OOP 组队大作业

Project : MST ((II) Paper: On Constructing Minimum Spanning Trees in k-dimensional Spaces and Related Problems)

Author : 李晨昊 2017011466

基本介绍

实现论文 *ON CONSTRUCTING MINIMUM SPANNING TREES IN k-DIMENSIONAL SPACES AND RELATED PROBLEMS*，从而能在$o(n^2)$的时间内计算出一个二维平面上的图的最小生成树。

实现了哪些算法： (按运行速度降序，20000个随机生成的点，开启O2优化)

算法	时间复杂度	耗时
delaunay三角剖分	$\Theta(nlgn)$	0.06s
姚期智的算法	$\Theta((nlgn)^{\frac{5}{3}})$	1.08s
配对堆优化的prim算法	$\Theta(n^2)$	1.46s
斐波那契堆优化的prim算法	$\Theta(n^2)$	1.55s
朴素的prim算法	$\Theta(n^2)$	2.58s
二叉堆优化的prim算法	$\Theta(n^2lgn)$	2.99s
朴素的kruskal算法	$\Theta(n^2lgn)$	MLE

delaunay三角剖分可以在$\Theta(nlgn)$的时间内计算出最小生成树，这比姚期智的算法在理论和实践上都优秀的多。

姚期智的算法的算法理论复杂度为$\Theta((nlgn)^{\frac{5}{3}})$，这是通过取参数$s=n^{\frac{1}{3}}$达到的。但是这样的取值下常数过于巨大(也可能是我实现的太烂)，所以我改成了$s=n^{0.13}$。这时依据论文中的公式容易得到时间复杂度是$\Theta(n^{1.87})$其实这样复杂度已经和平方没什么区别了，在O2优化下只比朴素的prim快两倍左右。(不开O2优化大致能快4倍)

斐波那契堆和配对堆优化的prim算法在一般的图上有很好表现，均为$\Theta(E+VlgV)$，在完全图中即为$\Theta(n^2)$。之所以比朴素的prim更快，可能是因为朴素的prim在循环内需要两次遍历点集(一次找点，一次更新)，而斐波那契堆和配对堆优化的prim算法只需要遍历一次(找点过程由堆$\Theta(1)$实现)

二叉堆优化的prim算法和kruskal算法在稠密图，尤其是完全图中表现很差，复杂度为$\Theta(n^2lgn)$，而且因为空间复杂度是$\Theta(n^2)$，可能导致MLE。

如何编译运行

首先确保你的编译器支持C++ 17

```
cd testcase
make
./main.exe
```

其中运行main.exe时可以提供一个命令行参数，表示生成点的数目(对从文件读取点的FileInputTest无效)。默认数目是20000。

架构&分析

所有MST算法都在其构造函数中执行准备操作，其operator()中计算MST。**不存在什么抽象基类，能在编译期判断的决不留到运行期，能用模板决不用虚函数。**

这些MST算法可以分成prim派和kruskal派。

prim派：

- PrimMST.h 提供了统一的入口
  - PrimMST<NormalHeap, false> 不支持节点减值的堆(二叉堆)优化的prim算法
  - PrimMST<DecreaseableHeap, true> 支持节点减值的堆(斐波那契堆，配对堆)优化的prim算法
  - PrimMST<NoHeap> 朴素的prim算法
- FibHeap.h 实现了斐波那契堆
  - template <K> FibHeap。均摊$\Theta(1)$节点减值，均摊$\Theta(lgn)$删除最小节点
- PairingHeap.h 实现了配对堆
  - template <K> PairingHeap。均摊$\Theta(1)$节点减值，均摊$\Theta(lgn)$删除最小节点
- HeapTraits.h 利用模板提供了堆的信息
  - bool IsDecreaseableHeapV<Heap> 编译期判断Heap是否可以节点减值(是否有成员名叫decKey)
  - bool IsMergeableHeapV<Heap> 编译期判断Heap是否可以合并(是否有成员名叫merge)

kruskal派：

- KruskalMST.h
  - class KruskalMST。基类，利用排好序的边列表生成MST。**不要找虚函数，没有的。**
- DelaunayMST.h
  - class DelaunayMST。在构造函数中完成三角剖分，生成长度为$\Theta(n)$的边列表。
- YaoMST.h
  - class YapMST。在构造函数中完成姚期智算法描述的选边过程。
- UFS.h
  - class UFS。只有路径压缩的并查集(反正这不是影响效率的部分)。

测试部分，不在乎速度，所以写几个虚函数还是没问题的(可能不要虚函数其实是一种洁癖？反正我不希望任何和性能相关的代码出现虚函数)：

- BaseTest.h
  - class BaseTest。抽象基类，留出纯虚函数generate。
- DistinctIntTest.h
  - class DistinctIntTest : public BaseTest。生成一定区间内的互不相同的整数。
- RandomDoubleTest.h
  - class RandomDoubleTest : public BaseTest。生成一定区间内的随机浮点数。
- FileInputTest.h
  - class FileInputTest : public BaseTest。从in.txt读取输入数据。
- GeneratorTest.h
  - class GeneratorTest : public BaseTest。接受一个生成器，利用生成器来生成点。