南京航空航天大学

一种基于桶，堆，表的单调优先队列及其对 Dijsktra 算法的优化应用

Buckets, Heaps, Lists, and Monotone Priority Queues[?]

学生姓名	李想
学　　号	161610323
学　　院	计算机科学与技术学院
专　　业	计算机软件培优班
班　　级	1616001
指导教师	陈松灿教授

二〇一八年八月

目 录

第一章 算法简介

1.1 简介

本文主要提供了一种由 multilevel bucket 与传统（对元素个数敏感）的堆组合而成的一种新型高效基数堆 Heap-on-top Priority Queue，由于其较传统基数堆，利用了优先队列与桶相结合后的数据结构再来优化堆，所以此种基数堆在数据量较大的情况下运行效率表现优秀。若其中的优先队列采用高效的 Thorup’s heaps[?], 其插入操作的时间复杂度能达到 $O(log^{\frac{1}{3}}C)$，删除某一元素操作的时间复杂度能达到 $O(1)$，取出最小值的时间复杂度能达到 $O(log^{\frac{1}{3}+\epsilon})$。由此易得在点集大小为 N，边集大小为 M，边权范围 [0,C] 的图上利用 Heap-on-top Priority Queue 优化的 Dijsktra 算法的时间复杂度可达 $O(m+n(logC)^{\frac{1}{3}+\epsilon})$。

1.2 算法步骤

1.2.1 基本操作

插入 (insert) 找到元素所应该插入的位置，并插入：

弹出最小值 (extract-min)：删除列表中的最小值

调整位置 (decrease-key)：删除元素后一些元素需要减小其位置（前移）

1.2.2 多层桶

考虑一个有 k 层的桶结构 B，k 是正整数。除了顶层有无限的桶外，其他层桶有 Δ 个桶我们把第 i 层的第 k 个桶叫做 $B(I,j)$，我们用双链表表示桶，从而可以做到常数时间插入和删除。

插入：为了插入 u，找到合适的位置，然后插入

调整位置：找到该点，从 $B(i,j)$ 里面取出来，然后把它的键值 ρ 赋新值，重插入合适位置。

删除最小值：我们先找到最小值，更新 μ，然后找到最低的非空桶 level i，将其中 $B(i,j)$ 的所有元素检查一遍，删除 $B(i,j)$ 中最小的 u，并令 $\mu=\rho(u)$。最后延展 $B(i,j)$，将最小值返回。

1.2.3 新型基数堆

插入：如果 H 为空或者待插入的 u，那么就把 u 按 $multilevelbucket$ 的方法插入到桶 B。否则，若 $c(a,b) < t$，将 u 插入 H 和 $B(a,b)$，若 $c(a,b) >= t$，将 $B(a,b)$ 置为 $inactive$，将 u 插入进去并扩展这个 $bucket$

删除：如果 u 在 H 中，删除 H 中的 u，否则在 B 中找到 u，然后删除 B 中的 u，并将 $\rho(u)$ 新值再插入进合适位置

调整位置：若 H 非空，返回 H 最小值。否则找到最低的非空 $level$，在这个 $level$ 里找到第一个非空 $bucket$，和前文的操作一样若 $i = 1$，删除 $B(i,j)$ 中的一个元素，设置 $\mu = \rho(u)$ 该值并返回若 $i > 1$，找到 $B(i,j)$ 最小的那个，然后返回当 $c(i,j) > t$ 的时候（$t =$ 某个参数，k 和 t 关系到算法的复杂度）就扩展桶 B，否则设置 B 活跃 $active$

1.2.4 时间复杂度

表 1.1 时间复杂度

Trorup heaps	Heap bounds	Hot queue bounds	hot queue, best k and t
Insert	$O(1)$	$O(k + \frac{kC^{\frac{1}{k}}}{t})$	$O(\log^{\frac{1}{3}} C)$
Decrease-key	$O(1)$	$O(1)$	$O(1)$
Extract-min	$O(\log^{\frac{1}{2}+\epsilon} N)$	$O(\log^{\frac{1}{2}+\epsilon} t)$	$O(\log^{\frac{1}{3}+\epsilon} C)$

第二章 实验仿真

2.1 数据设置

我们使用 GRIDGEN 生成器[?] 生成，其两种图的类型及参数如表**??**所示。

表 2.1 Graph Types[1]

Graph Type	Graph Family	Range of values	Other values
long grid	Modifying C	$C = 1\,to\,1,000,000$	$x = 8192$
	Modifying x	$x = 512\,to\,32,768$	$C = 16$
			$C = 10,000$
			$C = 100,000,000$
	Modifying C and x	$x = 512\,to\,32,678$	$C = x$
wide grid	Modifying C	$C = 1\,to\,1,000,000$	$y = 8192$
	Modifying y	$y = 512\,to\,32,768$	$C = 16$
			$C = 10,000$
			$C = 100,000,000$
	Modifying C and y	$y = 512\,to\,32,678$	$C = y$

2.2 测试环境

系统：Windows 10 Pro 1803 17134.112

处理器：Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz 2.50GHz

运行内存：8.00 GB

编译环境：TDM64-GCC 4.9.2

2.3 实验结果

2.3.1 Long grids

表 2.2 运行时间 ($C = 16$)

n	8193	16385	32769	65537	131073	262145	524289
h3	0.03s	0.06s	0.11s	0.22s	0.45s	0.89s	1.78s

表 2.3 运行时间 ($C = 100,000,000$)

n	8193	16385	32769	65537	131073	262145	524289
h3	0.03s	0.06s	0.12s	0.23s	0.46s	0.91s	1.80s

表 2.4 运行时间 ($b = 131,073$)

C	100	1000	10000	100000	1000000	9999994	99999937
h3	0.47s	0.48s	0.48s	0.46s	0.47s	0.46s	0.46s

2.3.2 Wide grids

表 2.5 运行时间 ($C = 100,000,000$)

n	8193	16385	32769	65537	131073	262145	524289
h3	0.03s	0.07s	0.14s	0.31s	0.68s	1.45s	3.05s

表 2.6 运行时间 ($n = 131,073$)

C	100	1000	10000	100000	1000000	9999994	99999937
h3	0.61s	0.63s	0.64s	0.66s	0.68s	0.67s	0.68s

第三章　结论和总结

3.1　对本算法的总结与思考

由桶的定义可知，本文提出的算法仅适用于于整数型边权值数据的图，并且由表**??**，表**??**，表**??**可得在 C 较小的情况下 1-level bucket 较 k-level bucket 时间复杂度更小，在 C 较大时 k-level bucket 才有优化效果。此种算法将所有桶建立成一个堆，从而快速寻找最值，是我了解到了优先队列的一种新的用法。

通过研究本文算法，使我对如何对一个算法进行优化，优化的方向，以及如何结合不同的数据结构有了更深的理解。我认为，许多算法的改进都是建立在前面已有的算法知识的组合之上的，关键就在于如何确定优化的目标，确定目标后寻找适合的替代算法，还要注意不同算法之间结合的方式。