实验报告

数据集

数据的数据特征

对于poj与hdu分别构造三组数据，为poj_0, poj_50, poj_100, hdu_0, hdu_50, hdu_100。文件名前部分为源数据文件名，后部分为查询输入查询命令中，必定插入的数据的比重。例如poj_50代表从poj数据集构造的输入数据集，且50%的插入来自已插入的数据，其余50%来自全部数据集。

数据的构造方法

首先将原数据从文件读入内存中，对于输入数据集，每50条源数据对应50条插入，同时50条插入对应50条查询。对于查询的对象的数据来源，依据对于每个输入数据集中设定的查询数据占已插入数据的比例，利用C++的random库，随机一个随机数，通过随机数与设定比例的大小关系决定从一插入数据集中选取还是全部数据集中选取。

哈希函数的实现思路

HashTable类：将插入与查询委托给Collection类与Hash类，实现多态。

Hash抽象类：定义hash函数接口与模数P，这里P使其等于131。

Hash4Ascii类：ascii码字符串哈希函数，对于每个字符串str，定义哈希函数

$$hash(str) = \sum_{i=0}^{len-1} b[i] * P^i \equiv 2^{64}$$

为了取模的方便，定义哈希函数返回值为unsigned long long，使其自然溢出。

Hash4UTF_8类：对于utf-8字符，首先依据每一个字节的最高两位的值，将不同的utf-8字符进行分割，之后将分割后的字符转化为unsigned int形式，采用ascii码字符串的哈希函数进行哈希。

处理冲突的方法

Collection抽象类：定义接口与用于哈希表扩容的两个整数hasInsert， *capacity。当插入数据量大于容量的一半时，找到大于* capacity的两倍的第一个质数作为容量。每一个元素存储其数据、哈希值、字符串。

Collection4Link类：利用散列表作为哈希表，每一个桶中存储一个列表，用于从存储冲突的元素，每一次查询时，将计算出来的哈希值与容量取余，后插入同种；当发生冲突时，将冲突的元素挂在相同的桶下面；当发生扩容时，每一个元素根据扩容后的容量与元素中存取的哈希值重新更新散列表的情况。

Collection4Array抽象类：利用vector作为容器，通过开放寻址法处理冲突。每一次查询从计算出来的哈希值与容量取余后的下标为地址，每一次地址通过next函数迭代到下一个地址。当发生冲突时，即第一次找到的值得的地址非空且与查询不等，即迭代到下一个地址，当发生扩容时，每一个元素根据扩容后的容量与元素中存取的哈希值嗲用insert接口重新插入到扩容后的哈希表中。

Collection4Array1类：重写迭代地址方法：单向逐步试探

$$(index + 1) \equiv \_capacity;$$

Collection4Array2类：重写迭代地址方法：双向平方试探

$$(index \pm t * t) \equiv \_capacity;$$

测试结果表格

利用以下三个方法尽量避免误差：

1. 通过迭代200次取得运行时间后平均值
2. 避开利用空间局部性导致cahe加速运行使得测试时间出现偏差
3. 利用C++11的计时器使得程序运行时间精确到微秒级

time\s	Hash4Ascii&Close	Hash4UTF_8&Close	Hash4Ascii&Step	Hash4UTF_8&Step	Hash4Ascii&DoubleSquare	Hash4UTF_8&DoubleSquare
poj_0	0.799321	0.823660	0.726715	0.744146	0.738459	0.751950
poj_50	0.799349	0.824825	0.723894	0.739042	0.734276	0.751169
poj_100	0.801125	0.823641	0.723086	0.737293	0.735677	0.751566
hdu_0	0.902822	0.931126	0.895616	0.915244	0.874508	0.896617
hdu_50	0.906557	0.936065	0.909983	0.927354	0.882948	0.903768
hdu_100	0.908349	0.936907	0.913208	0.930527	0.882792	0.902849

问题

(一)

使用ascii哈希函数实际效果略好于使用utf-8哈希函数，可能的原因是因为utf-8哈希函数将多个字节的字符合成一个utf-8字符，循环体中嵌套使用一重while循环，增加分支预测错误的代价，同时在utf-8字符中，合并字节使用的模数P为256，不为质数，可能使得字符串哈希的冲突增加。

(二)

开放散列处理冲突比封闭散列处理冲突的性能要好，因为每次封闭散列表的地址不是连续的，没有很好地应用cahe进行内存读取加速；而对于开放散列， 逐步探测的双平方探测的性能要好，因为双平方探测涉及的内存跨度较大，空间局部性较差，而此数据集随机，没有出现数据稠密的现象，使用双平方探测快速跳过数据稠密区的成效不是恒显著。

(三)

字符串中字符大量重复可能使字符串哈希计算值趋同，同时使得导致哈希值没有字符串相对应，加大哈希冲突的可能性，使得插入与查询的迭代次数增多，常数加大。

(四)

当输入的字符串中只包含两种字符时，可以用bitMap代替hashMap，可以为字符串压缩8倍空间且对应到唯一整数。对于字符串哈希的时间复杂度为$O(len)$，而用bitMap映射到的时间复杂度为$O(1)$，之后再用散列表存储映射。