上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

程序设计 II

Project: Database

田嘉禾 5130379056

基本实现

- key长度 1-16，data长度 1-256，随机生成
- 对硬盘直接操作，定期刷新
- 索引采用B+树，存在内存中，退出时保存
- 每个internal node有degree个children
- leaf存储degree-1条record
- record存储key和对应datafile中的位置

部分代码


```
class Database{
public:
    explicit Database(const string& name, int degree);
    ~Database();
    void open();          // open the datafile
    void close();         // close the datafile
    FLAG store(const string& key, const string& data, FLAG fl);     // store a piece of data
    string fetch(const string& key);          // fetch a piece of data
    void remove(const string& key);           // remove a piece of data
    void delfile();           // delete the datafile
    bool randremove();        // randomly remove a piece of data
    void randfetch();         // randomly fetch a piece of data
    bool randreplace();       // randomly replace a piece of data
    void rebuild();           // if the database is half empty, rebuild it to a new file

private:
    string name;              // name of database
    BplusTree index;          // keep index in memory
    fstream datafile;         // read & write datafile
    int count;                // number of data that has been stored in the datafile
    void copyto(ofstream& newfile, node* p);
};
```

部分代码


```
class BplusTree{
public:
    explicit BplusTree(const string& name, int degree);
    ~BplusTree();
    bool isEmpty() const{ return count == 0; }
    void insert(const rec& r);        // insert a record to the tree
    rec* fetch(const string& key);    // fetch a record from tree
    void remove(const string& key); // remove a record from tree
    void clear();                     // clear all nodes
    string randkey();                 // randomly pick a key
    friend class Database;

private:
    string name;     // file name
    int degree;      // < degree records per node
    node* root;      // root of B+ Tree
    bool dirty;      // if the index has been changed
    int count;       // number of records
```

部分代码


```
// either an internal node with pointer
// or a leaf with offset and datlen
struct rec{
    string key;
    union{  // either an internal node or a
        node* ptr;                  // points to a node
        unsigned long pos;          // position in datafile
    };
    rec(const string& key = "", node* ptr = NULL)
        :key(key), ptr(ptr){}
    rec(const string& key, unsigned long pos)
        :key(key), pos(pos){}
};
// struct of the node with an array of records
struct node{
    int num;        // number of keys, actual size of recs
    node* first;    // if internal, points to leftmost child, else, NULL
    rec* recs;      // key-node or key-pos
    node(int degree, node* p)
        :num(0), first(p){
        recs = new rec[degree - 1];
    }
    void clear();   // clear all children
};
```

测试

- 两部分测试函数：正确性与性能

正确性测试

- 随机存储nrec条数据
- 每次fetch并比较是否正确
- 每13次remove和replace一条数据
- 检查remove后是否还能获取数据
- 检查replace后数据是否一致

性能测试

- 随机写nrec条记录
- 随机读nrec条记录
- 执行下面的循环nrec×5次：

  (a) 随机读一条记录

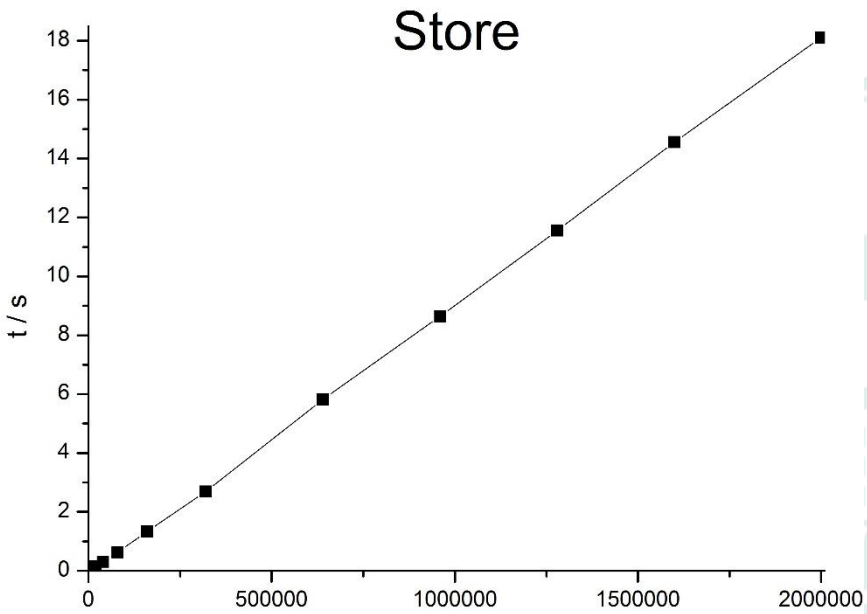
  (b) 每循环37次，随机删除一条记录

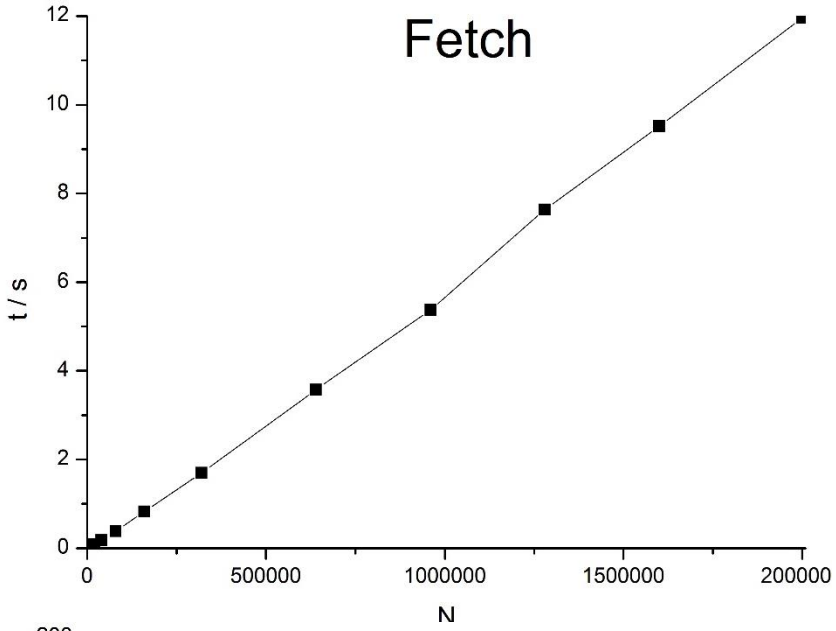
  (c) 每循环11次，随机添加一条记录并读取这条记录

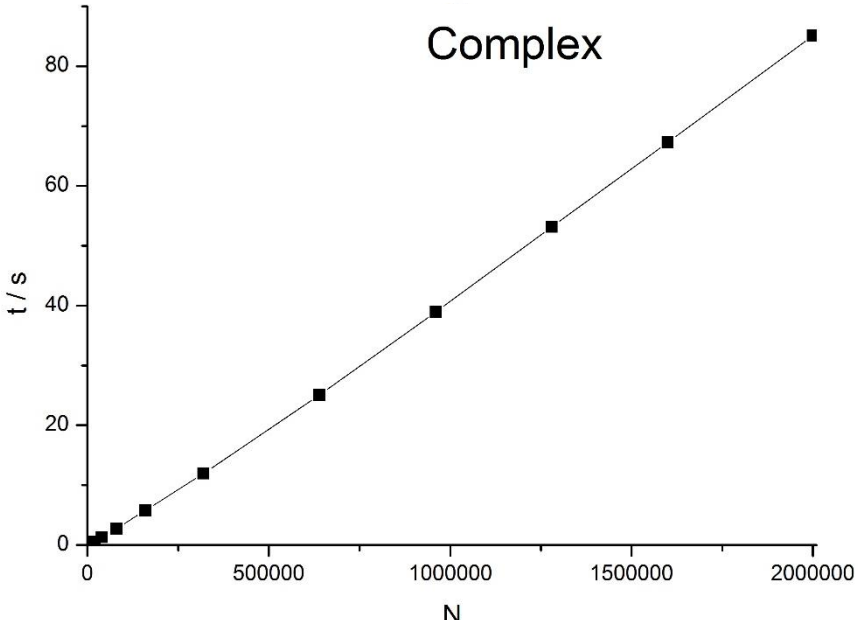
  (d) 每循环17次，随机替换一条记录为新记录
- 删除所有记录，每删除一条记录，随机查找10条记录

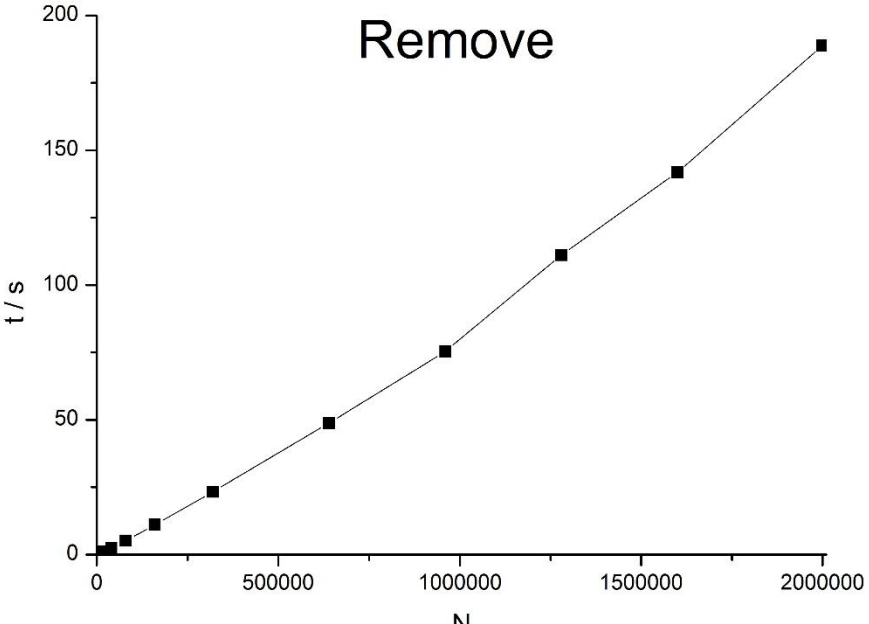
测试结果


1896
1920
1987
2006


Store
t / s
0
2
4
6
8
10
12
14
16
18
0
500000
1000000
1500000
2000000

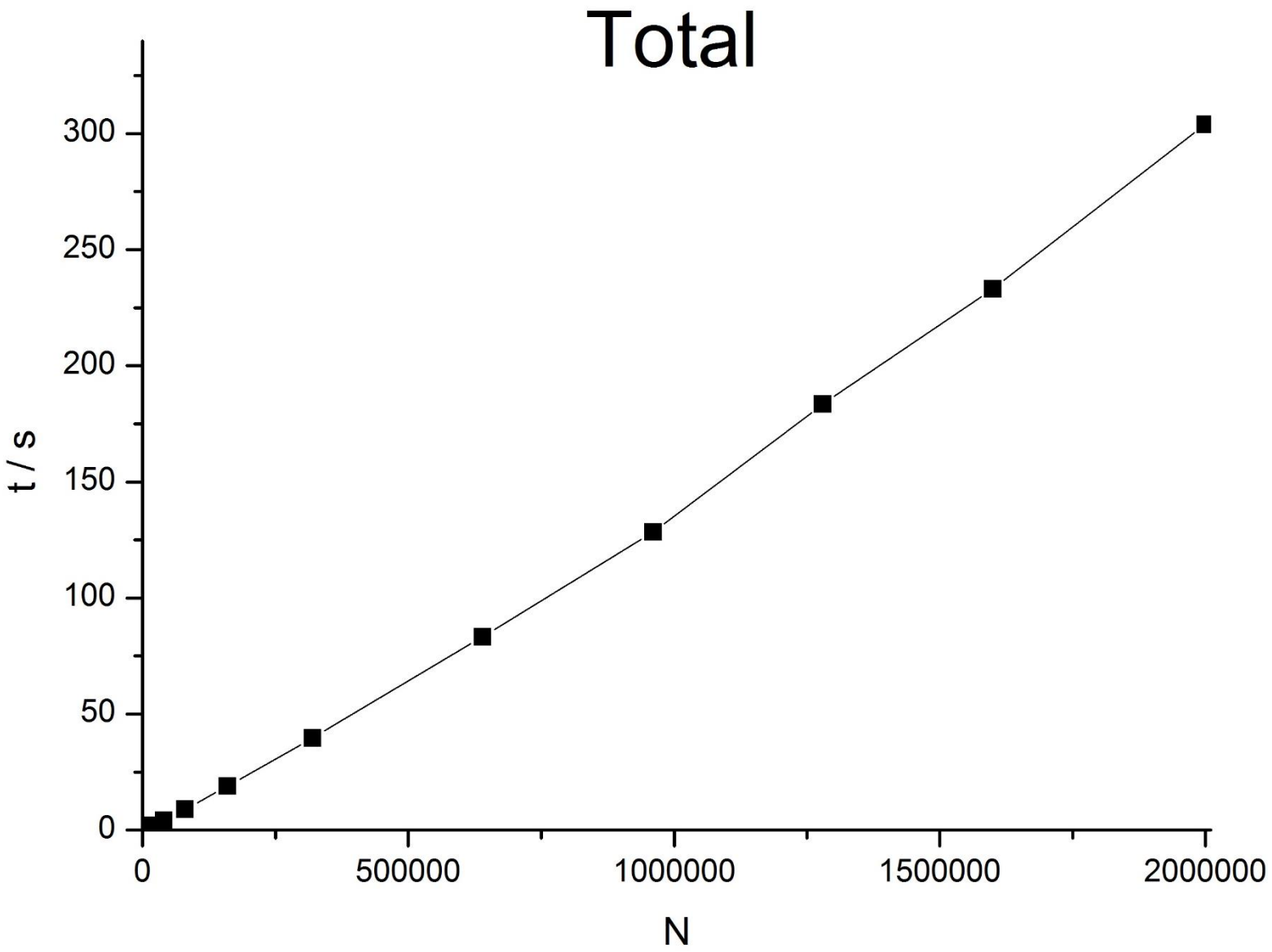

Fetch
t / s
0
2
4
6
8
10
12
0
500000
1000000
1500000
2000000
N


Complex
t / s
0
20
40
60
80
0
500000
1000000
1500000
2000000
N


Remove
t / s
0
50
100
150
200
0
500000
1000000
1500000
2000000
N

测试结果


1896
1920
1987
2006


Total
t / s
300
250
200
150
100
50
0
0
500000
1000000
1500000
2000000
N

解释分析

- degree=16
- nrec=1000000
- level≤6
- **字符串比较次数 < 6*15=90**
- **平均需 6*（15*16/2）/15=48次字符串比较**

解释分析

- store操作花费时间约为fetch操作的1.5倍：
  - 对硬盘进行读写而产生的开销
- Part4花费时间约为Part3的两倍：
  - 前者fetch的次数约为后者的两倍

解释分析

- 当nrec=1000000时，总时间约130秒，共进行
  - 1450000次store，
  - 20000000次fetch，
  - 1500000次remove，
  - 300000次replace.
- 索引文件大小约为40MB，数据文件大小约为250MB，这是由randkey()和randdata()函数生成字符串大小决定的

解释分析

- 时间随数据规模呈近似线性的增长
- 当数据量达到某些临界值，每次操作所花费的时间略有增长
  - 一方面，由于B+树中节点数增加引起的开销
  - 另一方面，因为对数据库进行刷新所花费的时间

解释分析

- 由于索引存在内存里，B+树优势不明显，层数增加带来的开销不明显
- 若索引存在磁盘上，则每次多对磁盘进行一次读写花费的额外时间会很显著

谢　谢