrow[d] & 7 \arrow[d] \\
37 \arrow[r] & 19 \arrow[r] & 11 \arrow[r] & 0
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "3"] & B \arrow[d, "4"] & C & D \\
E & F \arrow[r, dotted] & G \arrow[r, "9"] \arrow[d, dotted] & H \arrow[d, dotted] \\
I & J & K \arrow[r, dotted] & L
\end{tikzcd}$$

问题转化

- 至此，我们将问题转化为：求第 k 小的满足性质 1 的边的序列。

问题转化

- 至此，我们将问题转化为：求第 k 小的满足性质 1 的边的序列。
- 即，序列中任意一条边 e 都有 $e \in G - T$；任意相邻的两条边 e, f，都满足 $head(f)$ 是 $tail(e)$ 在 T 上的祖先或 $head(f) = tail(e)$。

问题转化

- 至此，我们将问题转化为：求第 k 小的满足性质 1 的边的序列。
- 即，序列中任意一条边 e 都有 $e \in G - T$；任意相邻的两条边 e, f，都满足 $head(f)$ 是 $tail(e)$ 在 T 上的祖先或 $head(f) = tail(e)$。
- 其中，序列 q 的权值定义为 $w(q) = \sum_{e \in q} \delta(e)$。

Outline

算法一

- 维护一个权值从小到大的优先队列，最初只有一个空序列。

算法一

- 维护一个权值从小到大的优先队列，最初只有一个空序列。
- 每次从队头取出一个序列 q，令 q 最后一条边的 $tail$ 为 v（若是空序列则 $v = s$）。

算法一

- 维护一个权值从小到大的优先队列，最初只有一个空序列。
- 每次从队头取出一个序列 q，令 q 最后一条边的 $tail$ 为 v（若是空序列则 $v = s$）。
- 在 q 之后加入一条边 e 得到新序列 q'（$head(e)$ 在 T 中 v 到 t 的路径上且 $e \in G - T$）。

算法一

- 维护一个权值从小到大的优先队列，最初只有一个空序列。
- 每次从队头取出一个序列 q，令 q 最后一条边的 $tail$ 为 v（若是空序列则 $v = s$）。
- 在 q 之后加入一条边 e 得到新序列 q'（$head(e)$ 在 T 中 v 到 t 的路径上且 $e \in G - T$）。
- $w(q') = w(q) + \delta(e)$，将 q' 加入优先队列。

算法一

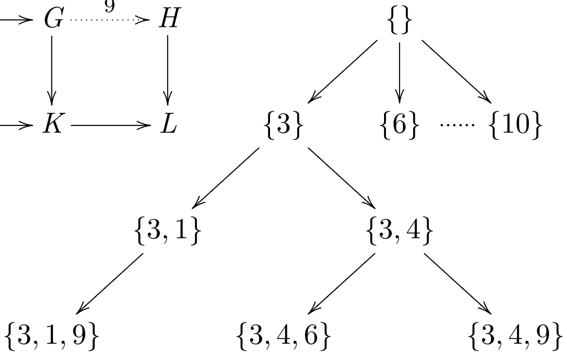
- 维护一个权值从小到大的优先队列，最初只有一个空序列。
- 每次从队头取出一个序列 q，令 q 最后一条边的 $tail$ 为 v（若是空序列则 $v = s$）。
- 在 q 之后加入一条边 e 得到新序列 q'（$head(e)$ 在 T 中 v 到 t 的路径上且 $e \in G - T$）。
- $w(q') = w(q) + \delta(e)$，将 q' 加入优先队列。
- 重复 k 次。

算法一

- 维护一个权值从小到大的优先队列，最初只有一个空序列。
- 每次从队头取出一个序列 q，令 q 最后一条边的 $tail$ 为 v（若是空序列则 $v = s$）。
- 在 q 之后加入一条边 e 得到新序列 q'（$head(e)$ 在 T 中 v 到 t 的路径上且 $e \in G - T$）。
- $w(q') = w(q) + \delta(e)$，将 q' 加入优先队列。
- 重复 k 次。
- 如果默认第一步的最短路算法复杂度为 $O(n\log n + m)$，则总复杂度为 $O(n\log n + km(\log k + \log m))$。

算法一

$$\begin{tikzcd}
A \arrow[r, dotted, "3"] \arrow[d] & B \arrow[r] \arrow[d, dotted, "4"] & C \arrow[r] \arrow[d, dotted, "1"] & D \arrow[d] \\
E \arrow[r] \arrow[d, dotted, "10"] & F \arrow[r] \arrow[d, dotted, "6"] & G \arrow[r, dotted, "9"] \arrow[d] & H \arrow[d] \\
I \arrow[r] & J \arrow[r] & K \arrow[r] & L
\end{tikzcd}$$



算法二

- 对于点 v 建立一张有序表 $g(v)$，按权值从小到大记录所有在 $G-T$ 中且满足 $head(e)$ 在 T 中 v 到 t 的路径上的边 e。

算法二

- 对于点 v 建立一张有序表 $g(v)$，按权值从小到大记录所有在 $G-T$ 中且满足 $head(e)$ 在 T 中 v 到 t 的路径上的边 e。
- 每次对于从队头取出的序列 q，令其最后一条边为 e、$v=tail(e)$、倒数第二条边的 $tail$ 为 u。

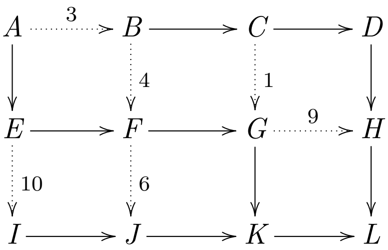
算法二

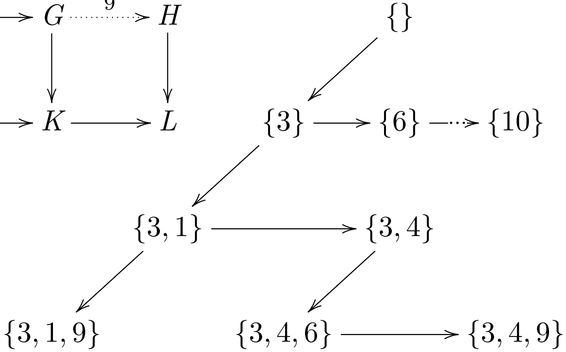
- 对于点 v 建立一张有序表 $g(v)$，按权值从小到大记录所有在 $G-T$ 中且满足 $head(e)$ 在 T 中 v 到 t 的路径上的边 e。
- 每次对于从队头取出的序列 q，令其最后一条边为 e、$v=tail(e)$、倒数第二条边的 $tail$ 为 u。
- 我们将 e 替换为 $g(u)$ 中 e 的后一条边或者在 q 中新加入 $g(v)$ 中的第一条边得到新序列 q'。

算法二

- 对于点 v 建立一张有序表 $g(v)$，按权值从小到大记录所有在 $G-T$ 中且满足 $head(e)$ 在 T 中 v 到 t 的路径上的边 e。
- 每次对于从队头取出的序列 q，令其最后一条边为 e、$v = tail(e)$、倒数第二条边的 $tail$ 为 u。
- 我们将 e 替换为 $g(u)$ 中 e 的后一条边或者在 q 中新加入 $g(v)$ 中的第一条边得到新序列 q'。
- 复杂度为 $O(n\log n + nm\log m + k\log k)$。

算法二





算法三

- 上一个算法的瓶颈在于对每个点建立有序表。

算法三

- 上一个算法的瓶颈在于对每个点建立有序表。
- 我们注意到 $g(v)$ 和 $g(next_T(v))$ 是有大部分相同的。

算法三

- 上一个算法的瓶颈在于对每个点建立有序表。
- 我们注意到 $g(v)$ 和 $g(next_T(v))$ 是有大部分相同的。
- 事实上我们是在 $g(next_T(v))$ 中添上所有在 $G-T$ 的由 v 连出去的边来求得 $g(v)$ 的。

算法三

- 上一个算法的瓶颈在于对每个点建立有序表。
- 我们注意到 $g(v)$ 和 $g(next_T(v))$ 是有大部分相同的。
- 事实上我们是在 $g(next_T(v))$ 中添上所有在 $G-T$ 的由 v 连出去的边来求得 $g(v)$ 的。
- 但是有碍于有序表的添加复杂度，我们不能有效得来求得 g。

算法三

- 如果我们对于点 v 不建立 $g(v)$ 转而建立一个堆 $H(v)$，并通过在 $H(next_T(v))$ 中**可持久地**加入所有在 $G-T$ 的由 v 连出去的边来得到 $H(v)$，那么我们就有能解决上面的问题。

算法三

- 如果我们对于点 v 不建立 $g(v)$ 转而建立一个堆 $H(v)$，并通过在 $H(next_T(v))$ 中**可持久地**加入所有在 $G-T$ 的由 v 连出去的边来得到 $H(v)$，那么我们就有能解决上面的问题。
- 而对于上一算法中“将 e 替换为 $g(u)$ 中 e 的后一条边”这一步，我们直接将 e 在 $H(u)$ 中的两个儿子替换 e 即可。

算法三

- 如果我们对于点 v 不建立 $g(v)$ 转而建立一个堆 $H(v)$，并通过在 $H(next_T(v))$ 中**可持久地**加入所有在 $G-T$ 的由 v 连出去的边来得到 $H(v)$，那么我们就有能解决上面的问题。
- 而对于上一算法中 “将 e 替换为 $g(u)$ 中 e 的后一条边” 这一步，我们直接将 e 在 $H(u)$ 中的两个儿子替换 e 即可。
- 复杂度 $O(n\log n+m\log m+k\log k)$。

算法四

- 能不能再优化呢？

算法四

- 能不能再优化呢？
- 对于点 v 用堆 $h(v)$ 维护所有在 $G-T$ 的由 v 连出去的边。

算法四

- 能不能再优化呢？
- 对于点 v 用堆 $h(v)$ 维护所有在 $G-T$ 的由 v 连出去的边。
- 并将 $h(v)$ 中最小元素的权值作为 $h(v)$ 的权值。

算法四

- 能不能再优化呢？
- 对于点 v 用堆 $h(v)$ 维护所有在 $G-T$ 的由 v 连出去的边。
- 并将 $h(v)$ 中最小元素的权值作为 $h(v)$ 的权值。
- 把 $h(v)$ 当作一个节点可持久地插入 $H(next_T(v))$ 得到堆的堆 $H(v)$。

算法四

- 能不能再优化呢？
- 对于点 v 用堆 $h(v)$ 维护所有在 $G - T$ 的由 v 连出去的边。
- 并将 $h(v)$ 中最小元素的权值作为 $h(v)$ 的权值。
- 把 $h(v)$ 当作一个节点可持久地插入 $H(next_T(v))$ 得到堆的堆 $H(v)$。
- 建 $h(v)$ 的时间为 $O(m)$，建 $H(v)$ 变为 $O(n \log n)$。

算法四

- 能不能再优化呢？
- 对于点 v 用堆 $h(v)$ 维护所有在 $G-T$ 的由 v 连出去的边。
- 并将 $h(v)$ 中最小元素的权值作为 $h(v)$ 的权值。
- 把 $h(v)$ 当作一个节点可持久地插入 $H(next_T(v))$ 得到堆的堆 $H(v)$。
- 建 $h(v)$ 的时间为 $O(m)$，建 $H(v)$ 变为 $O(n\log n)$。
- 总复杂度为 $O(n\log n+m+k\log k)$。

Outline

什么样的堆可能可以可持久化

- 众所周知，带均摊复杂度的数据结构是不能完全可持久化的。

什么样的堆可能可以可持久化

- 众所周知，带均摊复杂度的数据结构是不能完全可持久化的。
- 例如：左偏树，斜堆，斐波那契堆，配对堆

什么样的堆可能可以可持久化

- 众所周知，带均摊复杂度的数据结构是不能完全可持久化的。
- 例如：左偏树，斜堆，斐波那契堆，配对堆
- 那剩下还有什么？

什么样的堆可能可以可持久化

- 众所周知，带均摊复杂度的数据结构是不能完全可持久化的。
- 例如：左偏树，斜堆，斐波那契堆，配对堆
- 那剩下还有什么？
- 二叉堆[3]，二项堆[4]，Brodal queue[2]

什么样的堆可能可以可持久化

- 众所周知，带均摊复杂度的数据结构是不能完全可持久化的。
- 例如：左偏树，斜堆，斐波那契堆，配对堆
- 那剩下还有什么？
- 二叉堆[3]，二项堆[4]，Brodal queue[2]
- 堆的基本操作：INSERT, FINDMIN, DELETEMIN, DECREASEKEY, [MERGE(MELD)]

Outline

Brodal Queue

- 这是什么？

Brodal Queue

- 这是什么?
- http://en.wikipedia.org/wiki/Brodal_queue

Brodal Queue

- 这是什么？
- http://en.wikipedia.org/wiki/Brodal_queue
- 插入、查询最小值、合并、减小某个元素的值都是最坏复杂度 $O(1)$ 的，只有删除操作是 $O(\log n)$ 的。

Brodal Queue

- 这是什么？
- http://en.wikipedia.org/wiki/Brodal_queue
- 插入、查询最小值、合并、减小某个元素的值都是最坏复杂度 $O(1)$ 的，只有删除操作是 $O(\log n)$ 的。
- 并且它能够可持久化。

Brodal Queue

- 这是什么？
- http://en.wikipedia.org/wiki/Brodal_queue
- 插入、查询最小值、合并、减小某个元素的值都是最坏复杂度 $O(1)$ 的，只有删除操作是 $O(\log n)$ 的。
- 并且它能够可持久化。
- 很可惜，它的发明者 Brodal 也不得不承认它 "quite complicated" and "[not] applicable in practice."。

Outline

二叉堆

- 二叉堆是最基础的堆，但是要将其可持久化也是需要一些技巧的。

二叉堆

- 二叉堆是最基础的堆，但是要将其可持久化也是需要一些技巧的。
- 在可持久化数据结构中，如果一个节点被修改，那么会产生了一个新的节点保留修改后的信息，原来的节点不变。

二叉堆

- 二叉堆是最基础的堆，但是要将其可持久化也是需要一些技巧的。
- 在可持久化数据结构中，如果一个节点被修改，那么会产生了一个新的节点保留修改后的信息，原来的节点不变。
- 所以，如果有一个节点中记录了被修改的节点的地址，那么这个节点也需要修改。

二叉堆

- 二叉堆是最基础的堆，但是要将其可持久化也是需要一些技巧的。
- 在可持久化数据结构中，如果一个节点被修改，那么会产生了一个新的节点保留修改后的信息，原来的节点不变。
- 所以，如果有一个节点中记录了被修改的节点的地址，那么这个节点也需要修改。
- 所以，如果记录左右儿子，那么一个节点被修改后，需要连带他的父亲被修改；

二叉堆

- 二叉堆是最基础的堆，但是要将其可持久化也是需要一些技巧的。
- 在可持久化数据结构中，如果一个节点被修改，那么会产生了一个新的节点保留修改后的信息，原来的节点不变。
- 所以，如果有一个节点中记录了被修改的节点的地址，那么这个节点也需要修改。
- 所以，如果记录左右儿子，那么一个节点被修改后，需要连带他的父亲被修改；
- 如果记录父亲，那么一个节点被修改后，整棵子树都需要被修改。

二叉堆

- 后一条是我们不能接受的，所以我们不能记录父亲，只能记录左右儿子。

二叉堆

- 后一条是我们不能接受的，所以我们不能记录父亲，只能记录左右儿子。
- 不知道父亲那应该如何“连带他的父亲被修改”？

二叉堆

- 后一条是我们不能接受的，所以我们不能记录父亲，只能记录左右儿子。
- 不知道父亲那应该如何“连带他的父亲被修改”？
- 二叉堆有一个优异的性质，它能对每个位置的节点标号！

二叉堆

- 后一条是我们不能接受的，所以我们不能记录父亲，只能记录左右儿子。
- 不知道父亲那应该如何“连带他的父亲被修改”?
- 二叉堆有一个优异的性质，它能对每个位置的节点标号!
- 如果知道一个节点在位置 i，那么我们就能知道它是它父亲的左儿子还是右儿子，

二叉堆

- 后一条是我们不能接受的，所以我们不能记录父亲，只能记录左右儿子。
- 不知道父亲那应该如何“连带他的父亲被修改”？
- 二叉堆有一个优异的性质，它能对每个位置的节点标号！
- 如果知道一个节点在位置 i，那么我们就能知道它是它父亲的左儿子还是右儿子，
- 知道它父亲是它祖父的左儿子还是右儿子……

二叉堆

- 后一条是我们不能接受的，所以我们不能记录父亲，只能记录左右儿子。
- 不知道父亲那应该如何“连带他的父亲被修改”？
- 二叉堆有一个优异的性质，它能对每个位置的节点标号！
- 如果知道一个节点在位置 i，那么我们就能知道它是它父亲的左儿子还是右儿子，
- 知道它父亲是它祖父的左儿子还是右儿子……
- 所以，只要记录根是什么，我们就可以知道根到该节点的整条路径。

二叉堆

- 还有问题

二叉堆

- 还有问题
- 假设我们需要修改某个时刻某个点的值，我们还不知道其当时对应的堆中的节点。

二叉堆

- 还有问题
- 假设我们需要修改某个时刻某个点的值，我们还不知道其当时对应的堆中的节点。
- 再维护一个可持久化数据结构，来记录每个点当时对应的堆中的节点。

二叉堆

- 还有问题
- 假设我们需要修改某个时刻某个点的值，我们还不知道其当时对应的堆中的节点。
- 再维护一个可持久化数据结构，来记录每个点当时对应的堆中的节点。
- 但是复杂度多了一个 log。

二叉堆

- 还有问题
- 假设我们需要修改某个时刻某个点的值，我们还不知道其当时对应的堆中的节点。
- 再维护一个可持久化数据结构，来记录每个点当时对应的堆中的节点。
- 但是复杂度多了一个 log。
- 如果可以离线的话，可以直接对每个点维护一个栈来解决。

二叉堆

- 总的来说，每次函数式二叉堆的操作只会影响最多两条链上的节点。

二叉堆

- 总的来说，每次函数式二叉堆的操作只会影响最多两条链上的节点。
- 节点上记录左儿子、右儿子、在二叉堆中的位置、节点原本的编号。

二叉堆

- 总的来说，每次函数式二叉堆的操作只会影响最多两条链上的节点。
- 节点上记录左儿子、右儿子、在二叉堆中的位置、节点原本的编号。
- 一个节点修改过后，我需要将其放入对应的数据结构中，以便之后修改它的时候用。

二叉堆

- 总的来说，每次函数式二叉堆的操作只会影响最多两条链上的节点。
- 节点上记录左儿子、右儿子、在二叉堆中的位置、节点原本的编号。
- 一个节点修改过后，我需要将其放入对应的数据结构中，以便之后修改它的时候用。
- 记录每个时刻的根节点也不在话下。

二叉堆

- 总的来说，每次函数式二叉堆的操作只会影响最多两条链上的节点。
- 节点上记录左儿子、右儿子、在二叉堆中的位置、节点原本的编号。
- 一个节点修改过后，我需要将其放入对应的数据结构中，以便之后修改它的时候用。
- 记录每个时刻的根节点也不在话下。
- 所以操作都可以非递归实现。

Outline

二项堆

- 二项堆是可并堆，意义很大，但是可持久化的难度大了不少。

二项堆

- 二项堆是可并堆，意义很大，但是可持久化的难度大了不少。
- 但是如果不需要使用 DecreaseKey，它的可持久化也是不难的。

二项堆

- 二项堆是可并堆，意义很大，但是可持久化的难度大了不少。
- 但是如果不需要使用 DecreaseKey，它的可持久化也是不难的。
- 为什么说不需要使用 DecreaseKey呢？

二项堆

- 二项堆是可并堆，意义很大，但是可持久化的难度大了不少。
- 但是如果不需要使用 DecreaseKey，它的可持久化也是不难的。
- 为什么说不需要使用 DecreaseKey呢?
- 因为如果是函数式的可并堆，那么一个节点的 t_1 时刻版本和 t_2 时刻的版本可能在某个时刻同时出现，所以修改某一特定编号的节点的值是没有意义的。

二项堆

- 二项堆是可并堆，意义很大，但是可持久化的难度大了不少。
- 但是如果不需要使用 DecreaseKey，它的可持久化也是不难的。
- 为什么说不需要使用 DecreaseKey呢?
- 因为如果是函数式的可并堆，那么一个节点的 t_1 时刻版本和 t_2 时刻的版本可能在某个时刻同时出现，所以修改某一特定编号的节点的值是没有意义的。
- 去年冬令营上王启圣讲解过二项堆，这里不详细讲，但是略微提一下。

二项树

- 如果用 T_i 表示树高为 i 的二项树的形状，那么：

二项树

- 如果用 T_i 表示树高为 i 的二项树的形状，那么：
- T_0 为只有一个节点的树。

二项树

- 如果用 T_i 表示树高为 i 的二项树的形状，那么：
- T_0 为只有一个节点的树。
- T_i 是在 T_{i-1} 的根节点上再加一棵形状为 T_{i-1} 的子树。

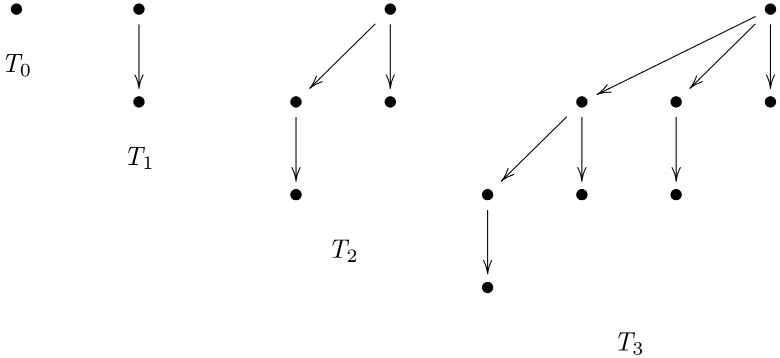
二项树

- 如果用 T_i 表示树高为 i 的二项树的形状，那么：
- T_0 为只有一个节点的树。
- T_i 是在 T_{i-1} 的根节点上再加一棵形状为 T_{i-1} 的子树。
- 所以，我们可以快速地合并两棵高度相同的二项树。

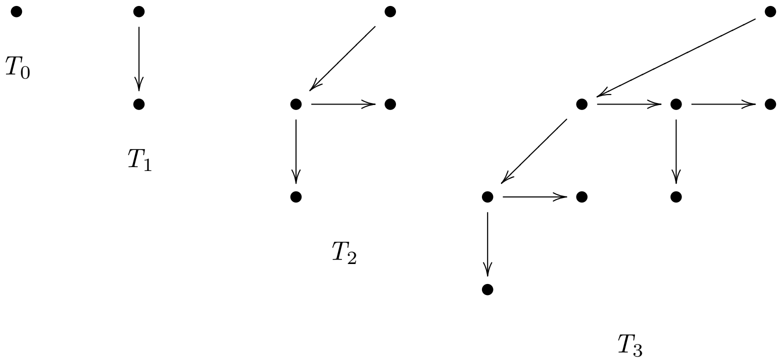
二项树

- 如果用 T_i 表示树高为 i 的二项树的形状，那么：
- T_0 为只有一个节点的树。
- T_i 是在 T_{i-1} 的根节点上再加一棵形状为 T_{i-1} 的子树。
- 所以，我们可以快速地合并两棵高度相同的二项树。
- 注意：我们不需要记录每个点的所有儿子，只需要记录比它高度恰好小 1 的兄弟和它最大的儿子。

二项树



二项树



二项堆

- 二项堆是若干棵大小不同的满足堆性质的二项树的集合。

二项堆

- 二项堆是若干棵大小不同的满足堆性质的二项树的集合。
- 所以如果要询问最小值的话，直接取所有二项树的根的最小值即可。

二项堆

- 二项堆是若干棵大小不同的满足堆性质的二项树的集合。
- 所以如果要询问最小值的话，直接取所有二项树的根的最小值即可。
- 二项堆的关键操作在于合并。

二项堆

- 二项堆是若干棵大小不同的满足堆性质的二项树的集合。
- 所以如果要询问最小值的话，直接取所有二项树的根的最小值即可。
- 二项堆的关键操作在于合并。
- 因为一个点也可以是一个二项堆，所以插入操作可以用合并来解决。

二项堆

- 二项堆是若干棵大小不同的满足堆性质的二项树的集合。
- 所以如果要询问最小值的话，直接取所有二项树的根的最小值即可。
- 二项堆的关键操作在于合并。
- 因为一个点也可以是一个二项堆，所以插入操作可以用合并来解决。
- 因为将一棵二项树的根删去，其所有子树也是一个二项堆，所以删除最小值操作也可以用合并来解决。

二项堆

- 二项堆是若干棵大小不同的满足堆性质的二项树的集合。
- 所以如果要询问最小值的话，直接取所有二项树的根的最小值即可。
- 二项堆的关键操作在于合并。
- 因为一个点也可以是一个二项堆，所以插入操作可以用合并来解决。
- 因为将一棵二项树的根删去，其所有子树也是一个二项堆，所以删除最小值操作也可以用合并来解决。
- 删去二项树的根的时候，有什么改变了？

合并

- 两个二项堆的合并就是两组二项树合并。

合并

- 两个二项堆的合并就是两组二项树合并。
- 如果没有两个相同高度的二项树，那么合并就结束了。

合并

- 两个二项堆的合并就是两组二项树合并。
- 如果没有两个相同高度的二项树，那么合并就结束了。
- 如果有两个高度相同的二项树，我们就将其合并得到高度大一的二项树。

合并

- 两个二项堆的合并就是两组二项树合并。
- 如果没有两个相同高度的二项树，那么合并就结束了。
- 如果有两个高度相同的二项树，我们就将其合并得到高度大一的二项树。
- 直到没有相同高度的二项树为止。

合并

- 两个二项堆的合并就是两组二项树合并。
- 如果没有两个相同高度的二项树，那么合并就结束了。
- 如果有两个高度相同的二项树，我们就将其合并得到高度大一的二项树。
- 直到没有相同高度的二项树为止。
- 为了保证复杂度，我们需要从小到大合并二项树。

二项树合并

- 两棵二项树合并的时候，只需要比较两者根节点关键字的大小。

二项树合并

- 两棵二项树合并的时候，只需要比较两者根节点关键字的大小。
- 其中含小关键字的节点成为结果树的根节点，另一棵树变成结果树的子树。

二项树合并

- 两棵二项树合并的时候，只需要比较两者根节点关键字的大小。
- 其中含小关键字的节点成为结果树的根节点，另一棵树变成结果树的子树。
- 这时候小关键字的节点的最大儿子被修改了、大关键字有了一个新兄弟。

二项树合并

- 两棵二项树合并的时候，只需要比较两者根节点关键字的大小。
- 其中含小关键字的节点成为结果树的根节点，另一棵树变成结果树的子树。
- 这时候小关键字的节点的最大儿子被修改了、大关键字有了一个新兄弟。
- 为了可持久化，我们需要新建两个点。

例题

- 给出 n 个 multiset，一开始每个集合内只有一个值。

例题

- 给出 n 个 multiset，一开始每个集合内只有一个值。
- 给出 q 个操作，每次新建一个集合为之前某两个集合的并，或者输出某个集合的最小值并将最小值加上一个读入的数。

例题

- 给出 n 个 multiset，一开始每个集合内只有一个值。
- 给出 q 个操作，每次新建一个集合为之前某两个集合的并，或者输出某个集合的最小值并将最小值加上一个读入的数。
- $n, q \leq 10^5$，保证所有集合大小不超过 2^{100}。

例题

- 直接套用即可。

例题

- 直接套用即可。
- 给最小值加上一个读入的数，可以先删除最小值，然后插入修改后的数。

例题

- 直接套用即可。
- 给最小值加上一个读入的数，可以先删除最小值，然后插入修改后的数。
- 复杂度 $O(n + q\log MaxSize)$。

参考资料

- D. Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998.
- Wikipedia. Brodal queue.
- Wikipedia. Binary heap.
- Wikipedia. Binomial heap.
- 王启圣、李煜东。理想王国的数据结构 —二项堆与斐波那契堆。WC2012 营员讨论。

- 本幻灯片使用 beamer + CTEX 制作，并用 XƎLATEX 编译。

- 本幻灯片使用 beamer + CTEX 制作，并用 XƎLATEX 编译。
- 本幻灯片中所有图片均使用 XY-pic 制作。

- 本幻灯片使用 beamer + CTeX 制作，并用 XeLaTeX 编译。
- 本幻灯片中所有图片均使用 XY-pic 制作。
- 感谢您的细心聆听，有任何问题都可以联系我。