华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析	**年级：**19 级	**上机实践成绩：**
指导教师：金澈清	**姓名：**龚敬洋	
上机实践名称：元素查找	**学号：**	**上机实践日期：**
	10195501436	2020/12/4
上机实践编号：No.9	**组号：**1-436	

一、目的

1．熟悉算法设计的基本思想

2．掌握 Dynamic table 的方法

二、内容与设计思想

有一个公司想开发一个关于花卉的百科全书，用户只要输入花卉的名称，就能够输出花卉的详细信息。花卉包括：牡丹、芍药、茶花、菊花、梅花、兰花、月季、杜鹃花、郁金香、茉莉花、海棠、荷花、栀子花、莲花、百合、康乃馨、玫瑰、格桑花等 1000 种。这个公司想提升花卉检索和存储效率，打算采用动态表（dynamic table）来实现。由于花卉的数量可能会增加，也可能会减少，所实现的动态表需要有如下功能：

1. 能够插入数据
2. 能够删除数据
3. 能够检索数据
4. 能够按照参数扩展规模或者缩减规模

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

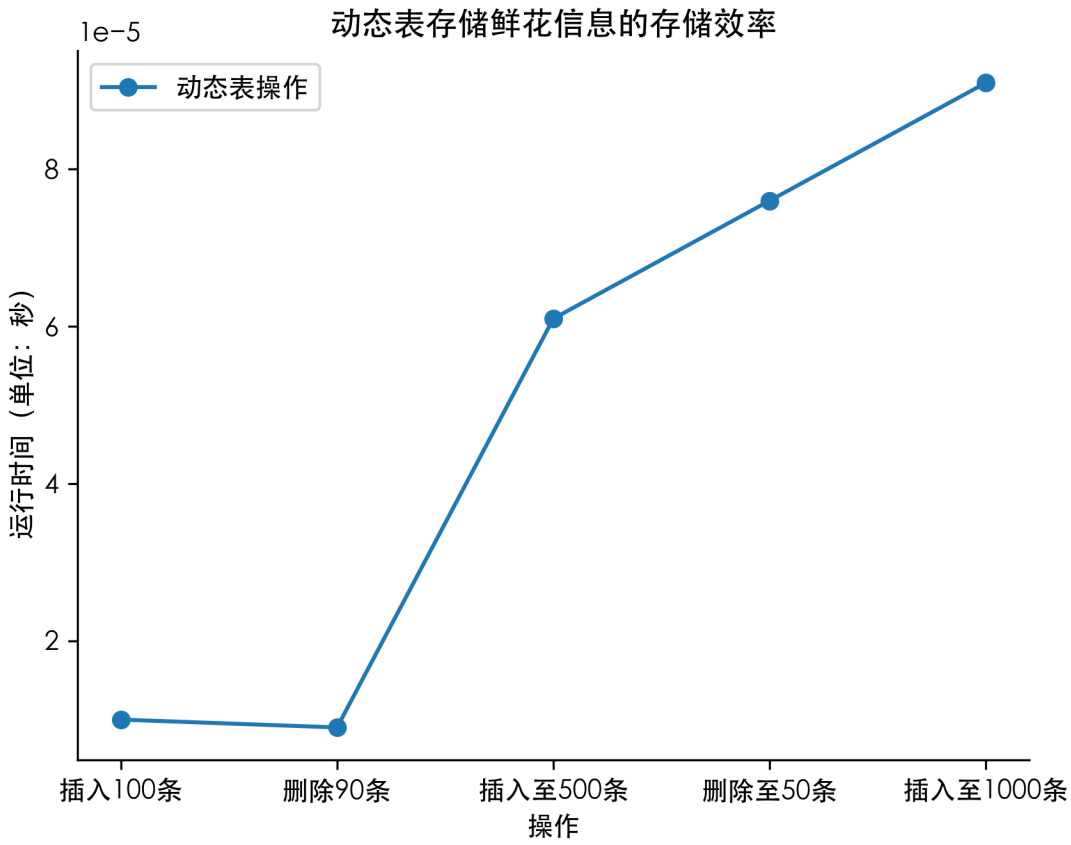
1. 写出实验的源代码

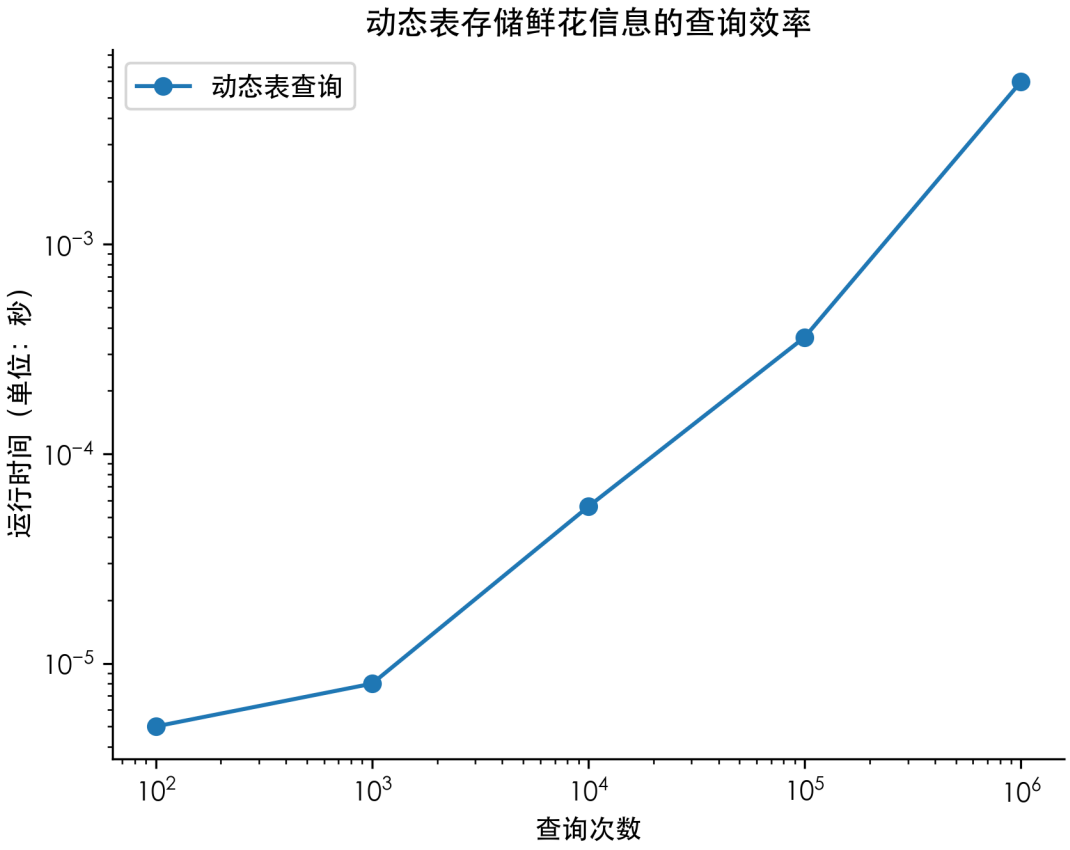
```
#include <iostream>
#include <cstdlib>
using namespace std;
struct table_info{
    int *p;
    int size;
    int num;
};
table_info *insert(table_info *table, int description){
    if (table->size == 0){
        table->p = new int[1];
        table->size = 1;
```

```
    }
    if (table->num == table->size){
        int *ntable = new int[2 * table->size];
        for (int i = 0; i < table->num; i++){
            ntable[i] = table->p[i];
        }
        delete[] table->p;
        table->p = ntable;
        table->size = 2 * table->size;
    }
    table->p[table->num] = description;
    table->num++;
    return table;
}
// You can add a parameter to insert the actual flower infos
table_info *multi_insert(table_info *table, int n){
    int t = table->num;
    for (int i = 0; i < n; i++){
        insert(table, t + i);
    }
    return table;
}
table_info *remove(table_info *table){
    if (table->size == 0){
        return table;
    }
    if (table->num <= table->size / 2){
        int *ntable = new int[table->size / 2];
        for (int i = 0; i < table->num; i++){
            ntable[i] = table->p[i];
        }
        delete[] table->p;
        table->p = ntable;
        table->size = table->size / 2;
    }
    table->num--;
    return table;
}
table_info *multi_remove(table_info *table, int n){
    for (int i = 0; i < n; i++) {
        remove(table);
    }
    return table;
}
int query(table_info *table, int i){
    if (i >= table->num) return 0;
    return table->p[i];
}
int main() {
    int op, num;
    clock_t start, stop;
    table_info t = {nullptr, 0, 0};
    cin>>op>>num;
    //op: 1 - insert; 2 - delete; 3 - query; 0 - exit
    while (op != 0) {
        if (op == 1){
            start = clock();
            multi_insert(&t, num);
            stop = clock();
            cout<<"Capacity: "<<t.size<<endl;
            cout<<"Time: "<<(double)(stop - start) / CLOCKS_PER_SEC<<endl;
        }
        if (op == 2){
            start = clock();
            multi_remove(&t, num);
            stop = clock();
```

```
            cout<<"Capacity: "<<t.size<<endl;
            cout<<"Time: "<<(double)(stop - start) / CLOCKS_PER_SEC<<endl;
        }
        if (op == 3){
            start = clock();
            int r = query(&t, num);
            stop = clock();
            cout<<"description: "<<r<<endl;
            cout<<"Time: "<<(double)(stop - start) / CLOCKS_PER_SEC<<endl;
        }
        cin >> op >> num;
    }
    return 0;
}
```

2. 分别画出各个实验结果的折线图（存储效率和查询效率）





五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。

动态表执行n次插入和删除操作的摊还代价为$O(n)$，但在实际运行过程中，当插入或删除的数据规模越过2的幂次时，运行时间会发生显著的增长，且幂次越高，运行时间的增长幅度越大，这与实验结果相吻合。实验中使用了直接寻址表作为动态表的存储结构，故理论上可以在$O(1)$的时间内查询给定的花卉信息。考虑到不同存储单元的访问代价略有不同，实验结果基本与理论吻合。