## 数据结构课程设计报告

班级：1618103 学号：051810217 姓名：杨通达

### 建立大多数任务可能均用到的公用函数：

function.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_FUNCTION\_H  
#define DESIGN\_FUNCTION\_H  
  
#include <string>  
#include <vector>  
#include <queue>  
#include <stack>  
#define LENG 100  
  
std::string getString(const std::string& out = "name");  
int getInt(const std::string& out = "name");  
float getFloat(const std::string& out = "name");  
bool compareOperate(char c1,char c2);//true c2 > c1  
double computeTwoNumber(double n1,double n2,char c1);  
void mergeSort(std::vector<int> &sq);  
void selectionSort(std::vector<int> &sq);  
void bubbleSort(std::vector<int> &sq);  
bool fastSort(std::vector<int> &sq);  
bool heapSort(std::vector<int> &sq);  
bool radixSort(std::vector<int> &sq,int n);  
bool straightSort(std::vector<int> &sq);  
bool shellSort(std::vector<int> &sq);  
#endif //DESIGN\_FUNCTION\_H

function.cpp

//  
// Created by 97927 on 2019/12/13.  
//  
  
#include "function.h"  
#include <iostream>  
#include <vector>  
#include "my\_Queue.h"  
#include <ctime>  
#include <math.h>  
using namespace std;  
  
//return a string get fro stdin  
std::string getString(const std::string& out) {  
 std::string str;  
 cout<<"Enter the "<<out<<" : ";  
 getline(cin,str);  
 return str;  
}  
//return a int number and clear stdin  
int getInt(const string &out) {  
 int num;  
 cout<<"Enter the "<<out<<" : ";  
 while(true)  
 {  
 cin>>num;  
 if(cin.fail())  
 {  
 cout<<"Error!Please enter the integer : ";  
 cin.clear();  
 continue;  
 }  
 break;  
 }  
 cin.ignore();  
  
 return num;  
}  
//return a float number and clear stdin  
float getFloat(const std::string& out)  
{  
 float num;  
 cout<<"Enter the "<<out<<" : ";  
 while(true)  
 {  
 cin>>num;  
 if(cin.fail())  
 {  
 cout<<"Error!Please enter the integer : ";  
 cin.clear();  
 continue;  
 }  
 break;  
 }  
  
 return num;  
}  
//compare two operators  
bool compareOperate(char c1,char c2)  
{  
 switch(c1)  
 {  
 case '-' :  
 case '+' :  
 {  
 return !(c2 == '-' || c2 == '+' || c2 == '(');  
 }  
 case '\*' :  
 case '/' :  
 {  
 return c2 == ')';  
 }  
 case '(' :  
 {  
 return c2 != ')';  
 }  
 case ')' :  
 {  
 return false;  
 }  
 }  
}  
//compute  
double computeTwoNumber(double n1, double n2,char c1)  
{  
 switch(c1)  
 {  
 case '+' :  
 return n1 + n2;  
 case '-' :  
 return n1 - n2;  
 case '\*' :  
 return n1 \* n2;  
 case '/' :  
 if(n2 == 0)//if divide bt 0  
 return -65535;  
 return n1 / n2;  
 }  
 return -1;  
}  
//sort two orderly array , sq is the total array , a1 / a2 is the first array's begin and end  
void collectionSort(vector<int> &sq,int a1,int a2,int b1,int b2)  
{  
 while(1)  
 {  
 if(a1 > a2 || b1 > b2)  
 break;  
 if(sq[a1] >= sq[b1])  
 {  
 a1++;  
 }  
 else{//sq[a1] < sq[b1]  
 int pause = sq[b1];  
 sq.erase(sq.begin() + b1);  
 sq.insert(sq.begin() + a1,pause);  
 a1++;  
 b1++;  
 a2++;  
 }  
 }  
}  
  
void mergeSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int length = sq.size();  
 //store the begin site and end site  
 my\_Queue q;  
 for(int i = 0;i < length;i++)  
 {  
 q.enQueue(i);  
 q.enQueue(i);  
 }  
 while(q.getlength() > 2)  
 {  
 int a1,a2,b1,b2;  
 q.deQueue(&a1);  
 q.deQueue(&a2);  
 q.getHead(&b1);  
 //decide weather this two arrays is order  
 if(b1 - 1 == a2)  
 {  
 q.deQueue(&b1);  
 q.deQueue(&b2);  
 collectionSort(sq,a1,a2,b1,b2);  
 //pop in the array1 begin and array2 end  
 q.enQueue(a1);  
 q.enQueue(b2);  
 } else{  
 q.enQueue(a1);  
 q.enQueue(a2);  
 }  
 }  
 last = clock();  
 printf("Merge Sort Finish.");  
 cout<<"Using "<<last - current<<" ms"<<endl;  
}  
  
void selectionSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int times = sq.size();  
 int max;  
 int max\_site;  
 for(int i = 0;i < times;i++)  
 {  
 max = sq[i];  
 for(int j = i + 1;j < times;j++)  
 {  
 if(sq[j] < max)  
 {  
 max = sq[j];  
 max\_site = j;  
 }  
 }  
 sq[max\_site] = sq[i];  
 sq[i] = max;  
 }  
 last = clock();  
 printf("Selection Sort Finish.");  
 cout<<"Using "<<last - current<<" ms"<<endl;  
}  
  
void bubbleSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int times = sq.size();  
 for(int i = 0;i < times;i++)  
 {  
 for(int j = times;j > i;j--)  
 {  
 if(sq[j] < sq[j - 1])  
 {  
 int pause = sq[j];  
 sq[j] = sq[j - 1];  
 sq[j - 1] = pause;  
 }  
 }  
 }  
 last = clock();  
 printf("Bubble Sort Finish.");  
 cout<<"Using "<<last - current<<" ms"<<endl;  
}  
  
bool straightSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int times = sq.size();  
 for(int i = 1;i < times;i++)  
 {  
 for(int k = 0;k < i;k++)  
 {  
 if(sq[k] > sq[i])  
 {  
 int nm = sq[i];  
 sq.erase(sq.begin() + i);  
 sq.insert(sq.begin() + k,nm);  
 break;  
 }  
 }  
 }  
 last = clock();  
 printf("Straight Sort Finish.");  
 cout<<"Using "<<last - current<<" ms"<<endl;  
  
 return true;  
}  
//straight sort used by shell sort  
bool straight(vector<int> &sq)  
{  
 int times = sq.size();  
 for(int i = 1;i < times;i++)  
 {  
 for(int k = 0;k < i;k++)  
 {  
 if(sq[k] > sq[i])  
 {  
 int nm = sq[i];  
 sq.erase(sq.begin() + i);  
 sq.insert(sq.begin() + k,nm);  
 break;  
 }  
 }  
 }  
  
 return true;  
}  
//used bt fast sort  
int changeSort(vector<int> &sq,int bin,int end)  
{  
 //if just have two elem  
 if((end - bin) <= 1)  
 {  
 if(bin >= end)  
 return -1;  
 if(sq[bin] < sq[end])  
 {  
 int pause = sq[bin];  
 sq[bin] = sq[end];  
 sq[end] = pause;  
 }  
 return -1;  
 }  
 //store the number  
 int pause = sq[bin];  
 sq.erase(sq.begin() + bin);  
 int i = bin;  
 int k = end - 1;  
 while(1)  
 {  
 //end condition  
 if(i >= k)  
 {  
 if(sq[i] <= pause)  
 {  
 k = i;  
 } else{  
 k++;  
 i = k;  
 }  
 sq.insert(sq.begin() + k,pause);  
 break;  
 }  
 //find the number smaller than pause from the end  
 if(sq[k] > pause)  
 {  
 k--;  
 continue;  
 }  
 //find the number biger than pause from the begin  
 if(sq[i] > pause)  
 {  
 //swap  
 int data = sq[k];  
 sq[k] = sq[i];  
 sq[i] = data;  
 k--;  
 i++;  
 } else{  
 i++;  
 continue;  
 }  
 }  
 //return next position should be sort  
 return k;  
}  
  
bool fastSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 queue<int> q;//store the position which should be sort  
 q.push(0);  
 q.push(sq.size() - 1);  
 while(!q.empty())  
 {  
 int bin,end;  
 bin = q.front();  
 q.pop();  
 end = q.front();  
 q.pop();  
 int middle = changeSort(sq,bin,end);  
 if(middle != -1)  
 {  
 //store the begin to middle , middle to end  
 q.push(bin);  
 q.push(middle - 1);  
 q.push(middle + 1);  
 q.push(end);  
 }  
 }  
 last = clock();  
 cout<<"Faste Sort Finish.Using "<<last - current<<" ms"<<endl;  
  
 return true;  
}  
//swap two number  
void swap(int \*num1,int \* num2)  
{  
 int pause = \*num1;  
 \*num1 = \*num2;  
 \*num2 = pause;  
}  
//compare the root and its two leaves  
//return the position should be solved , -1 means don't need to compare  
int maxHeap(vector<int> &sq,int bin,int end)  
{  
 int end1 = 2 \* bin + 1;  
 if(end1 >= end)  
 {  
 return -1;  
 }  
 if(end1 + 1 < end && sq[end1] < sq[end1 + 1])  
 {  
 end1++;  
 }  
 if(sq[end1] > sq[bin])  
 {  
 swap(&sq[end1],&sq[bin]);  
 return end1;  
 }  
 return -1;  
}  
  
bool heapSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int i = (int)(sq.size() / 2);  
 int end = sq.size();  
 //create the max heap  
 while(i >= 0)  
 {  
 int k;  
 k = maxHeap(sq, i,end);  
 while(k > 0)  
 {  
 k = maxHeap(sq,k,end);  
 }  
 i--;  
 }  
 while(1)  
 {  
 /\*cout<<endl;  
 for(int i = 0;i < sq.size();i++)  
 {  
 cout<<sq[i]<<" ";  
 if((i % 10) == 9)  
 cout<<endl;  
 }\*/  
 int pause = sq[0];  
 //swap the top and the bottom  
 swap(&sq[0],&sq[end - 1]);  
 if(end == 1)  
 break;  
 end--;  
 //create the max heap again  
 int k = maxHeap(sq,0,end);  
 while(k > 0)  
 {  
 k = maxHeap(sq,k,end);  
 }  
 }  
 last = clock();  
 cout<<"HeapSort Finish.Using "<<last - current<<" ms"<<endl;  
  
 return true;  
}  
//inputing n stands for the bit of the bigest number  
bool radixSort(std::vector<int> &sq,int n)  
{  
 clock\_t current,last;  
 current = clock();  
 vector<int> pauses[10];  
 int k = 0;  
  
 while(k < n)  
 {  
 //push back begin from singles  
 while(!sq.empty())  
 {  
 int nm = sq.back();  
 int st = nm;  
 for(int j = 0;j < k;j++)  
 {  
 st /= 10;  
 }  
 sq.pop\_back();  
 pauses[st % 10].push\_back(nm);  
 }  
 k++;  
 //pop and sort again  
 for(int i = 0;i < 10;i++)  
 {  
 while(!pauses[i].empty())  
 {  
 sq.push\_back(pauses[i].back());  
 pauses[i].pop\_back();  
 }  
 }  
 }  
 last = clock();  
 cout<<"RadixSort Finish.Using "<<last - current<<" ms"<<endl;  
  
 return true;  
}  
  
bool shellInsert(std::vector<int> &sq,int n, int length)  
{  
 //straight sort as the path of n  
 for(int j = 0;j < length - n;j++)  
 {  
 if(sq[j] > sq[j + n])  
 {  
 swap(&sq[j],&sq[j + n]);  
 for(int k = j - n;k > -1;k -= n)  
 {  
 if(sq[k] > sq[k + n])  
 {  
 swap(&sq[k],&sq[k + n]);  
 continue;  
 }  
 break;  
 }  
 }  
 }  
 return true;  
}  
  
bool shellSort(std::vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int length = sq.size();  
 int t = (int)log(length);  
 int dr = 2^(t + 1) - 1;//path  
 //end when the path smaller than 1  
 for(int i = 0;dr > 0;i++,dr = 2^(t - i + 1) - 1)  
 {  
 shellInsert(sq,dr,length);  
 }  
 last = clock();  
 cout<<"ShellSort Finish.Using "<<last - current<<" ms"<<endl;  
  
 return 0;  
}

my\_Queue.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_MY\_QUEUE\_H  
#define DESIGN\_MY\_QUEUE\_H  
  
class QNode{  
public:  
 int data;  
 QNode \* next;  
 QNode(int n = 0,QNode\* nxt = nullptr);  
 ~QNode();  
};  
class Qnode\_more :public QNode {  
public:  
 int data1;//time begin  
 int data2;//time eng  
 Qnode\_more \* frnt;  
 Qnode\_more \* next;  
 Qnode\_more(int n1 = 0,int n2 = 0,int n3 =0):QNode(n1){//initially  
 data1 = n2;  
 data2 = n3;  
 frnt = nullptr;  
 next = nullptr;  
 };  
 ~Qnode\_more()= default;  
};  
class my\_Queue {  
private:  
 QNode \* frt;  
 QNode \* lst;  
 int length;  
public:  
 my\_Queue();  
 bool empty();  
/\*Length Finish\*/  
 int getlength();  
/\*Get Finish\*/  
 bool getHead(int \*e);  
/\*Clear Finish\*/  
 void clearQueue();  
/\*En Finsh\*/  
 bool enQueue(int e);  
/\*De Finish\*/  
 bool deQueue(int \*e);  
/\*Traverse Finsh\*/  
 bool traverse();  
 ~my\_Queue();  
};  
class my\_Queue\_Plus{  
private:  
 Qnode\_more \* frt;  
 Qnode\_more \* lst;  
 int length;  
  
public:  
 my\_Queue\_Plus();//initially  
 bool empty();//true when all thing finish  
 int getlength();//return length  
 bool getHead(int \*e);//get the first thing  
 void clearQueue();  
 bool enQueue(int e1,int e2,int e3);//insert thing to back  
 bool deQueue(int \*e,int \*e2,int \*e3);//pop the first thing  
 bool traverse();//show queue  
 bool sort();//sort as begin time  
 int lastTime();//return the last thing end time  
 bool timeEvent(int t,my\_Queue\_Plus & q);//deal with time n  
 ~my\_Queue\_Plus();  
};  
  
#endif //DESIGN\_MY\_QUEUE\_H

my\_Queue.cpp

//  
// Created by 97927 on 2019/12/13.  
//  
  
#include "my\_Queue.h"  
#include <iostream>  
  
bool my\_Queue::empty() {  
 return length == 0;  
}  
  
my\_Queue::my\_Queue() {  
 frt = new QNode();  
 lst = frt;  
 length = 0;  
}  
  
int my\_Queue::getlength() {  
 return length;  
}  
  
bool my\_Queue::getHead(int \*e) {  
 if(empty())  
 return false;  
 \*e = frt->next->data;  
 return true;  
}  
  
void my\_Queue::clearQueue() {  
 QNode \* p = frt->next;  
 while(p)  
 {  
 QNode \* m = p->next;  
 delete p;  
 p = m;  
 }  
 lst = frt;  
 length = 0;  
}  
  
bool my\_Queue::enQueue(int e) {  
 QNode \* p = new QNode;  
 if(!p)  
 return false;  
  
 p->data = e;  
 p->next = nullptr;  
 lst->next = p;  
 lst = p;  
 length++;  
  
 return true;  
}  
  
bool my\_Queue::deQueue(int \*e) {  
 QNode \* p = frt->next;  
 if(p)  
 {  
 if(p == lst)  
 {  
 lst = frt;  
 }  
 \*e = p->data;  
 frt->next = p->next;  
 delete p;  
 }  
 else  
 return false;  
 length--;  
 return true;  
}  
  
bool my\_Queue::traverse() {  
 using std::cout;  
 using std::endl;  
 QNode \* p = frt->next;  
 int k = 1;  
 while(p)  
 {  
 cout<<"data "<<k<<" : "<<p->data<<" ";  
 p = p->next;  
 k++;  
 if(k % 10 == 0)  
 cout<<endl;  
 }  
 cout<<"queue traverse finish."<<endl;  
 return true;  
}  
  
my\_Queue::~my\_Queue() {  
 QNode \* p = frt;  
 while(p)  
 {  
 QNode \* m = p->next;  
 delete p;  
 p = m;  
 }  
 lst = nullptr;  
 frt = nullptr;  
 length = 0;  
}  
  
QNode::QNode(int n,QNode \* nxt) {  
 data = n;  
 QNode::next = nxt;  
}  
  
QNode::~QNode() = default;  
  
  
my\_Queue\_Plus::my\_Queue\_Plus() {  
 frt = new Qnode\_more();  
 lst = frt;  
 length = 0;  
}  
  
bool my\_Queue\_Plus::empty() {  
 return length == 0;  
}  
  
int my\_Queue\_Plus::getlength() {  
 return length;  
}  
  
bool my\_Queue\_Plus::getHead(int \*e) {  
 if(empty())  
 return false;  
 \*e = frt->next->data;  
 return true;  
}  
  
void my\_Queue\_Plus::clearQueue() {  
 Qnode\_more \* p = frt->next;  
 while(p)  
 {  
 Qnode\_more \* m = p->next;  
 delete p;  
 p = m;  
 }  
 lst = frt;  
 length = 0;  
}  
  
bool my\_Queue\_Plus::enQueue(int e1, int e2, int e3) {  
 Qnode\_more \* p = new Qnode\_more;  
 if(!p)  
 return false;  
  
 p->data = e1;  
 p->data1 = e2;  
 p->data2 = e3;  
 p->frnt = lst;  
 p->next = nullptr;  
 length++;  
 lst->next = p;  
 lst = p;  
  
 return true;  
}  
  
bool my\_Queue\_Plus::deQueue(int \*e, int \*e2, int \*e3) {  
 using std::cout;  
 using std::endl;  
 Qnode\_more \* p = frt->next;  
 if(p)  
 {  
 if(p == lst)  
 {  
 lst = frt;  
 }  
 \*e = p->data;  
 \*e2 = p->data1;  
 \*e3 = p->data2;  
 frt->next = p->next;  
 if(p->next)  
 p->next->frnt = frt;  
 delete p;  
 }  
 else  
 return false;  
 length--;  
 return true;  
}  
  
bool my\_Queue\_Plus::traverse() {  
 using std::cout;  
 using std::endl;  
 Qnode\_more \* p = frt->next;  
 int k = 1;  
 while(p)  
 {  
 cout<<"data "<<k<<" : "<<p->data<<","<<p->data1<<","<<p->data2 - p->data1<<" ";  
 p = p->next;  
 k++;  
 if(k % 10 == 0)  
 cout<<endl;  
 }  
 cout<<"queue traverse finish."<<endl;  
 return true;  
}  
  
my\_Queue\_Plus::~my\_Queue\_Plus(){  
 Qnode\_more \* p = frt;  
 while(p)  
 {  
 Qnode\_more \* m = p->next;  
 delete p;  
 p = m;  
 }  
 lst = nullptr;  
 frt = nullptr;  
 length = 0;  
}  
  
bool my\_Queue\_Plus::sort() {  
 Qnode\_more \* p = frt->next;  
 while(p)  
 {  
 Qnode\_more \* min = p;  
 Qnode\_more \* q = p->next;  
 while(q)  
 {  
 //find the min begin time  
 if(q->data1 < min->data1)  
 {  
 min = q;  
 }  
 //if begin time is equal , compare end time  
 else if(q->data1 == min->data1)  
 {  
 if(q->data2 < min->data2)  
 {  
 min = q;  
 }  
 else if(q->data2 == min->data2)  
 {  
 if(q->data < min->data)  
 {  
 min = q;  
 }  
 }  
 }  
 q = q->next;  
 }  
 if(min == p)  
 {  
 p = p->next;  
 continue;  
 }  
 min->frnt->next = min->next;  
 if(min->next)  
 min->next->frnt = min->frnt;  
 p->frnt->next = min;  
 min->frnt = p->frnt;  
 p->frnt = min;  
 min->next = p;  
 }  
  
 return true;  
}  
  
int my\_Queue\_Plus::lastTime() {  
 return lst->data2;  
}  
//find event happen at this second  
bool my\_Queue\_Plus::timeEvent(int t,my\_Queue\_Plus & q) {  
 using std::cout;  
 using std::endl;  
 Qnode\_more \* p = frt->next;  
 bool reFlag = false;  
 if(p->data1 > t)  
 return false;  
 while(p)  
 {  
 if(p->data1 > t)  
 break;  
 if(p->data1 == t || p->data2 == t)  
 {  
 int key\_n = p->data;  
 int ta\_t = p->data1;  
 int re\_t = p->data2;  
 q.enQueue(key\_n,ta\_t,re\_t);  
 reFlag = true;  
 }  
 //if this event is end , pop  
 else if(p->data2 < t)  
 {  
 p->frnt->next = p->next;  
 if(p->next)  
 p->next->frnt = p->frnt;  
 cout<<"event "<<" : "<<p->data<<","<<p->data1<<","<<p->data2 - p->data1<<" end"<<endl;  
 length--;  
 Qnode\_more \* pause = p->next;  
 delete p;  
 p = pause;  
 continue;  
 }  
 p = p->next;  
 }  
  
 return reFlag;  
}

### 必做题第一题

###### 题目：系统进程统计（必做）（链表）

[问题描述]

设计一个程序，每秒统计一次当前系统的进程状况，并按照内存使用自多到少排序打印输出相关信息。对已经结束的进程，另外给出一个列表，并显示该进程的结束时间和持续时间。

[基本要求]

（1） 该题目要求使用两个链式线性表。一个链表存储当前活动进程，要求使用双向链表，排序要求是按照内存使用自多到少排序。另外一个链表存储已结束进程，要求使用单向链表，按照结束时间离当前时间的关系排序，最近的最前，最远的最后。

（2） 每秒在窗口内更新一次当前系统进程情况，输出内容包括：进程名，持续时间，内存使用情况。

（3） 每秒在窗口内更新一次已结束进程情况，输出内容包括：进程名，持续时间，结束时间。

（4） 注意进程在这两个链表中的切换，一个进程既可被结束，也可以过一段时间后再被运行。

###### 数据结构：链表

###### 算法思想：

建立三个链表  
list1存储上一时刻进程  
list2存储该时刻进程  
list3存储结束进程  
每秒进行一次判断：  
1.list1为空：将所有进程存入list1，执行步骤3  
2.list1不为空：对比list1与list2  
 2.1.list1中存在，list2中不存在，则从list1中删除该记录，存入list3中，记录结束时间  
 2.2.list1中不存在，list2中存在，则在list3中查找该进程  
 2.2.1.list3中存在，则将该进程的已经运行时间记录为原有运行时间，从list3中删除该进程，加入 list1中  
 2.2.2.list3不存在，直接将该进程加入到list1中  
 执行步骤3  
3.将list1按内存排序后，输出list1与list3

###### 算法时间复杂度：

在排序过程中使用了选择排序，时间复杂度为(n^2)

###### 实现过程：

my\_List.h

//  
// Created by 97927 on 2019/12/29.  
//  
  
#ifndef DESIGN\_MY\_LIST\_H  
#define DESIGN\_MY\_LIST\_H  
  
#include <cstring>  
#include <ctime>  
class task {  
public:  
 char name[128];//name of task  
 int memory;//store memory  
 time\_t bgn;//store the begin time  
 int lastTime;//store the task last time  
 time\_t end;//store the end time  
 task(char\* t) : memory(0)//initial  
 {  
 std::time(&bgn);  
 lastTime = 0;  
 end = 0;  
 char\* b = t + 1, \* p = t + 1;  
 while (\*p) {  
 if (\*p == '"')  
 {  
 \*p = 0;  
 strcpy(name, b);  
 break;  
 }  
 p++;  
 }  
 for (b = p + 1; p[1] != 'K'; p++)  
 if (\*p == '"')  
 b = p;  
 for (char\* c = b + 1; c != p; c++)//calculate the memory  
 if (\*c == ',')  
 continue;  
 else  
 memory = 10 \* memory + \*c - '0';  
 }  
 bool operator<(task& t) { return this->memory < t.memory; }  
};  
class LiNode{  
public:  
 task data;  
 LiNode \* next;  
 LiNode(task n):data(n)  
 {  
 next = nullptr;  
 }  
 ~LiNode();  
};  
class LiNode\_B {  
public:  
 task data;  
 LiNode\_B \* pre;  
 LiNode\_B \* next;  
 LiNode\_B(task n):data(n)  
 {  
 pre = nullptr;  
 next = nullptr;  
 }  
 ~LiNode\_B();  
};  
class my\_List{  
public:  
 LiNode \* frt;  
 LiNode \* lst;  
 int length;  
 my\_List();//initial  
 ~my\_List();//delete  
 LiNode \* preLiNode(LiNode \* n);//find the particular node t's pre  
 bool enList(task t);//insert t to frt  
 bool deList();//delete the last node  
 bool dele(LiNode \* t);//delete the particular node t  
};  
class my\_List\_B {  
public:  
 LiNode\_B \* frt;  
 LiNode\_B \* lst;  
 int length;  
 my\_List\_B();//initial  
 ~my\_List\_B();//delete  
 bool empty();//return true is empty  
 bool enList(task t);//insert t to last  
 bool deList();//delete the last node  
 LiNode\_B \* dete(LiNode\_B \* t);//delete the particular node t  
 bool sortList();//sort as task memory  
 bool compare(my\_List\_B &b,my\_List &ndl);//compare two memory standing for all task in each second  
};  
  
  
  
#endif //DESIGN\_MY\_LIST\_H

my\_List.cpp

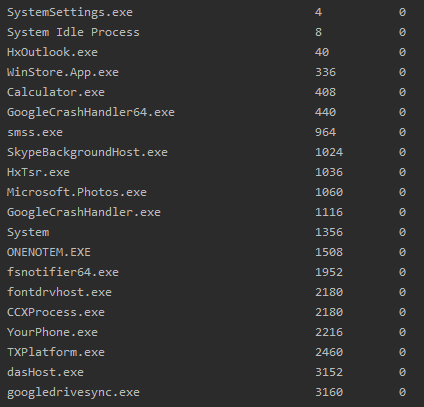
//  
// Created by 97927 on 2019/12/29.  
//  
  
#include "my\_List.h"  
#include <ctime>  
  
my\_List\_B::my\_List\_B() {  
 frt = nullptr;  
 lst = nullptr;  
 length = 0;  
}  
  
my\_List\_B::~my\_List\_B() {  
 LiNode\_B \* p = frt;  
 while (p)  
 {  
 LiNode\_B \* q = p->next;  
 delete p;  
 p = q;  
 }  
 frt = nullptr;  
 lst = nullptr;  
}  
  
bool my\_List\_B::enList(task t) {  
 length++;  
 LiNode\_B \* nnd = new LiNode\_B(t);//new node  
 if(!nnd)  
 return false;  
 if(!frt)//if list is empty  
 {  
 frt = nnd;  
 lst = nnd;  
 } else{  
 lst->next = nnd;  
 nnd->pre = lst;  
 lst = nnd;  
 }  
 return true;  
}  
  
bool my\_List\_B::deList() {  
 if(!lst)  
 {  
 return false;  
 } else{  
 LiNode\_B \* p = lst->pre;  
 delete lst;  
 if(p)//if last' pre node exist  
 {  
 p->next = nullptr;  
 lst = p;  
 } else{  
 frt = nullptr;  
 lst = nullptr;  
 }  
 }  
 length--;  
 return true;  
}  
  
bool my\_List\_B::sortList() {  
 //selection sort  
 LiNode\_B \* p = frt;  
 while(p)  
 {  
 int min = p->data.memory;//store min memory  
 LiNode\_B \* min\_s = p;//store min site  
 LiNode\_B \* q = p->next;  
 while(q)  
 {  
 if(q->data.memory < min)//if q is miner than min , swap  
 {  
 min = q->data.memory;  
 min\_s = q;  
 }  
 q = q->next;  
 }  
 if(min\_s == p)  
 {  
 p = p->next;  
 continue;  
 }  
 min\_s->pre->next = min\_s->next;  
 if(min\_s->next)//if min is last  
 min\_s->next->pre = min\_s->pre;  
 else  
 lst = min\_s->pre;  
 if(p->pre)  
 {  
 p->pre->next = min\_s;  
 min\_s->pre = p->pre;  
 }  
 min\_s->next = p;  
 p->pre = min\_s;  
 if(p == frt)//if p is first  
 {  
 frt = min\_s;  
 }  
  
 }  
 return true;  
}  
  
bool my\_List\_B::compare(my\_List\_B &b,my\_List &ndl) {  
 LiNode\_B \* p = frt;  
 //compare this and b  
 while(p)  
 {  
 LiNode\_B \* q = b.frt;  
 while(q)//weather the task in this is also in b  
 {  
 if(strcmp(q->data.name,p->data.name) == 0)  
 {  
 break;  
 }  
 q = q->next;  
 }  
 //find fail  
 if(q == nullptr)  
 {  
 std::time(&p->data.end);//note the ending time  
 ndl.enList(p->data);//push in ending list  
 p = dete(p);//delete this task  
 }  
 else{  
 p->data.lastTime++;//last time ++  
 p->data.memory = q->data.memory;//refresh memory  
 p = p->next;  
 b.dete(q);//delete this task in b  
 }  
 }  
 //compare b and ending list  
 if(!b.empty())  
 {  
 LiNode\_B \* q = b.frt;  
 while(q)  
 {  
 LiNode \* pause = ndl.frt;  
 while(pause)//weather this new task have run  
 {  
 if(strcmp(q->data.name,pause->data.name) == 0)  
 {  
 break;  
 }  
 pause = pause->next;  
 }  
 if(pause == nullptr)  
 {  
 } else{  
 //have run  
 q->data.lastTime = pause->data.lastTime;//refresh last time  
 ndl.dele(pause);  
 }  
 enList(q->data);//push in this list  
 q = q->next;  
 }  
 }  
  
 return true;  
}  
  
bool my\_List\_B::empty() {  
 return frt == nullptr;  
}  
  
LiNode\_B \*my\_List\_B::dete(LiNode\_B \*t) {  
 length--;  
 if(t == frt)  
 {  
 frt = t->next;  
 }  
 if(t == lst)  
 {  
 lst = t->pre;  
 }  
 LiNode\_B \* preNode = t->next;  
 if(t->pre)  
 t->pre->next = t->next;  
 if(t->next)  
 t->next->pre = t->pre;  
 delete t;  
 return preNode;  
}  
  
my\_List::my\_List() {  
 frt = nullptr;  
 lst = nullptr;  
 length = 0;  
}  
  
my\_List::~my\_List() {  
 LiNode \* p = frt;  
 while (p)  
 {  
 LiNode \* q = p->next;  
 delete p;  
 p = q;  
 }  
 frt = nullptr;  
 lst = nullptr;  
}  
  
bool my\_List::enList(task t) {  
 length++;  
 LiNode \* nnd = new LiNode(t);  
 if(!nnd)  
 return false;  
 nnd->next = frt;  
 frt = nnd;  
 if(!lst)  
 lst = nnd;  
 return true;  
}  
  
bool my\_List::deList() {  
 length--;  
 if(!lst)  
 {  
 return false;  
 } else{  
 LiNode \* p = preLiNode(lst);  
 delete lst;  
 if(p)  
 {  
 p->next = nullptr;  
 lst = p;  
 } else{  
 frt = nullptr;  
 lst = nullptr;  
 }  
 return true;  
 }  
}  
  
LiNode \*my\_List::preLiNode(LiNode \*n) {  
 LiNode \* p = frt;  
 while(p->next)  
 {  
 if(p->next == n)  
 break;  
 p = p->next;  
 }  
 if(p->next == n)  
 return p;  
 return nullptr;  
}  
  
bool my\_List::dele(LiNode \*t) {  
 length--;  
 LiNode \* pr = preLiNode(t);  
 if(pr)  
 {  
 pr->next = t->next;  
 }  
 if(t == frt)  
 {  
 frt = t->next;  
 }  
 if(t == lst)  
 {  
 lst = pr;  
 }  
 delete t;  
  
 return true;  
}  
  
LiNode::~LiNode() = default;  
  
LiNode\_B::~LiNode\_B() = default;

task1.cpp

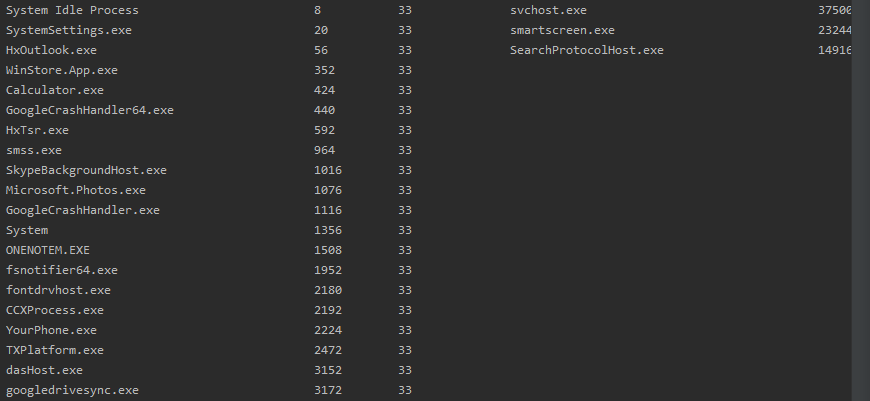
//  
// Created by 97927 on 2019/12/29.  
//  
#include <iostream>  
#include <iomanip>  
#include<list>  
#include <cstdlib>  
#include "my\_List.h"  
#include <windows.h>  
using namespace std;  
  
#define SLEEPTIME 2000  
//#define YTD  
  
#ifndef YTD  
int main() {  
 //list to store task last time  
 my\_List\_B q;  
 //list to store task end  
 my\_List ndl;  
 cout<<std::left;  
 while(true)  
 {  
 std::system("clear");  
 cout<<endl;  
 char buffer[128], cmd[] = "tasklist /FO CSV";  
 //store task now  
 my\_List\_B b;  
 FILE\* pipe = popen(cmd, "r");  
 if (!pipe)  
 return -1;  
  
 fgets(buffer, 128, pipe);  
 //last is empty , store in last and do not compare  
 if(q.empty())  
 {  
 while (!feof(pipe))  
 {  
 if (fgets(buffer, 128, pipe))  
 {  
 q.enList(task(buffer));  
 }  
 }  
 }  
 //compare two seconds task  
 else{  
 while (!feof(pipe))  
 {  
 if (fgets(buffer, 128, pipe))  
 {  
 b.enList(task(buffer));  
 }  
 }  
 q.compare(b,ndl);  
 }  
  
 pclose(pipe);  
 //sort as for memory  
 q.sortList();  
 LiNode \* nd = ndl.frt;  
 int k = 0;  
 for (LiNode\_B \* it = q.frt;k < 20; it = it->next,k++)  
 {  
 cout <<setw(40)<< it->data.name << "\t" ;  
 cout<<setw(10)<< it->data.memory<<"\t"<<it->data.lastTime;  
 if(nd)  
 {  
 cout <<"\t\t\t\t"<<setw(40)<< nd->data.name << "\t" ;  
 cout<<setw(10)<< nd->data.memory<<"\t"<<nd->data.end;  
 nd = nd->next;  
 }  
 cout<<endl;  
 }  
 Sleep(SLEEPTIME);  
 }  
 return 0;  
}  
#endif

###### 运行结果：

初始运行：



有任务结束时：



### 必做题第二题

###### 题目：2、算术表达式求值 (必做) （栈）

[问题描述]

一个算术表达式是由操作数(operand)、运算符(operator)和界限符(delimiter)组成的。假设操作数是正实数，运算符只含加减乘除等四种运算符，界限符有左右括号和表达式起始、结束符“#”，如：#6+15\*（21-8/4）#。引入表达式起始、结束符是为了方便。编程利用“运算符优先法”求算术表达式的值。

[基本要求]

（1） 从键盘或文件读入一个合法的算术表达式，输出正确的结果。

（2） 显示输入序列和栈的变化过程。

（3） 考虑算法的健壮性，当表达式错误时，要给出错误原因的提示。

（4） 实现非整数的处理（\*）。

###### 数据结构：栈

###### 算法思想：

建立两个栈  
stack1存放数字  
stack2存放运算符  
一个个读入字符，进行判断：  
1.该字符为数字，进行数字统计，统计后加入stack1  
2.读取到运算符  
 2.1.如果运算符栈为空——直接加入stack2  
 2.2.如果该运算符为（——直接进栈  
 2.3.如果该运算符为）——取出栈顶元素，如果不是（，则取出数字栈栈顶两个数字进行运算，结果存入数字栈 中，重复此过程直到取出了（  
 2.4.取出运算符栈的栈顶元素与该运算符比较，若栈顶元素的优先级不小于该运算符，则取出数字栈的两个栈顶 元素进行运算，结果存入数字栈中，继续该过程，直到栈为空或者栈顶元素优先级小于该运算符  
  
3.读取到#，结束读取  
  
取出stack1中两个数字，stack2中一个运算符进行运算，结果存储在stack1中：  
1.当stack1只剩余一个数字，stack2为空，运算结束，输出结果，结束程序  
2.当stack1大于一个数字，stack2为空，输入的运算符过少  
3.当stack1只剩余一个数字，stack2不为空，输入中有连续的运算符或以运算符开头/结尾  
4.有（，没有输入足够的）  
  
输出错误原因

###### 实现过程：

my\_Stack.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_MY\_STACK\_H  
#define DESIGN\_MY\_STACK\_H  
  
#include <vector>  
  
class StNode{  
public:  
 double data;  
 StNode(double n = 0);  
 ~StNode();  
};  
class my\_stack {  
private:  
 std::vector<StNode> frt;  
 int length;  
public:  
 my\_stack();//initial  
 bool empty();//true while stack is empty  
 int getlength();//get stack length at this time  
 void clearStack();//clear stack  
 bool getTop(double \* e);//get top num and store at e  
 bool getTop(int \* e);  
 double getTop();  
 bool push(double e);//push back e  
 bool push(int e);  
 bool pop(double \* e);//pop top elem and store at e  
 double pop();  
 bool pop(int \* e);  
 bool traverse();//show all stack elem  
};  
  
  
#endif //DESIGN\_MY\_STACK\_H

my\_Stack.cpp

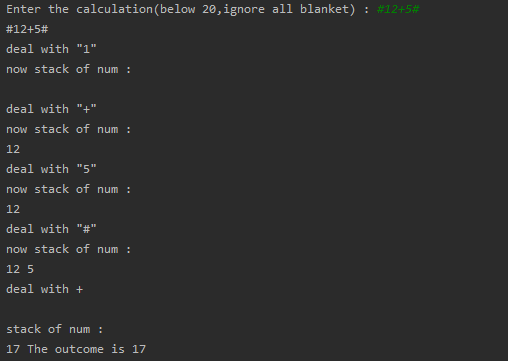
//  
// Created by YTD on 2019/12/13.  
//  
  
#include "my\_stack.h"  
#include <iostream>  
  
StNode::~StNode() = default;  
  
my\_stack::my\_stack() {  
 length = 0;  
}  
  
bool my\_stack::empty() {  
 return length == 0;  
}  
  
int my\_stack::getlength() {  
 return length;  
}  
  
void my\_stack::clearStack() {  
 frt.clear();  
 length = 0;  
}  
  
bool my\_stack::getTop(double \*e) {  
 if(length == 0)  
 {  
 return false;  
 }  
 \*e = frt[length - 1].data;  
  
 return true;  
}  
  
bool my\_stack::push(double e) {  
 StNode nw\_node(e);  
 frt.push\_back(nw\_node);  
 length++;  
  
 return true;  
}  
bool my\_stack::push(int e) {  
 StNode nw\_node(e);  
 frt.push\_back(nw\_node);  
 length++;  
  
 return true;  
}  
  
bool my\_stack::pop(double \*e) {  
 if(length == 0)  
 return false;  
 \*e = frt[length - 1].data;  
 length--;  
 frt.pop\_back();  
  
 return true;  
}  
bool my\_stack::pop(int \*e) {  
 if(length == 0)  
 return false;  
 \*e = frt[length - 1].data;  
 length--;  
 frt.pop\_back();  
  
 return true;  
}  
  
bool my\_stack::traverse() {  
 using std::cout;  
 using std::endl;  
 if(length == 0)  
 return false;  
 for(int i = 0;i < length;i++)  
 {  
 //cout<<"data "<<i<<" : "<<frt[i].data<<" ";  
 cout<<frt[i].data<<" ";;  
 /\*if(i%10 < 1)  
 cout<<endl;\*/  
 }  
  
 return true;  
}  
  
bool my\_stack::getTop(int \*e) {  
 if(length == 0)  
 {  
 return false;  
 }  
 \*e = frt[length - 1].data;  
  
 return true;  
}  
  
double my\_stack::getTop() {  
 if(length == 0)  
 {  
 return -1;  
 }  
 return frt[length - 1].data;  
}  
  
double my\_stack::pop() {  
 if(length == 0)  
 return -1;  
 double e = frt[length - 1].data;  
 length--;  
 frt.pop\_back();  
  
 return e;  
}  
  
StNode::StNode(double n) {  
 data = n;  
}

task2.cpp

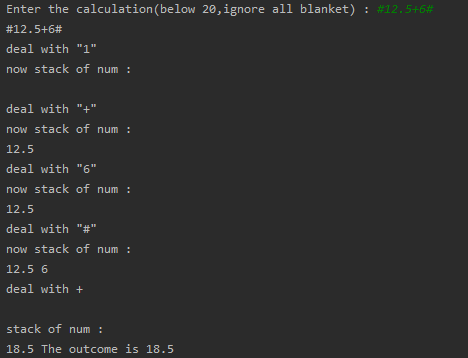
//  
// Created by YTD on 2019/12/19.  
//  
#include <iostream>  
#include <cstdlib>  
#include <cstdio>  
#include "stack"  
#include "my\_stack.h"  
#include <string>  
#include "function.h"  
  
using namespace std;  
//#define YTD  
double midToOut(string str)  
{  
 stack<char> oprt1;// stack operators  
 my\_stack nmb;// stack number  
 int i = str.find('#');//find first #  
 if(i == string::npos)//do not have #  
 {  
 cout<<"Don't have # at beginning."<<endl;  
 return -1;  
 }  
  
 int len = str.length();  
 i++;  
 while(i < len)  
 {  
 //output the char this circle get and the stack number  
 cout<<"deal with \""<<str[i]<<"\""<<endl;  
 cout<<"now stack of num :"<<endl;  
 nmb.traverse();  
 cout<<endl;  
 //if the char is # , end  
 if(str[i] == '#')  
 {  
 break;  
 }  
 //generate number  
 if(str[i] >= '0' && str[i] <= '9')  
 {  
 double integer = 0;  
 double decimal = 0;  
 bool flag = true;  
 for(double k = 0.1;str[i] == ' ' || str[i] == '.' ||(str[i] >= '0' && str[i] <= '9');i++) {  
 if (str[i] == ' ')//skip the blanket  
 continue;  
 if(str[i] == '.')//sign of the decimal part  
 {  
 flag = false;  
 continue;  
 }  
 if(str[i] - '0' > 9)  
 {  
 cout<<"enter wrong operate."<<endl;  
 return -1;  
 }  
 if (flag)  
 {  
 integer \*= 10;  
 integer += str[i] - '0';  
 } else{  
 decimal += (str[i] - '0') \* k;  
 k \*= 0.1;  
 }  
 }  
 nmb.push(integer + decimal);  
 continue;//do not need to add i  
 }  
 //skip the blanket  
 else if(str[i] == ' '){  
 }  
 //right brackets  
 else if(str[i] == ')')  
 {  
 if(oprt1.empty())  
 {  
 cout<<"Don have enough (."<<endl;  
 return -1;  
 }  
 char pause = oprt1.top();  
 while(pause != '(')  
 {  
 //do not have enough numberd  
 if(nmb.getlength() < 2)  
 {  
 cout<<"Too many operators."<<endl;  
 return -1;  
 }  
 double n1 = nmb.pop();  
 double n2 = nmb.pop();  
 n1 = computeTwoNumber(n2,n1,pause);  
 if(n1 == -65535)  
 {  
 cout<<"divide by 0"<<endl;  
 return -1;  
 }  
 nmb.push(n1);  
 oprt1.pop();  
 if(oprt1.empty())  
 {  
 cout<<"Don have enough (."<<endl;  
 return -1;  
 }  
 pause = oprt1.top();  
 }  
 oprt1.pop();  
 }  
 //operators + - / \* (  
 else if(str[i] == '(' || str[i] == '+' || str[i] == '-' || str[i] == '/' || str[i] == '\*'){  
 char pause;  
 if(!oprt1.empty())  
 pause = oprt1.top();  
 //empty or left brackets add  
 if(oprt1.empty() || pause == '(' || str[i] == '(')  
 {  
 oprt1.push(str[i]);  
 } else if(compareOperate(pause,str[i]))  
 {  
 oprt1.push(str[i]);  
 } else{  
 while(!compareOperate(pause,str[i]))  
 {  
 if(nmb.getlength() < 2)  
 {  
 cout<<"Too many operators."<<endl;  
 return -1;  
 }  
 double n1 = nmb.pop();  
 double n2 = nmb.pop();  
 n1 = computeTwoNumber(n2,n1,pause);  
 if(n1 == -65535)  
 {  
 cout<<"divide by 0"<<endl;  
 return -1;  
 }  
 nmb.push(n1);  
 oprt1.pop();  
 if(oprt1.empty())  
 break;  
 pause = oprt1.top();  
 }  
 oprt1.push(str[i]);  
 }  
 }  
 //other opraters  
 else{  
 cout<<"enter wrong operate."<<endl;  
 return -1;  
 }  
 i++;  
 }  
 //finish the handle of string  
 while(!oprt1.empty())  
 {  
 char pause = oprt1.top();  
 cout<<"deal with "<<pause<<endl;  
 if(pause == '(')  
 {  
 cout<<"Dont't enough )."<<endl;  
 return -1;  
 }  
 if(nmb.getlength() < 2)  
 {  
 cout<<"End by operator."<<endl;  
 return -1;  
 }  
 double n1 = nmb.pop();  
 double n2 = nmb.pop();  
 n1 = computeTwoNumber(n2,n1,pause);  
 if(n1 == -65535)  
 {  
 cout<<"divide by 0"<<endl;  
 return -1;  
 }  
 nmb.push(n1);  
 oprt1.pop();  
 cout<<endl<<"stack of num :"<<endl;  
 nmb.traverse();  
 }  
 return nmb.getTop();  
}  
#ifndef YTD  
int main()  
{  
 std::string str = getString("calculation(below 20,ignore all blanket)");  
 my\_stack nmb;  
 cout<<str<<endl;  
 double outcome = midToOut(str);  
 if(outcome != -1)  
 {  
 cout<<"The outcome is "<<outcome<<endl;  
 } else{  
 cout<<"Error"<<endl;  
 }  
  
  
 return 0;  
}  
#endif

###### 测试样例与运行结果：

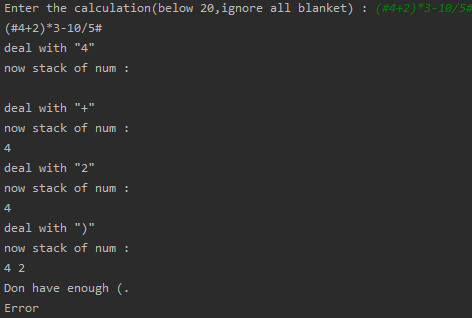
当输入正确整数样例：#12+5#



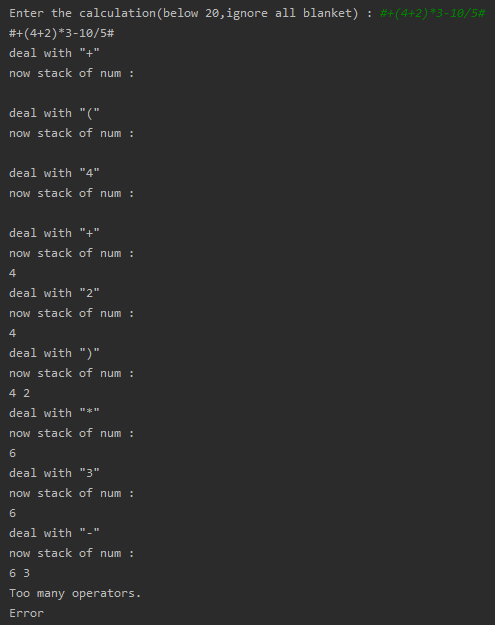
当输入正确的小数样例：#12.5+6#



当输入错误样例：(#4+2)\*3-10/5#



当输入错误样例：



### 必做题第三题

###### 题目：公共钥匙盒（必做）（线性表，栈，队列）

[问题描述]

有一个学校的老师共用N个教室，按照规定，所有的钥匙都必须放在公共钥匙盒里，老师不能带钥匙回家。每次老师上课前，都从公共钥匙盒里找到自己上课的教室的钥匙去开门，上完课后，再将钥匙放回到钥匙盒中。

钥匙盒一共有N个挂钩，从左到右排成一排，用来挂N个教室的钥匙。一串钥匙没有固定的悬挂位置，但钥匙上有标识，所以老师们不会弄混钥匙。

每次取钥匙的时候，老师们都会找到自己所需要的钥匙将其取走，而不会移动其他钥匙。每次还钥匙的时候，还钥匙的老师会找到最左边的空的挂钩，将钥匙挂在这个挂钩上。如果有多位老师还钥匙，则他们按钥匙编号从小到大的顺序还。如果同一时刻既有老师还钥匙又有老师取钥匙，则老师们会先将钥匙全还回去再取出。

今天开始的时候钥匙是按编号从小到大的顺序放在钥匙盒里的。有K位老师要上课，给出每位老师所需要的钥匙、开始上课的时间和上课的时长，假设下课时间就是还钥匙时间，请问最终钥匙盒里面钥匙的顺序是怎样的？

[基本要求]

输入格式

输入的第一行包含两个整数N, K。

接下来K行，每行三个整数w, s, c，分别表示一位老师要使用的钥匙编号、开始上课的时间和上课的时长。可能有多位老师使用同一把钥匙，但是老师使用钥匙的时间不会重叠。

保证输入数据满足输入格式，你不用检查数据合法性。

输出格式

输出一行，包含N个整数，相邻整数间用一个空格分隔，依次表示每个挂钩上挂的钥匙编号。

样例输入

5 2

4 3 3

2 2 7

样例输出

1 4 3 2 5

样例说明

第一位老师从时刻3开始使用4号教室的钥匙，使用3单位时间，所以在时刻6还钥匙。第二位老师从时刻2开始使用钥匙，使用7单位时间，所以在时刻9还钥匙。

每个关键时刻后的钥匙状态如下（X表示空）：

时刻2后为1X345；

时刻3后为1X3X5；

时刻6后为143X5；

时刻9后为14325。

课程设计要求：

（1）要求从文本文件中输入；

（2）根据时间进程，将取走钥匙和归还钥匙分别视为事件，放入队列中，然后通过每个事件的先后发生对钥匙盒的状态进行变更；

（3）严格按照要求的输入输出格式进行数据的输入、输出（训练CSP考试中的格式化输入输出的正确性）；

（4）选做：通过图形界面来显示钥匙盒的即时状态，以及事件队列的状态。

###### 数据结构：队列

###### 算法思想：通过文件中输入的数据建立时刻表，按时间顺序递增检查该时间是否发生事件

建立box类存储钥匙  
建立事件队列q  
从文件中读取初始钥匙数建立box  
从文件中读取事件开始时间，结束时间，取的钥匙建立事件队列q  
对所有事件按发生时间顺序排序  
进行时间递增，观察该时间点是否有事件发生：  
 1.无事件发生，继续下一个时间  
 2.如果有事件的结束时间在该时刻，先将钥匙放入box，将该事件删除  
 3.如果有事件的开始时间在该时刻，将钥匙从box中取出

###### 具体实现：

my\_Queue.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_MY\_QUEUE\_H  
#define DESIGN\_MY\_QUEUE\_H  
  
class QNode{  
public:  
 int data;  
 QNode \* next;  
 QNode(int n = 0,QNode\* nxt = nullptr);  
 ~QNode();  
};  
class Qnode\_more :public QNode {  
public:  
 int data1;//time begin  
 int data2;//time eng  
 Qnode\_more \* frnt;  
 Qnode\_more \* next;  
 Qnode\_more(int n1 = 0,int n2 = 0,int n3 =0):QNode(n1){//initially  
 data1 = n2;  
 data2 = n3;  
 frnt = nullptr;  
 next = nullptr;  
 };  
 ~Qnode\_more()= default;  
};  
class my\_Queue {  
private:  
 QNode \* frt;  
 QNode \* lst;  
 int length;  
public:  
 my\_Queue();  
 bool empty();  
/\*Length Finish\*/  
 int getlength();  
/\*Get Finish\*/  
 bool getHead(int \*e);  
/\*Clear Finish\*/  
 void clearQueue();  
/\*En Finsh\*/  
 bool enQueue(int e);  
/\*De Finish\*/  
 bool deQueue(int \*e);  
/\*Traverse Finsh\*/  
 bool traverse();  
 ~my\_Queue();  
};  
class my\_Queue\_Plus{  
private:  
 Qnode\_more \* frt;  
 Qnode\_more \* lst;  
 int length;  
  
public:  
 my\_Queue\_Plus();//initially  
 bool empty();//true when all thing finish  
 int getlength();//return length  
 bool getHead(int \*e);//get the first thing  
 void clearQueue();  
 bool enQueue(int e1,int e2,int e3);//insert thing to back  
 bool deQueue(int \*e,int \*e2,int \*e3);//pop the first thing  
 bool traverse();//show queue  
 bool sort();//sort as begin time  
 int lastTime();//return the last thing end time  
 bool timeEvent(int t,my\_Queue\_Plus & q);//deal with time n  
 ~my\_Queue\_Plus();  
};  
  
#endif //DESIGN\_MY\_QUEUE\_H

my\_Queue.cpp

//  
// Created by 97927 on 2019/12/13.  
//  
  
#include "my\_Queue.h"  
#include <iostream>  
  
bool my\_Queue::empty() {  
 return length == 0;  
}  
  
my\_Queue::my\_Queue() {  
 frt = new QNode();  
 lst = frt;  
 length = 0;  
}  
  
int my\_Queue::getlength() {  
 return length;  
}  
  
bool my\_Queue::getHead(int \*e) {  
 if(empty())  
 return false;  
 \*e = frt->next->data;  
 return true;  
}  
  
void my\_Queue::clearQueue() {  
 QNode \* p = frt->next;  
 while(p)  
 {  
 QNode \* m = p->next;  
 delete p;  
 p = m;  
 }  
 lst = frt;  
 length = 0;  
}  
  
bool my\_Queue::enQueue(int e) {  
 QNode \* p = new QNode;  
 if(!p)  
 return false;  
  
 p->data = e;  
 p->next = nullptr;  
 lst->next = p;  
 lst = p;  
 length++;  
  
 return true;  
}  
  
bool my\_Queue::deQueue(int \*e) {  
 QNode \* p = frt->next;  
 if(p)  
 {  
 if(p == lst)  
 {  
 lst = frt;  
 }  
 \*e = p->data;  
 frt->next = p->next;  
 delete p;  
 }  
 else  
 return false;  
 length--;  
 return true;  
}  
  
bool my\_Queue::traverse() {  
 using std::cout;  
 using std::endl;  
 QNode \* p = frt->next;  
 int k = 1;  
 while(p)  
 {  
 cout<<"data "<<k<<" : "<<p->data<<" ";  
 p = p->next;  
 k++;  
 if(k % 10 == 0)  
 cout<<endl;  
 }  
 cout<<"queue traverse finish."<<endl;  
 return true;  
}  
  
my\_Queue::~my\_Queue() {  
 QNode \* p = frt;  
 while(p)  
 {  
 QNode \* m = p->next;  
 delete p;  
 p = m;  
 }  
 lst = nullptr;  
 frt = nullptr;  
 length = 0;  
}  
  
QNode::QNode(int n,QNode \* nxt) {  
 data = n;  
 QNode::next = nxt;  
}  
  
QNode::~QNode() = default;  
  
  
my\_Queue\_Plus::my\_Queue\_Plus() {  
 frt = new Qnode\_more();  
 lst = frt;  
 length = 0;  
}  
  
bool my\_Queue\_Plus::empty() {  
 return length == 0;  
}  
  
int my\_Queue\_Plus::getlength() {  
 return length;  
}  
  
bool my\_Queue\_Plus::getHead(int \*e) {  
 if(empty())  
 return false;  
 \*e = frt->next->data;  
 return true;  
}  
  
void my\_Queue\_Plus::clearQueue() {  
 Qnode\_more \* p = frt->next;  
 while(p)  
 {  
 Qnode\_more \* m = p->next;  
 delete p;  
 p = m;  
 }  
 lst = frt;  
 length = 0;  
}  
  
bool my\_Queue\_Plus::enQueue(int e1, int e2, int e3) {  
 Qnode\_more \* p = new Qnode\_more;  
 if(!p)  
 return false;  
  
 p->data = e1;  
 p->data1 = e2;  
 p->data2 = e3;  
 p->frnt = lst;  
 p->next = nullptr;  
 length++;  
 lst->next = p;  
 lst = p;  
  
 return true;  
}  
  
bool my\_Queue\_Plus::deQueue(int \*e, int \*e2, int \*e3) {  
 using std::cout;  
 using std::endl;  
 Qnode\_more \* p = frt->next;  
 if(p)  
 {  
 if(p == lst)  
 {  
 lst = frt;  
 }  
 \*e = p->data;  
 \*e2 = p->data1;  
 \*e3 = p->data2;  
 frt->next = p->next;  
 if(p->next)  
 p->next->frnt = frt;  
 delete p;  
 }  
 else  
 return false;  
 length--;  
 return true;  
}  
  
bool my\_Queue\_Plus::traverse() {  
 using std::cout;  
 using std::endl;  
 Qnode\_more \* p = frt->next;  
 int k = 1;  
 while(p)  
 {  
 cout<<"data "<<k<<" : "<<p->data<<","<<p->data1<<","<<p->data2 - p->data1<<" ";  
 p = p->next;  
 k++;  
 if(k % 10 == 0)  
 cout<<endl;  
 }  
 cout<<"queue traverse finish."<<endl;  
 return true;  
}  
  
my\_Queue\_Plus::~my\_Queue\_Plus(){  
 Qnode\_more \* p = frt;  
 while(p)  
 {  
 Qnode\_more \* m = p->next;  
 delete p;  
 p = m;  
 }  
 lst = nullptr;  
 frt = nullptr;  
 length = 0;  
}  
  
bool my\_Queue\_Plus::sort() {  
 Qnode\_more \* p = frt->next;  
 while(p)  
 {  
 Qnode\_more \* min = p;  
 Qnode\_more \* q = p->next;  
 while(q)  
 {  
 //find the min begin time  
 if(q->data1 < min->data1)  
 {  
 min = q;  
 }  
 //if begin time is equal , compare end time  
 else if(q->data1 == min->data1)  
 {  
 if(q->data2 < min->data2)  
 {  
 min = q;  
 }  
 else if(q->data2 == min->data2)  
 {  
 if(q->data < min->data)  
 {  
 min = q;  
 }  
 }  
 }  
 q = q->next;  
 }  
 if(min == p)  
 {  
 p = p->next;  
 continue;  
 }  
 min->frnt->next = min->next;  
 if(min->next)  
 min->next->frnt = min->frnt;  
 p->frnt->next = min;  
 min->frnt = p->frnt;  
 p->frnt = min;  
 min->next = p;  
 }  
  
 return true;  
}  
  
int my\_Queue\_Plus::lastTime() {  
 return lst->data2;  
}  
//find event happen at this second  
bool my\_Queue\_Plus::timeEvent(int t,my\_Queue\_Plus & q) {  
 using std::cout;  
 using std::endl;  
 Qnode\_more \* p = frt->next;  
 bool reFlag = false;  
 if(p->data1 > t)  
 return false;  
 while(p)  
 {  
 if(p->data1 > t)  
 break;  
 if(p->data1 == t || p->data2 == t)  
 {  
 int key\_n = p->data;  
 int ta\_t = p->data1;  
 int re\_t = p->data2;  
 q.enQueue(key\_n,ta\_t,re\_t);  
 reFlag = true;  
 }  
 //if this event is end , pop  
 else if(p->data2 < t)  
 {  
 p->frnt->next = p->next;  
 if(p->next)  
 p->next->frnt = p->frnt;  
 cout<<"event "<<" : "<<p->data<<","<<p->data1<<","<<p->data2 - p->data1<<" end"<<endl;  
 length--;  
 Qnode\_more \* pause = p->next;  
 delete p;  
 p = pause;  
 continue;  
 }  
 p = p->next;  
 }  
  
 return reFlag;  
}

task3.cpp

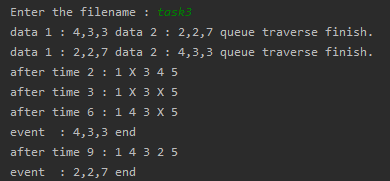
//  
// Created by 97927 on 2019/12/21.  
//  
#include <iostream>  
#include <cstdlib>  
#include <cstdio>  
#include <fstream>  
#include "my\_Queue.h"  
#include <string>  
#include "function.h"  
using namespace std;  
  
//#define YTD  
  
class box{  
private:  
 vector<int> key\_store;//box to store key  
 int box\_s;//box size  
public:  
 box(int n = 1)//initially  
 {  
 box\_s = n;  
 for(int i = 0;i < n;i++)  
 {  
 key\_store.push\_back(i + 1);  
 }  
 }  
 bool pop(int n)//pop key n  
 {  
 for(int i = 0;i < box\_s;i++)  
 {  
 //find key site  
 if(key\_store[i] == n)  
 {  
 //pop  
 key\_store[i] = 0;  
 return true;  
 }  
 }  
 return false;  
 }  
 bool push(int n)//push key n  
 {  
 for(int i = 0;i < box\_s;i++)  
 {  
 //find box empty site  
 if(key\_store[i] == 0)  
 {  
 //push  
 key\_store[i] = n;  
 return true;  
 }  
 }  
 return false;  
 }  
 bool show()//show box  
 {  
 for(int i = 0;i < box\_s;i++)  
 {  
 if(key\_store[i] == 0)  
 {  
 cout<<"X ";  
 } else{  
 cout<<key\_store[i]<<" ";  
 }  
 }  
 cout<<endl;  
  
 return true;  
 }  
 bool show(ofstream &out)//show out to file  
 {  
 for(int i = 0;i < box\_s;i++)  
 {  
 if(key\_store[i] == 0)  
 {  
 out<<"X ";  
 } else{  
 out<<key\_store[i]<<" ";  
 }  
 }  
 out<<endl;  
  
 return true;  
 }  
 ~box()  
 {  
 key\_store.clear();  
 }  
};  
bool timePass(box &b,my\_Queue\_Plus &p,ofstream &out)  
{  
 int n = p.lastTime();//get end time  
 int i = 0;  
 my\_Queue\_Plus q;  
 while(!p.empty())  
 {  
 i++;  
 if(!p.timeEvent(i,q))//this time do not happen any things  
 {  
 continue;  
 }  
 while(!q.empty())//deal with things happen this time  
 {  
 int key\_n = 0;  
 int ta\_t = 0;  
 int re\_t = 0;  
 q.deQueue(&key\_n,&ta\_t,&re\_t);  
 if(ta\_t == i)//if star time == this time  
 {  
 b.pop(key\_n);  
 } else if (re\_t == i)//if end time == this time  
 {  
 b.push(key\_n);  
 }  
 }  
 cout<<"after time "<<i<<" : ";  
 out<<"after time "<<i<<" : ";  
 b.show();  
 b.show(out);  
 q.clearQueue();  
 }  
  
 return true;  
}  
#ifndef YTD  
int main()  
{  
 //open the file to input and output  
 ifstream in\_tx;  
 ofstream out\_tx("task3result.txt");  
 std::string fileName = getString("filename");  
 fileName += ".txt";  
 in\_tx.open(fileName);  
 if(!in\_tx.is\_open())  
 {  
 cout<<"wrong"<<endl;  
  
 return 0;  
 }  
 //init keybox and timetable  
 int total\_long = 0;  
 in\_tx>>total\_long;  
 box B(total\_long);  
 my\_Queue\_Plus timetable;  
 int n;  
 in\_tx>>n;  
 for(int i = 0;i < n;i++)  
 {  
 int key\_n;  
 int ta\_t;  
 int re\_t;  
 in\_tx>>key\_n>>ta\_t>>re\_t;  
 timetable.enQueue(key\_n,ta\_t,re\_t + ta\_t);  
 }  
 //output timetable  
 timetable.traverse();  
 //sort timetable  
 timetable.sort();  
 //output result  
 timetable.traverse();  
 timePass(B,timetable,out\_tx);  
 in\_tx.close();  
 out\_tx.close();  
 return 0;  
}  
#endif

###### 测试样例与运行结果：

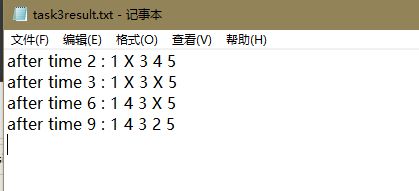
文件中数据：

5 2 4 3 3 2 2 7

运行结果：



存储在文件中的结果：



### 必做题第四题

###### 题目：家谱管理系统（必做） （树）

[问题描述]

实现具有下列功能的家谱管理系统。

[基本要求]

（1）输入文件以存放最初家谱中各成员的信息，成员的信息中均应包含以下内容：姓名、出生日期、婚否、地址、健在否、死亡日期（若其已死亡），也可附加其它信息、但不是必需的。

（2）实现数据的文件存储和读取。

（3）以图形方式显示家谱。

（4）显示第n 代所有人的信息。

（5）按照姓名查询，输出成员信息（包括其本人、父亲、孩子的信息）。

（6）按照出生日期查询成员名单。

（7）输入两人姓名，确定其关系。

（8）某成员添加孩子。

（9）删除某成员（若其还有后代，则一并删除）。

（10）修改某成员信息。

（11）要求建立至少20个成员的数据，以较为直观的方式显示结果，并提供文稿形式以便检查。

（12）界面要求：有合理的提示，每个功能可以设立菜单，根据提示，可以完成相关的功能要求。

（13）存储结构：根据系统功能要求自行设计，但是要求相关数据要存储在数据文件中。测试数据：要求使用1、全部合法数据；2、局部非法数据。进行程序测试，以保证程序的稳定。

###### 数据结构：孩子兄弟树

###### 算法思想：

建立tree存储整个家谱信息  
从文件中读取信息建立初始家谱系统  
输出选项信息，进行功能选择：  
 1.增加一个人  
 2.删除一个人  
 3.展示一个人的所有信息  
 4.改变一个人的信息  
 5.展示某一层的所有信息  
 6.显示所有人  
 7.确定两个人的关系  
 8.按出生日期排序显示所有人  
1：调用tree中assign类函数进行增加  
2：调用tree中deletePeople类函数进行删除  
3：调用tree中find类函数进行查找  
4：获取需要改变的人名，输出改变选项  
 4.1.姓名  
 4.2.出生日期  
 4.3.地址  
 4.4.婚否  
 4.5.健在否  
 4.6.死亡日期  
 调用对应的类函数进行改变。在健在否由存活变为死亡死，自动调用更改死亡日期的类函数，获取该信息进行更改  
5：调用showTreeLayer类函数，利用层次遍历的思想，当层次等于相应的层次时，显示该层信息，显示完后退出  
6：调用showAll类函数按层次遍历的思想显示每一层的信息  
7：获取两个人A，B的姓名，通过类函数获取两个人的信息，比较两个人的辈分（layer）：  
 7.1.A的layer与B相等，两个人为兄弟  
 7.2.B的layer大于A：  
 7.2.1.B的父母为A，输出A为B的父母  
 7.2.2.B的父母的父母为A，输出A为B的祖父母  
 7.3.A的layer大于B：  
 7.3.1.A的父母为B，输出B为A的父母  
 7.3.2.A的父母的父母为B，输出B为A的zufumu  
 7.4.输出两人没有关系  
8.遍历整个树将每个节点的指针存储在数字中，用选择排序将数组按出生日期排序  
  
将现有的树按层次遍历输出到文件中

###### 算法时间复杂度：在按出生日期排序时，选择排序复杂度为（n^2)

###### 具体实现：

my\_BrotherTree.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_MY\_BROTHERTREE\_H  
#define DESIGN\_MY\_BROTHERTREE\_H  
  
  
#include <string>  
  
class my\_BrotherTree {  
private:  
 int layer;  
 std::string name;  
 my\_BrotherTree \* parent;  
 my\_BrotherTree \* child;  
 my\_BrotherTree \* brother;  
protected:  
 virtual my\_BrotherTree \* findPeople(std::string nm);//locate the people  
public:  
 my\_BrotherTree(std::string nm);//initial  
 my\_BrotherTree();//initial  
 virtual bool assign(my\_BrotherTree \* t,std::string parent\_name);//assign parent\_name chile  
 virtual bool showTreeLayer(int layer);//show the nth layer people  
 bool show();//show this node information  
 bool showLevel();//show all people information as level  
 int layers();//return layer  
 std::string names();//return name  
 bool showName();//show this node's name  
 bool showLayer();//shwo this node's layer  
 bool setLayer(int n);//change this node's layer  
 bool setName(std::string nm1);//change this node's name  
  
 ~my\_BrotherTree();  
};  
class my\_BrotherTree\_plus : public my\_BrotherTree{  
private:  
 std::string borndate;  
 std::string deathdate;  
 std::string addre;  
 bool marige;  
 bool live;  
 my\_BrotherTree\_plus \* parent;  
 my\_BrotherTree\_plus \* child;  
 my\_BrotherTree\_plus \* brother;  
protected:  
 my\_BrotherTree\_plus \* findPeople(std::string nm);//locate people  
public:  
 my\_BrotherTree\_plus(std::string nm,std::string born,\  
 std::string addre,bool marige,bool l,std::string death = "");//initial  
 my\_BrotherTree\_plus(std::string filename);//initial from file  
 bool assign(my\_BrotherTree\_plus \* t,std::string parent\_name);//assign parent\_name's child  
 bool showTreeLayer(int layer);//show nth layer people  
 bool deletePeople(std::string peo);//delete a people and all his childeren  
 bool find(std::string nm);//show a people all information  
 bool setBorn(std::string nm,std::string born);//set born data  
 bool setAddress(std::string nm,std::string add);//set address  
 bool setMar(std::string nm,bool mrg);//set marige  
 bool setLive(std::string nm,bool lf);//set live condition  
 bool setDeathdate(std::string nm,std::string death);//set death date  
 bool setNames(std::string nm,std::string nm1);//set name  
 bool showAll();//show all people as level  
 bool outToFile();//out to file  
 ~my\_BrotherTree\_plus();  
 bool showBron();//show as born date oder  
 int size();//return the number of nodes  
 int relationship(std::string nm1,std::string nm2);//relationship between two people  
};  
  
#endif //DESIGN\_MY\_BROTHERTREE\_H

my\_BrotherTree.cpp

//  
// Created by 97927 on 2019/12/13.  
//  
  
#include "my\_BrotherTree.h"  
#include <stack>  
#include <string>  
#include <iostream>  
#include <queue>  
#include <fstream>  
#include "function.h"  
#include <fstream>  
#include <iomanip>  
using namespace std;  
  
my\_BrotherTree::my\_BrotherTree(std::string nm) {  
 name = nm;  
 layer = 1;  
 parent = nullptr;  
 child = nullptr;  
 brother = nullptr;  
}  
  
bool my\_BrotherTree::assign(my\_BrotherTree \*t, std::string parent\_name) {  
 my\_BrotherTree \* pa = findPeople(parent\_name);  
 //set t's layer  
 t->layer = pa->layer + 1;  
 //if parent have child  
 if(pa->child)  
 {  
 pa = pa->child;  
 //find the last child  
 while(pa->brother)  
 {  
 pa = pa->brother;  
 }  
 pa->brother = t;  
 } else{  
 pa->child = t;  
 }  
 return true;  
}  
  
bool my\_BrotherTree::showLevel() {  
 using std::cout;  
 using std::endl;  
 //store the node  
 std::queue<my\_BrotherTree \*> q;  
 q.push(this);  
 while(!q.empty())  
 {  
 my\_BrotherTree \* t = q.front();  
 q.pop();  
 cout<<"name : "<<name<<" "<<layer;  
 if(t->parent)  
 cout<<" parent : "<<t->parent->name;  
 if(t->child)  
 {  
 cout<<endl;  
 //if child exist  
 my\_BrotherTree \* p = t->child;  
 while(p)  
 {  
 q.push(p);  
 cout<<p->name<<" ";  
 p = p->brother;  
 }  
 cout<<endl;  
 }  
 }  
 cout<<"Show level end."<<endl;  
  
 return true;  
}  
  
bool my\_BrotherTree::showTreeLayer(int layer) {  
 using std::cout;  
 using std::endl;  
 //store node  
 std::queue<my\_BrotherTree \*> q;  
 q.push(this);  
 while(!q.empty())  
 {  
 my\_BrotherTree \* t = q.front();  
 q.pop();  
 //if layer == n , output  
 if(t->layer == layer)  
 {  
 cout<<"name : "<<name<<" "<<layer;  
 if(t->parent)  
 cout<<" parent : "<<t->parent->name;  
 cout<<endl;  
 }  
  
 if(t->child)  
 {  
 my\_BrotherTree \* p = t->child;  
 while(p)  
 {  
 q.push(p);  
 if(t->layer == layer)  
 cout<<p->name<<" ";  
 p = p->brother;  
 }  
 cout<<endl;  
 }  
 }  
  
 return true;  
}  
  
my\_BrotherTree \*my\_BrotherTree::findPeople(std::string nm) {  
 //store node  
 std::queue<my\_BrotherTree \*> q;  
 my\_BrotherTree \* t = nullptr;  
 q.push(this);  
 while(!q.empty())  
 {  
 t = q.front();  
 q.pop();  
 if(t->name == nm)  
 break;  
 if(t->child)  
 {  
 my\_BrotherTree \* p = t->child;  
 while(p)  
 {  
 q.push(p);  
 p = p->brother;  
 }  
 }  
 }  
 //if find success  
 if(t->name == nm)  
 return t;  
 return nullptr;  
}  
  
int my\_BrotherTree::layers() {  
 return layer;  
}  
  
std::string my\_BrotherTree::names() {  
 return name;  
}  
  
bool my\_BrotherTree::showName() {  
 using std::cout;  
 cout<<name;  
 return true;  
}  
  
bool my\_BrotherTree::showLayer() {  
 using std::cout;  
 cout<<layer;  
  
 return true;  
}  
  
bool my\_BrotherTree::show() {  
 using std::cout;  
 using std::endl;  
 cout<<"name : "<<name<<" "<<layer<<" ";  
 if(parent)  
 cout<<" parent : "<<parent->name;  
 if(child)  
 {  
 cout<<endl;  
 my\_BrotherTree \* p = child;  
 while(p)  
 {  
 cout<<p->name<<" ";  
 p = p->brother;  
 }  
 cout<<endl;  
 }  
  
 return true;  
}  
  
bool my\_BrotherTree::setLayer(int n) {  
 layer = n;  
 return true;  
}  
  
bool my\_BrotherTree::setName(std::string nm1) {  
 name = nm1;  
 return true;  
}  
  
my\_BrotherTree::my\_BrotherTree() {  
 layer = 1;  
 parent = nullptr;  
 child = nullptr;  
 brother = nullptr;  
}  
  
my\_BrotherTree::~my\_BrotherTree() = default;  
  
  
my\_BrotherTree\_plus::my\_BrotherTree\_plus(std::string nm, std::string born,std::string addre,  
 bool marige, bool l,std::string death) :my\_BrotherTree(nm){  
 borndate = born;  
 deathdate = death;  
 my\_BrotherTree\_plus::addre = addre;  
 my\_BrotherTree\_plus::marige = marige;  
 live = l;  
 parent = nullptr;  
 brother = nullptr;  
 child = nullptr;  
}  
  
bool my\_BrotherTree\_plus::showTreeLayer(int layer) {  
 using std::cout;  
 using std::endl;  
  
 //store node  
 std::queue<my\_BrotherTree\_plus \*> q;  
 q.push(this);  
 while(!q.empty())  
 {  
 my\_BrotherTree\_plus \* t = q.front();  
 q.pop();  
 int n\_lay = t->layers();  
 std::string names = t->names();  
 //if layer == n , output  
 if(n\_lay == layer)  
 {  
 cout<<endl<<"name : "<<names<<" "<<layer;  
 if(t->parent)  
 {  
 cout<<" parent : ";  
 t->parent->showName();  
 }  
  
 cout<<endl;  
 }  
  
 if(t->child)  
 {  
 my\_BrotherTree\_plus \* p = t->child;  
 while(p)  
 {  
 q.push(p);  
 if(n\_lay == layer)  
 {  
 cout<<endl<<"child : ";  
 p->showName();  
 cout<<" ";  
 }  
 p = p->brother;  
 }  
 }  
 }  
  
 return true;  
}  
  
bool my\_BrotherTree\_plus::find(std::string nm) {  
 using std::cout;  
 using std::endl;  
 my\_BrotherTree\_plus \* t = findPeople(nm);  
 //if don't have the people  
 if(!t)  
 return false;  
 t->show();  
 cout<<"born :"<<t->borndate<<endl;  
 if(!live)  
 cout<<"dead : "<<t->deathdate<<endl;  
 cout<<"address :"<<t->addre;  
 cout<<endl;  
  
 return true;  
}  
my\_BrotherTree\_plus \*my\_BrotherTree\_plus::findPeople(std::string nm) {  
 using std::cout;  
 using std::endl;  
 //store node  
 std::queue<my\_BrotherTree\_plus \*> q;  
 my\_BrotherTree\_plus \* t = nullptr;  
 std::string names;  
 q.push(this);  
 while(!q.empty())  
 {  
 t = q.front();  
 q.pop();  
 names = t->names();  
 if(names == nm)  
 break;  
 if(t->marige)  
 {  
 my\_BrotherTree\_plus \* p = t->child;  
 while(p)  
 {  
 q.push(p);  
 p = p->brother;  
 }  
 }  
 }  
 //if find success  
 if(t->names() == nm)  
 return t;  
 return nullptr;  
}  
  
bool my\_BrotherTree\_plus::assign(my\_BrotherTree\_plus \*t, std::string parent\_name) {  
 my\_BrotherTree\_plus \* pa = findPeople(parent\_name);  
 if(!pa)  
 {  
 return false;  
 }  
 //assign to parent  
 if(!pa->child)  
 {  
 pa->child = t;  
 } else{  
 my\_BrotherTree\_plus \* ch = pa->child;  
 while(ch->brother)  
 {  
 ch = ch->brother;  
 }  
 ch->brother = t;  
 }  
 //set t's layer  
 t->setLayer(pa->layers() + 1);  
 //set t's parent  
 t->parent = pa;  
 return true;  
}  
  
bool my\_BrotherTree\_plus::deletePeople(std::string peo) {  
 using std::cout;  
 using std::endl;  
 my\_BrotherTree\_plus \* thepro = findPeople(peo);  
 if(!thepro)  
 {  
 cout<<"don't have this people."<<endl;  
 return false;  
 }  
 //certain the choice  
 cout<<"certain the delete command.(1 means yes)"<<endl<<"delete : ";  
 thepro->show();  
 int cer = getInt("command");  
 //if quit  
 if(cer != 1)  
 return false;  
 std::queue<my\_BrotherTree\_plus \*> q;  
 //delete from parent  
 my\_BrotherTree\_plus \* t = thepro->parent->child;  
 //if this node's parent have two or more children  
 if(t->brother)  
 {  
 //find brother  
 while(t->brother)  
 {  
 if(t->brother == thepro)  
 {  
 t->brother = t->brother->brother;  
 break;  
 }  
 }  
 }  
 //have one child  
 else{  
 thepro->parent->child = nullptr;  
 }  
 std::string names;  
 q.push(thepro);  
 //delete all children  
 while(!q.empty())  
 {  
 t = q.front();  
 q.pop();  
 if(t->marige)  
 {  
 my\_BrotherTree\_plus \* p = t->child;  
 while(p)  
 {  
 q.push(p);  
 p = p->brother;  
 }  
 }  
 delete t;  
 }  
 return true;  
}  
  
bool my\_BrotherTree\_plus::setBorn(std::string nm,std::string born) {  
 my\_BrotherTree\_plus \* p = findPeople(nm);  
 if(!p)  
 {  
 cout<<"don't have this people."<<endl;  
 return false;  
 }  
 p->borndate = born;  
 return true;  
}  
  
bool my\_BrotherTree\_plus::setAddress(std::string nm,std::string add) {  
 my\_BrotherTree\_plus \* p = findPeople(nm);  
 if(!p)  
 {  
 cout<<"don't have this people."<<endl;  
 return false;  
 }  
 p->addre = add;  
 return true;  
}  
  
bool my\_BrotherTree\_plus::setMar(std::string nm,bool mrg) {  
 my\_BrotherTree\_plus \* p = findPeople(nm);  
 if(!p)  
 {  
 cout<<"don't have this people."<<endl;  
 return false;  
 }  
 p->marige = mrg;  
 return true;  
}  
  
bool my\_BrotherTree\_plus::setLive(std::string nm,bool lf) {  
 my\_BrotherTree\_plus \* p = findPeople(nm);  
 if(!p)  
 {  
 cout<<"don't have this people."<<endl;  
 return false;  
 }  
 bool pause = p->live;  
 p->live = lf;  
 if(pause)  
 {  
 //if change from exist to death  
 if(!lf)  
 {  
 std::string death = getString("deathdate");  
 deathdate = death;  
 }  
 } else{  
 if(lf)  
 {  
 deathdate = "";  
 }  
 }  
  
 return true;  
}  
  
bool my\_BrotherTree\_plus::setDeathdate(std::string nm, std::string death) {  
 my\_BrotherTree\_plus \* p = findPeople(nm);  
 if(!p)  
 {  
 cout<<"don't have this people."<<endl;  
 return false;  
 }  
 //if people exist  
 if(p->live)  
 {  
 using std::cout;  
 using std::endl;  
 cout<<nm<<" is alive."<<endl;  
 return false;  
 }  
 p->deathdate = death;  
  
 return true;  
}  
  
my\_BrotherTree\_plus::my\_BrotherTree\_plus(std::string filename) {  
 std::ifstream in(filename);  
 std::string names;  
 in>>names;  
 setName(names);  
 std::string born;  
 in>>born;  
 borndate = born;  
 std::string add;  
 in>>add;  
 addre = add;  
 int lv;  
 std::string death;  
 in>>lv;  
 live = (lv == 1);  
 if(!lv)  
 {  
 in>>death;  
 deathdate = death;  
 }  
 int mar;  
 in>>mar;  
 marige = (mar == 1);  
 parent = nullptr;  
 child = nullptr;  
  
 while(!in.eof())  
 {  
 in>>names;  
 in>>born;  
 in>>add;  
 in>>lv;  
 if(!lv)  
 {  
 in>>death;  
 }  
 in>>mar;  
 std::string pa;  
 in>>pa;  
 my\_BrotherTree\_plus \* p = new my\_BrotherTree\_plus(names,born,add,mar == 1,lv == 1,death);  
 assign(p,pa);  
 }  
 in.close();  
}  
  
bool my\_BrotherTree\_plus::setNames(std::string nm, std::string nm1) {  
 my\_BrotherTree\_plus \* p = findPeople(nm);  
 if(p)  
 {  
 p->setName(nm1);  
 return true;  
 }  
 return false;  
}  
  
bool my\_BrotherTree\_plus::showAll() {  
 using std::cout;  
 using std::endl;  
 int lay = 1;  
 //store node  
 std::queue<my\_BrotherTree\_plus \*> q;  
 q.push(this);  
 while(!q.empty())  
 {  
 my\_BrotherTree\_plus \* t = q.front();  
 q.pop();  
 //if this node is at next layer  
 if(t->layers() != lay)  
 {  
 cout<<endl;  
 lay = t->layers();  
 }  
 cout<<t->showName()<<" ";  
 if(t->child)  
 {  
 my\_BrotherTree\_plus \* p = t->child;  
 while(p)  
 {  
 q.push(p);  
 p = p->brother;  
 }  
 }  
 }  
 cout<<"Show Finish."<<endl;  
  
 return true;  
}  
  
my\_BrotherTree\_plus::~my\_BrotherTree\_plus() = default;  
  
bool my\_BrotherTree\_plus::outToFile() {  
 std::ofstream out\_tx("task4result.txt");  
 using std::endl;  
 std::queue<my\_BrotherTree\_plus \*> q;  
 q.push(this);  
 my\_BrotherTree\_plus \* p;  
 while (!q.empty())  
 {  
 p = q.front();  
 q.pop();  
 out\_tx<<p->names()<<" "<<p->borndate<<" "<<p->addre<<" ";  
 if(p->live == 0)  
 out\_tx<<"0 "<<p->deathdate<<" ";  
 if(p->marige == 1)  
 out\_tx<<"1 ";  
 else  
 out\_tx<<"0 ";  
 if(p->parent)  
 out\_tx<<p->parent->names();  
 out\_tx<<'\n';  
 if(p->child)  
 {  
 my\_BrotherTree\_plus \* t = p->child;  
 while(t)  
 {  
 q.push(t);  
 t = t->brother;  
 }  
 }  
 }  
 out\_tx.close();  
 return true;  
}  
  
bool inVector(vector<my\_BrotherTree\_plus \*> &v,string nm)  
{  
 for(int i = 0;i < v.size();i++)  
 {  
 if(v[i]->names() == nm)  
 return true;  
 }  
 return false;  
}  
  
bool my\_BrotherTree\_plus::showBron() {  
 cout<<std::left;  
 //store node  
 vector<my\_BrotherTree\_plus \*> v;  
 std::queue<my\_BrotherTree\_plus \*> q;  
 q.push(this);  
 int courn = 0;//node number  
 while(!q.empty())  
 {  
 my\_BrotherTree\_plus \* t = q.front();  
 v.push\_back(t);  
 q.pop();  
 courn++;  
 if(t->child)  
 {  
 my\_BrotherTree\_plus \* p = t->child;  
 while(p)  
 {  
 q.push(p);  
 p = p->brother;  
 }  
 }  
 }  
 for(int i = 0;i < courn;i++)  
 {  
 my\_BrotherTree\_plus \* min = v[i];  
 int min\_s = i;  
 for(int k = i + 1;k < courn;k++)  
 {  
 if(min->borndate > v[k]->borndate)  
 {  
 min = v[k];  
 min\_s = k;  
 }  
 }  
 if(min\_s == i)  
 continue;  
 v.erase(v.begin() + min\_s);  
 v.insert(v.begin() + i,min);  
 }  
 cout <<setw(20)<<"name"<<"\t\t";  
 cout <<setw(20)<<"born date"<<endl;  
 for(int i = 0;i < courn;i++)  
 {  
 cout <<setw(20)<<v[i]->names()<<"\t\t";  
 cout <<setw(20)<<v[i]->borndate<<endl;  
 }  
  
 return false;  
}  
  
int my\_BrotherTree\_plus::size() {  
 std::queue<my\_BrotherTree\_plus \*> q;  
 q.push(this);  
 int courn = 0;  
 while(!q.empty())  
 {  
 my\_BrotherTree\_plus \* t = q.front();  
 q.pop();  
 courn++;  
 if(t->child)  
 {  
 my\_BrotherTree\_plus \* p = t->child;  
 while(p)  
 {  
 q.push(p);  
 p = p->brother;  
 }  
 }  
 }  
  
 return courn;  
}  
//1 = nm2 parent \ 2 = nm1 parent \ 3 = nm2 grandparent \ 4 = nm1 grandparent \ 5 = brother  
int my\_BrotherTree\_plus::relationship(std::string nm1, std::string nm2) {  
 my\_BrotherTree\_plus \* pro1 = findPeople(nm1);  
 my\_BrotherTree\_plus \* pro2 = findPeople(nm2);  
 if(pro1->layers() == pro2->layers())  
 {  
 return 5;  
 }  
 if(pro1->layers() > pro2->layers())  
 {  
 if(pro1->parent == pro2)  
 return 1;  
 if(pro1->parent->parent == pro2)  
 return 3;  
 }  
 else{  
 if(pro2->parent == pro1)  
 return 2;  
  
 if(pro2->parent->parent == pro1)  
 return 4;  
 }  
  
 return 0;  
}

task4.cpp

//  
// Created by 97927 on 2019/12/22.  
//  
#include <iostream>  
#include <cstdlib>  
#include <cstdio>  
#include <fstream>  
#include "my\_BrotherTree.h"  
#include <string>  
#include "function.h"  
  
using namespace std;  
//#define YTD  
  
bool sysControl()//choices  
{  
 std::string str[9]{"1.Assign a people",  
 "2.Delete a people",  
 "3.Show a people's information",  
 "4.Change a people's information",  
 "5.Show a layer all people' information",  
 "6.Show all people",  
 "7.Two people's relationship",  
 "8.Show as borndate",  
 "Enter q to quit"};  
 for(int i = 0;i < 9;i++)  
 {  
 cout<<str[i]<<endl;  
 }  
 return true;  
}  
//change choices  
bool changeInformation(my\_BrotherTree\_plus &root\_tree,std::string nm)  
{  
 std::string str[8]{"Which information do you want to change : ",  
 "1.name",  
 "2.borndate",  
 "3.address",  
 "4.mariage",  
 "5.alive",  
 "6.deathdate",  
 "Enter q to quit"};  
 for(int i = 0;i < 8;i++)  
 {  
 cout<<str[i]<<endl;  
 }  
 int choise;  
 cin>>choise;  
 cin.ignore();  
 //input error  
 if(cin.fail())  
 {  
 cin.clear();  
 return false;  
 }  
   
 switch (choise){  
 case 1 :  
 {  
 string nm\_new = getString("new name");  
 root\_tree.setNames(nm,nm\_new);  
 }  
 break;  
 case 2 :  
 {  
 std::string born = getString("borndate");  
 root\_tree.setBorn(nm,born);  
 }  
 break;  
 case 3 :  
 {  
 std::string add = getString("address");  
 root\_tree.setAddress(nm,add);  
 }  
 break;  
 case 4 :  
 {  
 cout<<"marriaged ?(1 for yes)"<<endl;  
 int judge = getInt("answer");  
 root\_tree.setMar(nm,judge == 1);  
 }  
 break;  
 case 5 :  
 {  
 cout<<"still alive ?(1 for yes)"<<endl;  
 int judge = getInt("answer");  
 root\_tree.setLive(nm,judge == 1);  
 }  
 case 6 :  
 {  
 std::string death = getString("deathdate(q to quit)");  
 if(death == "q")  
 break;  
 root\_tree.setDeathdate(nm,death);  
 }  
 break;  
 }  
  
 return true;  
}  
//carry out the choices  
bool carryOutControl(my\_BrotherTree\_plus &root\_tree,int n)  
{  
 switch(n) {  
 case 1 :  
 {  
 //get information of the people  
 std::string name = getString();  
 std::string born = getString("borndate");  
 std::string add = getString("address");  
 bool lv;  
 bool mar;  
 cout<<"still alive ?(1 for yes)"<<endl;  
 int judge = getInt("answer");  
 std::string death;  
 if(judge != 1)  
 {  
 lv = false;  
 death = getString("deathdate");  
 } else{  
 lv = true;  
 }  
 cout<<"marriaged ?(1 for yes)"<<endl;  
 judge = getInt("answer");  
 mar = judge == 1;  
 std::string pa = getString("parent");  
 my\_BrotherTree\_plus \* p = new my\_BrotherTree\_plus(name,born,add,mar,lv,death);  
 //assign to pa  
 root\_tree.assign(p,pa);  
 }  
 break;  
 case 2 :  
 {  
 std::string name = getString();  
 //delete a people  
 root\_tree.deletePeople(name);  
 }  
 break;  
 case 3 :  
 {  
 std::string name = getString();  
 //show a people all information  
 root\_tree.find(name);  
 }  
 break;  
 case 4 :  
 {  
 std::string name = getString();  
 //change information  
 changeInformation(root\_tree,name);  
 }  
 break;  
 case 5 :  
 {  
 int layer = getInt("layer");  
 //show nth layer all people  
 root\_tree.showTreeLayer(layer);  
 }  
 break;  
 case 6 :  
 {  
 root\_tree.showAll();  
 }  
 break;  
 case 8 :  
 {  
 root\_tree.showBron();  
 }  
 break;  
 case 7 :  
 {  
 string nm\_new1 = getString("name1");  
 string nm\_new2 = getString("name2");  
 switch (root\_tree.relationship(nm\_new1,nm\_new2)){  
 case 1 :  
 {  
 cout<<nm\_new2<<" is "<<nm\_new1<<" parent"<<endl;  
 }  
 break;  
 case 2 :  
 {  
 cout<<nm\_new1<<" is "<<nm\_new2<<" parent"<<endl;  
 }  
 break;  
 case 3 :  
 {  
 cout<<nm\_new2<<" is "<<nm\_new1<<" grandparent"<<endl;  
 }  
 break;  
 case 4 :  
 {  
 cout<<nm\_new1<<" is "<<nm\_new2<<" grandparent"<<endl;  
 }  
 break;  
 case 5 :  
 {  
 cout<<nm\_new1<<" is "<<nm\_new2<<" brother"<<endl;  
 }  
 break;  
 default:{  
 cout<<"Do not have relationship."<<endl;  
 }  
 }  
 }  
 }  
}  
#ifndef YTD  
int main()  
{  
 /\*std::string fileName = getString("filename");  
 fileName += ".txt";\*/  
 my\_BrotherTree\_plus root\_tree("familyin.txt");  
 while(1)  
 {  
 //clear screen  
 system("clear");  
 sysControl();  
 int choise;  
 cin>>choise;  
 cin.ignore();  
 //enter below the range  
 if(cin.fail())  
 {  
 cin.clear();  
 break;  
 }  
 carryOutControl(root\_tree,choise);  
 //system pause  
 system("read -p 'Press Enter to continue...' var");  
 }  
 //out to file  
 root\_tree.outToFile();  
  
 return 0;  
}  
  
#endif

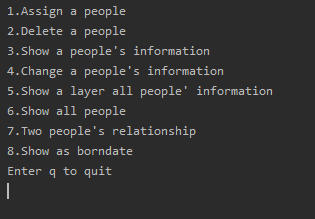
###### 测试样例与运行结果：

测试样例：

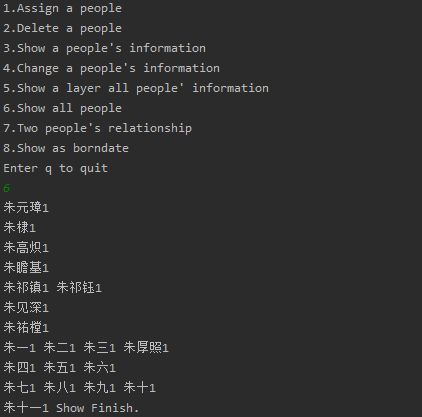
朱元璋 1328 明皇宫 0 1398 1 朱棣 1360 明皇宫 0 1424 1 朱元璋 朱高炽 1378 明皇宫 0 1425 1 朱棣 朱瞻基 1398 明皇宫 0 1435 1 朱高炽 朱祁镇 1427 明皇宫 0 1464 1 朱瞻基 朱祁钰 1428 明皇宫 0 1457 1 朱瞻基 朱见深 1447 明皇宫 0 1487 1 朱祁镇 朱祐樘 1470 明皇宫 0 1505 1 朱见深 朱一 1492 明皇宫 0 1521 1 朱祐樘 朱二 1493 明皇宫 0 1521 1 朱祐樘 朱三 1494 明皇宫 0 1521 1 朱祐樘 朱四 1491 明皇宫 0 1521 1 朱二 朱五 1491 明皇宫 0 1521 1 朱二 朱六 1491 明皇宫 0 1521 1 朱三 朱七 1491 明皇宫 0 1521 1 朱四 朱八 1491 明皇宫 0 1521 1 朱四 朱九 1491 明皇宫 0 1521 1 朱四 朱十 1491 明皇宫 0 1521 1 朱五 朱十一 1491 明皇宫 0 1521 1 朱九 朱厚照 1491 明皇宫 0 1521 1 朱祐樘

运行结果：

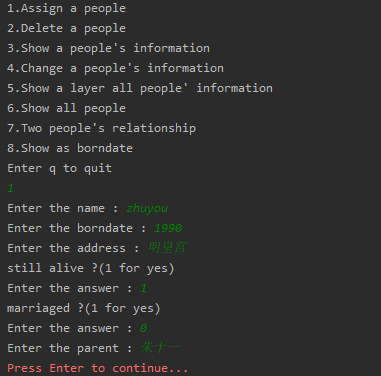
选择菜单

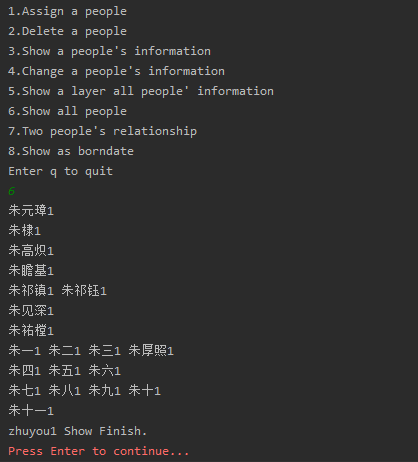


显示所有人：

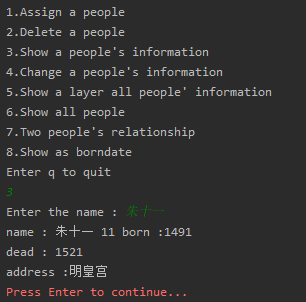


增加一个人：

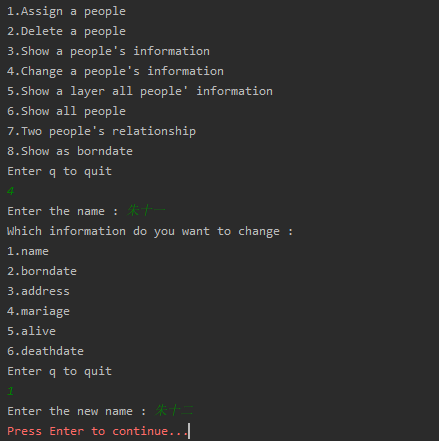


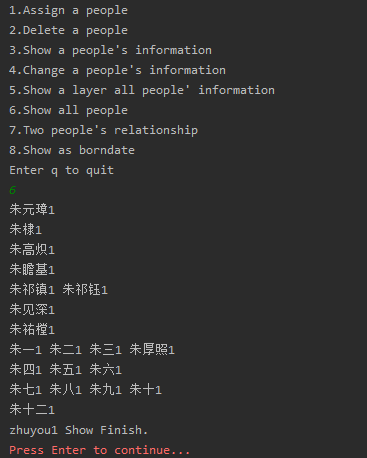


显示一个人所有信息：

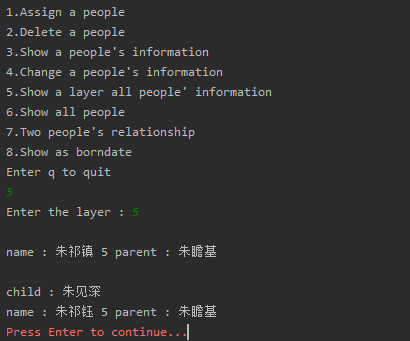


改变一个人的信息：

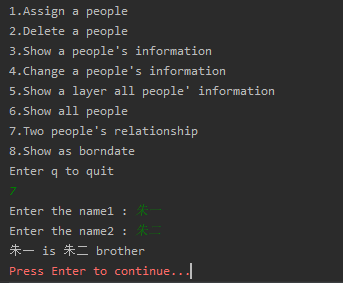




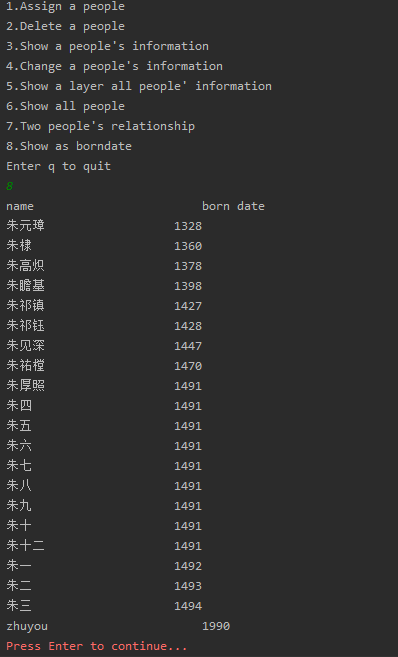
显示某一辈所有人：



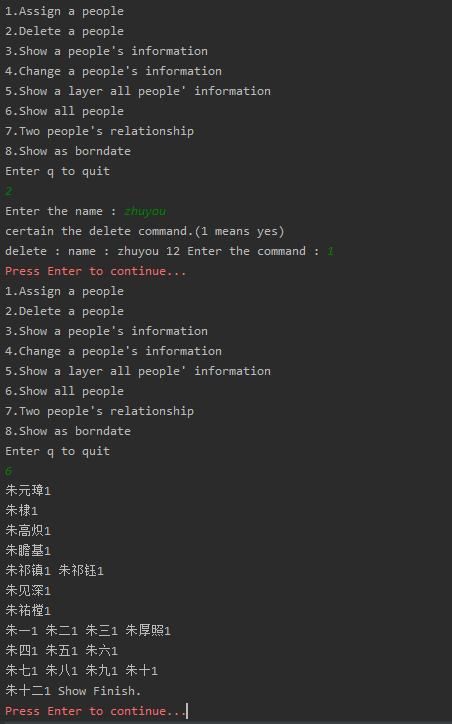
显示两人关系：



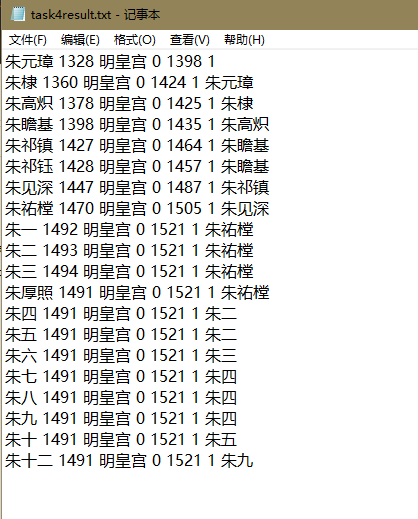
按出生日期排序：



删除一个人：



结束后文件中的结果：



### 必做题第五题

###### 题目：Huffman编码与解码(必做)（Huffman编码、二叉树）

[问题描述]

对一篇不少于5000字符的英文文章（source.txt），统计各字符出现的次数，实现Huffman编码(code.dat)，以及对编码结果的解码(recode.txt)。

[基本要求]

（1） 输出每个字符出现的次数和编码,并存储文件(Huffman.txt)。

（2） 在Huffman编码后，英文文章编码结果保存到文件中(code.dat)，编码结果必须是二进制形式，即0 1的信息用比特位表示，不能用字符’0’和’1’表示（\*）。

（3） 实现解码功能。

###### 数据结构：二叉树

###### 算法思想：

构建huffman编码：  
打开文件  
读取文件中的每个字符：  
 1.该字符未出现，建立新树存储该字符，权重为1  
 2.该字符已出现，权重加1  
对已有的树按权重排序  
重复执行：  
 取出权重最小的两棵树，建立一棵新树，存储字符‘\0’，左子树为权重较小的一棵树，右 子树为权重较大的一棵树，权重为两个子树的和，将该树插入到原有的有序树中  
 当只剩余一颗树时，结束  
将根节点的编码置为空，重复执行：  
 1.左子树的编码为该子树的pa加‘0’  
 2.右子树的编码为该子树的pa加‘0’  
 当该树的字符不为'\0'时，输出该字符、权重、编码到文件中  
 当遍历完整棵树时，结束  
编码到文件中：  
打开文件  
建立无符号整型变量n  
读取文件中的每个字符，取出该字符编码，从高位到低位遍历编码：  
 1.如果为0，将n\*2  
 2.如果为1，将n\*2后，n++  
 如果操作次数为31次，输出n，将n置为0，重新进行计数  
当读取结束后，继续对n进行\*2操作，知道操作次数到达31次，输出  
解码：  
设置查找节点遍历tree = root  
打开文件  
读取无符号整型数字，分解为二进制形式  
将该二进制数从高位向低位读取，从根节点出发：  
 1.该位为0，tree = tree->left  
 2.该位为1，tree = tree->right  
 若tree的字符不为'\0'，输出该字符，tree = root

###### 具体实现：

my\_BinaryTree.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_MY\_BINARYTREE\_H  
#define DESIGN\_MY\_BINARYTREE\_H  
  
#include <string>  
  
class my\_BinaryTree {  
private:  
 int site;  
 int weight;  
 char data;  
 std::string code;  
 my\_BinaryTree \* pa;  
 my\_BinaryTree \* lft\_child;  
 my\_BinaryTree \* rght\_child;  
protected:  
 my\_BinaryTree \* locateTree(int st);  
 my\_BinaryTree \* leftBottom();  
 my\_BinaryTree \* rightBottom();  
public:  
 my\_BinaryTree(char ch,int n = 0);//initial  
 ~my\_BinaryTree();  
 int getWeight();//return weight  
 int getSite();//return site  
 int depth();//return tree depth  
 char getData();//get this node's data  
 bool assign(int st,char ch);//assign a new tree  
 my\_BinaryTree \* parent(int st);//find parent  
 my\_BinaryTree \* leftChild(int st);//find leftchild  
 my\_BinaryTree \* rightChild(int st);//find rightchild  
 bool inserChild(int st,int LR,my\_BinaryTree \* ct);//LR 1 right  
 bool setParent(my\_BinaryTree \* pr);//change parent  
 bool deleteChild(int st,int LR);  
 bool preOrderTraverse();  
 bool inOrderTraverse();  
 bool postOrderTraverse();  
 bool levelOrderTraverse();  
 bool show();//show this node's information  
 bool coding(std::string file);//code  
 std::string getCode();//return code  
 bool deCodeFile(std::string filename);//decode from file  
};  
  
  
#endif //DESIGN\_MY\_BINARYTREE\_H

my\_BinaryTree.cpp

//  
// Created by 97927 on 2019/12/13.  
//  
  
#include "my\_BinaryTree.h"  
#include <iostream>  
#include <queue>  
#include <stack>  
#include <fstream>  
using std::cout;  
using std::endl;  
  
my\_BinaryTree::my\_BinaryTree(char ch,int n) {  
 site = 0;  
 data = ch;  
 weight = n;  
 pa = nullptr;  
 lft\_child = nullptr;  
 rght\_child = nullptr;  
}  
  
my\_BinaryTree::~my\_BinaryTree() {}  
  
int my\_BinaryTree::depth() {  
 //store node  
 std::queue<my\_BinaryTree \*> q;  
 int level = 1;  
  
 my\_BinaryTree pu\_tr(-1);  
 if(lft\_child)  
 q.push(lft\_child);  
 if(rght\_child)  
 q.push(rght\_child);  
 //empty node  
 q.push(&pu\_tr);  
 while(q.size() != 1)  
 {  
 my\_BinaryTree \* tree;  
 tree = q.front();  
 q.pop();  
 //if find empty node  
 if(tree->site == -1)  
 {  
 level++;  
 q.push(tree);  
 //if traverse is finished  
 if(q.empty())  
 break;  
 continue;  
 }  
 if(tree->lft\_child)  
 {  
 q.push(tree->lft\_child);  
 }  
 if(tree->rght\_child)  
 {  
 q.push(tree->rght\_child);  
 }  
 }  
  
 return level;  
}  
  
int my\_BinaryTree::getSite() {  
 return site;  
}  
  
my\_BinaryTree \*my\_BinaryTree::leftChild(int st) {  
 return lft\_child;  
}  
  
my\_BinaryTree \*my\_BinaryTree::rightChild(int st) {  
 return rght\_child;  
}  
  
my\_BinaryTree \*my\_BinaryTree::parent(int st) {  
 return pa;  
}  
  
char my\_BinaryTree::getData() {  
 return data;  
}  
  
bool my\_BinaryTree::inserChild(int st, int LR, my\_BinaryTree \*ct) {  
 my\_BinaryTree \* tree = locateTree(st);  
 if(!tree)  
 return false;  
 switch (LR){  
 case 1 :  
 {  
 //if this have right child  
 if(tree->rght\_child)  
 {  
 //add this's right child to this's right bottom  
 my\_BinaryTree \* pause\_tree = tree->rght\_child;  
 tree->rightBottom()->rght\_child = pause\_tree;  
 }  
  
 tree->rght\_child = ct;  
 ct->site = tree->site \* 2 + 1;  
 }  
 break;  
 default:  
 {  
 //if this have left child  
 if(tree->lft\_child)  
 {  
 //add this's left child to this's left bottom  
 my\_BinaryTree \* pause\_tree = tree->lft\_child;  
 tree->leftBottom()->lft\_child = pause\_tree;  
 }  
 tree->lft\_child = ct;  
 ct->site = tree->site \* 2;  
 }  
 }  
 tree->weight += ct->weight;  
 ct->pa = tree;  
 return true;  
}  
  
bool my\_BinaryTree::setParent(my\_BinaryTree \* pr) {  
 pa = pr;  
  
 return true;  
}  
  
my\_BinaryTree \*my\_BinaryTree::locateTree(int st) {  
 std::queue<my\_BinaryTree \*> q;  
 q.push(this);  
 my\_BinaryTree \* tree = nullptr;  
 if(lft\_child)  
 q.push(lft\_child);  
 if(rght\_child)  
 q.push(rght\_child);  
 while(!q.empty())  
 {  
 tree = q.front();  
 q.pop();  
 //if find this node  
 if(tree->site == st)  
 break;  
 if(tree->lft\_child)  
 {  
 q.push(tree->lft\_child);  
 }  
 if(tree->rght\_child)  
 {  
 q.push(tree->rght\_child);  
 }  
 }  
 if(!tree || tree->site != st)  
 return nullptr;  
 return tree;  
}  
  
my\_BinaryTree \*my\_BinaryTree::rightBottom() {  
 my\_BinaryTree \* q = rght\_child;  
 while(q->rght\_child)  
 {  
 q = q->rght\_child;  
 }  
  
 return q;  
}  
  
my\_BinaryTree \*my\_BinaryTree::leftBottom() {  
 my\_BinaryTree \* q = lft\_child;  
 while(q->lft\_child)  
 {  
 q = q->lft\_child;  
 }  
 return q;  
}  
  
bool my\_BinaryTree::assign(int st,char ch) {  
 my\_BinaryTree \* tree = locateTree(st);  
 if(!tree)  
 return false;  
 tree->data = ch;  
 return true;  
}  
  
bool my\_BinaryTree::deleteChild(int st, int LR) {  
 my\_BinaryTree \* tree = locateTree(st);  
 if(!tree)  
 return false;  
 switch (LR){  
 case 1 :  
 {  
 tree->weight -= tree->lft\_child->weight;  
 tree->lft\_child = nullptr;  
 }  
 break;  
 default:  
 {  
 tree->weight -= tree->rght\_child->weight;  
 tree->rght\_child = nullptr;  
 }  
 }  
 return true;  
}  
  
bool my\_BinaryTree::preOrderTraverse() {  
 std::stack<my\_BinaryTree \*> s;  
 my\_BinaryTree \* tree = this;  
  
 do{  
 //do not have left child  
 if(!tree)  
 {  
 tree = s.top();  
 s.pop();  
 tree = tree->rght\_child;  
 continue;  
 }  
 cout<<"site "<<tree->site<<" , data : "<<tree->data<<" "<<endl;  
 s.push(tree);  
 tree = tree->lft\_child;  
 }while(!s.empty() || tree);  
 cout<<endl<<"preOrderTraverse finish."<<endl;  
  
 return true;  
}  
  
bool my\_BinaryTree::inOrderTraverse() {  
 std::stack<my\_BinaryTree \*> s;  
 my\_BinaryTree \* tree = this;  
  
 do{  
 //do not have left child  
 if(!tree)  
 {  
 tree = s.top();  
 cout<<"site "<<tree->site<<" , data : "<<tree->data<<" ";  
 s.pop();  
 tree = tree->rght\_child;  
 continue;  
 }  
 s.push(tree);  
 tree = tree->lft\_child;  
 }while(!s.empty());  
 cout<<endl<<"inOrderTraverse finish."<<endl;  
  
 return true;  
  
}  
  
bool my\_BinaryTree::postOrderTraverse() {  
 std::stack<my\_BinaryTree \*> s;  
 my\_BinaryTree \* tree = this;  
  
 do{  
 //do not have child  
 if(!tree)  
 {  
 tree = s.top();  
 s.pop();  
 cout<<"site "<<tree->site<<" , data : "<<tree->data<<" ";  
 tree = tree->rght\_child;  
 continue;  
 }  
 s.push(tree);  
 if(tree->lft\_child)  
 tree = tree->lft\_child;  
 else if (tree->rght\_child)  
 tree = tree->rght\_child;  
 else  
 tree = nullptr;  
 }while(!s.empty());  
 cout<<endl<<"postOrderTraverse finish."<<endl;  
  
 return true;  
}  
  
bool my\_BinaryTree::levelOrderTraverse() {  
 //store node  
 std::queue<my\_BinaryTree \*> q;  
 cout<<"site "<<site<<" , data : "<<data<<" "<<" , weight : "<<weight<<endl;  
 my\_BinaryTree \* tree = nullptr;  
 if(lft\_child)  
 q.push(lft\_child);  
 if(rght\_child)  
 q.push(rght\_child);  
 while(!q.empty())  
 {  
 tree = q.front();  
 cout<<"site "<<tree->site<<" , data : "<<tree->data<<" "<<" , weight : "<<tree->weight<<endl;  
 q.pop();  
 if(tree->lft\_child)  
 {  
 q.push(tree->lft\_child);  
 }  
 if(tree->rght\_child)  
 {  
 q.push(tree->rght\_child);  
 }  
 }  
 cout<<endl<<"levelOrderTraverse finish."<<endl;  
  
 return tree;  
}  
  
int my\_BinaryTree::getWeight() {  
 return weight;  
}  
  
bool my\_BinaryTree::show() {  
 cout<<"site "<<site<<" , data : "<<data<<" "<<" , weight : "<<weight;  
 return true;  
}  
  
bool my\_BinaryTree::coding(std::string file) {  
 std::ofstream out(file);  
 std::queue<my\_BinaryTree \*> q;  
 my\_BinaryTree \* tree = nullptr;  
 q.push(this);  
 while(!q.empty())  
 {  
 tree = q.front();  
 if(tree->data != '\0')  
 {  
 out<<tree->data<<" "<<tree->weight<<" "<<tree->code<<endl;  
 }  
 q.pop();  
 //left add 0  
 if(tree->lft\_child)  
 {  
 q.push(tree->lft\_child);  
 tree->lft\_child->code = tree->code + '0';  
 }  
 //right add 1  
 if(tree->rght\_child)  
 {  
 q.push(tree->rght\_child);  
 tree->rght\_child->code = tree->code + '1';  
 }  
 }  
 out.close();  
  
  
 return true;  
}  
  
std::string my\_BinaryTree::getCode() {  
 return code;  
}  
  
std::string splitInt(unsigned int n)  
{  
 std::string str(32,'0');  
 for(int i = 31;n != 0;i--)  
 {  
 str[i] = '0' + n % 2;  
 n /= 2;  
 }  
  
 return str;  
}  
bool my\_BinaryTree::deCodeFile(std::string filename) {  
 std::ifstream in(filename);  
 std::string str;  
 my\_BinaryTree \* p = this;  
 int i = 0;  
 while(!in.eof())  
 {  
 i = 0;  
 unsigned int nm;  
 in>>nm;  
 str = splitInt(nm);  
 while(true)  
 {  
 //output the data  
 if(p->data != '\0')  
 {  
 if(p->data != '\r')  
 cout<<p->data;  
 p = this;  
 }  
 //this int decode finish  
 if(i > 31)  
 break;  
 //0 turn left  
 if(str[i] == '0')  
 {  
 p = p->lft\_child;  
 }  
 //1 turn right  
 else{  
 p = p->rght\_child;  
 }  
 i++;  
 }  
 }  
 in.close();  
 return true;  
}

task5.cpp

//  
// Created by 97927 on 2019/12/24.  
//  
#include <iostream>  
#include <cstdlib>  
#include "my\_BinaryTree.h"  
#include <vector>  
#include <fstream>  
#define TOT 126 - 10  
//#define YTD  
  
using namespace std;  
//sort as weight  
bool sortVector(vector<my\_BinaryTree \*> &huff)  
{  
 int length = huff.size();  
 for(int k = 0;k < length;k++)  
 {  
 int minW = huff[k]->getWeight();  
 int minS = k;  
 for(int i = k + 1;i < length;i++)  
 {  
 if(huff[i]->getWeight() > minW)  
 {  
 minW = huff[i]->getWeight();  
 minS = i;  
 }  
 }  
 if(minS == k)  
 continue;  
 my\_BinaryTree \* pause\_tree = huff[minS];  
 huff.erase(huff.begin() + minS);  
 huff.insert(huff.begin() + k,pause\_tree);  
 }  
 return true;  
}  
//create huffman tree  
bool huffmanEnCode(vector<my\_BinaryTree \*> &huff)  
{  
 int length = huff.size();  
 sortVector(huff);  
 while(true)  
 {  
 //create new tree   
 my\_BinaryTree \* p = new my\_BinaryTree('\0');  
 //insert the two min weight  
 p->inserChild(0,0,huff.back());  
 huff.pop\_back();  
 p->inserChild(0,1,huff.back());  
 huff.pop\_back();  
 length = huff.size();  
 if(length == 0)  
 {  
 huff.push\_back(p);  
 break;  
 }  
 for(int i = 0;i < length;i++)  
 {  
 if(huff[i]->getWeight() < p->getWeight())  
 {  
 huff.insert(huff.begin() + i,p);  
 break;  
 }  
 }  
 if(p->getWeight() < huff.back()->getWeight())  
 {  
 huff.push\_back(p);  
 }  
 }  
  
 return true;  
}  
  
#ifndef YTD  
int main() {  
 //open read file to code  
 ifstream in("txtsourse.txt");  
 if(!in.is\_open())  
 {  
 cout<<"Open fails."<<endl;  
 return 0;  
 }  
 //courn and haxi exchange  
 int courn[TOT] = {0};  
 //to create a hufftree  
 vector<my\_BinaryTree \*> huff;  
 //store every letter  
 vector<my\_BinaryTree \*> huff\_letter;  
 //calculate the letter  
 char ch;  
 while (!in.eof()) {  
 in.get(ch);  
 courn[ch - '\n']++;  
 }  
 //create tree  
 for (int i = 0; i < TOT; i++)  
 {  
 if(courn[i] == 0)  
 continue;  
 my\_BinaryTree \* p = new my\_BinaryTree('\n' + i,courn[i]);  
 courn[i] = huff.size();  
 huff.push\_back(p);  
 huff\_letter.push\_back(p);  
 }  
 in.close();  
 //create huffman tree  
 huffmanEnCode(huff);  
 //code letter  
 huff[0]->coding("huffman.txt");  
 //code to file  
 std::string infile("example/tesk.txt");  
 std::string outfile("example/re.txt");  
 ifstream inf(infile);  
 ofstream outf(outfile);  
 //ofstream outf(outfile);  
 unsigned int nm = 0;  
 int times = 0;  
 int m = 1;  
 while(!inf.eof())  
 {  
 inf.get(ch);  
 int site = courn[ch - '\n'];  
 std::string pause = huff\_letter[courn[ch - '\n']]->getCode();  
 for(int i = 0;i < pause.size();i++)  
 {  
 if(pause[i] == '1')  
 {  
 nm += 1;  
 }  
 times++;  
 if(times == 32)  
 {  
 outf<<nm<<endl;  
 nm = 0;  
 times = 0;  
 m++;  
 }  
 else{  
 nm \*= 2;  
 }  
 }  
 }  
 if(time != 0)  
 {  
 for(int k = times;k < 31;k++)  
 {  
 nm \*= 2;  
 }  
 }  
 outf<<nm<<endl;  
 inf.close();  
 outf.close();  
 huff[0]->deCodeFile("example/re.txt");  
  
 return 0;  
}  
  
#endif

###### 测试样例与运行结果：

文件中文本：

Chinese Tea Culture  
China is a country with a time-honored civilization and also a land of ceremony and decorum. Whenever guests visit, it is necessary to make and serve tea to them. Before serving tea, you may ask them for their preferences as to what kind of tea they fancy, and serve them the tea in the most appropriate teacups.In the course of serving tea, the host should take careful note of how much water remains in the guests' cups.   
Usually, if the tea is made in a teacup, boiling water should be added into the cup when half of the tea in it has been consumed; and thus the cup is kept filled and the tea retains the same bouquet.  
  
The Chinese Knot  
The Chinese knot is a kind of traditional and typical folk hand-woven decoration in China. In Chinese language, "jie (a knot)" means reunion, amity, peace and love, etc., so the Chinese knot is often used to express good wishes. Each knot is usually woven with only one silk cord or silk rope, and named according to its shape and meaning. Chinese knots are mainly made of various cords which can be silk, cotton, linen or nylon and so on. The Chinese knot is both practical and decorative, fully reflecting the charm of Chinese culture.  
  
Tang Poetry  
Tang poetry generally refers to poems written during the Tang Dynasty (618 A.D.-907A.D.). Tang poetry is one of the most valuable cultural heritages of the Han Chinese. Meanwhile,it also has a great influence on the cultural development of neighboring ethnic groups and nations. The most widely spread among Tang poems are definitely the poems that are included in the "Three Hundred Poems of the Tang Dynasty", many of which are quite popular with people of later generations. There are lots of poets in Tang Dynasty, among whom Li Bai and Du Fu are world-famous. Many of the two great poets' works are household poems.  
  
Mencius  
Mencius is a great ideologist and educator in the Warring States Period. A hundred years after Confucius died, Mencius taught and expanded Confucian doctrine.His philosophical thinking has its origin in Confucius' thoughts. Mencius believes that human nature is essentially good, and it is society's influence that causes the degradation of morality. For that reason, he attaches great importance to moral education.Mencius' mother plays an important role in his life. She moved home three times only to find an environment that she felt was suitable for the child's upbringing.  
  
The West Lake  
The West Lake, located in the western area of Hangzhou's center, is one of the top three lakes in the regions south of the Yangtze River. Because of it, Hangzhou has been acclaimed as "a heaven on earth" since ancient times. The West Lake is like a shining pearl inlaid on the vast land, renowned for its beautiful scenery, well-known historical sites, brilliant culture,and plentiful local specialties. The literary giant Su Shi in Song Dynasty left a poetic masterpiece through the ages there: The West Lake is like the beauty Xi Shi, who is always charming with either light or heavy make-up (rainy or shiny). The Legend of White Snake also brings the West Lake an air of mystery.  
  
Jiaozi  
Jiaozi, or dumpling is a kind of food stuffed with filling inside a wheat wrapper. As a quite traditional food in Northern China, it is very popular among Han Chinese. Jiaozi is absolutely necessary when the Chinese people celebrate the Spring Festival and other festivals. It is said that jiaozi was invented by China's famous ancient doctor Zhang Zhongjing. The reason for naming the food as "jiaozi" is very simple: because "jiaozi" is homophonous to the Chinese phrase "change of year",which means the transition from the old to the new. As the Spring Festival marks the start of a new year,   
people choose to eat jiaozi to express their best wishes for the New Year. This tradition has remained unbroken despite the change of times.  
  
Human Flesh Search  
"Human Flesh Search" may be one of the hottest words in China's cyberspace.   
It refers to the massive collaboration of the netizens to identify and release as much information on a targeted individual or group as possible. For the vast number of Chinese Internet users, the human flesh search has become an effective tool in exposing corruption, fraudulence and injustice.However, every coin has two sides. The human flesh search may sometimes be misused for ulterior purposes. Therefore, laws and regulations must be made to ensure its healthy development.  
  
TV Dating Show  
A TV dating show is an entertainment reality show featuring blind dates and marriage seeking. The topics discussed in the show have gone far beyond the scope of love or marriage including those popular topics in the current Chinese society,such as money worship, housing mortgage slave, and 3S ladies,etc. For example, a female contestant in one dating show said when she rejected the courting of her suitor, said: "I'd rather sit and cry in a BMW than laugh on the backseat of a bicycle." Despite all its criticism, a TV dating show provides a brand-new dating platform for busy young single people.  
  
Positive Energy  
Positive energy, originally as a term in physics, has been given a new meaning recently and become popular rapidly. In contrast to negative energy, positive energy refers to a kind of healthy, optimistic and vigorous power and emotion,giving us confidence and hope and encouraging us to pursue a happy life constantly. This term is particularly popular on the Internet currently. Large amounts of stories and pictures full of positive energy appear on the Internet frequently. All the vigorous and inspiring people and stories are labeled with "positive energy" by netizens.  
  
Shanzhai (Copycatting)  
"Shanzhai" (copycatting) is a new and popular term used to describe a certain social phenomenon.Main features of "shanzhai" products are as follows: they are counterfeits; they are produced quickly and their consumers are ordinary people. These products are mainly produced by individual workshops to quickly imitate famous brands of various fields such as cell phones, digital product and video game players. Nowadays, everything on the Internet has its "shanzhai" version such as shanzhai cell phones, shanzhai computers, shanzhai "Bird's Nest" and even shanzhai star.

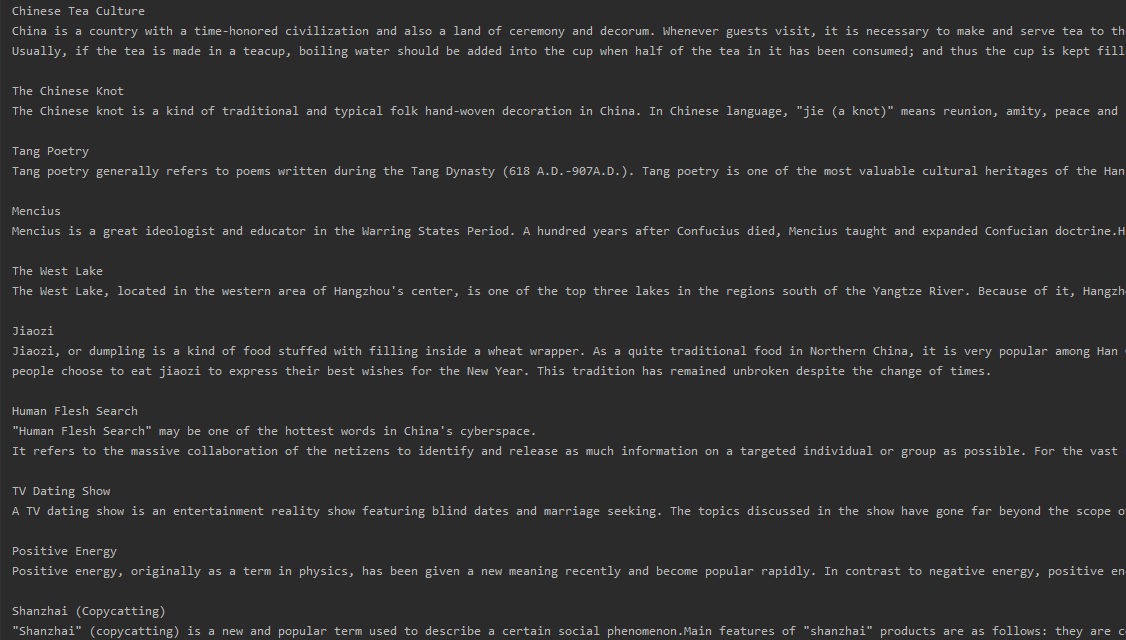
字符编码：

e 622 001  
 1015 111  
s 346 0101  
o 354 0110  
i 387 0111  
t 390 1000  
n 393 1001  
a 422 1011  
c 146 00001  
u 154 00010  
d 159 00011  
l 177 01001  
h 264 11010  
r 277 11011  
y 96 101000  
m 102 101001  
f 103 101010  
g 110 110000  
p 120 110010  
  
 31 0000000  
  
 31 0000001  
k 38 0100000  
b 43 0100010  
, 53 1010110  
v 54 1010111  
w 58 1100011  
. 60 1100110  
z 21 01000011  
C 24 01000111  
" 26 11000100  
T 30 11001110  
j 8 000001001  
M 9 000001010  
H 9 000001011  
W 9 000001101  
D 9 000001110  
I 10 010000101  
' 11 010001100  
S 15 110011110  
: 4 0000011000  
x 5 0000011001  
) 5 0000011110  
( 5 0000011111  
P 5 0100001000  
B 5 0100001001  
q 6 0100011011  
A 7 1100010101  
- 7 1100010110  
L 8 1100111110  
F 8 1100111111  
Z 2 00000100000  
Y 2 00000100001  
V 3 01000110101  
J 3 11000101000  
N 4 11000101111  
R 1 000001000100  
K 1 000001000101  
X 1 000001000110  
U 1 000001000111  
0 1 010001101000  
E 2 110001010010  
; 2 110001010011  
3 1 0100011010010  
1 1 0100011010011  
7 1 1100010111000  
6 1 1100010111001  
9 1 1100010111010  
8 1 1100010111011

编码后结果：

1205061970  
4190566968  
1233218688  
21484492  
4222598678  
174290044  
932019184  
2051943846  
1265405828  
4124913286  
3094781810  
531934074  
4108095412  
2920526666  
2794418750  
420179028  
1936748172  
1299431296  
1164017014  
2881842928  
4005447754  
1459046278  
2053964278  
2448767918  
1041102958  
2367320948  
308964766  
1404759864  
1008446952  
817706822  
3667980706  
2815745008  
3514694838  
891132168  
2947936478  
2404850946  
3833441008  
937641614  
2867082326  
2073163066  
3611178088  
4174508142  
3732695290  
749858604  
1533851382  
2212550724  
3379765606  
3813115268  
3667842420  
2648144322  
3772608282  
530995434  
3540591600  
3020392182  
889798234  
129326924  
3353940084  
4176350974  
1721465182  
2045879280  
145500722  
3257702198  
2147617002  
363148374  
2007746190  
3250514920  
3637475830  
4033618634  
1537369930  
3876556512  
2010598436  
1204887394  
2358230210  
3160993930  
2546443070  
2873974186  
2027024494  
3732654036  
3617728120  
757738056  
2024271430  
4053872524  
2399711982  
1585505566  
1403331710  
1916738110  
117361754  
3850135796  
3710362208  
2444302540  
262172  
1989101522  
3836338314  
2524971122  
3661438798  
2460459158  
1198913488  
508025518  
2379800092  
3546937122  
4171441932  
3555353760  
2103217942  
1667950200  
1680718726  
3036126862  
3553486568  
754220348  
1251908508  
45878638  
3288644064  
2130315624  
64528018  
3110056488  
3142414196  
1579724274  
453637922  
4105532250  
3423024494  
1458676688  
4202244730  
78935902  
1788923666  
1224488736  
1731040606  
3247996698  
3611652988  
693663184  
631487216  
2488702160  
4175083934  
3337727698  
1293788404  
3125833910  
1676525230  
1242543816  
1803260408  
3710156482  
191655736  
1010268254  
1566459868  
1206466350  
2620617852  
2655690370  
755756902  
3921122512  
4107849578  
1589505496  
635530636  
3206454330  
1893528646  
3587343030  
2241993322  
3126742262  
1875708264  
4220646766  
1775104948  
1025462052  
2812319374  
1978178100  
4184785678  
229104782  
3326270392  
1030970324  
307447768  
3566344312  
3789768582  
2883403178  
1954364564  
2651923972  
3654025224  
1895324  
487088922  
3489664628  
3876557592  
1854898578  
3718591612  
3444009952  
3743823022  
3822830616  
4052447032  
1013513450  
3457948996  
1835175438  
4050816316  
788409240  
248351586  
1955403960  
1656324568  
3237375900  
3116623046  
463715062  
1240291212  
2409993788  
3181663880  
2478974144  
1538587932  
3727622844  
3581465056  
1573865426  
3836333504  
2742259534  
1236070892  
2507123550  
3213282758  
3168047384  
2230755098  
504440324  
3718745302  
2494977176  
3345661078  
3785641398  
3457992356  
3791154564  
3380463868  
1847436658  
3195441060  
3407813516  
1397185718  
910128308  
1895421752  
1045195102  
3181261650  
3856705166  
2367656730  
736319262  
2067653672  
2288287206  
3813139046  
3048372268  
689883386  
140724982  
1440958438  
2926633194  
325784674  
729443944  
1990976466  
3283057468  
2374623484  
1301320570  
4235139832  
2338624474  
1464713694  
3764711136  
3979058150  
3027664364  
2052724660  
2945652830  
2046416696  
943360436  
3306536402  
1774445522  
2851730420  
326102598  
3772873980  
796606234  
3063014100  
2929070822  
1884671260  
3581465072  
3336506778  
3351874602  
435271376  
400478162  
2225556624  
4266420572  
1610629120  
2190247364  
1342193990  
561065844  
4227362012  
999076442  
2209119120  
4179133296  
1844944666  
504217068  
4085763908  
3769582094  
1858326000  
2949686394  
2449083884  
3203040190  
1198106688  
3096939976  
1975003336  
774238562  
1638463266  
4060931442  
421480296  
3558915534  
2374179704  
2629875550  
3410925142  
1705446822  
3813615646  
947739382  
3270958048  
2045721884  
3545268962  
708036248  
741673882  
3242607330  
793006822  
1555008860  
1048923598  
3451958906  
3616887600  
2068474104  
428301276  
1205746606  
2869283920  
1177516456  
2434337264  
3586592136  
2776393968  
3457921500  
497530206  
1265554160  
2738343898  
3206199546  
2606144222  
2750292674  
3512698572  
3820828982  
1903301600  
3734429094  
3368295302  
3034973464  
2244257998  
2778941310  
2456716790  
2666849498  
3318513074  
666697018  
2053022618  
4091166358  
1461975658  
667334222  
3285342132  
1397196532  
2655514232  
3447184204  
2469520214  
2394570402  
1283225276  
2725819464  
2673045446  
1200049808  
3515598226  
383676536  
3784835088  
3791420  
220929854  
1510080526  
994542410  
1676138112  
1802809454  
75360198  
1206070022  
3120018300  
3580911054  
71118992  
3319870214  
3135100852  
1332395112  
2083937844  
3644667520  
1472002256  
4254866852  
3176681902  
1790732528  
565821968  
1736582102  
2649155656  
2261034932  
2935053186  
2000167400  
3271210472  
1154461828  
2608482556  
3621937104  
3715317158  
3454940552  
3941733358  
561197536  
4103458290  
3661371990  
524203014  
3715783936  
4226525078  
3458147194  
2814158556  
1054073670  
3956983628  
3832921804  
1528599034  
1304359100  
2295898602  
155009636  
2001039244  
1385978116  
1528626490  
1639375284  
2059146668  
3904301386  
3999499336  
3233180508  
1207060888  
1027755850  
2204756880  
2246914276  
3116267150  
2782981878  
2749429552  
4192304374  
1339981432  
3548841808  
2606277540  
1787789094  
410057324  
2960626576  
1334668676  
3277730940  
4029408070  
2987162860  
2399573390  
3484221934  
1586970748  
880419682  
1159467422  
4091181790  
2404244860  
3549358174  
246379708  
1895358004  
3856877436  
2546160090  
3215160808  
2053964228  
3268117498  
3687652790  
3953629696  
4134509986  
4193370312  
2108350684  
2651836236  
3491719510  
1952152000  
3191109684  
1446935912  
131922812  
4216711584  
2960606816  
2097378  
1138434488  
1855734  
1681370586  
3190903698  
792229726  
3196183676  
3581581702  
3942820514  
533131694  
2849150876  
501570662  
4024684984  
2088484018  
501676202  
2113170510  
2212722230  
3283409532  
2791438238  
3320690896  
4000988830  
928758902  
2947446598  
3839428918  
3753532016  
1885196102  
3924077004  
4171361680  
1876786594  
723800146  
2747861332  
3739810036  
870533026  
4109522172  
1169312224  
2492750776  
521813820  
3413063804  
2119573180  
3555608538  
440136980  
3285693614  
927123316  
4122419084  
2922915308  
3362779610  
2077061936  
602450152  
2101140016  
3198920068  
1601312562  
2385906742  
2081450726  
2265193064  
3758717378  
2615634172  
3450556328  
3690165436  
1894884330  
856677116  
539356560  
1871306108  
3118275934  
1313415366  
2720117032  
4232072154  
2430538682  
2124060362  
2794853112  
931995278  
3553186802  
1802938338  
61578254  
3042870138  
3797338022  
2271122290  
1595555382  
3109521256  
4206286072  
1760971004  
465997714  
3348882090  
4054053848  
3069413326  
3848399834  
1050401284  
4174506774  
3698764766  
3385321526  
3714973710  
584656112  
1959569346  
3115901078  
3676422022  
1913447858  
1441899258  
4098188258  
3135811496  
3690046448  
3169050756  
1870249590  
1321088866  
3158003668  
3622622062  
2453547332  
3676320710  
731103728  
3521238478  
64663310  
2762891264  
536904388  
2800216040  
2513948442  
3963224070  
2282427702  
2678039726  
1597796192  
3943945076  
975428174  
3042849918  
1292916286  
460090334  
1025462070  
1177488648  
1198232964  
1936719952  
1492858140  
3619911888  
4201469418  
3473614134  
1155250294  
1332395112  
2090366002  
2515590598  
856642846  
1916777106  
3040794578  
276160106  
921286774  
1332344802  
2076229918  
3163417528  
1656026846  
3786806878  
1811047766  
2722745842  
4218406542  
3619000098  
1384705972  
2928650130  
1406186246  
3107046566  
3048072006  
4096421500  
2761283410  
3179765590  
1586005416  
2675166548  
2288118686  
2254763934  
550390958  
2620939130  
2334684826  
1867967554  
2300985980  
3833455634  
314803430  
47913390  
1001246066  
3709611250  
4206227852  
3713820018  
3195441104  
1423768210  
732866294  
247281478  
2874302952  
2515046308  
3928302580  
2608613762  
3686202946  
1864782520  
3745204070  
1399641832  
3728078626  
4254860442  
3283398564  
630098170  
909369784  
3398133692  
401136226  
2855849330  
1264929176  
1711277056  
242365920  
1975766558  
2664653568  
59440356  
904116344  
3788983694  
3716009778  
2209920922  
644082906  
789870034  
2985987520  
1534706594  
635731062  
2193548414  
1307537272  
257052732  
1892545460  
1041966858  
2027766308  
2837412336  
3521881798  
4206553056  
1765714910  
2721330318  
3813124186  
2448250698  
2879314932  
1870363584  
2076725792  
3731943846  
704198210  
653779378  
945274104  
1760594810  
2576256934  
3377714562  
1930712656  
2209311720  
3030007350  
1836382550  
2100336102  
2275784472  
788780342  
2926279568  
4197757916  
2602276268  
807198526  
3069773422  
1313420732  
4249262920  
4038377816  
1556576748  
2479321330  
3283000094  
2914582076  
2364920314  
2420706320  
4234735792  
1535614010  
2868526058  
331574966  
2914566542  
3292701446  
4223193566  
2881992990  
2272556958  
3196355200  
3743070696  
3635468136  
4174506156  
336227898  
2868367672  
872752950  
327387876  
2017383070  
2019433916  
472822998  
2079967446  
3346068358  
2929401802  
3064914730  
4157697764  
1012551230  
499371550  
2456693356  
3925161722  
278302530  
2235069816  
3767792172  
3374788608  
268456070  
734254910  
2320660356  
2147504262  
734254908  
1693387306  
3144544376  
3662890932  
4226840398  
3170678082  
1546345450  
1808864062  
2211918830  
3840144538  
3419283300  
321952316  
3833401610  
2839488200  
618594030  
2475504742  
1949102260  
1668723168  
3729659662  
2927203260  
341244210  
1579071974  
662196726  
891137926  
2078809670  
3981955508  
1666471660  
3391002456  
947767548  
3152075616  
2548357326  
2128903378  
1752603352  
515624476  
635529640  
3811709934  
603614350  
3700398346  
2243674598  
2266167406  
3361427742  
3215706922  
489640416  
3031194818  
879163866  
989346726  
3716044728  
944020442  
880581408  
2474372414  
440208176  
999496738  
1868980532  
863828726  
3237717380  
2556351860  
3278752506  
3372109336  
191200578  
624798456  
2573082296  
4079450192  
4153820918  
3980276858  
187725606  
514692204  
590387762  
3190461688  
1761275778  
3066191730  
528047262  
3080470084  
3643079568  
4207111626  
2112681178  
290004934  
2027406536  
3404789198  
2570830476  
658851730  
2269328056  
3221258242  
3479889166  
2877165474  
3676963696  
2275148256  
3709420  
2924541294  
3303077978  
2499917958  
3456169844  
4226970106  
3372109200  
1237186618  
3552879742  
231101498  
3122917252  
1997992916  
3246084042  
2737654166  
1314936252  
2101796954  
2515359684  
783175924  
3083139436  
3272630652  
4135432104  
3377161040  
836978318  
3181254164  
1626252016  
1549005218  
2749288394  
3055059526  
3906509864  
161733774  
3813013442  
1766937206  
1601912246  
466480782  
3367389496  
3745203534  
3853916290  
2247603176  
3687475448  
3066237000  
2098535908  
934208276  
3555833552  
390959864  
927384800  
2689408488  
3160153940  
3915460420  
3720927156  
2942236396  
318413988  
904536298  
3984083538  
4184450220  
3072057584  
3036460738  
2284973638  
4124861886  
195361938  
3121534774  
3807823714  
3708509082  
1941578574  
2621132684  
2403519516  
3838301614  
3157247068  
2924541294  
3304443604  
3980215324  
3182386524  
1358262210  
693095244  
2461383402  
1921493886  
190025888  
1991621354  
1921493886  
3292684130  
2736251634  
1489303330  
4082945884  
2924541294  
3944201432  
2147483648

解码后结果：



### 必做题第六题

###### 题目：最小生成树 (必做) （图）

[问题描述]

利用普利姆算法和克鲁斯卡尔算法实现最小生成树问题。

[基本要求]

（1）自行建立图的数据文件，第一行是顶点个数，然后依次是顶点名，接下来是边，用float表示边的权值；

（2）以邻接表或者邻接矩阵表示图皆可；

（3）分别利用prim和Kruskal算法实现最小生成树；

（4）输出最小生成树的权值之和，及所用的边。

###### 数据结构：图，邻接表

###### 算法思想：

prim（普利姆算法）：  
 点集v1含有所有点，v2为空，边集合e为空  
 选择最短的边，加入到e中，将两个端点从v1中删除，加入v2中  
 重复执行，寻找最短的边，一个端点在v1中，一个端点在v2中，加入到e中，将不在v2中 的点加入到v2中，直到v1为空  
kruskal (克鲁斯卡尔算法) ：  
 边集合e含所有边，点集v为空  
 将e按大小排序  
 重复执行，取出距离最小的边，若有其中一个端点不在v中，存入该边，直到e为空

###### 算法时间复杂度：prim 算法 n^2 （n为点的个数） / kruskal 算法 e^2 （e为边的个数）

###### 具体实现：

my\_Graph.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_MY\_GRAPH\_H  
#define DESIGN\_MY\_GRAPH\_H  
  
#include <string>  
#include <vector>  
#include <fstream>  
#define MAX 65535  
  
class edge{  
public:  
 int node1,node2;  
 float distance;  
 edge(int n1,int n2,float d)//initial  
 {  
 node1 = n1;  
 node2 = n2;  
 distance = d;  
 }  
 ~edge()= default;;  
};  
class GNode{  
private:  
 int site;  
 std::string data;  
protected:  
public:  
 std::vector<float> out\_e;//out egde , undirected graph only use this  
 std::vector<float> in\_e;//in edge  
 GNode(int s, std::string d)//initial  
 {  
 site = s;  
 data = d;  
 }  
 GNode(int s, std::string d, int n)//initial  
 {  
 site = s;  
 data = d;  
 out\_e.assign(n,MAX);  
 }  
 bool setSite(int s)//change this node's site  
 {  
 site = s;  
 return true;  
 }  
 bool setData(std::string da)//change this node's data  
 {  
 data = da;  
 }  
 float distanceNode(int st)//return between this node and st  
 {  
 return out\_e[st];  
 }  
 bool enOutEdge(int st,float dis)//set distance between this node and st  
 {  
 out\_e[st] = dis;  
  
 return true;  
 }  
 int outEdgeNumber()//return out edge number  
 {  
 return out\_e.size();  
 }  
 bool show();//show this node information  
 std::string getData();//return this node data  
 int firstAdjVex();//return the nearest node  
 int nextAdjVex(int node1);//return the nearest node after node1  
 bool deleteVex(int node1);//delete edge with node1  
 ~GNode()= default;  
};  
class my\_Graph {  
private:  
 std::vector<GNode> frt;  
 int nd\_num;  
 int eg\_num;  
public:  
 my\_Graph(int nd\_n,int eg\_n);//initial  
 my\_Graph(int nd\_n,int eg\_n,float dis);//initial  
 my\_Graph(std::string filename);//initial  
 ~my\_Graph();  
 int locateVex(std::string str);//locate node  
 std::string getVex(int site);//return node's data  
 bool putVex(int site,std::string str);//change the data of the node which st == site  
 GNode \* firstAdjVex(int site);//return the nearest node of the ndoe st == site  
 GNode \* nextAdjVex(int node1,int node2);//return the nearest node after node1 of the ndoe st == site  
 float primSpanningTree();//prim  
 float kruskalSpinningTree();//kruskal  
 bool InsertVex();//add node  
 bool DeleteVex(int site);//delete node  
 bool InsertArc(int node1,int node2);//add edge  
 bool BFSTraverse(int bgn = 0);//BFS traverse the graph  
 int getNodeNumber();//return the node number  
 bool dijkstra(std::vector<float> & T,int bgn);//find short distance from bgn to other node  
};  
class gNode\_plus : public GNode{  
private:  
public:  
 std::vector<int> bus;  
 gNode\_plus(int s, std::string d):GNode(s,d){}//initial  
 gNode\_plus(int s, std::string d, int n):GNode(s, d, n){}//initial  
 ~gNode\_plus(){}  
 bool enBus(int n);//add bus line  
 bool deBus(int n);//delete bus line  
};  
class my\_Graph\_Bus {  
private:  
 std::vector<gNode\_plus> frt;  
 int nd\_num;  
 int eg\_num;  
public:  
 my\_Graph\_Bus(std::string filename);//initial  
 int locateVex(std::string str);//locate node  
 bool BFSTraverse(int bgn = 0);//BFS traverse the graph  
 bool bfsGo(std::string node1,std::string node2);//find the fast way  
 bool dfsGo(std::string node1,std::string node2);//find the easiest way  
 int someLine(int node1,int node2,int limit = 0);//return the same line between after limit  
};  
class bfsNode  
{  
public:  
 int nw;  
 int pa;  
 bfsNode(int n = -1,int p = -1){  
 nw = n;  
 pa = p;  
 }  
 bool operator=(const bfsNode &b) {  
 nw = b.nw;  
 pa = b.pa;  
 }  
};  
#endif //DESIGN\_MY\_GRAPH\_H

my\_Graph.cpp

//  
// Created by 97927 on 2019/12/13.  
//  
  
#include "my\_Graph.h"  
#include <iostream>  
#include "function.h"  
#include <queue>  
#include <fstream>  
#include <vector>  
using std::cout;  
using std::cin;  
using std::endl;  
#define YTD  
  
bool sortVector(std::vector<edge> &e)  
{  
 int length = e.size();  
 for(int k = 0;k < length;k++)  
 {  
 int minW = e[k].distance;  
 int minS = k;  
 for(int i = k + 1;i < length;i++)  
 {  
 if(e[i].distance > minW)  
 {  
 minW = e[i].distance;  
 minS = i;  
 }  
 }  
 if(minS == k)  
 continue;  
 edge pause = e[minS];  
 e.erase(e.begin() + minS);  
 e.insert(e.begin() + k,pause);  
 }  
 return true;  
}  
  
my\_Graph::my\_Graph(int nd\_n,int eg\_n) {  
 nd\_num = nd\_n;  
 eg\_num = eg\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str = getString();  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 }  
 for(int i = 0;i < eg\_n;i++)  
 {  
 int n\_st1 = getInt("node1 site");  
 int n\_st2 = getInt("node2 site");  
 float dis = getFloat("distance");  
 frt[n\_st1].enOutEdge(n\_st2,dis);  
 frt[n\_st2].enOutEdge(n\_st1,dis);  
 }  
}  
  
int my\_Graph::locateVex(std::string str) {  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[i].getData() == str)  
 return i;  
 }  
 return -1;  
}  
  
std::string my\_Graph::getVex(int site) {  
 return frt[site].getData();  
}  
  
bool my\_Graph::putVex(int site, std::string str) {  
 frt[site].setData(str);  
  
 return true;  
}  
  
GNode \*my\_Graph::firstAdjVex(int site) {  
 int st = frt[site].firstAdjVex();  
 if(st == -1)  
 return nullptr;  
 return &frt[st];  
}  
  
GNode \*my\_Graph::nextAdjVex(int node1, int node2) {  
 int st = frt[node1].nextAdjVex(node2);  
 if(st == -1)  
 return nullptr;  
 return &frt[st];  
}  
  
bool my\_Graph::InsertVex() {  
 std::string str = getString();  
 GNode n\_node(nd\_num,str,nd\_num++);  
 frt.push\_back(n\_node);  
 for(int i = 0;i < nd\_num;i++)  
 {  
 frt[i].out\_e.push\_back(MAX);  
 }  
 frt[nd\_num].show();  
  
 return true;  
}  
  
bool my\_Graph::DeleteVex(int site) {  
 char c;  
 frt[site].show();  
 cout<<"concern the command : ( y to yes)";  
 //decide the delete  
 cin>>c;  
 if(c != 'y')  
 return false;  
 frt.erase(frt.begin() + site);  
 nd\_num--;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 frt[i].deleteVex(site);  
 }  
  
 return true;  
}  
  
bool my\_Graph::InsertArc(int node1, int node2) {  
 float dis = getFloat("distance");  
 frt[node1].enOutEdge(node2,dis);  
 frt[node2].enOutEdge(node1,dis);  
  
 return true;  
}  
  
bool my\_Graph::BFSTraverse(int bgn) {  
 //sign the node  
 std::vector<int> D(nd\_num,0);  
 //store the node  
 std::queue<int> P;  
 frt[bgn].show();  
 cout<<endl;  
 D[bgn] = 1;  
 //find near node  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[bgn].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 while(!P.empty()){  
 int node1 = P.front();  
 P.pop();  
 frt[node1].show();  
 cout<<endl;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i] || frt[node1].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 }  
  
 return true;  
}  
  
my\_Graph::my\_Graph(std::string filename) {  
 std::ifstream in(filename);  
 int nd\_n;  
 in>>nd\_n;  
 nd\_num = nd\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str;  
 in>>str;  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 cout<<endl;  
 }  
 while(!in.eof())  
 {  
 eg\_num++;  
 int n\_st1,n\_st2;  
 std::string n1,n2;  
 float dis;  
 in>>n1>>n2>>dis;  
 n\_st1 = locateVex(n1);  
 n\_st2 = locateVex(n2);  
 if(n\_st1 == -1 || n\_st2 == -1)  
 continue;  
 frt[n\_st1].enOutEdge(n\_st2,dis);  
 frt[n\_st2].enOutEdge(n\_st1,dis);  
 }  
 in.close();  
}  
  
int my\_Graph::getNodeNumber() {  
 return nd\_num;  
}  
  
float my\_Graph::primSpanningTree() {  
 //vector to concern the node have past  
 std::vector<bool> D(nd\_num,false);  
 D[0] = true;  
 float sum = 0;  
 while(true)  
 {  
 float min = MAX;  
 int node1 = 0,node2 = 0;  
 bool flag = false;  
 //find the nearest node  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 {  
 for(int k = 0;k < nd\_num;k++)  
 {  
 if(D[k])  
 continue;  
 if(frt[i].out\_e[k] < min)  
 {  
 flag = true;  
 node1 = i;  
 node2 = k;  
 min = frt[i].out\_e[k];  
 }  
 }  
 }  
 }  
 //do not have , break  
 if(!flag)  
 {  
 break;  
 }  
 //output edge  
 if(D[node1])  
 {  
#ifndef YTD  
 cout<<"enter ";  
 frt[node2].show();  
#endif  
 D[node2] = true;  
 cout<<"edge node1 : "<<frt[node1].getData()<<" node2 : "<<frt[node2].getData();  
 cout<<" distance : "<<min<<endl;  
 } else{  
#ifndef YTD  
 cout<<"enter node : ";  
 frt[node1].show();  
#endif  
 D[node1] = true;  
 cout<<endl<<"edge node1 : "<<frt[node1].getData()<<" node2 : "<<frt[node2].getData();  
 cout<<" distance : "<<min<<endl;  
 }  
 sum += min;  
 }  
 cout<<"prim end."<<endl;  
 return sum;  
}  
  
float my\_Graph::kruskalSpinningTree() {  
 //vector to concern the node have past ?  
 std::vector<bool> D(nd\_num,false);  
 float sum = 0;  
 std::vector<edge> P;  
 //sort edge  
 for(int i = 0;i < nd\_num;i++)  
 {  
 for(int k = i + 1;k < nd\_num;k++)  
 {  
 if(frt[i].out\_e[k] != MAX)  
 {  
 edge nw\_eg(i,k,frt[i].out\_e[k]);  
 P.push\_back(nw\_eg);  
 }  
 }  
 }  
 sortVector(P);  
 while(!P.empty())  
 {  
 edge pause = P.back();  
 P.pop\_back();  
 //find the shortest edge  
 if(!D[pause.node1] || !D[pause.node2])  
 {  
 sum += pause.distance;  
 D[pause.node1] = true;  
 D[pause.node2] = true;  
 cout<<"edge node1 : "<<frt[pause.node1].getData()<<" node2 : "<<frt[pause.node2].getData();  
 cout<<" distance : "<<pause.distance<<endl;  
 } else{  
 continue;  
 }  
 }  
 cout<<"kru end."<<endl;  
 return sum;  
}  
  
my\_Graph::my\_Graph(int nd\_n, int eg\_n, float dis) {  
 nd\_num = nd\_n;  
 eg\_num = eg\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str;  
 str += 'a' + i;  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 }  
 for(int i = 0;i < eg\_n;i++)  
 {  
 int n\_st1 = getInt("node1 site");  
 int n\_st2 = getInt("node2 site");;  
 frt[n\_st1 - 1].enOutEdge(n\_st2 - 1,dis);  
 frt[n\_st2 - 1].enOutEdge(n\_st1 - 1,dis);  
 }  
}  
  
bool my\_Graph::dijkstra(std::vector<float> & T,int bgn) {  
 std::vector<bool> D(nd\_num,false);  
 for(int i = 0;i < nd\_num;i++)  
 {  
 T.push\_back(frt[bgn].out\_e[i]);  
 }  
 int p = bgn;  
 D[bgn] = true;  
 while(true)  
 {  
 int min = -1;  
 //find the nearest node do not be traverse  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 continue;  
 if(min == -1 || T[i] < min)  
 {  
 min = T[i];  
 p = i;  
 }  
 }  
 if(min == -1)  
 break;  
 //refresh the distance  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 continue;  
 if(T[i] > T[p] + frt[p].out\_e[i])  
 {  
 T[i] = T[p] + frt[p].out\_e[i];  
 }  
 }  
 D[p] = true;  
 }  
 return true;  
}  
  
my\_Graph::~my\_Graph() = default;  
  
bool GNode::show() {  
 cout<<"node : "<<data<<" site : "<<site;  
  
 return true;  
}  
  
std::string GNode::getData() {  
 return data;  
}  
  
int GNode::firstAdjVex() {  
 int i = 0;  
 for(;i < out\_e.size();i++)  
 {  
 if(out\_e[i] != MAX)  
 {  
 break;  
 }  
 }  
 if(out\_e[i] != MAX)  
 return i;  
 return -1;  
}  
  
int GNode::nextAdjVex(int node1) {  
 int i = node1 + 1;  
 for(;i < out\_e.size();i++)  
 {  
 if(out\_e[i] != MAX)  
 break;  
 }  
 if(out\_e[i] != MAX)  
 return i;  
 return -1;  
}  
  
bool GNode::deleteVex(int node1) {  
 out\_e.erase(out\_e.begin() + node1);  
 return true;  
}  
  
bool gNode\_plus::enBus(int n) {  
 int i = 0;  
 for(;i < bus.size();i++)  
 {  
 if(bus[i] == n)  
 return false;  
 if(bus[i] > n)  
 break;  
 }  
 bus.insert(bus.begin() + i,n);  
  
 return true;  
}  
  
bool gNode\_plus::deBus(int n) {  
 int i = 0;  
 for(;i < bus.size();i++)  
 {  
 if(bus[i] == n)  
 break;  
 if(bus[i] > n)  
 return false;  
 }  
 if(bus[i] == n)  
 bus.erase(bus.begin() + i);  
  
 return true;  
}  
  
my\_Graph\_Bus::my\_Graph\_Bus(std::string filename)  
{  
 std::ifstream in(filename);  
 nd\_num = 0;  
 eg\_num = 0;  
 while(!in.eof())  
 {  
 int bsln;  
 in>>bsln;  
 int str\_last = -1;  
 while(true)  
 {  
 std::string str;  
 in>>str;  
 if(str == "z")  
 break;  
 int st = locateVex(str);  
 if(st == -1)  
 {  
 frt.push\_back(gNode\_plus (nd\_num,str,5652));  
 st = nd\_num;  
 nd\_num++;  
 }  
 frt[st].enBus(bsln);  
 if(str\_last != -1)  
 {  
 frt[st].enOutEdge(str\_last,1);  
 frt[str\_last].enOutEdge(st,1);  
 eg\_num++;  
 }  
 str\_last = st;  
 }  
 }  
  
 in.close();  
}  
  
int my\_Graph\_Bus::locateVex(std::string str) {  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[i].getData() == str)  
 return i;  
 }  
 return -1;  
}  
  
bool my\_Graph\_Bus::BFSTraverse(int bgn) {  
 std::vector<int> D(nd\_num,0);  
 std::queue<int> P;  
 frt[bgn].show();  
 cout<<endl;  
 D[bgn] = 1;  
 //find the near node  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[bgn].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 while(!P.empty()){  
 int node1 = P.front();  
 P.pop();  
 frt[node1].show();  
 cout<<endl;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i] || frt[node1].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 }  
  
 return true;  
}  
  
bool my\_Graph\_Bus::bfsGo(std::string node1, std::string node2) {  
 int nst1 = locateVex(node1);  
 int nst2 = locateVex(node2);  
  
 std::vector<bool> D(nd\_num, false);  
 std::queue<bfsNode> P;  
 std::stack<bfsNode> s;  
 D[nst1] = true;  
 P.push(bfsNode(nst1));  
  
 while(!P.empty()){  
 bfsNode nd = P.front();  
 P.pop();  
 s.push(nd);  
 if(nd.nw == nst2)  
 break;  
 //find near nodes  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i] || frt[nd.nw].out\_e[i] == MAX)  
 continue;  
 P.push(bfsNode(i,nd.nw));  
 D[i] = true;  
 }  
 }  
 if(P.empty())  
 {  
 cout<<"Don't have way."<<endl;  
 return false;  
 }  
 //find the node's pa from pause stack  
 int pa = s.top().pa;  
 bfsNode nd = s.top();  
 int dr = someLine(nd.nw,nd.pa);  
 int last\_node = nd.nw;  
 int count = 1;  
 cout<<"最快 : "<<endl;  
 cout<<frt[nd.nw].getData()<<" -> line "<<dr<<" : ";  
 s.pop();  
 while(!s.empty())  
 {  
 nd = s.top();  
 if(nd.nw == pa)  
 {  
 count++;  
 //when the line is changed  
 if(dr != someLine(last\_node,pa))  
 {  
 dr = someLine(last\_node,pa);  
 cout<<frt[last\_node].getData()<<" "<<count<<"zhan"<<endl<<frt[last\_node].getData()<<" -> ";  
 cout<<"line "<<dr<<" : ";  
 count = 1;  
 }  
 last\_node = pa;  
 pa = nd.pa;  
 }  
 s.pop();  
 if(s.empty())  
 {  
 cout<<frt[nd.nw].getData();  
 cout<<endl;  
 }  
 }  
 return true;  
}  
  
bool dirPass(std::vector<int> dr,int dir)  
{  
 for(int i = 0;i < dr.size();i++)  
 {  
 if(dr[i] == dir)  
 return true;  
 }  
 return false;  
}  
  
class dfsNode{  
public:  
 int nw;  
 int bus;  
 dfsNode(int n1 = -1,int n2 = -1)  
 {  
 nw = n1;  
 bus = n2;  
 }  
 ~dfsNode();  
};  
  
dfsNode::~dfsNode() {  
}  
  
bool my\_Graph\_Bus::dfsGo(std::string node1, std::string node2) {  
 std::stack<dfsNode> s;  
 std::vector<bool> D(nd\_num,false);  
 std::vector<int> dr;//direction have go  
 int site1 = locateVex(node1);  
 D[site1] = true;  
 int site2 = locateVex(node2);  
 s.push(dfsNode(site1,frt[site1].bus[0]));  
 bool flag = true;  
 bool end = false;  
 dr.push\_back(frt[site1].bus[0]);  
 dfsNode p;  
 while(!s.empty()) {  
 //do not have the node  
 if(!flag){  
 end = false;  
 p = s.top();  
 //change direction  
 for(int i = 0;i < frt[p.nw].out\_e.size();i++) {  
 if (frt[p.nw].out\_e[i] != 1 || D[i])  
 continue;  
 int dir = someLine(p.nw, i,p.bus);  
 if (dirPass(dr, dir)) {  
 continue;  
 } else {  
 dr.push\_back(dir);  
 s.push(dfsNode(i,dir));  
 end = true;  
 flag = true;  
 break;  
 }  
 }  
 if(!end)  
 {  
 D[s.top().nw] = false;  
 s.pop();  
 }  
 continue;  
 }  
 flag = false;  
 p = s.top();  
 //find the next node along line dir  
 int dir = p.bus;  
 int k;  
 int i;  
 for(i = 0;i < frt[p.nw].out\_e.size();i++)  
 {  
 if(frt[p.nw].out\_e[i] > 2 || D[i])  
 continue;  
 for(k = 0;k < frt[i].bus.size();k++)  
 {  
 if(frt[i].bus[k] == dir)  
 {  
 s.push(dfsNode(i,dir));  
 D[i] = true;  
 flag = true;  
 /\*cout<<frt[i].show()<<endl;\*/  
 break;  
 }  
 }  
 if(flag)  
 break;  
 }  
 if(i == site2)  
 break;  
 }  
 if(s.empty())  
 {  
 cout<<"don't have way"<<endl;  
 return false;  
 }  
 else{  
 cout<<"最少换程 : "<<endl;  
 int last\_node = s.top().nw;  
 int count = 1;  
 cout<<frt[s.top().nw].getData();  
 int dr = s.top().bus;  
 cout<<" -> line "<<dr<<" : ";  
 while(!s.empty())  
 {  
 count ++;  
 //if change bus  
 if(dr != s.top().bus)  
 {  
 dr = s.top().bus;  
 cout<<frt[s.top().nw].getData()<<" "<<count<<"zhan"<<endl;  
 cout<<frt[s.top().nw].getData()<<" -> line "<<dr<<" : ";  
 count = 1;  
 }  
 if(s.size() == 1)  
 cout<<frt[s.top().nw].getData()<<endl;  
 s.pop();  
 }  
 }  
 return true;  
}  
  
int my\_Graph\_Bus::someLine(int node1, int node2, int n) {  
 int i = 0;  
 int k = 0;  
 for(;i < frt[node1].bus.size();)  
 {  
 if(frt[node1].bus[i] == frt[node2].bus[k])  
 {  
 if(frt[node1].bus[i] < n)  
 {  
 i++;  
 k++;  
 continue;  
 }  
 return frt[node1].bus[i];  
 } else if (frt[node2].bus[k] < frt[node1].bus[i])  
 {  
 k++;  
 } else{  
 i++;  
 }  
 }  
  
 return -1;  
}

task6.cpp

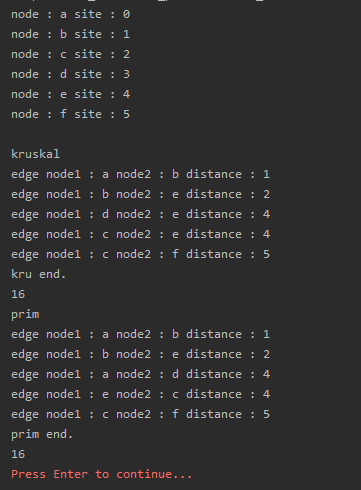
//  
// Created by 97927 on 2019/12/25.  
//  
#include <iostream>  
#include <cstdio>  
#include <cstdlib>  
#include "my\_Graph.h"  
#include <vector>  
#include "function.h"  
using namespace std;  
  
int main()  
{  
 my\_Graph g("spanningtree.txt");  
 //g.BFSTraverse();  
 cout<<endl;  
 cout<<"kruskal"<<endl<<g.kruskalSpinningTree()<<endl;  
 cout<<"prim"<<endl<<g.primSpanningTree()<<endl;  
  
 system("read -p 'Press Enter to continue...' var");  
 return 0;  
}

###### 测试样例与运行结果：

测试样例：

6 a b c d e f a b 1 a d 4 a e 3 b d 4 b e 2 c e 4 c f 5 d e 4 e f 7

运行结果：



### 必做题第七题

###### 题目：公交线路提示 (必做) （图）

[问题描述]

上网下载真实南京公交线路图，建立南京主要公交线路图的存储结构。

[基本要求]

（1）输入任意两站点，给出转车次数最少的乘车路线。

（2）输入任意两站点，给出经过站点最少的乘车路线。

###### 数据结构：栈

###### 算法思想：

转车次数最少：  
 使用深度优先搜索的策略，先沿着某一条公交路线进行搜索，直到终点站，在返回，换其他路线，直到找到目的地  
经过站点最少：  
 使用广度优先搜索的策略，从起始站出发，遍历与起始站相邻的所有点，在遍历与现在所遍历的点的所有相邻点，直到找到目的地

###### 算法时间复杂度：n

###### 具体实现：

my\_Graph.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_MY\_GRAPH\_H  
#define DESIGN\_MY\_GRAPH\_H  
  
#include <string>  
#include <vector>  
#include <fstream>  
#define MAX 65535  
  
class edge{  
public:  
 int node1,node2;  
 float distance;  
 edge(int n1,int n2,float d)//initial  
 {  
 node1 = n1;  
 node2 = n2;  
 distance = d;  
 }  
 ~edge()= default;;  
};  
class GNode{  
private:  
 int site;  
 std::string data;  
protected:  
public:  
 std::vector<float> out\_e;//out egde , undirected graph only use this  
 std::vector<float> in\_e;//in edge  
 GNode(int s, std::string d)//initial  
 {  
 site = s;  
 data = d;  
 }  
 GNode(int s, std::string d, int n)//initial  
 {  
 site = s;  
 data = d;  
 out\_e.assign(n,MAX);  
 }  
 bool setSite(int s)//change this node's site  
 {  
 site = s;  
 return true;  
 }  
 bool setData(std::string da)//change this node's data  
 {  
 data = da;  
 }  
 float distanceNode(int st)//return between this node and st  
 {  
 return out\_e[st];  
 }  
 bool enOutEdge(int st,float dis)//set distance between this node and st  
 {  
 out\_e[st] = dis;  
  
 return true;  
 }  
 int outEdgeNumber()//return out edge number  
 {  
 return out\_e.size();  
 }  
 bool show();//show this node information  
 std::string getData();//return this node data  
 int firstAdjVex();//return the nearest node  
 int nextAdjVex(int node1);//return the nearest node after node1  
 bool deleteVex(int node1);//delete edge with node1  
 ~GNode()= default;  
};  
class my\_Graph {  
private:  
 std::vector<GNode> frt;  
 int nd\_num;  
 int eg\_num;  
public:  
 my\_Graph(int nd\_n,int eg\_n);//initial  
 my\_Graph(int nd\_n,int eg\_n,float dis);//initial  
 my\_Graph(std::string filename);//initial  
 ~my\_Graph();  
 int locateVex(std::string str);//locate node  
 std::string getVex(int site);//return node's data  
 bool putVex(int site,std::string str);//change the data of the node which st == site  
 GNode \* firstAdjVex(int site);//return the nearest node of the ndoe st == site  
 GNode \* nextAdjVex(int node1,int node2);//return the nearest node after node1 of the ndoe st == site  
 float primSpanningTree();//prim  
 float kruskalSpinningTree();//kruskal  
 bool InsertVex();//add node  
 bool DeleteVex(int site);//delete node  
 bool InsertArc(int node1,int node2);//add edge  
 bool BFSTraverse(int bgn = 0);//BFS traverse the graph  
 int getNodeNumber();//return the node number  
 bool dijkstra(std::vector<float> & T,int bgn);//find short distance from bgn to other node  
};  
class gNode\_plus : public GNode{  
private:  
public:  
 std::vector<int> bus;  
 gNode\_plus(int s, std::string d):GNode(s,d){}//initial  
 gNode\_plus(int s, std::string d, int n):GNode(s, d, n){}//initial  
 ~gNode\_plus(){}  
 bool enBus(int n);//add bus line  
 bool deBus(int n);//delete bus line  
};  
class my\_Graph\_Bus {  
private:  
 std::vector<gNode\_plus> frt;  
 int nd\_num;  
 int eg\_num;  
public:  
 my\_Graph\_Bus(std::string filename);//initial  
 int locateVex(std::string str);//locate node  
 bool BFSTraverse(int bgn = 0);//BFS traverse the graph  
 bool bfsGo(std::string node1,std::string node2);//find the fast way  
 bool dfsGo(std::string node1,std::string node2);//find the easiest way  
 int someLine(int node1,int node2,int limit = 0);//return the same line between after limit  
};  
class bfsNode  
{  
public:  
 int nw;  
 int pa;  
 bfsNode(int n = -1,int p = -1){  
 nw = n;  
 pa = p;  
 }  
 bool operator=(const bfsNode &b) {  
 nw = b.nw;  
 pa = b.pa;  
 }  
};  
#endif //DESIGN\_MY\_GRAPH\_H

my\_Graph.cpp

//  
// Created by 97927 on 2019/12/13.  
//  
  
#include "my\_Graph.h"  
#include <iostream>  
#include "function.h"  
#include <queue>  
#include <fstream>  
#include <vector>  
using std::cout;  
using std::cin;  
using std::endl;  
#define YTD  
  
bool sortVector(std::vector<edge> &e)  
{  
 int length = e.size();  
 for(int k = 0;k < length;k++)  
 {  
 int minW = e[k].distance;  
 int minS = k;  
 for(int i = k + 1;i < length;i++)  
 {  
 if(e[i].distance > minW)  
 {  
 minW = e[i].distance;  
 minS = i;  
 }  
 }  
 if(minS == k)  
 continue;  
 edge pause = e[minS];  
 e.erase(e.begin() + minS);  
 e.insert(e.begin() + k,pause);  
 }  
 return true;  
}  
  
my\_Graph::my\_Graph(int nd\_n,int eg\_n) {  
 nd\_num = nd\_n;  
 eg\_num = eg\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str = getString();  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 }  
 for(int i = 0;i < eg\_n;i++)  
 {  
 int n\_st1 = getInt("node1 site");  
 int n\_st2 = getInt("node2 site");  
 float dis = getFloat("distance");  
 frt[n\_st1].enOutEdge(n\_st2,dis);  
 frt[n\_st2].enOutEdge(n\_st1,dis);  
 }  
}  
  
int my\_Graph::locateVex(std::string str) {  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[i].getData() == str)  
 return i;  
 }  
 return -1;  
}  
  
std::string my\_Graph::getVex(int site) {  
 return frt[site].getData();  
}  
  
bool my\_Graph::putVex(int site, std::string str) {  
 frt[site].setData(str);  
  
 return true;  
}  
  
GNode \*my\_Graph::firstAdjVex(int site) {  
 int st = frt[site].firstAdjVex();  
 if(st == -1)  
 return nullptr;  
 return &frt[st];  
}  
  
GNode \*my\_Graph::nextAdjVex(int node1, int node2) {  
 int st = frt[node1].nextAdjVex(node2);  
 if(st == -1)  
 return nullptr;  
 return &frt[st];  
}  
  
bool my\_Graph::InsertVex() {  
 std::string str = getString();  
 GNode n\_node(nd\_num,str,nd\_num++);  
 frt.push\_back(n\_node);  
 for(int i = 0;i < nd\_num;i++)  
 {  
 frt[i].out\_e.push\_back(MAX);  
 }  
 frt[nd\_num].show();  
  
 return true;  
}  
  
bool my\_Graph::DeleteVex(int site) {  
 char c;  
 frt[site].show();  
 cout<<"concern the command : ( y to yes)";  
 //decide the delete  
 cin>>c;  
 if(c != 'y')  
 return false;  
 frt.erase(frt.begin() + site);  
 nd\_num--;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 frt[i].deleteVex(site);  
 }  
  
 return true;  
}  
  
bool my\_Graph::InsertArc(int node1, int node2) {  
 float dis = getFloat("distance");  
 frt[node1].enOutEdge(node2,dis);  
 frt[node2].enOutEdge(node1,dis);  
  
 return true;  
}  
  
bool my\_Graph::BFSTraverse(int bgn) {  
 //sign the node  
 std::vector<int> D(nd\_num,0);  
 //store the node  
 std::queue<int> P;  
 frt[bgn].show();  
 cout<<endl;  
 D[bgn] = 1;  
 //find near node  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[bgn].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 while(!P.empty()){  
 int node1 = P.front();  
 P.pop();  
 frt[node1].show();  
 cout<<endl;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i] || frt[node1].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 }  
  
 return true;  
}  
  
my\_Graph::my\_Graph(std::string filename) {  
 std::ifstream in(filename);  
 int nd\_n;  
 in>>nd\_n;  
 nd\_num = nd\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str;  
 in>>str;  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 cout<<endl;  
 }  
 while(!in.eof())  
 {  
 eg\_num++;  
 int n\_st1,n\_st2;  
 std::string n1,n2;  
 float dis;  
 in>>n1>>n2>>dis;  
 n\_st1 = locateVex(n1);  
 n\_st2 = locateVex(n2);  
 if(n\_st1 == -1 || n\_st2 == -1)  
 continue;  
 frt[n\_st1].enOutEdge(n\_st2,dis);  
 frt[n\_st2].enOutEdge(n\_st1,dis);  
 }  
 in.close();  
}  
  
int my\_Graph::getNodeNumber() {  
 return nd\_num;  
}  
  
float my\_Graph::primSpanningTree() {  
 //vector to concern the node have past  
 std::vector<bool> D(nd\_num,false);  
 D[0] = true;  
 float sum = 0;  
 while(true)  
 {  
 float min = MAX;  
 int node1 = 0,node2 = 0;  
 bool flag = false;  
 //find the nearest node  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 {  
 for(int k = 0;k < nd\_num;k++)  
 {  
 if(D[k])  
 continue;  
 if(frt[i].out\_e[k] < min)  
 {  
 flag = true;  
 node1 = i;  
 node2 = k;  
 min = frt[i].out\_e[k];  
 }  
 }  
 }  
 }  
 //do not have , break  
 if(!flag)  
 {  
 break;  
 }  
 //output edge  
 if(D[node1])  
 {  
#ifndef YTD  
 cout<<"enter ";  
 frt[node2].show();  
#endif  
 D[node2] = true;  
 cout<<"edge node1 : "<<frt[node1].getData()<<" node2 : "<<frt[node2].getData();  
 cout<<" distance : "<<min<<endl;  
 } else{  
#ifndef YTD  
 cout<<"enter node : ";  
 frt[node1].show();  
#endif  
 D[node1] = true;  
 cout<<endl<<"edge node1 : "<<frt[node1].getData()<<" node2 : "<<frt[node2].getData();  
 cout<<" distance : "<<min<<endl;  
 }  
 sum += min;  
 }  
 cout<<"prim end."<<endl;  
 return sum;  
}  
  
float my\_Graph::kruskalSpinningTree() {  
 //vector to concern the node have past ?  
 std::vector<bool> D(nd\_num,false);  
 float sum = 0;  
 std::vector<edge> P;  
 //sort edge  
 for(int i = 0;i < nd\_num;i++)  
 {  
 for(int k = i + 1;k < nd\_num;k++)  
 {  
 if(frt[i].out\_e[k] != MAX)  
 {  
 edge nw\_eg(i,k,frt[i].out\_e[k]);  
 P.push\_back(nw\_eg);  
 }  
 }  
 }  
 sortVector(P);  
 while(!P.empty())  
 {  
 edge pause = P.back();  
 P.pop\_back();  
 //find the shortest edge  
 if(!D[pause.node1] || !D[pause.node2])  
 {  
 sum += pause.distance;  
 D[pause.node1] = true;  
 D[pause.node2] = true;  
 cout<<"edge node1 : "<<frt[pause.node1].getData()<<" node2 : "<<frt[pause.node2].getData();  
 cout<<" distance : "<<pause.distance<<endl;  
 } else{  
 continue;  
 }  
 }  
 cout<<"kru end."<<endl;  
 return sum;  
}  
  
my\_Graph::my\_Graph(int nd\_n, int eg\_n, float dis) {  
 nd\_num = nd\_n;  
 eg\_num = eg\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str;  
 str += 'a' + i;  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 }  
 for(int i = 0;i < eg\_n;i++)  
 {  
 int n\_st1 = getInt("node1 site");  
 int n\_st2 = getInt("node2 site");;  
 frt[n\_st1 - 1].enOutEdge(n\_st2 - 1,dis);  
 frt[n\_st2 - 1].enOutEdge(n\_st1 - 1,dis);  
 }  
}  
  
bool my\_Graph::dijkstra(std::vector<float> & T,int bgn) {  
 std::vector<bool> D(nd\_num,false);  
 for(int i = 0;i < nd\_num;i++)  
 {  
 T.push\_back(frt[bgn].out\_e[i]);  
 }  
 int p = bgn;  
 D[bgn] = true;  
 while(true)  
 {  
 int min = -1;  
 //find the nearest node do not be traverse  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 continue;  
 if(min == -1 || T[i] < min)  
 {  
 min = T[i];  
 p = i;  
 }  
 }  
 if(min == -1)  
 break;  
 //refresh the distance  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 continue;  
 if(T[i] > T[p] + frt[p].out\_e[i])  
 {  
 T[i] = T[p] + frt[p].out\_e[i];  
 }  
 }  
 D[p] = true;  
 }  
 return true;  
}  
  
my\_Graph::~my\_Graph() = default;  
  
bool GNode::show() {  
 cout<<"node : "<<data<<" site : "<<site;  
  
 return true;  
}  
  
std::string GNode::getData() {  
 return data;  
}  
  
int GNode::firstAdjVex() {  
 int i = 0;  
 for(;i < out\_e.size();i++)  
 {  
 if(out\_e[i] != MAX)  
 {  
 break;  
 }  
 }  
 if(out\_e[i] != MAX)  
 return i;  
 return -1;  
}  
  
int GNode::nextAdjVex(int node1) {  
 int i = node1 + 1;  
 for(;i < out\_e.size();i++)  
 {  
 if(out\_e[i] != MAX)  
 break;  
 }  
 if(out\_e[i] != MAX)  
 return i;  
 return -1;  
}  
  
bool GNode::deleteVex(int node1) {  
 out\_e.erase(out\_e.begin() + node1);  
 return true;  
}  
  
bool gNode\_plus::enBus(int n) {  
 int i = 0;  
 for(;i < bus.size();i++)  
 {  
 if(bus[i] == n)  
 return false;  
 if(bus[i] > n)  
 break;  
 }  
 bus.insert(bus.begin() + i,n);  
  
 return true;  
}  
  
bool gNode\_plus::deBus(int n) {  
 int i = 0;  
 for(;i < bus.size();i++)  
 {  
 if(bus[i] == n)  
 break;  
 if(bus[i] > n)  
 return false;  
 }  
 if(bus[i] == n)  
 bus.erase(bus.begin() + i);  
  
 return true;  
}  
  
my\_Graph\_Bus::my\_Graph\_Bus(std::string filename)  
{  
 std::ifstream in(filename);  
 nd\_num = 0;  
 eg\_num = 0;  
 while(!in.eof())  
 {  
 int bsln;  
 in>>bsln;  
 int str\_last = -1;  
 while(true)  
 {  
 std::string str;  
 in>>str;  
 if(str == "z")  
 break;  
 int st = locateVex(str);  
 if(st == -1)  
 {  
 frt.push\_back(gNode\_plus (nd\_num,str,5652));  
 st = nd\_num;  
 nd\_num++;  
 }  
 frt[st].enBus(bsln);  
 if(str\_last != -1)  
 {  
 frt[st].enOutEdge(str\_last,1);  
 frt[str\_last].enOutEdge(st,1);  
 eg\_num++;  
 }  
 str\_last = st;  
 }  
 }  
  
 in.close();  
}  
  
int my\_Graph\_Bus::locateVex(std::string str) {  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[i].getData() == str)  
 return i;  
 }  
 return -1;  
}  
  
bool my\_Graph\_Bus::BFSTraverse(int bgn) {  
 std::vector<int> D(nd\_num,0);  
 std::queue<int> P;  
 frt[bgn].show();  
 cout<<endl;  
 D[bgn] = 1;  
 //find the near node  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[bgn].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 while(!P.empty()){  
 int node1 = P.front();  
 P.pop();  
 frt[node1].show();  
 cout<<endl;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i] || frt[node1].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 }  
  
 return true;  
}  
  
bool my\_Graph\_Bus::bfsGo(std::string node1, std::string node2) {  
 int nst1 = locateVex(node1);  
 int nst2 = locateVex(node2);  
  
 std::vector<bool> D(nd\_num, false);  
 std::queue<bfsNode> P;  
 std::stack<bfsNode> s;  
 D[nst1] = true;  
 P.push(bfsNode(nst1));  
  
 while(!P.empty()){  
 bfsNode nd = P.front();  
 P.pop();  
 s.push(nd);  
 if(nd.nw == nst2)  
 break;  
 //find near nodes  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i] || frt[nd.nw].out\_e[i] == MAX)  
 continue;  
 P.push(bfsNode(i,nd.nw));  
 D[i] = true;  
 }  
 }  
 if(P.empty())  
 {  
 cout<<"Don't have way."<<endl;  
 return false;  
 }  
 //find the node's pa from pause stack  
 int pa = s.top().pa;  
 bfsNode nd = s.top();  
 int dr = someLine(nd.nw,nd.pa);  
 int last\_node = nd.nw;  
 int count = 1;  
 cout<<"最快 : "<<endl;  
 cout<<frt[nd.nw].getData()<<" -> line "<<dr<<" : ";  
 s.pop();  
 while(!s.empty())  
 {  
 nd = s.top();  
 if(nd.nw == pa)  
 {  
 count++;  
 //when the line is changed  
 if(dr != someLine(last\_node,pa))  
 {  
 dr = someLine(last\_node,pa);  
 cout<<frt[last\_node].getData()<<" "<<count<<"zhan"<<endl<<frt[last\_node].getData()<<" -> ";  
 cout<<"line "<<dr<<" : ";  
 count = 1;  
 }  
 last\_node = pa;  
 pa = nd.pa;  
 }  
 s.pop();  
 if(s.empty())  
 {  
 cout<<frt[nd.nw].getData();  
 cout<<endl;  
 }  
 }  
 return true;  
}  
  
bool dirPass(std::vector<int> dr,int dir)  
{  
 for(int i = 0;i < dr.size();i++)  
 {  
 if(dr[i] == dir)  
 return true;  
 }  
 return false;  
}  
  
class dfsNode{  
public:  
 int nw;  
 int bus;  
 dfsNode(int n1 = -1,int n2 = -1)  
 {  
 nw = n1;  
 bus = n2;  
 }  
 ~dfsNode();  
};  
  
dfsNode::~dfsNode() {  
}  
  
bool my\_Graph\_Bus::dfsGo(std::string node1, std::string node2) {  
 std::stack<dfsNode> s;  
 std::vector<bool> D(nd\_num,false);  
 std::vector<int> dr;//direction have go  
 int site1 = locateVex(node1);  
 D[site1] = true;  
 int site2 = locateVex(node2);  
 s.push(dfsNode(site1,frt[site1].bus[0]));  
 bool flag = true;  
 bool end = false;  
 dr.push\_back(frt[site1].bus[0]);  
 dfsNode p;  
 while(!s.empty()) {  
 //do not have the node  
 if(!flag){  
 end = false;  
 p = s.top();  
 //change direction  
 for(int i = 0;i < frt[p.nw].out\_e.size();i++) {  
 if (frt[p.nw].out\_e[i] != 1 || D[i])  
 continue;  
 int dir = someLine(p.nw, i,p.bus);  
 if (dirPass(dr, dir)) {  
 continue;  
 } else {  
 dr.push\_back(dir);  
 s.push(dfsNode(i,dir));  
 end = true;  
 flag = true;  
 break;  
 }  
 }  
 if(!end)  
 {  
 D[s.top().nw] = false;  
 s.pop();  
 }  
 continue;  
 }  
 flag = false;  
 p = s.top();  
 //find the next node along line dir  
 int dir = p.bus;  
 int k;  
 int i;  
 for(i = 0;i < frt[p.nw].out\_e.size();i++)  
 {  
 if(frt[p.nw].out\_e[i] > 2 || D[i])  
 continue;  
 for(k = 0;k < frt[i].bus.size();k++)  
 {  
 if(frt[i].bus[k] == dir)  
 {  
 s.push(dfsNode(i,dir));  
 D[i] = true;  
 flag = true;  
 /\*cout<<frt[i].show()<<endl;\*/  
 break;  
 }  
 }  
 if(flag)  
 break;  
 }  
 if(i == site2)  
 break;  
 }  
 if(s.empty())  
 {  
 cout<<"don't have way"<<endl;  
 return false;  
 }  
 else{  
 cout<<"最少换程 : "<<endl;  
 int last\_node = s.top().nw;  
 int count = 1;  
 cout<<frt[s.top().nw].getData();  
 int dr = s.top().bus;  
 cout<<" -> line "<<dr<<" : ";  
 while(!s.empty())  
 {  
 count ++;  
 //if change bus  
 if(dr != s.top().bus)  
 {  
 dr = s.top().bus;  
 cout<<frt[s.top().nw].getData()<<" "<<count<<"zhan"<<endl;  
 cout<<frt[s.top().nw].getData()<<" -> line "<<dr<<" : ";  
 count = 1;  
 }  
 if(s.size() == 1)  
 cout<<frt[s.top().nw].getData()<<endl;  
 s.pop();  
 }  
 }  
 return true;  
}  
  
int my\_Graph\_Bus::someLine(int node1, int node2, int n) {  
 int i = 0;  
 int k = 0;  
 for(;i < frt[node1].bus.size();)  
 {  
 if(frt[node1].bus[i] == frt[node2].bus[k])  
 {  
 if(frt[node1].bus[i] < n)  
 {  
 i++;  
 k++;  
 continue;  
 }  
 return frt[node1].bus[i];  
 } else if (frt[node2].bus[k] < frt[node1].bus[i])  
 {  
 k++;  
 } else{  
 i++;  
 }  
 }  
  
 return -1;  
}

task7.cpp

//  
// Created by 97927 on 2019/12/30.  
//  
#include <iostream>  
#include <cstdlib>  
#include <cstdio>  
#include <string>  
#include "my\_Graph.h"  
#include "function.h"  
using namespace std;  
  
int main()  
{  
 string str = "1234.txt";//bus line file  
 my\_Graph\_Bus g(str);  
 string node1 = getString("node1");  
 string node2 = getString("node2");  
 cout<<endl;  
 while(node1 != "p" && node2 != "p")  
 {  
 //the fast line  
 g.bfsGo(node1,node2);  
 cout<<endl;  
 //the easiest line  
 g.dfsGo(node1,node2);  
 node1 = getString("node1");  
 node2 = getString("node2");  
 }  
  
  
 return 0;  
}

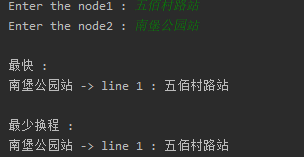
###### 测试样例与运行结果：

测试样例：

南堡公园站

五佰村路站

运行结果：

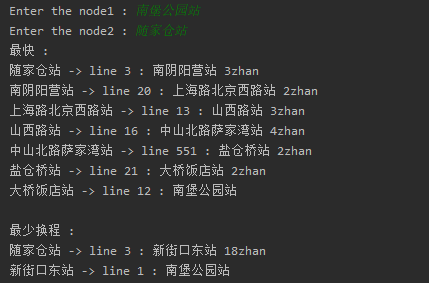


测试样例：

随家仓站

南堡公园站

运行结果：



### 必做题第八题

###### 题目：排序算法比较 （必做）（排序）

[问题描述]

利用随机函数产生10个样本，每个样本有50000个随机整数（并使第一个样本是正序，第二个样本是逆序），利用直接插入排序、希尔排序，冒泡排序、快速排序、选择排序、堆排序，归并排序、基数排序8种排序方法进行排序（结果为由小到大的顺序），并统计每一种排序算法对不同样本所耗费的时间。

[基本要求]

（1） 原始数据存在文件中，用相同样本对不同算法进行测试；

（2） 屏幕显示每种排序算法对不同样本所花的时间；

###### 数据结构：线性表、栈

###### 算法思想：

直接插入排序：  
 起始点i，终点n  
 循环数k从i+1开始，比较两个数：  
 1.num[k] < num[k - 1],k--  
 2.num[k] >= num[k - 1],在这里插入num，结束该次循环  
 3.k < i，在最前面插入num，结束该次循环  
 直到k = n + 1，结束  
希尔排序：  
 起始点i，终点n  
 建立递减步伐数组p，p的最后一个值为1  
 循环数k从i开始，以步伐p，比较两个数：  
 1.num[k] < num[k + p],k++  
 2.num[k] >= num[k + p],交换两个数，循环数j从k开始，以步伐 p，向前比较：  
 2.1.num[j] > num[j - p],结束该次比较，k++  
 2.2.num[j] <= num[j - p],交换两个数，j-=p  
 2.3.j < i，结束该次比较，k++  
 3.k > n - p,结束该次循环  
 直到步伐数组p的所有步伐全部比较完后，结束  
冒泡排序：  
 起始点i，终点n  
 循环数k每次从i开始，比较两个数：  
 1.num[k] < num[k + 1],k++  
 2.num[k] >= num[k + 1],交换两个数  
 3.k > n - 循环次数，结束该次循环  
 循环n次，结束  
快速排序：  
 起始点i，终点n  
 选择第一个数为标记p，前置循环数k从i开始向后比较：  
 1.num[k] < p,k++  
 2.结束该次查找  
 后置循环数j从n开始向前比较：  
 1.num[j] > p,j--  
 2.结束该次查找  
 交换两个数num[k],num[j]，继续上述查找过程  
 直到k >= j，结束该次循环  
 以i，k为起始点与终点继续上述过程，直到不可分割  
 以k，n为起始点与终点继续上述过程，直到不可分割  
选择排序：  
 起始点i，终点n  
 循环数k从i开始，记录min = num[k],向后比较：  
 1.min > num[k],min = num[k],记录该点，k++  
 2.k++  
 交换记录点的数与该次起点的数，结束该次循环  
 直到k = n，结束  
堆排序：  
 建堆：  
 起始点k，终点n1  
 比较num[k],num[k\*2],num[k\*2+1],若num[k\*2]ornum[k\*2+1] > num[k]，交换两个数，从发生改变的位置开始继续向下比较，直到 到达n1  
 起始点i，终点n  
 从i / 2开始，以n为终点，建堆  
 交换第i个数与第n个数，n--  
 以i为起点，n为终点建堆  
 重复上述过程，直到n = i，结束  
归并排序：  
 总个数n，以每个数为一个序列，分为n个序列  
 比较相邻两个序列，将两个序列有序合并，将序列分为[n /2]个  
 比较相邻两个序列，将两个序列有序合并，将序列分为[n / 4]个  
 ...  
 直到只剩余一个序列，结束  
基数排序：  
 建立十个栈  
 从个位开始，按该位数存入栈中  
 按该位数从小到大从栈中取出  
 重复上述过程，直到最高位也执行过

###### 算法时间复杂度：

直接插入排序、冒泡排序、选择排序：n^2

快速排序、堆排序、归并排序、基数排序：nlogn

###### 具体实现：

function.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_FUNCTION\_H  
#define DESIGN\_FUNCTION\_H  
  
#include <string>  
#include <vector>  
#include <queue>  
#include <stack>  
#define LENG 100  
  
std::string getString(const std::string& out = "name");  
int getInt(const std::string& out = "name");  
float getFloat(const std::string& out = "name");  
bool compareOperate(char c1,char c2);//true c2 > c1  
double computeTwoNumber(double n1,double n2,char c1);  
void mergeSort(std::vector<int> &sq);  
void selectionSort(std::vector<int> &sq);  
void bubbleSort(std::vector<int> &sq);  
bool fastSort(std::vector<int> &sq);  
bool heapSort(std::vector<int> &sq);  
bool radixSort(std::vector<int> &sq,int n);  
bool straightSort(std::vector<int> &sq);  
bool shellSort(std::vector<int> &sq);  
#endif //DESIGN\_FUNCTION\_H

function.cpp

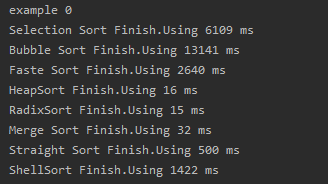
//  
// Created by 97927 on 2019/12/13.  
//  
  
#include "function.h"  
#include <iostream>  
#include <vector>  
#include "my\_Queue.h"  
#include <ctime>  
#include <math.h>  
using namespace std;  
  
//return a string get fro stdin  
std::string getString(const std::string& out) {  
 std::string str;  
 cout<<"Enter the "<<out<<" : ";  
 getline(cin,str);  
 return str;  
}  
//return a int number and clear stdin  
int getInt(const string &out) {  
 int num;  
 cout<<"Enter the "<<out<<" : ";  
 while(true)  
 {  
 cin>>num;  
 if(cin.fail())  
 {  
 cout<<"Error!Please enter the integer : ";  
 cin.clear();  
 continue;  
 }  
 break;  
 }  
 cin.ignore();  
  
 return num;  
}  
//return a float number and clear stdin  
float getFloat(const std::string& out)  
{  
 float num;  
 cout<<"Enter the "<<out<<" : ";  
 while(true)  
 {  
 cin>>num;  
 if(cin.fail())  
 {  
 cout<<"Error!Please enter the integer : ";  
 cin.clear();  
 continue;  
 }  
 break;  
 }  
  
 return num;  
}  
//compare two operators  
bool compareOperate(char c1,char c2)  
{  
 switch(c1)  
 {  
 case '-' :  
 case '+' :  
 {  
 return !(c2 == '-' || c2 == '+' || c2 == '(');  
 }  
 case '\*' :  
 case '/' :  
 {  
 return c2 == ')';  
 }  
 case '(' :  
 {  
 return c2 != ')';  
 }  
 case ')' :  
 {  
 return false;  
 }  
 }  
}  
//compute  
double computeTwoNumber(double n1, double n2,char c1)  
{  
 switch(c1)  
 {  
 case '+' :  
 return n1 + n2;  
 case '-' :  
 return n1 - n2;  
 case '\*' :  
 return n1 \* n2;  
 case '/' :  
 if(n2 == 0)//if divide bt 0  
 return -65535;  
 return n1 / n2;  
 }  
 return -1;  
}  
//sort two orderly array , sq is the total array , a1 / a2 is the first array's begin and end  
void collectionSort(vector<int> &sq,int a1,int a2,int b1,int b2)  
{  
 while(1)  
 {  
 if(a1 > a2 || b1 > b2)  
 break;  
 if(sq[a1] >= sq[b1])  
 {  
 a1++;  
 }  
 else{//sq[a1] < sq[b1]  
 int pause = sq[b1];  
 sq.erase(sq.begin() + b1);  
 sq.insert(sq.begin() + a1,pause);  
 a1++;  
 b1++;  
 a2++;  
 }  
 }  
}  
  
void mergeSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int length = sq.size();  
 //store the begin site and end site  
 my\_Queue q;  
 for(int i = 0;i < length;i++)  
 {  
 q.enQueue(i);  
 q.enQueue(i);  
 }  
 while(q.getlength() > 2)  
 {  
 int a1,a2,b1,b2;  
 q.deQueue(&a1);  
 q.deQueue(&a2);  
 q.getHead(&b1);  
 //decide weather this two arrays is order  
 if(b1 - 1 == a2)  
 {  
 q.deQueue(&b1);  
 q.deQueue(&b2);  
 collectionSort(sq,a1,a2,b1,b2);  
 //pop in the array1 begin and array2 end  
 q.enQueue(a1);  
 q.enQueue(b2);  
 } else{  
 q.enQueue(a1);  
 q.enQueue(a2);  
 }  
 }  
 last = clock();  
 printf("Merge Sort Finish.");  
 cout<<"Using "<<last - current<<" ms"<<endl;  
}  
  
void selectionSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int times = sq.size();  
 int max;  
 int max\_site;  
 for(int i = 0;i < times;i++)  
 {  
 max = sq[i];  
 for(int j = i + 1;j < times;j++)  
 {  
 if(sq[j] < max)  
 {  
 max = sq[j];  
 max\_site = j;  
 }  
 }  
 sq[max\_site] = sq[i];  
 sq[i] = max;  
 }  
 last = clock();  
 printf("Selection Sort Finish.");  
 cout<<"Using "<<last - current<<" ms"<<endl;  
}  
  
void bubbleSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int times = sq.size();  
 for(int i = 0;i < times;i++)  
 {  
 for(int j = times;j > i;j--)  
 {  
 if(sq[j] < sq[j - 1])  
 {  
 int pause = sq[j];  
 sq[j] = sq[j - 1];  
 sq[j - 1] = pause;  
 }  
 }  
 }  
 last = clock();  
 printf("Bubble Sort Finish.");  
 cout<<"Using "<<last - current<<" ms"<<endl;  
}  
  
bool straightSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int times = sq.size();  
 for(int i = 1;i < times;i++)  
 {  
 for(int k = 0;k < i;k++)  
 {  
 if(sq[k] > sq[i])  
 {  
 int nm = sq[i];  
 sq.erase(sq.begin() + i);  
 sq.insert(sq.begin() + k,nm);  
 break;  
 }  
 }  
 }  
 last = clock();  
 printf("Straight Sort Finish.");  
 cout<<"Using "<<last - current<<" ms"<<endl;  
  
 return true;  
}  
//straight sort used by shell sort  
bool straight(vector<int> &sq)  
{  
 int times = sq.size();  
 for(int i = 1;i < times;i++)  
 {  
 for(int k = 0;k < i;k++)  
 {  
 if(sq[k] > sq[i])  
 {  
 int nm = sq[i];  
 sq.erase(sq.begin() + i);  
 sq.insert(sq.begin() + k,nm);  
 break;  
 }  
 }  
 }  
  
 return true;  
}  
//used bt fast sort  
int changeSort(vector<int> &sq,int bin,int end)  
{  
 //if just have two elem  
 if((end - bin) <= 1)  
 {  
 if(bin >= end)  
 return -1;  
 if(sq[bin] < sq[end])  
 {  
 int pause = sq[bin];  
 sq[bin] = sq[end];  
 sq[end] = pause;  
 }  
 return -1;  
 }  
 //store the number  
 int pause = sq[bin];  
 sq.erase(sq.begin() + bin);  
 int i = bin;  
 int k = end - 1;  
 while(1)  
 {  
 //end condition  
 if(i >= k)  
 {  
 if(sq[i] <= pause)  
 {  
 k = i;  
 } else{  
 k++;  
 i = k;  
 }  
 sq.insert(sq.begin() + k,pause);  
 break;  
 }  
 //find the number smaller than pause from the end  
 if(sq[k] > pause)  
 {  
 k--;  
 continue;  
 }  
 //find the number biger than pause from the begin  
 if(sq[i] > pause)  
 {  
 //swap  
 int data = sq[k];  
 sq[k] = sq[i];  
 sq[i] = data;  
 k--;  
 i++;  
 } else{  
 i++;  
 continue;  
 }  
 }  
 //return next position should be sort  
 return k;  
}  
  
bool fastSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 queue<int> q;//store the position which should be sort  
 q.push(0);  
 q.push(sq.size() - 1);  
 while(!q.empty())  
 {  
 int bin,end;  
 bin = q.front();  
 q.pop();  
 end = q.front();  
 q.pop();  
 int middle = changeSort(sq,bin,end);  
 if(middle != -1)  
 {  
 //store the begin to middle , middle to end  
 q.push(bin);  
 q.push(middle - 1);  
 q.push(middle + 1);  
 q.push(end);  
 }  
 }  
 last = clock();  
 cout<<"Faste Sort Finish.Using "<<last - current<<" ms"<<endl;  
  
 return true;  
}  
//swap two number  
void swap(int \*num1,int \* num2)  
{  
 int pause = \*num1;  
 \*num1 = \*num2;  
 \*num2 = pause;  
}  
//compare the root and its two leaves  
//return the position should be solved , -1 means don't need to compare  
int maxHeap(vector<int> &sq,int bin,int end)  
{  
 int end1 = 2 \* bin + 1;  
 if(end1 >= end)  
 {  
 return -1;  
 }  
 if(end1 + 1 < end && sq[end1] < sq[end1 + 1])  
 {  
 end1++;  
 }  
 if(sq[end1] > sq[bin])  
 {  
 swap(&sq[end1],&sq[bin]);  
 return end1;  
 }  
 return -1;  
}  
  
bool heapSort(vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int i = (int)(sq.size() / 2);  
 int end = sq.size();  
 //create the max heap  
 while(i >= 0)  
 {  
 int k;  
 k = maxHeap(sq, i,end);  
 while(k > 0)  
 {  
 k = maxHeap(sq,k,end);  
 }  
 i--;  
 }  
 while(1)  
 {  
 /\*cout<<endl;  
 for(int i = 0;i < sq.size();i++)  
 {  
 cout<<sq[i]<<" ";  
 if((i % 10) == 9)  
 cout<<endl;  
 }\*/  
 int pause = sq[0];  
 //swap the top and the bottom  
 swap(&sq[0],&sq[end - 1]);  
 if(end == 1)  
 break;  
 end--;  
 //create the max heap again  
 int k = maxHeap(sq,0,end);  
 while(k > 0)  
 {  
 k = maxHeap(sq,k,end);  
 }  
 }  
 last = clock();  
 cout<<"HeapSort Finish.Using "<<last - current<<" ms"<<endl;  
  
 return true;  
}  
//inputing n stands for the bit of the bigest number  
bool radixSort(std::vector<int> &sq,int n)  
{  
 clock\_t current,last;  
 current = clock();  
 vector<int> pauses[10];  
 int k = 0;  
  
 while(k < n)  
 {  
 //push back begin from singles  
 while(!sq.empty())  
 {  
 int nm = sq.back();  
 int st = nm;  
 for(int j = 0;j < k;j++)  
 {  
 st /= 10;  
 }  
 sq.pop\_back();  
 pauses[st % 10].push\_back(nm);  
 }  
 k++;  
 //pop and sort again  
 for(int i = 0;i < 10;i++)  
 {  
 while(!pauses[i].empty())  
 {  
 sq.push\_back(pauses[i].back());  
 pauses[i].pop\_back();  
 }  
 }  
 }  
 last = clock();  
 cout<<"RadixSort Finish.Using "<<last - current<<" ms"<<endl;  
  
 return true;  
}  
  
bool shellInsert(std::vector<int> &sq,int n, int length)  
{  
 //straight sort as the path of n  
 for(int j = 0;j < length - n;j++)  
 {  
 if(sq[j] > sq[j + n])  
 {  
 swap(&sq[j],&sq[j + n]);  
 for(int k = j - n;k > -1;k -= n)  
 {  
 if(sq[k] > sq[k + n])  
 {  
 swap(&sq[k],&sq[k + n]);  
 continue;  
 }  
 break;  
 }  
 }  
 }  
 return true;  
}  
  
bool shellSort(std::vector<int> &sq)  
{  
 clock\_t current,last;  
 current = clock();  
 int length = sq.size();  
 int t = (int)log(length);  
 int dr = 2^(t + 1) - 1;//path  
 //end when the path smaller than 1  
 for(int i = 0;dr > 0;i++,dr = 2^(t - i + 1) - 1)  
 {  
 shellInsert(sq,dr,length);  
 }  
 last = clock();  
 cout<<"ShellSort Finish.Using "<<last - current<<" ms"<<endl;  
  
 return 0;  
}

task8.cpp

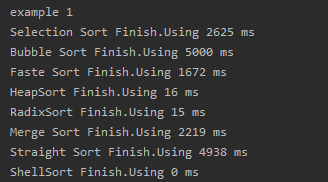
//  
// Created by 97927 on 2019/12/26.  
//  
#include <cstdlib>  
#include <cstdio>  
#include <iostream>  
#include <vector>  
#include <string>  
#include "function.h"  
#include <ctime>  
#include <fstream>  
using namespace std;  
  
bool compareSort(int n)  
{  
 std::string ifilename = "task8/source" ;  
 string sort[8] = {"mergesort","selectionsort","bubblesort",  
 "fastsort","heapsort","radixsort"};  
 ifilename += ('0' + n) ;  
 ifilename += ".txt";  
 ifstream in(ifilename);  
 int k = 0;  
 vector<int> test,p;  
 while(!in.eof())  
 {  
 int nm;  
 in>>nm;  
 test.push\_back(nm);  
 p.push\_back(nm);  
 }  
 in.close();  
 cout<<"example "<<n<<endl;  
 selectionSort(test);  
 test.assign(p.begin(),p.end());  
 bubbleSort(test);  
 test.assign(p.begin(),p.end());  
 fastSort(test);  
 test.assign(p.begin(),p.end());  
 heapSort(test);  
 test.assign(p.begin(),p.end());  
 radixSort(test,5);  
 test.assign(p.begin(),p.end());  
 mergeSort(test);  
 test.assign(p.begin(),p.end());  
 straightSort(test);  
 test.assign(p.begin(),p.end());  
 shellSort(test);  
  
 return 0;  
}  
int main()  
{  
 for(int i = 0;i < 10;i++)  
 {  
 compareSort(i);  
 }  
  
 return 0;  
}

###### 测试样例与运行结果：

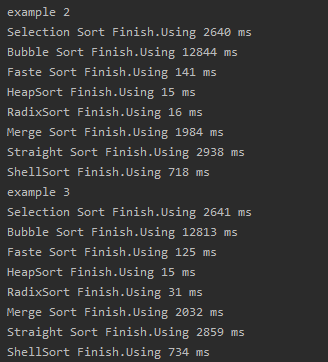
测试样例0（有序），运行结果：

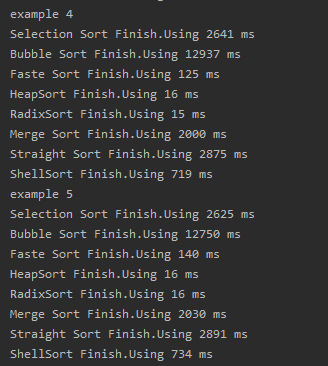


测试样例1（逆序），运行结果：

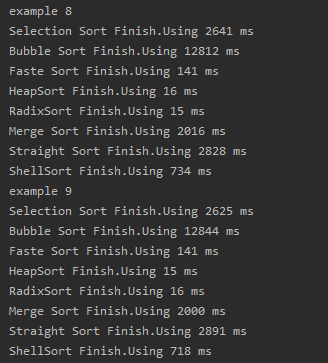


测试样例2-9（无序），运行结果：









### 选做题第11题

###### 题目：社交网络图中结点的“重要性”计算（选做）（图）

[问题描述]

在社交网络中，个人或单位（结点）之间通过某些关系（边）联系起来。他们受到这些关系的影响，这种影响可以理解为网络中相互连接的结点之间蔓延的一种相互作用，可以增强也可以减弱。而结点根据其所处的位置不同，在网络中体现的重要性也不尽相同。

“紧密度中心性”是用来衡量一个结点达到其他结点的“快慢”的指标，即一个有较高中心性的结点比有较低中心性的结点能够更快地（平均意义下）到达网络中的其他结点，因而在该网络的传播过程中有更重要的价值。在有N个结点的网络中，结点vi的“紧密度中心性” 数学上定义为 到其余所有结点(j!=i)的最短距离的平均值的倒数：

对于非连通图，所有结点的紧密度中心性都是0。

本实验给定一个无权的无向图以及其中的一组结点，要求计算这组结点中每个结点的紧密度中心性。

[基本要求]

（1）输入说明：输入的第一行给出两个正整数N (N<=1000)和M，其中N是图中结点个数，结点编号从1到N； M（M<=10000）是边的数目。随后的M行中，每行给出一条边的信息，即该边连接的两个结点编号，中间用空格分隔。最后一行给出需要计算紧密度中心性的这组结点的个数K（K<=100），以及K个结点的编号，用空格分隔。

（2）输出说明：按照“Cc(i)=x.xx”的格式输出K个给定结点的紧密度中心性，每个输出占一行，结果精确到小数点后2位。

（3）测试用例：

输入 5 8

1 2

1 3

1 4

2 3

3 4

4 5

2 5

3 5

2 4 3

输出 Cc(4) = 0.80

Cc(3) = 1.00

###### 数据结构：图

###### 算法思想：

运用迪杰斯特拉算法，计算输出的起点到其他各个点的最短距离

###### 算法时间复杂度：n^2

###### 具体实现：

my\_Graph.h

//  
// Created by 97927 on 2019/12/13.  
//  
  
#ifndef DESIGN\_MY\_GRAPH\_H  
#define DESIGN\_MY\_GRAPH\_H  
  
#include <string>  
#include <vector>  
#include <fstream>  
#define MAX 65535  
  
class edge{  
public:  
 int node1,node2;  
 float distance;  
 edge(int n1,int n2,float d)//initial  
 {  
 node1 = n1;  
 node2 = n2;  
 distance = d;  
 }  
 ~edge()= default;;  
};  
class GNode{  
private:  
 int site;  
 std::string data;  
protected:  
public:  
 std::vector<float> out\_e;//out egde , undirected graph only use this  
 std::vector<float> in\_e;//in edge  
 GNode(int s, std::string d)//initial  
 {  
 site = s;  
 data = d;  
 }  
 GNode(int s, std::string d, int n)//initial  
 {  
 site = s;  
 data = d;  
 out\_e.assign(n,MAX);  
 }  
 bool setSite(int s)//change this node's site  
 {  
 site = s;  
 return true;  
 }  
 bool setData(std::string da)//change this node's data  
 {  
 data = da;  
 }  
 float distanceNode(int st)//return between this node and st  
 {  
 return out\_e[st];  
 }  
 bool enOutEdge(int st,float dis)//set distance between this node and st  
 {  
 out\_e[st] = dis;  
  
 return true;  
 }  
 int outEdgeNumber()//return out edge number  
 {  
 return out\_e.size();  
 }  
 bool show();//show this node information  
 std::string getData();//return this node data  
 int firstAdjVex();//return the nearest node  
 int nextAdjVex(int node1);//return the nearest node after node1  
 bool deleteVex(int node1);//delete edge with node1  
 ~GNode()= default;  
};  
class my\_Graph {  
private:  
 std::vector<GNode> frt;  
 int nd\_num;  
 int eg\_num;  
public:  
 my\_Graph(int nd\_n,int eg\_n);//initial  
 my\_Graph(int nd\_n,int eg\_n,float dis);//initial  
 my\_Graph(std::string filename);//initial  
 ~my\_Graph();  
 int locateVex(std::string str);//locate node  
 std::string getVex(int site);//return node's data  
 bool putVex(int site,std::string str);//change the data of the node which st == site  
 GNode \* firstAdjVex(int site);//return the nearest node of the ndoe st == site  
 GNode \* nextAdjVex(int node1,int node2);//return the nearest node after node1 of the ndoe st == site  
 float primSpanningTree();//prim  
 float kruskalSpinningTree();//kruskal  
 bool InsertVex();//add node  
 bool DeleteVex(int site);//delete node  
 bool InsertArc(int node1,int node2);//add edge  
 bool BFSTraverse(int bgn = 0);//BFS traverse the graph  
 int getNodeNumber();//return the node number  
 bool dijkstra(std::vector<float> & T,int bgn);//find short distance from bgn to other node  
};  
class gNode\_plus : public GNode{  
private:  
public:  
 std::vector<int> bus;  
 gNode\_plus(int s, std::string d):GNode(s,d){}//initial  
 gNode\_plus(int s, std::string d, int n):GNode(s, d, n){}//initial  
 ~gNode\_plus(){}  
 bool enBus(int n);//add bus line  
 bool deBus(int n);//delete bus line  
};  
class my\_Graph\_Bus {  
private:  
 std::vector<gNode\_plus> frt;  
 int nd\_num;  
 int eg\_num;  
public:  
 my\_Graph\_Bus(std::string filename);//initial  
 int locateVex(std::string str);//locate node  
 bool BFSTraverse(int bgn = 0);//BFS traverse the graph  
 bool bfsGo(std::string node1,std::string node2);//find the fast way  
 bool dfsGo(std::string node1,std::string node2);//find the easiest way  
 int someLine(int node1,int node2,int limit = 0);//return the same line between after limit  
};  
class bfsNode  
{  
public:  
 int nw;  
 int pa;  
 bfsNode(int n = -1,int p = -1){  
 nw = n;  
 pa = p;  
 }  
 bool operator=(const bfsNode &b) {  
 nw = b.nw;  
 pa = b.pa;  
 }  
};  
#endif //DESIGN\_MY\_GRAPH\_H

my\_Graph.cpp

//  
// Created by 97927 on 2019/12/13.  
//  
  
#include "my\_Graph.h"  
#include <iostream>  
#include "function.h"  
#include <queue>  
#include <fstream>  
#include <vector>  
using std::cout;  
using std::cin;  
using std::endl;  
#define YTD  
  
bool sortVector(std::vector<edge> &e)  
{  
 int length = e.size();  
 for(int k = 0;k < length;k++)  
 {  
 int minW = e[k].distance;  
 int minS = k;  
 for(int i = k + 1;i < length;i++)  
 {  
 if(e[i].distance > minW)  
 {  
 minW = e[i].distance;  
 minS = i;  
 }  
 }  
 if(minS == k)  
 continue;  
 edge pause = e[minS];  
 e.erase(e.begin() + minS);  
 e.insert(e.begin() + k,pause);  
 }  
 return true;  
}  
  
my\_Graph::my\_Graph(int nd\_n,int eg\_n) {  
 nd\_num = nd\_n;  
 eg\_num = eg\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str = getString();  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 }  
 for(int i = 0;i < eg\_n;i++)  
 {  
 int n\_st1 = getInt("node1 site");  
 int n\_st2 = getInt("node2 site");  
 float dis = getFloat("distance");  
 frt[n\_st1].enOutEdge(n\_st2,dis);  
 frt[n\_st2].enOutEdge(n\_st1,dis);  
 }  
}  
  
int my\_Graph::locateVex(std::string str) {  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[i].getData() == str)  
 return i;  
 }  
 return -1;  
}  
  
std::string my\_Graph::getVex(int site) {  
 return frt[site].getData();  
}  
  
bool my\_Graph::putVex(int site, std::string str) {  
 frt[site].setData(str);  
  
 return true;  
}  
  
GNode \*my\_Graph::firstAdjVex(int site) {  
 int st = frt[site].firstAdjVex();  
 if(st == -1)  
 return nullptr;  
 return &frt[st];  
}  
  
GNode \*my\_Graph::nextAdjVex(int node1, int node2) {  
 int st = frt[node1].nextAdjVex(node2);  
 if(st == -1)  
 return nullptr;  
 return &frt[st];  
}  
  
bool my\_Graph::InsertVex() {  
 std::string str = getString();  
 GNode n\_node(nd\_num,str,nd\_num++);  
 frt.push\_back(n\_node);  
 for(int i = 0;i < nd\_num;i++)  
 {  
 frt[i].out\_e.push\_back(MAX);  
 }  
 frt[nd\_num].show();  
  
 return true;  
}  
  
bool my\_Graph::DeleteVex(int site) {  
 char c;  
 frt[site].show();  
 cout<<"concern the command : ( y to yes)";  
 //decide the delete  
 cin>>c;  
 if(c != 'y')  
 return false;  
 frt.erase(frt.begin() + site);  
 nd\_num--;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 frt[i].deleteVex(site);  
 }  
  
 return true;  
}  
  
bool my\_Graph::InsertArc(int node1, int node2) {  
 float dis = getFloat("distance");  
 frt[node1].enOutEdge(node2,dis);  
 frt[node2].enOutEdge(node1,dis);  
  
 return true;  
}  
  
bool my\_Graph::BFSTraverse(int bgn) {  
 //sign the node  
 std::vector<int> D(nd\_num,0);  
 //store the node  
 std::queue<int> P;  
 frt[bgn].show();  
 cout<<endl;  
 D[bgn] = 1;  
 //find near node  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[bgn].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 while(!P.empty()){  
 int node1 = P.front();  
 P.pop();  
 frt[node1].show();  
 cout<<endl;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i] || frt[node1].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 }  
  
 return true;  
}  
  
my\_Graph::my\_Graph(std::string filename) {  
 std::ifstream in(filename);  
 int nd\_n;  
 in>>nd\_n;  
 nd\_num = nd\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str;  
 in>>str;  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 cout<<endl;  
 }  
 while(!in.eof())  
 {  
 eg\_num++;  
 int n\_st1,n\_st2;  
 std::string n1,n2;  
 float dis;  
 in>>n1>>n2>>dis;  
 n\_st1 = locateVex(n1);  
 n\_st2 = locateVex(n2);  
 if(n\_st1 == -1 || n\_st2 == -1)  
 continue;  
 frt[n\_st1].enOutEdge(n\_st2,dis);  
 frt[n\_st2].enOutEdge(n\_st1,dis);  
 }  
 in.close();  
}  
  
int my\_Graph::getNodeNumber() {  
 return nd\_num;  
}  
  
float my\_Graph::primSpanningTree() {  
 //vector to concern the node have past  
 std::vector<bool> D(nd\_num,false);  
 D[0] = true;  
 float sum = 0;  
 while(true)  
 {  
 float min = MAX;  
 int node1 = 0,node2 = 0;  
 bool flag = false;  
 //find the nearest node  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 {  
 for(int k = 0;k < nd\_num;k++)  
 {  
 if(D[k])  
 continue;  
 if(frt[i].out\_e[k] < min)  
 {  
 flag = true;  
 node1 = i;  
 node2 = k;  
 min = frt[i].out\_e[k];  
 }  
 }  
 }  
 }  
 //do not have , break  
 if(!flag)  
 {  
 break;  
 }  
 //output edge  
 if(D[node1])  
 {  
#ifndef YTD  
 cout<<"enter ";  
 frt[node2].show();  
#endif  
 D[node2] = true;  
 cout<<"edge node1 : "<<frt[node1].getData()<<" node2 : "<<frt[node2].getData();  
 cout<<" distance : "<<min<<endl;  
 } else{  
#ifndef YTD  
 cout<<"enter node : ";  
 frt[node1].show();  
#endif  
 D[node1] = true;  
 cout<<endl<<"edge node1 : "<<frt[node1].getData()<<" node2 : "<<frt[node2].getData();  
 cout<<" distance : "<<min<<endl;  
 }  
 sum += min;  
 }  
 cout<<"prim end."<<endl;  
 return sum;  
}  
  
float my\_Graph::kruskalSpinningTree() {  
 //vector to concern the node have past ?  
 std::vector<bool> D(nd\_num,false);  
 float sum = 0;  
 std::vector<edge> P;  
 //sort edge  
 for(int i = 0;i < nd\_num;i++)  
 {  
 for(int k = i + 1;k < nd\_num;k++)  
 {  
 if(frt[i].out\_e[k] != MAX)  
 {  
 edge nw\_eg(i,k,frt[i].out\_e[k]);  
 P.push\_back(nw\_eg);  
 }  
 }  
 }  
 sortVector(P);  
 while(!P.empty())  
 {  
 edge pause = P.back();  
 P.pop\_back();  
 //find the shortest edge  
 if(!D[pause.node1] || !D[pause.node2])  
 {  
 sum += pause.distance;  
 D[pause.node1] = true;  
 D[pause.node2] = true;  
 cout<<"edge node1 : "<<frt[pause.node1].getData()<<" node2 : "<<frt[pause.node2].getData();  
 cout<<" distance : "<<pause.distance<<endl;  
 } else{  
 continue;  
 }  
 }  
 cout<<"kru end."<<endl;  
 return sum;  
}  
  
my\_Graph::my\_Graph(int nd\_n, int eg\_n, float dis) {  
 nd\_num = nd\_n;  
 eg\_num = eg\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str;  
 str += 'a' + i;  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 }  
 for(int i = 0;i < eg\_n;i++)  
 {  
 int n\_st1 = getInt("node1 site");  
 int n\_st2 = getInt("node2 site");;  
 frt[n\_st1 - 1].enOutEdge(n\_st2 - 1,dis);  
 frt[n\_st2 - 1].enOutEdge(n\_st1 - 1,dis);  
 }  
}  
  
bool my\_Graph::dijkstra(std::vector<float> & T,int bgn) {  
 std::vector<bool> D(nd\_num,false);  
 for(int i = 0;i < nd\_num;i++)  
 {  
 T.push\_back(frt[bgn].out\_e[i]);  
 }  
 int p = bgn;  
 D[bgn] = true;  
 while(true)  
 {  
 int min = -1;  
 //find the nearest node do not be traverse  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 continue;  
 if(min == -1 || T[i] < min)  
 {  
 min = T[i];  
 p = i;  
 }  
 }  
 if(min == -1)  
 break;  
 //refresh the distance  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 continue;  
 if(T[i] > T[p] + frt[p].out\_e[i])  
 {  
 T[i] = T[p] + frt[p].out\_e[i];  
 }  
 }  
 D[p] = true;  
 }  
 return true;  
}  
  
my\_Graph::~my\_Graph() = default;  
  
bool GNode::show() {  
 cout<<"node : "<<data<<" site : "<<site;  
  
 return true;  
}  
  
std::string GNode::getData() {  
 return data;  
}  
  
int GNode::firstAdjVex() {  
 int i = 0;  
 for(;i < out\_e.size();i++)  
 {  
 if(out\_e[i] != MAX)  
 {  
 break;  
 }  
 }  
 if(out\_e[i] != MAX)  
 return i;  
 return -1;  
}  
  
int GNode::nextAdjVex(int node1) {  
 int i = node1 + 1;  
 for(;i < out\_e.size();i++)  
 {  
 if(out\_e[i] != MAX)  
 break;  
 }  
 if(out\_e[i] != MAX)  
 return i;  
 return -1;  
}  
  
bool GNode::deleteVex(int node1) {  
 out\_e.erase(out\_e.begin() + node1);  
 return true;  
}  
  
bool gNode\_plus::enBus(int n) {  
 int i = 0;  
 for(;i < bus.size();i++)  
 {  
 if(bus[i] == n)  
 return false;  
 if(bus[i] > n)  
 break;  
 }  
 bus.insert(bus.begin() + i,n);  
  
 return true;  
}  
  
bool gNode\_plus::deBus(int n) {  
 int i = 0;  
 for(;i < bus.size();i++)  
 {  
 if(bus[i] == n)  
 break;  
 if(bus[i] > n)  
 return false;  
 }  
 if(bus[i] == n)  
 bus.erase(bus.begin() + i);  
  
 return true;  
}  
  
my\_Graph\_Bus::my\_Graph\_Bus(std::string filename)  
{  
 std::ifstream in(filename);  
 nd\_num = 0;  
 eg\_num = 0;  
 while(!in.eof())  
 {  
 int bsln;  
 in>>bsln;  
 int str\_last = -1;  
 while(true)  
 {  
 std::string str;  
 in>>str;  
 if(str == "z")  
 break;  
 int st = locateVex(str);  
 if(st == -1)  
 {  
 frt.push\_back(gNode\_plus (nd\_num,str,5652));  
 st = nd\_num;  
 nd\_num++;  
 }  
 frt[st].enBus(bsln);  
 if(str\_last != -1)  
 {  
 frt[st].enOutEdge(str\_last,1);  
 frt[str\_last].enOutEdge(st,1);  
 eg\_num++;  
 }  
 str\_last = st;  
 }  
 }  
  
 in.close();  
}  
  
int my\_Graph\_Bus::locateVex(std::string str) {  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[i].getData() == str)  
 return i;  
 }  
 return -1;  
}  
  
bool my\_Graph\_Bus::BFSTraverse(int bgn) {  
 std::vector<int> D(nd\_num,0);  
 std::queue<int> P;  
 frt[bgn].show();  
 cout<<endl;  
 D[bgn] = 1;  
 //find the near node  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[bgn].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 while(!P.empty()){  
 int node1 = P.front();  
 P.pop();  
 frt[node1].show();  
 cout<<endl;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i] || frt[node1].out\_e[i] == MAX)  
 continue;  
 P.push(i);  
 D[i] = 1;  
 }  
 }  
  
 return true;  
}  
  
bool my\_Graph\_Bus::bfsGo(std::string node1, std::string node2) {  
 int nst1 = locateVex(node1);  
 int nst2 = locateVex(node2);  
  
 std::vector<bool> D(nd\_num, false);  
 std::queue<bfsNode> P;  
 std::stack<bfsNode> s;  
 D[nst1] = true;  
 P.push(bfsNode(nst1));  
  
 while(!P.empty()){  
 bfsNode nd = P.front();  
 P.pop();  
 s.push(nd);  
 if(nd.nw == nst2)  
 break;  
 //find near nodes  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i] || frt[nd.nw].out\_e[i] == MAX)  
 continue;  
 P.push(bfsNode(i,nd.nw));  
 D[i] = true;  
 }  
 }  
 if(P.empty())  
 {  
 cout<<"Don't have way."<<endl;  
 return false;  
 }  
 //find the node's pa from pause stack  
 int pa = s.top().pa;  
 bfsNode nd = s.top();  
 int dr = someLine(nd.nw,nd.pa);  
 int last\_node = nd.nw;  
 int count = 1;  
 cout<<"最快 : "<<endl;  
 cout<<frt[nd.nw].getData()<<" -> line "<<dr<<" : ";  
 s.pop();  
 while(!s.empty())  
 {  
 nd = s.top();  
 if(nd.nw == pa)  
 {  
 count++;  
 //when the line is changed  
 if(dr != someLine(last\_node,pa))  
 {  
 dr = someLine(last\_node,pa);  
 cout<<frt[last\_node].getData()<<" "<<count<<"zhan"<<endl<<frt[last\_node].getData()<<" -> ";  
 cout<<"line "<<dr<<" : ";  
 count = 1;  
 }  
 last\_node = pa;  
 pa = nd.pa;  
 }  
 s.pop();  
 if(s.empty())  
 {  
 cout<<frt[nd.nw].getData();  
 cout<<endl;  
 }  
 }  
 return true;  
}  
  
bool dirPass(std::vector<int> dr,int dir)  
{  
 for(int i = 0;i < dr.size();i++)  
 {  
 if(dr[i] == dir)  
 return true;  
 }  
 return false;  
}  
  
class dfsNode{  
public:  
 int nw;  
 int bus;  
 dfsNode(int n1 = -1,int n2 = -1)  
 {  
 nw = n1;  
 bus = n2;  
 }  
 ~dfsNode();  
};  
  
dfsNode::~dfsNode() {  
}  
  
bool my\_Graph\_Bus::dfsGo(std::string node1, std::string node2) {  
 std::stack<dfsNode> s;  
 std::vector<bool> D(nd\_num,false);  
 std::vector<int> dr;//direction have go  
 int site1 = locateVex(node1);  
 D[site1] = true;  
 int site2 = locateVex(node2);  
 s.push(dfsNode(site1,frt[site1].bus[0]));  
 bool flag = true;  
 bool end = false;  
 dr.push\_back(frt[site1].bus[0]);  
 dfsNode p;  
 while(!s.empty()) {  
 //do not have the node  
 if(!flag){  
 end = false;  
 p = s.top();  
 //change direction  
 for(int i = 0;i < frt[p.nw].out\_e.size();i++) {  
 if (frt[p.nw].out\_e[i] != 1 || D[i])  
 continue;  
 int dir = someLine(p.nw, i,p.bus);  
 if (dirPass(dr, dir)) {  
 continue;  
 } else {  
 dr.push\_back(dir);  
 s.push(dfsNode(i,dir));  
 end = true;  
 flag = true;  
 break;  
 }  
 }  
 if(!end)  
 {  
 D[s.top().nw] = false;  
 s.pop();  
 }  
 continue;  
 }  
 flag = false;  
 p = s.top();  
 //find the next node along line dir  
 int dir = p.bus;  
 int k;  
 int i;  
 for(i = 0;i < frt[p.nw].out\_e.size();i++)  
 {  
 if(frt[p.nw].out\_e[i] > 2 || D[i])  
 continue;  
 for(k = 0;k < frt[i].bus.size();k++)  
 {  
 if(frt[i].bus[k] == dir)  
 {  
 s.push(dfsNode(i,dir));  
 D[i] = true;  
 flag = true;  
 /\*cout<<frt[i].show()<<endl;\*/  
 break;  
 }  
 }  
 if(flag)  
 break;  
 }  
 if(i == site2)  
 break;  
 }  
 if(s.empty())  
 {  
 cout<<"don't have way"<<endl;  
 return false;  
 }  
 else{  
 cout<<"最少换程 : "<<endl;  
 int last\_node = s.top().nw;  
 int count = 1;  
 cout<<frt[s.top().nw].getData();  
 int dr = s.top().bus;  
 cout<<" -> line "<<dr<<" : ";  
 while(!s.empty())  
 {  
 count ++;  
 //if change bus  
 if(dr != s.top().bus)  
 {  
 dr = s.top().bus;  
 cout<<frt[s.top().nw].getData()<<" "<<count<<"zhan"<<endl;  
 cout<<frt[s.top().nw].getData()<<" -> line "<<dr<<" : ";  
 count = 1;  
 }  
 if(s.size() == 1)  
 cout<<frt[s.top().nw].getData()<<endl;  
 s.pop();  
 }  
 }  
 return true;  
}  
  
int my\_Graph\_Bus::someLine(int node1, int node2, int n) {  
 int i = 0;  
 int k = 0;  
 for(;i < frt[node1].bus.size();)  
 {  
 if(frt[node1].bus[i] == frt[node2].bus[k])  
 {  
 if(frt[node1].bus[i] < n)  
 {  
 i++;  
 k++;  
 continue;  
 }  
 return frt[node1].bus[i];  
 } else if (frt[node2].bus[k] < frt[node1].bus[i])  
 {  
 k++;  
 } else{  
 i++;  
 }  
 }  
  
 return -1;  
}

opinion11.cpp

//  
// Created by 97927 on 2019/12/27.  
//  
#include "my\_Graph.h"  
#include <iostream>  
#include <vector>  
using namespace std;  
  
int main()  
{  
 int nd\_n,ed\_n;  
 cin>>nd\_n>>ed\_n;  
 my\_Graph g(nd\_n,ed\_n,1);  
 int n;  
 cin>>n;  
 for(int i = 0;i < n;i++)  
 {  
 int bgn;  
 cin>>bgn;  
 vector<float> T;  
 //find the shortest distance  
 g.dijkstra(T,bgn - 1);  
 float sum = 0;  
 for(int k = 0;k < nd\_n;k++)  
 {  
 if(k == bgn - 1)  
 continue;  
 if(T[k] == MAX)  
 {  
 sum = 0;  
 break;  
 }  
 sum += T[k];  
 }  
 if(sum != 0)  
 {  
 sum = (float)(nd\_n - 1) / sum;  
 }  
 cout.setf(ios::showpoint);  
 cout.flags(ios::fixed);  
 cout.precision(2);  
 cout<<"Cc("<<bgn<<") = "<<sum<<endl;  
 }  
}

###### 测试样例与运行结果：

测试样例：

5 8

1 2

1 3

1 4

2 3

3 4

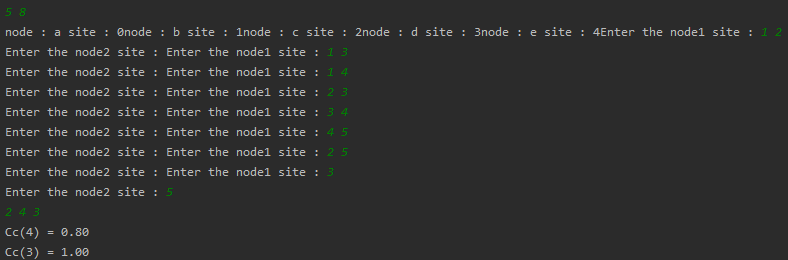
4 5

2 5

3 5

2 4 3

运行结果：



### 选做题第12题

###### 题目：魔法优惠券（选做）（排序）

[问题描述]

在火星上有个魔法商店，通过魔法优惠券。每个优惠券上印有一个整数面值K，表示若你在购买某商品使用这张优惠券，可以得到K倍该商品价值的回报。该商店还免费赠送一些有价值的商品，但如果你在领取免费赠品的时候使用面值为正的优惠券，则必须倒贴给商品K倍该商品价值的金额……但是不要紧，还有面值为负的优惠券可以用。

例如，给定一组优惠券，面值分别为1、2、4、-1；对应一组商品，价值为火星币7、6、-2、-3，其中负的价值表示该商品是免费赠品。我们可以将优惠券3（面值4）用在商品1（价值7）上，得到火星币28的回报。优惠券4（面值-1）用在商品4（价值-3）上，得到火星币3的回报。但是，如果一不小心把优惠券3（面值4）用到商品4（价值-3）上，你必须倒贴给商店火星币12个。同样，把优惠券4（面值-1）用到商品1（价值7）上，你必须倒贴给商店火星币7个。

规定每张优惠券和每件商品都只能最多被使用一次，求你可以得到的最大回报。

[基本要求]

（1）输入说明：输入有两行。第一行首先给出优惠券的个数N，随后给出N个优惠券的整数面值。 第二行首先给出商品的个数M，随后给出M个商品的整数价值。N和M在[1,106]之间，所有的数据大小不超过230，数字间以空格分隔。

（2）输出说明：输出可以得到的最大回报。

（3）测试用例：

输入 4 1 2 4 -1

4 7 6 -2 -3

输出 43

输入：4 3 2 6 1

3 2 6 3

输出：49

输入： 7 3 36 -1 73 2 3 6

6 -1 -1 -1 -1 -1 -1

输出： 1

###### 数据结构：线性表

###### 算法思想：

对输入的券的数值进行排序  
对输入的商品的数值进行排序  
从大到小选择券：  
 1.券的值大于0，选择值最大的商品  
 2.券的值小于0，选择值最小的商品  
如果两者相乘大于0，则加到总收益上，否则放弃该券  
直到券或商品全部消耗，结束

###### 算法时间复杂度：n^2

###### 具体实现：

opinion12.cpp

//  
// Created by 97927 on 2019/12/27.  
//  
#include "my\_Graph.h"  
#include <iostream>  
#include <vector>  
#include "function.h"  
using namespace std;  
  
int main()  
{  
 int n = getInt("number");  
 vector<int> object;  
 vector<int> quan;  
 for(int i = 0;i < n;i++)  
 {  
 int q;  
 cin>>q;  
 quan.push\_back(q);  
 }  
 n = getInt("number");  
 for(int i = 0;i < n;i++)  
 {  
 int q;  
 cin>>q;  
 object.push\_back(q);  
 }  
 heapSort(quan);  
 heapSort(object);  
 int sum = 0;  
 while(true)  
 {  
 if(object.empty() || quan.empty())  
 break;  
 int nm = quan.back();  
 quan.pop\_back();  
 int ob;  
 if(nm > 0)  
 {  
 ob = object.back();  
 if(ob >= 0)  
 {  
 object.pop\_back();  
 }  
 }  
 else{  
 ob = object.front();  
 if(ob <= 0)  
 {  
 object.erase(object.begin());  
 }  
 }  
 if(ob \* nm > 0)  
 {  
 sum += ob\*nm;  
 }  
 }  
 cout<<sum;  
  
 return 0;  
}

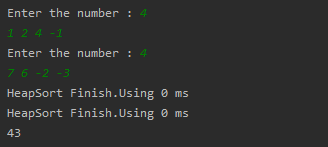
###### 测试样例与运行结果：

测试样例：

4 1 2 4 -1

4 7 6 -2 -3

运行结果：

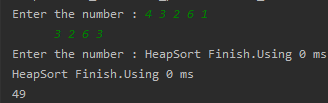


测试样例：

4 3 2 6 1

3 2 6 3

运行结果：

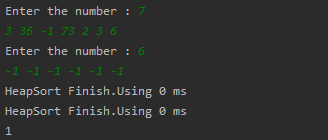


测试样例：

7 3 36 -1 73 2 3 6

6 -1 -1 -1 -1 -1 -1

运行结果：



### 选做题第13题

###### 题目：行车路线 (选做)（图）

[问题描述]

小明和小芳出去乡村玩，小明负责开车，小芳来导航。

小芳将可能的道路分为大道和小道。大道比较好走，每走1公里小明会增加1的疲劳度。小道不好走，如果连续走小道，小明的疲劳值会快速增加，连续走s公里小明会增加s2的疲劳度。

例如：有5个路口，1号路口到2号路口为小道，2号路口到3号路口为小道，3号路口到4号路口为大道，4号路口到5号路口为小道，相邻路口之间的距离都是2公里。如果小明从1号路口到5号路口，则总疲劳值为(2+2)2+2+22=16+2+4=22。

现在小芳拿到了地图，请帮助她规划一个开车的路线，使得按这个路线开车小明的疲劳度最小。

[基本要求]

输入格式：

输入的第一行包含两个整数n, m，分别表示路口的数量和道路的数量。路口由1至n编号，小明需要开车从1号路口到n号路口。

接下来m行描述道路，每行包含四个整数t, a, b, c，表示一条类型为t，连接a与b两个路口，长度为c公里的双向道路。其中t为0表示大道，t为1表示小道。保证1号路口和n号路口是连通的。

输出格式

输出一个整数，表示最优路线下小明的疲劳度。

样例输入

6 7

1 1 2 3

1 2 3 2

0 1 3 30

0 3 4 20

0 4 5 30

1 3 5 6

1 5 6 1

样例输出

76

样例说明

从1走小道到2，再走小道到3，疲劳度为52=25；然后从3走大道经过4到达5，疲劳度为20+30=50；最后从5走小道到6，疲劳度为1。总共为76。

课程设计要求：

（1）要求从文本文件中输入；

（2）采用适当的数据结构存储由输入数据中的道路所形成的图结构；

（3）编写尽可能优的算法，处理好连续走小道造成的疲劳值的指数增长（提示：基于迪杰斯特拉算法进行改进即可完成本题）；

（4）除严格按题目要求进行输出以外，还要求输出最优路线的路径，以及从出发点到各个点的最小疲劳值。

###### 数据结构：图

###### 算法思想：

从起始点出发，记录到其他点沿着大路的距离  
计算到其他点沿着小路的距离，若较短，则替换距离，记录两个点的疲劳度  
寻找距离最近的点，标记为以经过点，观察如何到达该点：  
 1.沿着大路，清空所有疲劳度  
 2.沿着小路，记录该疲劳度  
从该点出发，计算从起始点到该点的距离与该点到其他点的沿着大路距离和，若小于起始点到其他点的距离，更新  
从该点出发，计算从起始点到该点的距离与该点到其他点的沿着小路距离和：  
 1.上次沿着大路，直接计算疲劳度的平方  
 2.上次沿着小路，计算两次疲劳度和与差之积  
重复上述过程，直到标记所有点

###### 算法时间复杂度：n^2

###### 具体实现：

//  
// Created by 97927 on 2019/12/27.  
//  
#include <string>  
#include <vector>  
#include <fstream>  
#include <iostream>  
using namespace std;  
#define MAX 65535  
class GNode{  
private:  
 int site;  
 std::string data;  
protected:  
public:  
 std::vector<int> out\_e;//big road  
 std::vector<int> in\_e;//small road  
 GNode(int s, std::string d)//initial  
 {  
 site = s;  
 data = d;  
 }  
 GNode(int s, std::string d, int n)//initial  
 {  
 site = s;  
 data = d;  
 out\_e.assign(n,MAX);  
 in\_e.assign(n,MAX);  
 }  
 bool setSite(int s)//change this node's site  
 {  
 site = s;  
 return true;  
 }  
 bool setData(std::string da)//change this node's data  
 {  
 data = da;  
 }  
 bool enOutEdge(int st,int dis)//set distance between this node and st  
 {  
 out\_e[st] = dis;  
  
 return true;  
 }  
 int outEdgeNumber()//return out edge number  
 {  
 return out\_e.size();  
 }  
 bool enInEdge(int st,int dis)  
 {  
 in\_e[st] = dis;  
 return true;  
 }  
 bool show();//show this node information  
 ~GNode()= default;  
};  
class my\_Graph {  
private:  
 std::vector<GNode> frt;  
 int nd\_num;  
 int eg\_num;  
public:  
 my\_Graph(std::string filename);//initial  
 ~my\_Graph() = default;  
 bool dijkstra(std::vector<int> & T,int bgn);  
  
};  
my\_Graph::my\_Graph(string filename)  
{  
 std::ifstream in(filename);  
 int nd\_n;  
 in>>nd\_n;  
 nd\_num = nd\_n;  
 for(int i = 0;i < nd\_n;i++)  
 {  
 std::string str;  
 str += '0' + (i / 10);  
 str += '0' + (i % 10);  
 GNode n\_node(i,str,nd\_n);  
 frt.push\_back(n\_node);  
 frt[i].show();  
 cout<<endl;  
 }  
 in>>eg\_num;  
 while(!in.eof())  
 {  
 int tp;  
 int n\_st1,n\_st2;  
 int dis;  
 in>>tp;  
 in>>n\_st1>>n\_st2>>dis;  
 if(n\_st1 == -1 || n\_st2 == -1)  
 continue;  
 if(tp == 0)  
 {  
 frt[n\_st1 - 1].enOutEdge(n\_st2 - 1,dis);  
 frt[n\_st2 - 1].enOutEdge(n\_st1 - 1,dis);  
 } else{  
 frt[n\_st1 - 1].enInEdge(n\_st2 - 1,dis);  
 frt[n\_st2 - 1].enInEdge(n\_st1 - 1,dis);  
 }  
 }  
 in.close();  
}  
  
bool my\_Graph::dijkstra(std::vector<int> &T, int bgn) {  
 //sign the node have passed  
 std::vector<bool> D(nd\_num,false);  
 //double Fatigue value  
 std::vector<int> doubleT(nd\_num,0);  
 //initial distance  
 for(int i = 0;i < nd\_num;i++)  
 {  
 T.push\_back(frt[bgn].out\_e[i]);  
 }  
 //if small road is shorter  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(frt[bgn].in\_e[i] == MAX)  
 continue;  
 int dis = frt[bgn].in\_e[i] \* frt[bgn].in\_e[i];  
 if(dis < T[i])  
 {  
 T[i] = dis;  
 //store the fatigue value  
 doubleT[i] = frt[bgn].in\_e[i];  
 }  
 }  
 int p = bgn;  
 bool flag = false;  
 D[bgn] = true;  
 while(true)  
 {  
 int p\_last = p;  
 int min = -1;  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 continue;  
 if(min == -1 || T[i] < min)  
 {  
 min = T[i];  
 p = i;  
 //if drive on big road  
 flag = doubleT[i] > 0;  
 }  
 }  
 if(min == -1)  
 break;  
 cout<<"from "<<p\_last + 1<<" to "<<p + 1<<" by road ";  
 if(flag)  
 cout<<1;  
 else  
 cout<<0;  
 cout<<" totally cost "<<T[p]<<endl;  
 //if this time on small road  
 if(flag)  
 {  
 int t = doubleT[p];  
 for(int i = 0;i < nd\_num;i++)  
 {  
 doubleT[i] = t;  
 }  
 } else{  
 //clear the fatigue value  
 for(int i = 0;i < nd\_num;i++)  
 {  
 doubleT[i] = 0;  
 }  
 }  
 //refresh distance  
 for(int i = 0;i < nd\_num;i++)  
 {  
 if(D[i])  
 continue;  
 if(frt[p].in\_e[i] == MAX)  
 {  
 if(T[i] > T[p] + frt[p].out\_e[i])  
 {  
 T[i] = T[p] + frt[p].out\_e[i];  
 doubleT[i] = 0;  
 }  
 } else {  
 int dis = (frt[p].in\_e[i] + 2 \* doubleT[i]) \* frt[p].in\_e[i];  
 if (dis > frt[p].out\_e[i]) {  
 if (T[i] > T[p] + frt[p].out\_e[i]) {  
 T[i] = T[p] + frt[p].out\_e[i];  
 doubleT[i] = 0;  
 }  
 } else {  
 if (T[i] > T[p] + dis) {  
 T[i] = T[p] + dis;  
 doubleT[i] += frt[p].in\_e[i];  
 }  
 }  
 }  
 }  
 D[p] = true;  
 }  
 return true;  
}  
  
bool GNode::show() {  
 cout<<"node : "<<data<<" site : "<<site;  
  
 return true;  
}  
int main()  
{  
 std::string filename = "road13.txt";  
 my\_Graph g(filename);  
 std::vector<int> T;  
 g.dijkstra(T,0);  
 cout<<T.back();  
  
 return 0;  
}

###### 测试样例与运行结果：

测试样例：

6 7

1 1 2 3

1 2 3 2

0 1 3 30

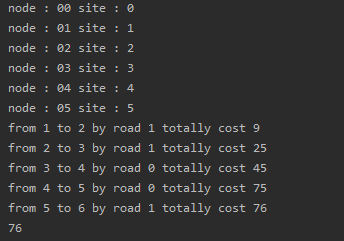
0 3 4 20

0 4 5 30

1 3 5 6

1 5 6 1

运行结果：



### 选做题第14题

###### 题目：跳一跳（选做）

问题描述

近来，跳一跳这款小游戏风靡全国，受到不少玩家的喜爱。 简化后的跳一跳规则如下：玩家每次从当前方块跳到下一个方块，如果没有跳到下一个方块上则游戏结束。 如果跳到了方块上，但没有跳到方块的中心则获得1分；跳到方块中心时，若上一次的得分为1分或这是本局游戏的第一次跳跃则此次得分为2分，否则此次得分比上一次得分多两分（即连续跳到方块中心时，总得分将+2，+4，+6，+8…）。 现在给出一个人跳一跳的全过程，请你求出他本局游戏的得分（按照题目描述的规则）。

输入格式

输入包含多个数字，用空格分隔，每个数字都是1，2，0之一，1表示此次跳跃跳到了方块上但是没有跳到中心，2表示此次跳跃跳到了方块上并且跳到了方块中心，0表示此次跳跃没有跳到方块上（此时游戏结束）。

输出格式

输出一个整数，为本局游戏的得分（在本题的规则下）。

样例输入

1 1 2 2 2 1 1 2 2 0

样例输出

22

数据规模和约定

对于所有评测用例，输入的数字不超过30个，保证0正好出现一次且为最后一个数字。

###### 数据结构：线性表

###### 算法思想：

建立变量n = 2，读取数字：  
 1.读取到1，总分加1，将n更新为2  
 2.读取到2，总分加n，n = n \* 2  
 3.读取到0，结束

###### 算法时间复杂度：n

###### 具体实现：

//  
// Created by 97927 on 2019/12/27.  
//  
#include <iostream>  
#include <vector>  
#include "my\_Queue.h"  
using namespace std;  
  
int sore[3] = {0,1,2};  
int main()  
{  
 int nm;  
 cin>>nm;  
 int sum = 0;  
 while(nm != 0)  
 {  
 if(nm == 1)  
 {  
 sum += 1;  
 sore[1] = 2;  
 }  
 else if(nm == 2)  
 {  
 sum += sore[1];  
 sore[1] += 2;  
 }  
 cin>>nm;  
 }  
 cout<<sum;  
  
 return 0;  
}

###### 测试样例与运行结果：

测试样例：

1 1 2 2 2 1 1 2 2 0

运行结果：



### 选做题第16题

###### 题目：迷宫问题（选做）（栈与队列），（深度搜索，广度搜索）

[问题描述]

利用栈操作实现迷宫问题求解。

[基本要求]

（1）从文件中读取数据，生成模拟迷宫地图，不少于10行10列。

（2）给出任意入口和出口，显示输出迷宫路线。

###### 数据结构：数组

###### 算法思想：

利用深度优先遍历，广度优先遍历来寻找从起点到终点的道路

###### 算法时间复杂度：n

###### 具体实现：

opinion.cpp

//  
// Created by 97927 on 2020/1/2.  
//  
#include <iostream>  
#include <cstdlib>  
#include <cstdio>  
#include <string>  
#include <stack>  
#include <queue>  
#include <fstream>  
#include "function.h"  
using namespace std;  
  
#define MAZELENGTH 10  
#define MAZEWIDE 10  
  
/\*0 = road / 1 = wall / 2 = beginning / 3 = ending / 4 = road have go / 5 = bfs road\*/  
class bfsNode{  
public:  
 int n1,n2;  
 int p1,p2;//last node position  
 bfsNode(int d1 = -1,int d2 = -1,int d3 = -1,int d4 = -1)  
 {  
 n1 = d1;  
 n2 = d2;  
 p1 = d3;  
 p2 = d4;  
 }  
 bool set(int d1 = -1,int d2 = -1,int d3 = -1,int d4 = -1)  
 {  
 n1 = d1;  
 n2 = d2;  
 p1 = d3;  
 p2 = d4;  
 }  
};  
class dfsNode{  
public:  
 int n1,n2;  
 int dir;//last direction  
 dfsNode(int d1 = -1,int d2 = -1,int d3 = -1)  
 {  
 n1 = d1;  
 n2 = d2;  
 dir = d3;  
 }  
 bool operator=(const dfsNode &d)  
 {  
 n1 = d.n1;  
 n2 = d.n2;  
 dir = d.dir;  
 return true;  
 }  
};  
  
bool bfsTraverse(int maze[][MAZELENGTH],int begin1,int begin2)  
{  
 queue<bfsNode> que;  
 stack<bfsNode> s;  
 bfsNode p;  
 que.push(bfsNode(begin1, begin2));  
 while(!que.empty())  
 {  
 p = que.front();  
 que.pop();  
 s.push(p);  
  
 if(p.n2 - 1 > -1)  
 {  
 //find the end  
 if(maze[p.n1][p.n2 - 1] == 3)  
 break;  
 if(maze[p.n1][p.n2 - 1] == 0)  
 {  
 bfsNode nw(p.n1, p.n2 - 1, p.n1, p.n2);  
 que.push(nw);  
 maze[p.n1][p.n2 - 1] = 4;  
 }  
 }  
 if(p.n1 - 1 > -1)  
 {  
 //find the end  
 if(maze[p.n1 - 1][p.n2] == 3)  
 break;  
 if(maze[p.n1 - 1][p.n2] == 0)  
 {  
 bfsNode nw(p.n1 - 1, p.n2, p.n1, p.n2);  
 que.push(nw);  
 maze[p.n1 - 1][p.n2] = 4;  
 }  
 }  
 if(p.n1 + 1 < MAZEWIDE)  
 {  
 //find the end  
 if(maze[p.n1 + 1][p.n2] == 3)  
 break;  
 if(maze[p.n1 + 1][p.n2] == 0)  
 {  
 bfsNode nw(p.n1 + 1, p.n2, p.n1, p.n2);  
 que.push(nw);  
 maze[p.n1 + 1][p.n2] = 4;  
 }  
 }  
 if(p.n2 + 1 < MAZELENGTH)  
 {  
 //find the end  
 if(maze[p.n1][p.n2 + 1] == 3)  
 break;  
 if(maze[p.n1][p.n2 + 1] == 0)  
 {  
 bfsNode nw(p.n1, p.n2 + 1, p.n1, p.n2);  
 que.push(nw);  
 maze[p.n1][p.n2 + 1] = 4;  
 }  
 }  
 }  
 if(!que.empty())  
 {  
 //find the road  
 int node1 = p.p1,node2 = p.p2;  
 maze[p.n1][p.n2] = 5;  
 while(!s.empty())  
 {  
 bfsNode pause;  
 pause = s.top();  
 s.pop();  
 if(maze[pause.n1][pause.n2] == 4)  
 maze[pause.n1][pause.n2] = 0;  
 //find the last node  
 if(pause.n1 == node1 && pause.n2 == node2)  
 {  
 maze[pause.n1][pause.n2] = 5;  
 node1 = pause.p1;  
 node2 = pause.p2;  
 }  
 }  
 }  
 else{  
 cout<<"don't have way"<<endl;  
 return false;  
 }  
 //restitute the maze  
 while(!que.empty())  
 {  
 bfsNode pause;  
 pause = que.front();  
 que.pop();  
 if(maze[pause.n1][pause.n2] == 4)  
 maze[pause.n1][pause.n2] = 0;  
 }  
  
 return true;  
}  
  
bool dfsTraverse(int maze[][MAZELENGTH],int begin1,int begin2)  
{  
 stack<dfsNode> s;  
 bool flag = true;  
 bool end = false;  
 s.push(dfsNode(begin1,begin2,0));  
 while(!s.empty()) {  
 dfsNode p;  
 int dir;  
 //don't need to change direction  
 if (flag) {  
 dir = 0;  
 } else {  
 dir = s.top().dir;  
 maze[s.top().n1][s.top().n2] = 0;  
 s.pop();  
 }  
 p = s.top();  
 switch (dir) {  
 case 0 : {  
 if (p.n1 + 1 < MAZEWIDE) {  
 if (maze[p.n1 + 1][p.n2] == 3)  
 {  
 end = true;  
 break;  
 }  
 if (maze[p.n1 + 1][p.n2] == 0) {  
 s.push(dfsNode(p.n1 + 1, p.n2, 1));  
 maze[p.n1 + 1][p.n2] = 5;  
 flag = true;  
 break;  
 }  
 }  
 }  
 case 1 : {  
 if (p.n2 - 1 > 0) {  
 if (maze[p.n1 + 1][p.n2] == 3)  
 {  
 end = true;  
 break;  
 }  
 if (maze[p.n1][p.n2 - 1] == 0) {  
 s.push(dfsNode(p.n1, p.n2 - 1, 2));  
 maze[p.n1][p.n2 - 1] = 5;  
 flag = true;  
 break;  
 }  
 }  
 }  
 case 2 : {  
 if (p.n1 - 1 > 0) {  
 if (maze[p.n1 + 1][p.n2] == 3)  
 {  
 end = true;  
 break;  
 }  
 if (maze[p.n1 - 1][p.n2] == 0) {  
 s.push(dfsNode(p.n1 - 1, p.n2, 3));  
 maze[p.n1 - 1][p.n2] = 5;  
 flag = true;  
 break;  
 }  
 }  
 }  
 case 3 : {  
 if (p.n2 + 1 < MAZEWIDE) {  
 if (maze[p.n1 + 1][p.n2] == 3)  
 {  
 end = true;  
 break;  
 }  
 if (maze[p.n1][p.n2 + 1] == 0) {  
 s.push(dfsNode(p.n1, p.n2 + 1, 4));  
 maze[p.n1][p.n2 + 1] = 5;  
 flag = true;  
 break;  
 }  
 }  
 }  
 default: {  
 flag = false;  
 }  
 }  
 if(end)  
 break;  
 }  
 if(s.empty())  
 {  
 cout<<"don't have way"<<endl;  
 return false;  
 }  
 return true;  
}  
  
int main()  
{  
 ifstream in("maze.txt");  
 int maze[MAZEWIDE][MAZELENGTH] = {0};  
 for(int i = 0;i < MAZEWIDE;i++)  
 {  
 for(int k = 0;k < MAZELENGTH;k++)  
 {  
 int node;  
 in>>node;  
 maze[i][k] = node;  
 }  
 }  
 in.close();  
 //printf the maze  
 for(int i = 0;i < MAZEWIDE;i++)  
 {  
 for(int k = 0;k < MAZELENGTH;k++)  
 {  
 switch (maze[i][k]) {  
 case 0 :  
 cout<<"o ";  
 break;  
 case 1 :  
 cout<<'\24'<<" ";  
 break;  
 case 2 :  
 cout<<"B ";  
 break;  
 case 3 :  
 cout<<"$ ";  
 break;  
 case 4 :  
 case 5 :  
 cout<<"\* ";  
 break;  
 }  
 }  
 cout<<endl;  
 }  
 int node1,node2,node3,node4;  
 while(true)  
 {  
 node1 = getInt("beginning node's raw");  
 node2 = getInt("beginning node's column");  
 //array site  
 if(maze[node1 - 1][node2 - 1] == 0)  
 {  
 maze[node1 - 1][node2 - 1] = 2;  
 break;  
 }  
 cout<<"This node is wall"<<endl;  
 }  
 while(true)  
 {  
 node3 = getInt("ending node's raw");  
 node4 = getInt("ending node's column");  
 if(maze[node3 - 1][node4 - 1] == 0)  
 {  
 maze[node3 - 1][node4 - 1] = 3;  
 break;  
 }  
 cout<<"This node is wall"<<endl;  
 }  
 //show the road  
 if(bfsTraverse(maze, node1 - 1, node2 - 1))  
 {  
 maze[node1 - 1][node2 - 1] = 2;  
 for(int i = 0;i < MAZEWIDE;i++)  
 {  
 for(int k = 0;k < MAZELENGTH;k++)  
 {  
 switch (maze[i][k]) {  
 case 0 :  
 cout<<"o ";  
 break;  
 case 1 :  
 cout<<'\24'<<" ";  
 break;  
 case 2 :  
 cout<<"B ";  
 break;  
 case 3 :  
 cout<<"$ ";  
 break;  
 case 4 :  
 case 5 :  
 cout<<"\* ";  
 break;  
 }  
 }  
 cout<<endl;  
 }  
 }  
  
 return 0;  
}

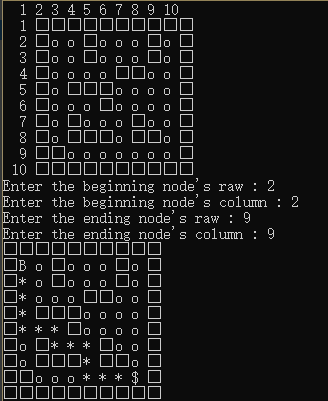
###### 测试样例与运行结果：

测试样例：

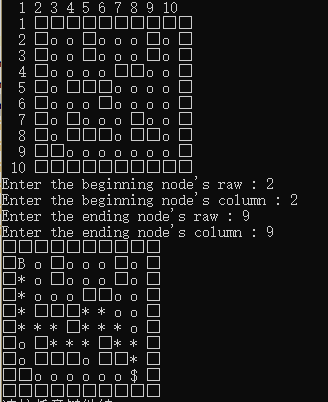
1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1

运行结果：

广度优先遍历：



深度优先遍历：



### 选做题第19题

###### 题目：B-树应用 （选做） （查找）

[问题描述]

设计并实现B-树的一种应用。

[基本要求]

（1）从文件中读取数据

（2）实现B-树的插入、删除、查找功能。

###### 数据结构：树

###### 算法思想：

每个节点存储个数为n  
插入：  
 查找该数字要插入的位置，进行判断：  
 1.该位置的数字个数 < n，直接插入该数字  
 2.该位置的数字个数 = n，将该位置的数与要插入的数字进行排序，取中位 数，建立两棵子树，左子树存储小于中位数的数，右子树存储大于中位数的 数，将中位数存储到该树的父节点中，两棵子树指向父节点，对父节点进行判 断：  
 2.1.父节点的数字个数 < n,直接 插入中位数  
 2.2.父节点的数字个数 = n，按上述过程分裂  
删除：  
 查找该数字的位置，进行判断：  
 1.该位置在叶节点，且该节点数字个数大于n / 2 + 1，直接删除  
 2.该位置在叶节点，且该节点数字个数为n / 2，对左兄弟树进行判断：  
 2.1.左兄弟树的数字个数大于n / 2 + 1,将父节点的数替换该数字， 左兄弟树最大的数替换父节点中的数  
 2.2.左兄弟树的数字个数等于n / 2 ,将父节点的数下移，与两棵子树 合并为一棵子树，指向父节点  
 3.该位置有子节点，用左子树的最大值代替该数，进行删除该最大值操作

###### 算法时间复杂度：

###### 具体实现：

opinion19.cpp

//  
// Created by 97927 on 2019/12/31.  
//  
#include <iostream>  
#include <cstdlib>  
#include <cstdio>  
#include <string>  
#include <vector>  
#include <queue>  
#include <fstream>  
  
using namespace std;  
  
class b\_Tree {  
private:  
 vector<int> data;  
 int size;//size of the tree  
 int nowsize;  
 vector<b\_Tree \*> child;  
 b\_Tree \*root;  
 b\_Tree \*pa;  
  
 b\_Tree \*upOn(b\_Tree \*p);//split to up  
 b\_Tree \*dowmDe(b\_Tree \* p,int &n);//merge to bottom  
 bool mergeTree(b\_Tree \*p1, b\_Tree \*p2, int pa\_dt);//merge the pa and child  
 //find this tree is the nth child of parent  
 int siteForParent(b\_Tree \* p)  
 {  
 int i = 0;  
 for(;i< p->pa->child.size();i++)  
 {  
 if(p->pa->child[i] == p)  
 break;  
 }  
 return i;  
 }  
 //find this data's site  
 int siteForDate(b\_Tree \* p,int dt)  
 {  
 for(int k = 0;k < p->data.size();k++)  
 {  
 if(p->data[k] == dt)  
 {  
 return k;  
 }  
 }  
 return -1;  
 }  
  
  
public:  
 //initial  
 b\_Tree(int dt, int sz, b\_Tree \*parent = nullptr) {  
 data.push\_back(dt);  
 size = sz;  
 nowsize = 1;  
 pa = parent;  
 root = this;  
 }  
 //initial  
 b\_Tree(int dt, int sz, b\_Tree \*parent, b\_Tree \*rt) {  
 data.push\_back(dt);  
 size = sz;  
 nowsize = 1;  
 pa = parent;  
 root = rt;  
 }  
 //initial  
 b\_Tree(vector<int> &dt, int sz, b\_Tree \*parent = nullptr) {  
 data.insert(data.begin(), dt.begin(), dt.end());  
 size = sz;  
 nowsize = 1;  
 pa = parent;  
 root = this;  
 }  
 //initial add child  
 bool inchild(vector<b\_Tree \*> &c) {  
 child.insert(child.end(), c.begin(), c.end());  
  
 return true;  
 }  
 //add new data  
 bool append(int dt) {  
 b\_Tree \*p = root;  
 while (p) {  
 //find the position  
 bool flag = false;  
 int i = 0;  
 while (i < p->data.size()) {  
 if (p->data[i] > dt) {  
 flag = true;  
 break;  
 }  
 i++;  
 }  
  
 if (p->child.empty()) {  
 //leaves  
 if (flag) {  
 //add to the ith position  
 p->data.insert(p->data.begin() + i, dt);  
 } else {  
 //add back  
 p->data.push\_back(dt);  
 }  
 //if this full out  
 b\_Tree \*q = upOn(p);  
 while (q) {  
 q = upOn(q);  
 }  
 break;  
 } else {  
 //find in child  
 if (!flag) {  
 p = p->child.back();  
 } else {  
 p = p->child[i];  
 }  
 }  
 }  
 }  
  
 bool show() {  
 std::queue<b\_Tree \*> q;  
 b\_Tree \*p = root;  
 cout << "(";  
 for (int i = 0; i < p->data.size(); i++) {  
 cout << p->data[i] << ",";  
 }  
 cout << ")" << endl;  
 if (!p->child.empty()) {  
 for (int i = 0; i < p->child.size(); i++) {  
 q.push(p->child[i]);  
 }  
 }  
 q.push(nullptr);  
 while (!q.empty()) {  
 p = q.front();  
 if (p == nullptr) {  
 q.pop();  
 cout << endl;  
 if (!q.empty())  
 q.push(nullptr);  
 continue;  
 }  
 cout << "(";  
 for (int i = 0; i < p->data.size(); i++) {  
 cout << p->data[i] << ",";  
 }  
 cout << ")" << "\t";  
 q.pop();  
 if (!p->child.empty()) {  
 for (int i = 0; i < p->child.size(); i++) {  
 q.push(p->child[i]);  
 }  
 }  
 }  
 cout << endl << "levelOrderTraverse finish." << endl;  
  
 return true;  
 }  
 //delete data  
 bool delt(int dt)  
 {  
 b\_Tree \* p = root;  
 int i = 0;  
 while(p)  
 {  
 bool flag = false;  
 i = 0;  
 //find data  
 while (i < p->data.size()) {  
 if (p->data[i] >= dt) {  
 flag = true;  
 break;  
 }  
 i++;  
 }  
 if(p->data[i] == dt)  
 break;  
 if(!p->child.empty())  
 {  
 if (!flag) {  
 p = p->child.back();  
 } else {  
 p = p->child[i];  
 }  
 }  
 }  
 if(p->data[i] == dt)  
 {  
 int n = dt;  
 while(p)  
 {  
 p = dowmDe(p,n);  
 }  
 }  
 return true;  
 }  
};  
  
b\_Tree \*b\_Tree::upOn(b\_Tree \*p) {  
 b\_Tree \* q = nullptr;  
 if(p->data.size() > p->size) {  
 //split to two trees  
 vector<int> dt2;  
 int dt\_mi = p->data[size / 2];  
 int st = size / 2;  
 dt2.insert(dt2.begin(), p->data.begin() + st + 1, p->data.end());  
 p->data.erase(p->data.begin() + st, p->data.end());  
 b\_Tree \* t2 = new b\_Tree(dt2, size, p->pa);  
 if (!p->child.empty()) {  
 vector<b\_Tree \*> c2;  
 for(int i = st + 1;i < p->child.size();i++)  
 {  
 c2.push\_back(p->child[i]);  
 p->child[i]->pa = t2;  
 }  
 t2->inchild(c2);  
 p->child.erase(p->child.begin() + st + 1, p->child.end());  
 }  
 q = p->pa;  
 if(q)  
 { int i = 0;  
 while(i < q->data.size())  
 {  
 if(dt\_mi < q->data[i])  
 break;  
 i++;  
 }  
 q->data.insert(q->data.begin() + i,dt\_mi);  
 q->child.insert(q->child.begin() + i + 1,t2);  
 } else{  
 q = new b\_Tree(dt\_mi,size);  
 q->child.push\_back(p);  
 q->child.push\_back(t2);  
 p->pa = q;  
 t2->pa = q;  
 root = q;  
 return nullptr;  
 }  
 }  
 //up on to the parent node and deal with the parent node  
 return q;  
}  
  
b\_Tree \*b\_Tree::dowmDe(b\_Tree \*p, int &n) {  
 b\_Tree \* parent = p->pa;  
 int st = size / 2 - 1;  
 int dt\_st = siteForDate(p,n);  
 //if it is leaf  
 if(p->child.empty())  
 {  
 p->data.erase(p->data.begin() + dt\_st);  
 //if have n / 2 + 1 numbers  
 if(p->data.size() > st)  
 {  
 return nullptr;  
 }  
 else{  
  
 bool flag = false;  
 int i = siteForParent(p);  
 //if left tree have n / 2 + 1 numbers  
 if(parent->child[i + 1]->data.size() > st + 1)  
 {  
 flag = true;  
 }  
 if(flag)  
 {  
 //replace the number by parent number , replace the parent number by left tree number  
 p->data.push\_back(parent->data[i]);  
 parent->data.erase(parent->data.begin() + i);  
 parent->data.insert(parent->data.begin() + i,parent->child[i + 1]->data.front());  
 n = parent->child[i + 1]->data.front();  
 return parent->child[i + 1];  
 }  
 else{  
 //merge two trees  
 mergeTree(p,parent->child[i + 1],parent->data[i]);  
 parent->child.erase(parent->child.begin() + i);  
 n = parent->data[i];  
 return parent;  
 }  
 }  
 } else{  
 int nw\_dt = p->child[dt\_st]->data.back();  
 n = nw\_dt;  
 p->data[dt\_st] = nw\_dt;  
 return p->child[dt\_st];  
 }  
}  
//merge two trees and the parent number  
bool b\_Tree::mergeTree(b\_Tree \*p1, b\_Tree \*p2, int pa\_dt) {  
 p1->data.push\_back(pa\_dt);  
 p1->data.insert(p1->data.end(),p2->data.begin(),p2->data.end());  
 delete p2;  
}  
  
int main()  
{  
 int nm;  
 string filename("opinion19.txt");  
 ifstream in(filename);  
 in>>nm;  
 b\_Tree t(nm,3);  
 while(!in.eof())  
 {  
 in>>nm;  
 cout<<nm<<endl;  
 t.append(nm);  
 t.show();  
 }  
 in.close();  
 t.show();  
 t.delt(11);  
 t.show();  
 return 0;  
}

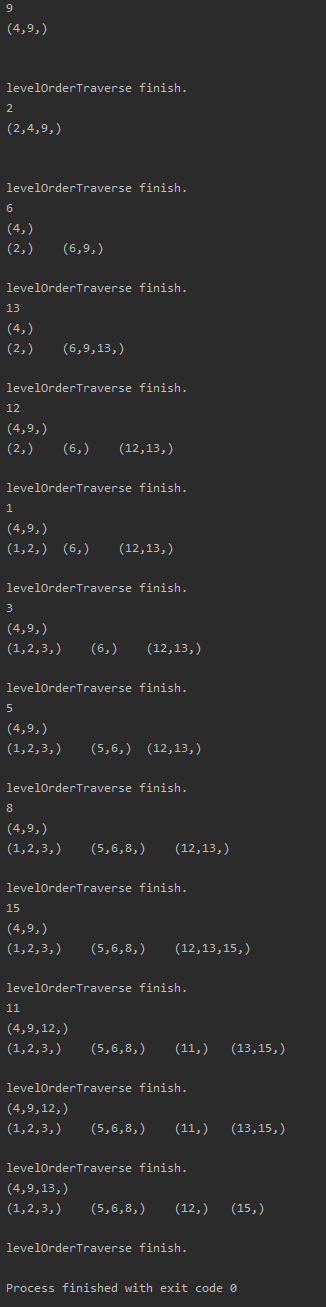
###### 测试样例与运行结果：

测试样例：

输入数据：4 9 2 6 13 12 1 3 5 8 15 11

删除：11

运行结果：



### 课设心得：

在本次课程设计中，我一共完成了14道题目，包括8道必做题，6道选做题。总共完成5200行代码的编写。

本次课程设计我收获巨大。首先在课程设计中，我强化了自己的c++代码编写了能力。在课程设计中编写了5200行代码，是我在实践上对c++语言进行训练。同时，在课程设计中遇到的问题，通过搜索资料解决的过程中，也加深了我对这些知识的记忆。例如在课设之前，我并不了解windows系统这些自带的api，也不知道运作原理。通过柴华溢和郭世祺两位同学分享的资料，加上我自己搜索的资料，与尝试，我明白系统进程的运行原理，以及windows自带的api的使用方法。最后，我成功的做出了第一道题。

在编写第二题的过程中，我并没有认真考虑自己程序的健壮性。在同学给了我一些错误的数据的帮助下，我加强了程序的健壮性，也开始对健壮性有了一些更多的考虑。同时，该题加强了我对后缀表达式，中缀表达式和前缀表达式的认识。

在面对必做题第七题，公交车问题时，我一开始一筹莫展，不知如何进行解决。后来通过认真的回顾课本知识，发现了在图那一节讲到的深度优先遍历和广度优先遍历两个算法刚好适用于这个情况，解决了算法问题。但是单单知道算法并不是就代表完成了，因为这道题有本身的公交车语境，在实际的深度优先遍历中，需要沿着同一个公交线路去查找，而不是直接按照第一个相邻的点去查找，给问题的解决带来了一定的困难。最后，通过设计前序点，终于解决了这个比较困难的问题，也让自己的编程水平得到了一定的提升。

本次课程设计中，我也发现了算法知识只是一个解决问题的方法，并不是万能的。在实际的问题中，需要我们对算法进行一定的变形，才能更好的解决问题。正如我选做的第13道选做题，行车路线。需要我们对迪杰斯特拉算法按照实际题目的要求进行改进，才能解决这道问题。同时理论知识与实际使用中也有一定的差距。像我做的第七道题，我在查阅书本后明白这道题可以选择使用深度优先遍历和广度优先遍历，但是实际运用上，并不是像我想法中那样容易编写，我必须通过设计一些其他的辅助结构等，才能解决这个问题。

同时，我也学会了在解决问题中把问题抽象化为某类问题，拆分该问题，而不是直接就编写程序。例如我在写选做题第19题时，一开始冥思苦想，想要解决插入中分裂的时候，需要不断分裂的问题。后来我发现可以把该问题抽象为继续插入的问题，就不需要编写大量的代码来解决不断分裂的问题，而是将分裂后的插入当作一次新的插入，这样重新开始一次新的循环，就可以解决所遇到的这些问题，也不用重复编写大量的代码。这个道理在删除过程中同样适用。在删除中存在大量的情况，我把这个情况分解为该数在叶节点，与该数不在叶节点的情况，当该数不在叶节点，我只需要把叶节点的数换上来代替该数，然后将这个问题转化为删除叶节点的那个数的问题，计算机就会不断的自己进行循环迭代解决问题。在不使用栈进行排序也是这个思想。我在堆排序中的建堆函数返回了发生变化的数的位置，这样每次改变一个数时，计算机就能通过循环一直向下调整，直到该部分的树符合我们的要求。通过这种拆分问题，归类问题的方式，不仅大大提高了我代码的运行速度，也减少了代码的数量，避免更多的问题。

通过此次课程设计，我深深感受到自己实际的代码编写能力还有欠缺，并不想我想象中那么好，理论与实际结合也仍有需要改进的地方。所以很感谢此次课程设计给我的锻炼，加强了我的实践能力，也加深了我对本学期所学习的知识的理解。