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：**10195501436	**上机实践日期：**
上机实践编号：No.8	**组号：**1-436	

一、目的

1. 熟悉算法设计的基本思想

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。公司也在试运行阶段发现这些花的访问频率不一，有些花经常性被访问，有些被访问的次数就少很多了。这 18 种花中，第 1 种的访问频率是 6，第 2-3 种的访问频率是 5，第 4-6 种的访问频率是 4，第 7-10 种的访问频率是 3，第 11-15 种的访问频率是 2，第 16-18 种的访问频率是 1。

这个公司想提升花卉检索效率，所以对比了三种方法。

1. 构建优化的二叉搜索树（optimal BST），进行搜索。
2. 将这些花卉按照访问频度从高到低放在一个数组中，并顺序访问来检索
3. 构建哈希表来存储这些数据，并基于哈希表来检索数据。

请实现这三种方法，并且通过实验来比较这三种方法的优劣。

你是否还能够想出其他高效的方法？

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. 写出各个实验的源代码

随机序列生成器：

```
#include<iostream>
#include<fstream>
#include<cstdlib>
#include<ctime>
#include<cstring>
using namespace std;
string a[10000005];
int shuffle(int length){
    return rand() % length;
}
int main(){
    string flower[18] = {"paeoniaSA", "paeoniaLP", "camellia", "chrysanthemum", "plum", "orchid",
                         "rose", "azalea", "tulip", "jasmine", "begonia", "lotus",
                         "gardenia", "lotus", "lily", "carnation", "rose", "gesang"};
    int freq[18] = {6, 5, 5, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1};
    srand(time(0));
    ofstream fout("data.txt");
    int n;
    cin>>n;
    for(int i = 0; i < 18; i++){
        for(int j = 0; j < (n * freq[i] / 53); j++){
            int t = shuffle(n);
            while(a[t] != ""){
                t = shuffle(n);
            }
            a[t] = flower[i];
        }
    }
    for(int i = 0; i < n; i++){
        fout<<a[i]<<" ";
    }
    fout.close();
    return 0;
}
```

最优二叉搜索树：

```
#include<iostream>
#include<fstream>
#include<algorithm>
#include<cstring>
#include<map>
using namespace std;
string a[10000005];
int e[20][20], w[20][20], root[20][20];
map<string, int> freq_table;
int cnt = 0;
struct node{
    string name;
    string description;
    struct node *lchild;
    struct node *rchild;
};
void optimal_bst(int n, string *flower){
    for(int i = 1; i <= n + 1; i++){
        e[i][i - 1] = 0;
        w[i][i - 1] = 0;
    }
    for(int l = 1; l <= n; l++){
        for(int i = 1; i <= n - l + 1; i++){
            int j = i + l - 1;
            e[i][j] = 999999;
            w[i][j] = w[i][j - 1] + freq_table[flower[j - 1]];
            for(int r = i; r <= j; r++){
                int t = e[i][r - 1] + e[r + 1][j] + w[i][j];
                if(t < e[i][j]){
                    e[i][j] = t;
                    root[i][j] = r;
                }
            }
        }
    }
}
node *build_bst(string *flower, string *description, int i, int j){
    struct node *rnode = new node();
    rnode->name = flower[root[i][j] - 1];
    rnode->description = description[root[i][j] - 1];
    if(i >= j) return rnode;
    struct node *lc = build_bst(flower, description, i, root[i][j] - 1);
    struct node *rc = build_bst(flower, description, root[i][j] + 1, j);
    rnode->lchild = lc;
    rnode->rchild = rc;
    return rnode;
}
string search(struct node *rnode,string *name){
    struct node *cur = rnode;
    while (cur){
        cnt++;
        if((*name) == cur->name) return cur->description;
        if (cur->lchild && ((*name) < cur->name)) cur = cur->lchild;
        else cur = cur->rchild;
    }
}
int main(){
    string flower[18] = {"paeoniaSA", "paeoniaLP", "camellia", "chrysanthemum", "plum", "orchid",
                         "rose", "azalea", "tulip", "jasmine", "begonia", "lotus",
                         "gardenia", "lotus", "lily", "carnation", "rose", "gesang"};
    //Simulate the description of flowers
    string description[18] = {"DpaeoniaSA", "DpaeoniaLP", "Dcamellia", "Dchrysanthemum", "Dplum", "Dorchid",
                              "Drose", "Dazalea", "Dtulip", "Djasmine", "Dbegonia", "Dlotus",
                              "Dgardenia", "Dlotus", "Dlily", "Dcarnation", "Drose", "Dgesang"};
    int freq[18] = {6, 5, 5, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1};
    for(int i = 0; i < 18; i++){
        freq_table[flower[i]] = freq[i];
    }
    sort(flower, flower + 18);
    sort(description, description + 18);
    ifstream fin("data.txt");
    clock_t start, stop;
    int n = 0;
    while(!fin.eof()){
        fin>>a[n];
        n++;
    }
    n--;
    optimal_bst(18, flower);
    struct node *troot = build_bst(flower, description, 1, 18);
    start = clock();
    for(int i = 0; i < n; i++){
        string t = search(troot, &a[i]);
        cout<<t<<" ";
    }
    stop = clock();
    cout<<endl;
    cout<<"Total count: "<<cnt<<endl;
    cout<<"Total time: "<<(double)(stop - start) / CLOCKS_PER_SEC<<"s"<<endl;
    return 0;
}
```

顺序访问：

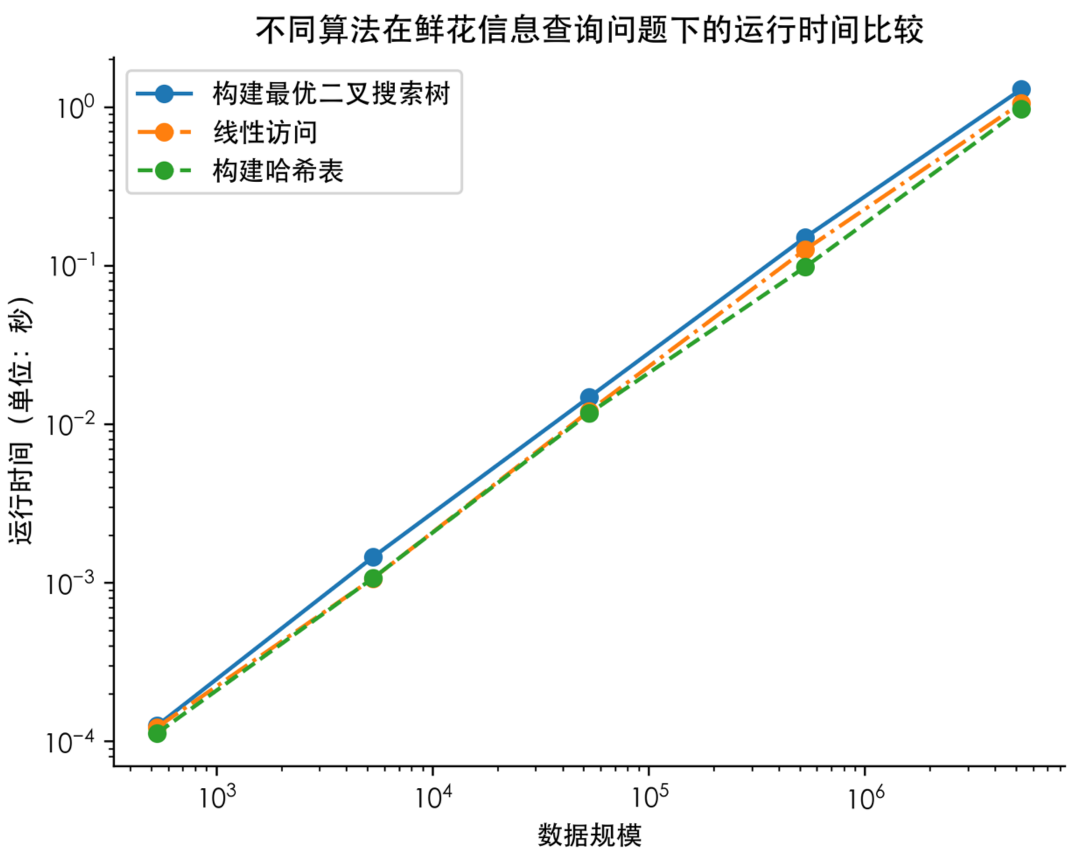
```
#include <iostream>
#include <fstream>
#include <cstring>
```

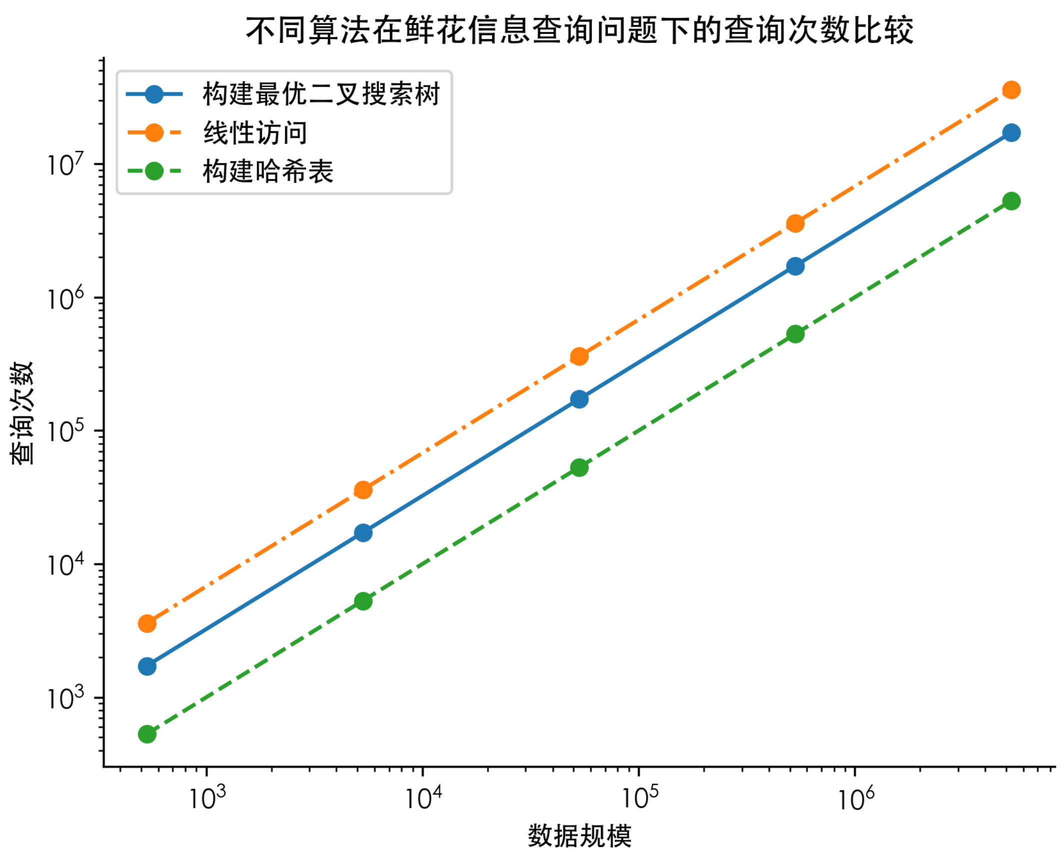
```cpp
#include <cstdlib>
using namespace std;
string a[10000005];
int main() {
    string flower[18] = {"paeoniaSA", "paeoniaLP", "camellia", "chrysanthemum", "plum", "orchid",
                         "rose", "azalea", "tulip", "jasmine", "begonia", "lotus",
                         "gardenia", "lotus", "lily", "carnation", "rose", "gesang"};
    //Simulate the description of flowers
    string description[18] = {"DpaeoniaSA", "DpaeoniaLP", "Dcamellia", "Dchrysanthemum", "Dplum", "Dorchid",
                              "Drose", "Dazalea", "Dtulip", "Djasmine", "Dbegonia", "Dlotus",
                              "Dgardenia", "Dlotus", "Dlily", "Dcarnation", "Drose", "Dgesang"};
    ifstream fin("data.txt");
    clock_t start, stop;
    int n = 0;
    while(!fin.eof()){
        fin>>a[n];
        n++;
    }
    n--;
    int cnt = 0;
    start = clock();
    for(int i = 0; i < n; i++) {
        for (int j = 0; j < 18; j++) {
            cnt++;
            if (a[i] == flower[j]) {
                string t = description[j];
                cout << t << " ";
                break;
            }
        }
    }
    stop = clock();
    cout<<endl;
    cout<<"Total count: "<<cnt<<endl;
    cout<<"Total time: "<<(double)(stop - start) / CLOCKS_PER_SEC<<"s"<<endl;
    return 0;
}
```

哈希表：

```cpp
#include<iostream>
#include<fstream>
#include<cstring>
#include<unordered_map>
using namespace std;
string a[10000005];
int main(){
    string flower[18] = {"paeoniaSA", "paeoniaLP", "camellia", "chrysanthemum", "plum", "orchid",
                         "rose", "azalea", "tulip", "jasmine", "begonia", "lotus",
                         "gardenia", "lotus", "lily", "carnation", "rose", "gesang"};
    //Simulate the description of flowers
    string description[18] = {"DpaeoniaSA", "DpaeoniaLP", "Dcamellia", "Dchrysanthemum", "Dplum", "Dorchid",
                              "Drose", "Dazalea", "Dtulip", "Djasmine", "Dbegonia", "Dlotus",
                              "Dgardenia", "Dlotus", "Dlily", "Dcarnation", "Drose", "Dgesang"};
    unordered_map<string, string> flower_map;
    for(int i = 0; i < 18; i++){
        flower_map[flower[i]] = description[i];
    }
    ifstream fin("data.txt");
    clock_t start, stop;
    int n = 0;
    while(!fin.eof()){
        fin>>a[n];
        n++;
    }
    n--;
    int cnt = 0;
    start = clock();
    for(int i = 0; i < n; i++){
        cnt++;
        string t = flower_map[a[i]];
        cout<<t<<" ";
    }
    stop = clock();
    cout<<endl;
    cout<<"Total count: "<<cnt<<endl;
    cout<<"Total time: "<<(double)(stop - start) / CLOCKS_PER_SEC<<"s"<<endl;
    return 0;
}
```

2. *分别画出各个实验结果的折线图*





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。

由于每种鲜花被查询的频率不同，为了测试各算法的运行效率，实验中首先根据不同鲜花的查询频率生成了一串不同规模的查询序列，通过比较查询序列所花费的总时间来比较各算法的运行效率。

从实验中可以发现，通过构建哈希表来查询鲜花信息的运行效率最高，且随着数据规模的增大优势越发明显，这与哈希表$O(1)$的理论查询时间相符。然而，实验中通过构建最优二叉搜索树进行查询的运行效率不如线性访问的运行效率，这与理论分析不符。通过进一步记录两种算法的查询次数发现在最优二叉搜索树上查询的次数的确小于线性访问的查询次数。由于实验中最优二叉搜索树的每个结点均申请在了堆上，而线性访问时所用数组申请在了栈上，故推测是堆的访问效率不如栈的访问效率导致了其运行效率不如线性访问。

由此可见，在鲜花信息查询问题上，构建哈希表查询是一个最为高效且可行的算法。