数据结构课程设计



班级： 1621501

学号： 162150109

姓名： 姚远

指导教师： 孙涵

**目录**

1、必做题

**1.1 菜鸟智慧系统**

1.1.1采用的数据结构 ………………………………………… 5

1.1.2算法设计思想 …………………………………………… 5

1.1.3关键代码 ………………………………………………… 5

1.1.4测试数据和结果 ………………………………………… 6

1.1.5.算法的时间复杂度及其改进方法 ……………………… 8

**1.2 算术表达式求值**

1.2.1采用的数据结构 ………………………………………… 8

1.2.2算法设计思想 …………………………………………… 8

1.2.3关键代码 ………………………………………………… 9

1.2.4测试数据和结果 ………………………………………… 10

1.2.5.算法的时间复杂度及其改进方法 ……………………… 12

**1.3 特殊路径统计**

1.3.1采用的数据结构 ………………………………………… 12

1.3.2算法设计思想 …………………………………………… 12

1.3.3关键代码 ………………………………………………… 12

1.3.4测试数据和结果 ………………………………………… 12

1.3.5.算法的时间复杂度及其改进方法 ……………………… 12

**1.4 公交线路提示**

1.4.1采用的数据结构 ………………………………………… 13

1.4.2算法设计思想 …………………………………………… 13

1.4.3关键代码 ………………………………………………… 13

1.4.4测试数据和结果 ………………………………………… 18

1.4.5.算法的时间复杂度及其改进方法 ……………………… 19

**1.5 Huffman编码与解码**

1.5.1采用的数据结构 ………………………………………… 20

1.5.2算法设计思想 …………………………………………… 20

1.5.3关键代码 ………………………………………………… 20

1.5.4测试数据和结果 ………………………………………… 24

1.5.5.算法的时间复杂度及其改进方法 ……………………… 25

**1.6 排序算法比较**

1.6.1采用的数据结构 ………………………………………… 25

1.6.2算法设计思想 …………………………………………… 25

1.6.3关键代码 ………………………………………………… 26

1.6.4测试数据和结果 ………………………………………… 31

1.6.5.算法的时间复杂度及其改进方法 ……………………… 31

2、选做题

**2.1 火车购票**

2.1.1采用的数据结构 ………………………………………… 32

2.1.2算法设计思想 …………………………………………… 32

2.1.3关键代码 ………………………………………………… 32

2.1.4测试数据和结果 ………………………………………… 33

2.1.5.算法的时间复杂度及其改进方法 ……………………… 33

**2.2 棋局评估**

2.2.1采用的数据结构 ………………………………………… 34

2.2.2算法设计思想 …………………………………………… 34

2.2.3关键代码 ………………………………………………… 34

2.2.4测试数据和结果 ………………………………………… 34

2.2.5.算法的时间复杂度及其改进方法 ……………………… 35

**2.3 通信管理系统**

2.3.1采用的数据结构 ………………………………………… 35

2.3.2算法设计思想 …………………………………………… 35

2.3.3关键代码 ………………………………………………… 35

2.3.4测试数据和结果 ………………………………………… 38

2.3.5.算法的时间复杂度及其改进方法 ……………………… 39

**2.4 词梯**

2.4.1采用的数据结构 ………………………………………… 39

2.4.2算法设计思想 …………………………………………… 39

2.4.3关键代码 ………………………………………………… 39

2.4.4测试数据和结果 ………………………………………… 41

2.4.5.算法的时间复杂度及其改进方法 ……………………… 42

**2.5 城市巡游赛**

2.5.1采用的数据结构 ………………………………………… 42

2.5.2算法设计思想 …………………………………………… 42

2.5.3关键代码 ………………………………………………… 42

2.5.4测试数据和结果 ………………………………………… 45

2.5.5.算法的时间复杂度及其改进方法 ……………………… 45

**2.6 平衡二叉树操作的演示**

2.6.1采用的数据结构 ………………………………………… 46

2.6.2算法设计思想 …………………………………………… 46

2.6.3关键代码 ………………………………………………… 46

2.6.4测试数据和结果 ………………………………………… 51

2.6.5.算法的时间复杂度及其改进方法 ……………………… 51

3、结束语………………………………………………………… 52

1、必做题

**1.1 菜鸟智慧系统**

**1.1.1 数据结构**

本题采用链表、队列。

**1.1.2 算法思想**

建立三个链表，对应不同的货架。

通过随机函数随机生成以下事件：

1、对三个货架生成不同数量的包裹，并按编号升序插入至链表

2、生成取包裹事件，其中再随机分为按取件码、按手机号取包裹，对应链表查找、删除操作

同时，我们一开始还需遍历三个链表，对过期的包裹进行删除操作。

然后使用存储空间为7的循环队列，维护最近7天内过期包裹的数量。

**1.1.3 关键代码**

Status PickListByID(LinkList &L, std::string &ID, Parcel &ret)

{

    LNode \*pre = L.Head, \*now = L.Head->next;

    while (now != NULL)

    {

        if (now->Data.ID == ID)

        {

            pre->next = now->next;

            ret = now->Data;

            --L.Size;

            L.Used[ret.id] = false;

            delete now;

            return OK;

            now = pre->next;

        }

        else

        {

            pre = now;

            now = now->next;

        }

    }

    return FAILPICK;

}

Status PickListByTel(LinkList &L, std::string Tel, int &ret)

{

    LNode \*pre = L.Head, \*now = L.Head->next;

    while (now != NULL)

    {

        if (now->Data.user.Tel == Tel)

        {

            ++ret;

            pre->next = now->next;

            Parcel ret = now->Data;

            --L.Size;

            L.Used[ret.id] = false;

            delete now;

            now = pre->next;

        }

        else

        {

            pre = now;

            now = now->next;

        }

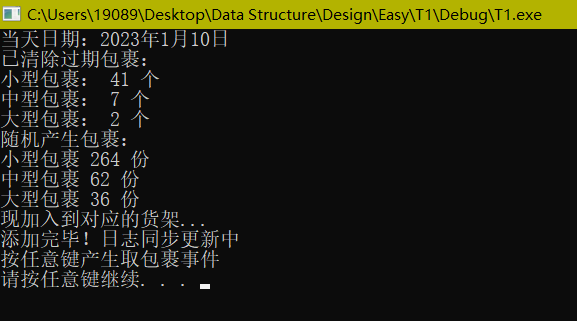
    }

    return OK;

}

**1.1.4 测试结果**

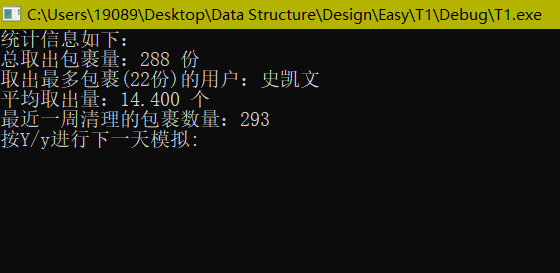
产生包裹：



取包裹：



统计信息：



**1.1.5 时间复杂度与改进**

时间复杂度：

链表插入、删除皆为O(n)

改进策略：

使用UI来展示内容，用命令行容易造成刷屏

**1.2 算术表达式求值**

**1.2.1 数据结构**

本题采用栈。

**1.2.2 算法思想**

对于计算过程，设立两个栈分别存储操作数与操作符。

若当前为操作数，压栈。

若当前为操作符，根据与上个操作符的优先级比较，进行分类讨论。

本题重点在于如何保证程序的鲁棒性，即对异常情况与特殊情况的讨论。

该程序讨论了以下异常情况，并支持输出相关错误信息：

1、出现非法字符（比如1+a）

2、操作符位置不合法（比如(-2\*-3)）

3、数字不合法（比如-2.5\*6.9.6+.3）

4、括号不合法（比如(2\*3+5))）

5、除数为0（比如3/(2-2)）

6、其它未知错误

**1.2.3 关键代码**

double cal()

{

    Cache[++n].pd = false; Cache[n].ch = ')';

    std::stack<double> OPND;

    std::stack<char> OPTR;

    OPTR.push('(');

    double a, b, c; char opt;

    int \_tot = 0;

    for (int i = 1; i <= n;)

    {

    //  printf("i = %d\n", i);

        if (Cache[i].pd)

            OPND.push(Cache[i].num), ++i,

            printf("第%d步操作：push OPND %lf\n", ++\_tot,OPND.top());

        else

        {

            switch (cmp(OPTR.top(), Cache[i].ch))

            {

            case '<':

                OPTR.push(Cache[i].ch);

                ++i;

                printf("第%d步操作：push OPTR %c\n", ++\_tot, OPTR.top());

                break;

            case '>':

                b = OPND.top(); OPND.pop(); printf("第%d步操作：pop OPND %lf\n", ++\_tot, b);

                a = OPND.top(); OPND.pop(); printf("第%d步操作：pop OPND %lf\n", ++\_tot, a);

                opt = OPTR.top(); OPTR.pop(); printf("第%d步操作：pop OPTR %c\n", ++\_tot, opt);

                c = Operate(a, b, opt);

                OPND.push(c); printf("第%d步操作：push OPND %lf\n", ++\_tot, c);

                break;

            case '=':

                OPTR.pop();

                printf("第%d步操作：pop OPTR %c\n", ++\_tot, '(');

                ++i;

                break;

            default:

                STATUS = ERROR;

                break;

            }

        }

        if (STATUS != OK)

            return 0;

    //  printf("OPND:");

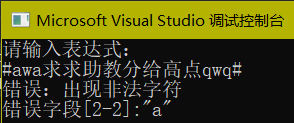
    }

    return OPND.top();

}

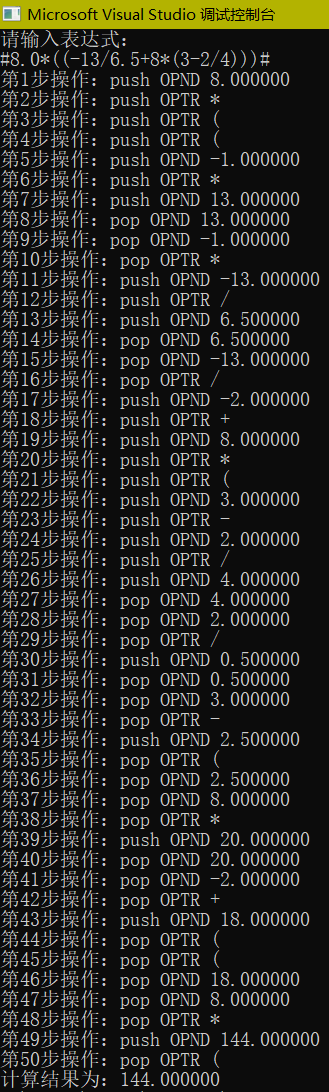
**1.2.4 测试结果**

部分异常情况：





正常情况：



**1.2.5 时间复杂度与改进**

时间复杂度：O(n)

改进：用表达式树

**1.3 特殊路径统计**

**1.3.1 数据结构**

本题采用图。

**1.3.2 算法思想**

枚举起始点，分别进行DFS，DFS的同时也记录上下界，若满足要求则答案+1。最后答案需除以二。

**1.3.3 关键代码**

void DFS(Graph &G, int u, int down, int up,int fa)

{

    if (u < down) down = u;

    if (u > up) up = u;

    if (down > now || up < now)

        return;

    if (u != now && (up == u && down == now || up == now && down == u))

        ++ans;

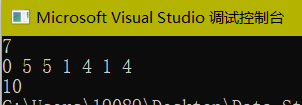
    for (auto \*k = G.V[u].head; k != NULL; k = k->next)

        if(k->to != fa)

            DFS(G, k->to, down, up,u);

}

**1.3.4 测试结果**



**1.3.5 时间复杂度与改进**

时间复杂度O(n²)

改进：若当前上下界中没有一个与始点相同，可剪枝，但总体时间复杂度并没有降低。

**1.4 公交线路提示**

**1.4.1 数据结构**

本题采用图、队列。

**1.4.2 算法思想**

建立两个图。

第一个图，我们建立车与车之间的图。每个结点为一辆车，每条边为相关的两辆车是否在公交线路上有交集。这样子在这张图上进行BFS可以得出转乘最少的路线。

输出方案的时候，我们寻找前后两辆车的交集，并取出其中一个站点作为转乘站点。

第二个图，则是常规的图。在这张图上进行BFS可以得出经过站点最少的路线。

输出方案的时候，我们每次二分出一个里当前站点最远的站点，并满足两个站点之间有交集，该站点也作为转乘站点。

**1.4.3 关键代码**

void Solve1(std::string from,std::string to)

{

    if (!M.count(from))

    {

        puts("错误：出发地不存在！");

        return;

    }

    if (!M.count(to))

    {

        puts("错误：目的地不存在！");

        return;

    }

    if (from == to)

    {

        puts("出发地即为目的地，你这坐个锤子车，是不是嫌钱不够花？不如vivo50(bushi");

        return;

    }

    bool \*vis = new bool[bus\_cnt + 1];

    int \*dis = new int[bus\_cnt + 1];

    int \*F = new int[bus\_cnt + 1];

    std::queue<int> Q;

    int des = -1;

    for (int i = 1; i <= bus\_cnt; ++i)

    {

        vis[i] = false;

        dis[i] = 0x3f3f3f3f;

        if (std::find(A[i].begin(), A[i].end(), from) != A[i].end())

        {

            Q.push(i);

            vis[i] = true;

            dis[i] = F[i] = 0;

        }

    }

    while (!Q.empty())

    {

        int u = Q.front();

        Q.pop();

        if (std::find(A[u].begin(), A[u].end(), to) != A[u].end())

        {

            des = u;

            break;

        }

        for (ArcNode \*k = G1.V[u].Head; k != NULL; k = k->next)

        {

            int v = k->v;

            if (vis[v])

                continue;

            vis[v] = true;

            dis[v] = dis[u] + 1;

            F[v] = u;

            Q.push(v);

        }

    }

    if (des == -1)

    {

        puts("未找到路线！");

        delete[] vis;

        delete[] F;

        delete[] dis;

        return;

    }

    if (dis[des] == 0)

    {

        std::cout << "只需乘坐 " << N[des] << " 路公交车从 " << from << " 到 " << to << " 即可，无需换乘！";

    }

    else

    {

        printf("乘车路线如下（总转车%d次）：\n",dis[des]);

        int \*cache = new int[dis[des] + 1];

        int u = des;

        int cnt = 0;

        while (u)

            cache[++cnt] = u,

            u = F[u];

        std::string nowSta = from;

        for (int i = cnt; i > 1; --i)

        {

            int a = cache[i], b = cache[i - 1];

            for (auto it = A[a].begin(); it < A[a].end(); ++it)

            {

                if (std::find(A[b].begin(), A[b].end(), \*it) != A[b].end())

                {

                    if (i == cnt)

                        std::cout << "首先";

                    else

                        std::cout << "然后";

                    std::cout << "在 " << nowSta << " 乘坐 " << N[a] << " 路公交车，至 " << \*it << " 下车；\n";

                    nowSta = \*it;

                    break;

                }

            }

        }

        std::cout << "最后在 " << nowSta << " 乘坐 " << N[cache[1]] << " 路公交车，至 " << to << "下车。\n";

    }

    delete[] vis;

    delete[] F;

    delete[] dis;

}

void Solve2(std::string from,std::string to)

{

    if (!M.count(from))

    {

        puts("错误：出发地不存在！");

        return;

    }

    if (!M.count(to))

    {

        puts("错误：目的地不存在！");

        return;

    }

    if (from == to)

    {

        puts("出发地即为目的地，你这坐个锤子车，是不是嫌钱不够花？不如vivo50(bushi");

        return;

    }

    int Dis[6666],f[6666];

    memset(Dis, 0x3f, sizeof Dis);

    Dis[M[from]] = 0; f[M[from]] = 0;

    std::queue<int> Q;

    Q.push(M[from]);

    while (!Q.empty())

    {

        int u = Q.front();

        if (u == M[to])

            break;

        Q.pop();

        for (auto \*k = G2.V[u].Head; k != NULL; k = k->next)

        {

            int v = k->v;

            if (Dis[v] < 0x3f3f3f3f)

                continue;

            Dis[v] = Dis[u] + 1;

            f[v] = u;

            Q.push(v);

        }

    }

    if (Dis[M[to]] == 0x3f3f3f3f)

    {

        puts("未找到路线！");

        return;

    }

    std::vector<int> W,Ans;

    Ans.push\_back(0);

    for (int i = M[to]; i; i = f[i])

        W.push\_back(i);

    std::reverse(W.begin(), W.end());

    int l = 0, r = W.size() - 1,ll,rr;

    while (l < W.size() - 1)

    {

        ll = l, rr = W.size() - 1;

        while (ll < rr)

        {

            int mid = ll + rr >> 1; ++mid;

            bool pd = false;

            for(auto it = B[W[l]].begin(); it != B[W[l]].end(); ++it)

                for(auto it2 = B[W[mid]].begin(); it2 != B[W[mid]].end(); ++it2)

                    if (\*it == \*it2)

                    {

                        pd = true;

                        break;

                    }

            if (pd) ll = mid;

            else rr = mid - 1;

        }

        Ans.push\_back(ll);

        l = ll;

    }

    for (int i = 0; i < Ans.size() - 1; ++i)

    {

        l = Ans[i], r = Ans[i + 1]; bool pd = false;

        for (auto it = B[W[l]].begin(); it != B[W[l]].end(); ++it)

        {

            for (auto it2 = B[W[r]].begin(); it2 != B[W[r]].end(); ++it2)

                if (\*it == \*it2)

                {

                    pd = true;

                    if (!i)

                        printf("首先");

                    else if (W[r] == M[to])

                        printf("最后");

                    else printf("然后");

                    std::cout << "在 " << C[W[l]] << " 乘坐 " << N[\*it] << " 路公交车，至 " << C[W[r]] << " 下车";

                    if (W[r] == M[to]) puts("。");

                    else puts("；");

                    break;

                }

            if (pd) break;

        }

        assert(pd);

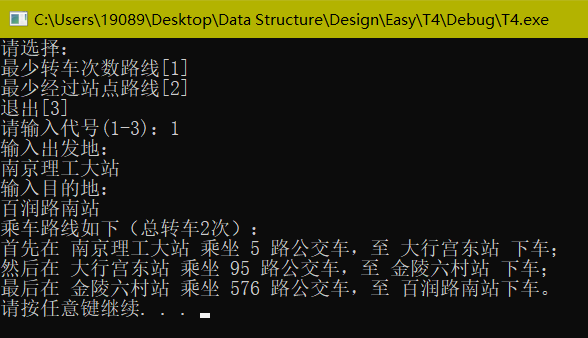
    }

    std::cout << "总经过站点：" << Dis[M[to]] + 1 << std::endl;

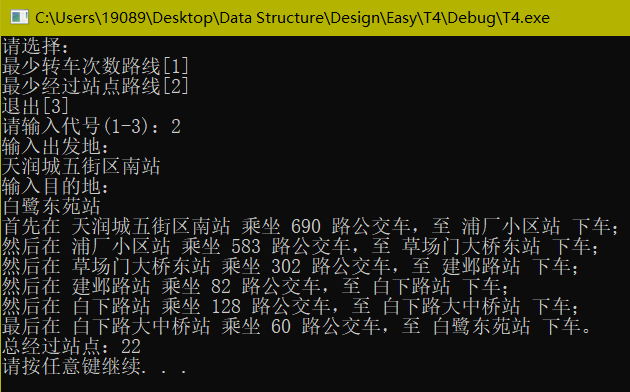
}

**1.4.4 测试结果**

转乘最少：



站点最少：



**1.4.5 时间复杂度与改进**

本做法的时间复杂度主要集中在建第一个图。

我们需要枚举每一对公交车，然后根据各自的站点数量暴力判断是否存在交集。

设公交车有n辆，各自的期望站点数量为m，总站点数量为，则时间复杂度为O(n²m²)。

转乘最少：O(n)

站点最少：O()

改进：

提前建图，存入文件，这样程序效率将会大大提高。

**1.5 Huffman编码与解码**

**1.5.1 数据结构**

本题采用树、堆。

**1.5.2 算法思想**

首先从文件中读取数据，记录各个字符的出现次数。然后按照合并果子的思想，用STL的priority\_queue取出当前频率最少的两个结点，合并成新的一个结点，直至剩下最后一个结点，作为huffman树的根。

编码过程中，每8位进行二进制输出。

**1.5.3 关键代码**

void EnCode()

{

    std::fstream FILE("code.dat",std::ios::out | std::ios::binary);

    std::fstream SRC("source.txt", std::ios::in);

    std::string str;

    int len = 0;

    while (SRC.good())

    {

        std::getline(SRC, str);

        for (auto it = str.begin(); it != str.end(); ++it)

        {

            for (auto it2 = Code[RH[\*it]].begin(); it2 != Code[RH[\*it]].end(); ++it2)

                tag[len++] = \*it2;

        }

        if(SRC.good())

        for (auto it2 = Code[RH['\n']].begin(); it2 != Code[RH['\n']].end(); ++it2)

            tag[len++] = \*it2;

    }

    for (auto it2 = Code[RH[128]].begin(); it2 != Code[RH[128]].end(); ++it2)

        tag[len++] = \*it2;

    SRC.close();

    int l = 0;

    while (true)

    {

        unsigned char buf = 0;

        for (int i = l, j = 0; j < 8; ++i, ++j)

        {

            if(i < len)

                buf = (buf << 1) | (tag[i] - '0');

            else buf <<= 1;

        }

        FILE.write((char \*)&buf, sizeof(buf));

        l += 8;

        if (l >= len)

            break;

    }

    FILE.close();

}

void DeCode()

{

    std::fstream FILE("code.dat", std::ios::in | std::ios::binary);

    std::fstream TGT("recode.txt", std::ios::out);

    int len = 0;

    while (!FILE.eof())

    {

        unsigned char t = 0;

        FILE.read((char \*)&t, sizeof(unsigned char));

        if (FILE.fail())

            break;

        for (int i = 7; ~i; --i)

            tag[len++] = (((1<<i) & t) > 0) + '0';

    }

    std::string mod;

    std::string zero, one;

    zero.append("0");

    one.append("1");

    for (int i = 0; i < len; ++i)

    {

        bool end = false;

        if (tag[i] == '0')

            mod.append(zero);

        else

            mod.append(one);

        for(int j = 0; j < ccnt; ++j)

            if (Code[j] == mod)

            {

                if (j == RH[128])

                {

                    end = true;

                    break;

                }

                mod.clear();

                TGT << H[j];

                break;

            }

        if (end) break;

    }

    TGT.close();

    FILE.close();

}

void HufCode()

{

    HT = (HufTree)malloc(sizeof(HTNode) \* 256);

    int cnt = 0;

    C[128] = 1; vis[128] = true; //128表示结束符号

    while (!Q.empty()) Q.pop();

    for (int i = 0; i <= 128; ++i)

        if (vis[i])

        {

            HT[cnt].lchild = HT[cnt].rchild = -1;

            HT[cnt].weight = C[i];

            H[cnt] = i; RH[i] = cnt;

            Q.push(std::make\_pair(C[i], cnt++));

        }

    ccnt = cnt;

    while (Q.size() > 1)

    {

        std::pair<int, int> a, b;

        a = Q.top(); Q.pop();

        b = Q.top(); Q.pop();

        HT[a.second].parent = HT[b.second].parent = cnt;

        HT[cnt].lchild = a.second;

        HT[cnt].rchild = b.second;

        HT[cnt].parent = -1;

        HT[cnt].weight = a.first + b.first;

        Q.push(std::make\_pair(a.first + b.first, cnt++));

    }

    for (int i = 0; i < ccnt; ++i)

    {

        int s = 0;

        int c, f;

        for (c = i, f = HT[c].parent; f != -1; c = f, f = HT[f].parent)

        {

            if (HT[f].lchild == c)

                Code[i].append("0");

            else

                Code[i].append("1");

        }

        std::reverse(Code[i].begin(), Code[i].end());

    //  std::cout << H[i] << ":" << Code[i] << std::endl;

    }

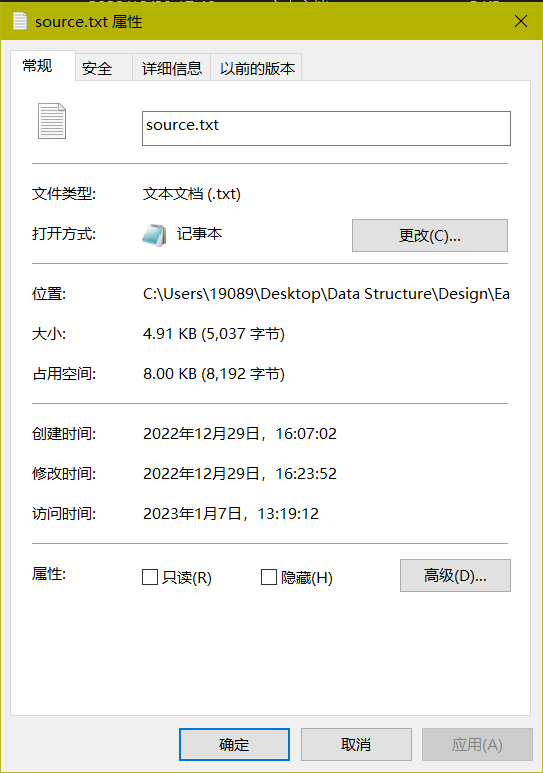
    free(HT);

    EnCode();

}

**1.5.4 测试结果**

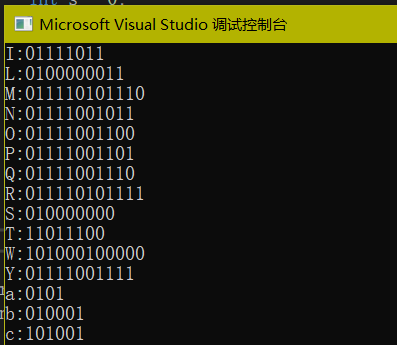
原文本大小：



压缩后大小：



部分编码结果：



**1.5.5 时间复杂度与改进**

设有效字符为n个。使用优先队列。

则编码的时间复杂度为O(nlogn)

改进：手写堆，降低常数。

**1.6 排序算法比较**

**1.6.1 数据结构**

堆排使用堆，基排使用队列。

**1.6.2 算法思想**

直接插入排序：选出当前最小的直接插入。

希尔排序：使用Hibbard增量序列，每个增量插入排序。

冒泡排序：把当前最大的交换到后面。

快速排序：选取中间点为基准点，一趟快排后分治。

选择排序：选择当前最小的与当前交换。

堆排序：维护个大顶堆。每次从堆顶取出元素与最后的交换。

归并排序：将两个较小的有序序列归并到一起。

基数排序：从个位到最高位进行关键字的桶排，并使用队列维持已排序关键字的相对顺序。

**1.6.3 关键代码**

void Insert(int i)

{

    int tmp = A[i];

    int j;

    for (j = i; j > 1; --j)

        if (tmp < A[j - 1])

            A[j] = A[j - 1];

        else break;

    A[j] = tmp;

}

void InsertSort()

{

    rep(i, 2, n)

        Insert(i);

}

void ShellInsert(int d)

{

    rep(i, d + 1, n)

    {

        if (A[i] < A[i - d])

        {

            int tmp = A[i], j;

            for (j = i; j > d; j -= d)

                if (tmp < A[j - d])

                    A[j] = A[j - d];

                else break;

            A[j] = tmp;

        }

    }

}

void ShellSort() //Hibbard增量序列

{

    int d[30], tot = 0;

    rep(j, 1, 20)

    {

        d[j] = (1 << j) - 1;

        if (d[j] >= n)

            break;

        ++tot;

    }

    per(i, 1, tot)

        ShellInsert(d[i]);

}

void BubbleSort()

{

    rep(i, 1, n)

    {

        rep(j, 1, n - i)

            if (A[j] > A[j + 1])

            {

                int tmp = A[j];

                A[j] = A[j + 1];

                A[j + 1] = tmp;

            }

    }

}

int Partition(int l, int r)

{

    int tmp = A[(l+r)>>1];

    A[(l + r) >> 1] = A[l];

    A[l] = tmp;

    while (l < r)

    {

        while (l < r && A[r] >= tmp)

            --r;

        A[l] = A[r];

        while (l < r && A[l] <= tmp)

            ++l;

        A[r] = A[l];

    }

    A[l] = tmp;

    return l;

}

void QSort(int l, int r)

{

    if (l < r)

    {

        int p = Partition(l, r);

        QSort(l, p-1);

        QSort(p + 1, r);

    }

}

void QuickSort()

{

    QSort(1, n);

}

void SelectSort()

{

    rep(i, 1, n)

    {

        int k = i;

        rep(j, i + 1, n)

            if (A[k] > A[j])

                k = j;

        int tmp = A[i];

        A[i] = A[k];

        A[k] = tmp;

    }

}

void HeapAdjust(int l, int r)

{

    int tmp = A[l];

    int i = l, j = i << 1;

    while (j <= r)

    {

        if (j + 1 <= r && A[j + 1] > A[j])

            ++j;

        if (tmp > A[j])

            break;

        A[i] = A[j];

        i = j;

        j <<= 1;

    }

    A[i] = tmp;

}

void HeapSort()

{

    per(i, 1, n / 2)

        HeapAdjust(i, n);

    per(i, 2, n)

    {

        swap(A[1], A[i]);

        HeapAdjust(1, i - 1);

    }

}

void Merge(int l, int mid, int r)

{

    int T[50010],tot = 0;

    int i = l, j = mid + 1;

    while (i <= mid && j <= r)

    {

        if (A[i] < A[j])

            T[++tot] = A[i++];

        else

            T[++tot] = A[j++];

    }

    while(i <= mid)

        T[++tot] = A[i++];

    while (j <= r)

        T[++tot] = A[j++];

    rep(k, l, r)

        A[k] = T[k - l + 1];

}

void MSort(int d)

{

    int i = 1;

    while (i + 2 \* d <= n)

    {

        Merge(i, i + d - 1, i + 2 \* d - 1);

        i += d << 1;

    }

    if (i + d <= n)

        Merge(i, i + d - 1, n);

}

void MergeSort()

{

    for (int i = 1; i <= n; i <<= 1)

        MSort(i);

}

int Q[50010][10]; //手写队列，在VS环境下比STL快100倍（

void RadixSort()

{

    int l[10], r[10];

//  queue<int> Q[10];

    ll bas = 1;

    rep(i, 1, 10)

    {

        bas \*= 10;

        rep(j, 0, 9)

            l[j] = 1, r[j] = 0;

        rep(j, 1, n)

        {

            int m = (A[j] % bas) / (bas / 10);

            Q[++r[m]][m] = A[j];

        }

        int k = 0;

        rep(j, 0, 9)

            while (l[j] <= r[j])

                A[++k] = Q[l[j]++][j];

    }

}

**1.6.4 测试结果**

****

**1.6.5 时间复杂度与改进**

O(n²)：直接插入、冒泡、选择

O()：希尔

O(nlogn)：堆排、快排、归并

O(dn)：基排（这里d为10）

改进：别用STL，常数很大，最好自己写堆、写队列

2、选做题

**2.1 火车购票**

**2.1.1 数据结构**

本题使用数组。

**2.1.2 算法思想**

先判断是否存在一排有着连续的空位，若不存在再按序填即可。纯模拟。

**2.1.3 关键代码**

int main()

{

    cin >> n;

    while (n--)

    {

        int c;

        cin >> c;

        bool tag;

        rep(i, 1, 20)

        {

            rep(j, 1, 5 - c + 1)

            {

                tag = true;

                rep(k, 1, c)

                    tag &= !vis[(i - 1) \* 5 + j + k - 1];

                if (tag)

                {

                    rep(k, 1, c)

                        cout << (i - 1) \* 5 + j + k - 1 << ' ',

                        vis[(i - 1) \* 5 + j + k - 1] = true;

                    cout << endl;

                    break;

                }

            }

            if (tag)

                break;

        }

        if (tag)

            continue;

        rep(i, 1, 100)

        {

            if (!vis[i])

            {

                vis[i] = true;

                --c;

                cout << i << ' ';

                if (!c)

                    break;

            }

        }

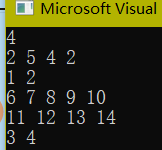
        cout << endl;

    }

    return 0;

}

**2.1.4 测试结果**



CSP官网测试100分

**2.1.5 时间复杂度与改进**

O(100n)

改进：可以发现，同一排中，不可能出现一个空的座位，它的后面座位被填补。也就是说每一排的空的座位一直都是连着的。故我们可以实时维护每排空座的数量，进而优化枚举，这样可以稍微降低点复杂度。

**2.2 棋局评估**

**2.2.1 数据结构**

本题使用数组。

**2.2.2 算法思想**

使用博弈论的对抗搜索。每次DFS的时候取对于当前玩家最优的策略返回即可，记得记忆化。

**2.2.3 关键代码**

int DFS(int x)

{

    int sta = Hash();

    if (vis[sta]) return F[sta];

    vis[sta] = true;

    if (Alice())

        return F[sta] = blank() + 1;

    else if (Bob())

        return F[sta] = -(blank() + 1);

    if (blank() == 0)

        return F[sta] = 0;

    int ret;

    if (x == 1) ret = -2333;

    else ret = 2333;

    rep(i,1,4)

        rep(j,1,4)

        if (!A[i][j])

        {

            A[i][j] = x;

            int d = DFS(0x3 ^ x);

            if (x == 1) CMAX(ret, d);

            else CMIN(ret, d);

            A[i][j] = 0;

        }

    return F[sta] = ret;

}

**2.2.4 测试结果**



CSP官网测试100分

**2.2.5 时间复杂度与改进**

状态量最多为O()

改进：可以采取Alpha-Beta剪枝进行优化，但没必要。

**2.3 通信管理系统**

**2.3.1 数据结构**

本题使用set。使用图的思想。

**2.3.2 算法思想**

首先考虑事件的存储。我们给每一天开个vector，存储一系列事件。

然后我们给每个计算机开个set，从大到小存储其余的计算机与当前计算机的每日可用额度。

然后我们实时维护通信孤岛与通信对的数量即可。对于通信对的数量需要分类讨论进行计算，较为麻烦。

**2.3.3 关键代码**

rep(i, 1, m)

    {

    //  printf("Day%d:\n", i);

        int k;

        read(k);

        while (k--)

        {

            int u, v, x, y;

            read(u); read(v); read(x); read(y);

        //  scanf("%d%d%d%d", &u, &v, &x, &y);

            A[i].pb(Node(u, v, x));

            if (i + y <= m)

                A[i + y].pb(Node(u, v, -x));

        }

        for (auto p : A[i])

        {

            int u = p.u, v = p.v;

            int u1 = 0, v1 = 0, u2 = 0, v2 = 0; //对应的

            int x1 = 0, y1 = 0, x2 = 0, y2 = 0; //对应的对应的

            ll x = p.x;

        //  printf("u = %d v = %d x = %lld\n", u, v, x);

            if (!M.count(mp(u, v)))

                M[mp(u, v)] = ++tot;

            if (!M.count(mp(v, u)))

                M[mp(v, u)] = ++tot;

            int t1 = M[mp(u, v)], t2 = M[mp(v, u)];

            int s1 = S[u].size(), s2 = S[v].size();

            if (s1)

            {

                u1 = S[u].begin()->first;

                if(S[u1].size())x1 = S[u1].begin()->first;

            }

            if (s2)

            {

                v1 = S[v].begin()->first;

                if (S[v1].size())y1 = S[v1].begin()->first;

            }

            auto it = S[u].find(mp(v,C[t1]));

            if (it == S[u].end())

            {

                S[u].insert(mp(v, x));

                C[t1] = x;

            }

            else

            {

                C[t1] += x;

                S[u].erase(it);

                if(C[t1]) S[u].insert(mp(v, C[t1]));

            }

            it = S[v].find(mp(u, C[t2]));

            if (it == S[v].end())

            {

                S[v].insert(mp(u, x));

                C[t2] = x;

            }

            else

            {

                C[t2] += x;

                S[v].erase(it);

                if(C[t2]) S[v].insert(mp(u, C[t2]));

            }

            int s11 = S[u].size(), s22 = S[v].size();

            if (s1 == 0 && s11) --cnt0;

            if (s1 && s11 == 0) ++cnt0;

            if (s2 == 0 && s22) --cnt0;

            if (s2 && s22 == 0) ++cnt0;

            if (s11)

            {

                u2 = S[u].begin()->first;

                if (S[u2].size())x2 = S[u2].begin()->first;

            }

            if (s22)

            {

                v2 = S[v].begin()->first;

                if (S[v2].size())y2 = S[v2].begin()->first;

            }

            if (u1 == v && v1 == u)

            {

                if (u2 == v && v2 == u)

                    continue;

                else --cnt1;

                if (u == x2) ++cnt1;

                if (v == y2) ++cnt1;

            }

            else if(u2 == v && v2 == u)

            {

                ++cnt1;

                if (u == x1) --cnt1;

                if (v == y1) --cnt1;

            }

            else

            {

                if (u == x1 && u != x2) --cnt1;

                if (u != x1 && u == x2) ++cnt1;

                if (v == y1 && v != y2) --cnt1;

                if (v != y1 && v == y2) ++cnt1;

            }

        //  printf("cnt0 = %d cnt1 = %d\n", cnt0, cnt1);

        }

        int l;

        read(l);

        while (l--)

        {

            int id;

            read(id);

            if (!S[id].size())

                puts("0");

            else

                printf("%d\n", S[id].begin()->first);

        }

        int p0, p1;

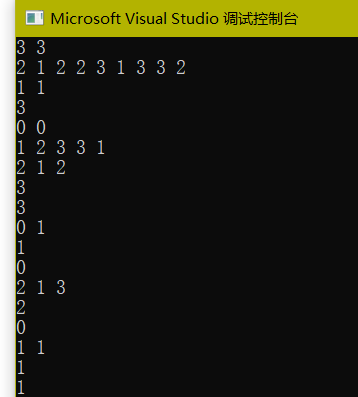
        read(p0); read(p1);

        if (p0) printf("%d\n", cnt0);

        if (p1) printf("%d\n", cnt1);

    }

**2.3.4 测试结果**



CSP官网测试100分

**2.3.5 时间复杂度与改进**

O((A+B)logn)，但STL常数较大，可能需要一些优化：

改进：使用快读，编译优化等等

**2.4 词梯**

**2.4.1 数据结构**

本题使用图的思想和set。

**2.4.2 算法思想**

使用BFS。对于当前单词，枚举它每一位上可能产生的变化，随后进行迭代。判断一个单词是否存在于单词表可以使用Hash，但由于数据量很大，为了保证正确性，这里使用set。

**2.4.3 关键代码**

int main()

{

    Init();

    puts("输入起始单词：");

    std::cin >> S;

    puts("输入终止单词：");

    std::cin >> T;

    if (S.size() != T.size())

    {

        puts("单词长度不相同！");

        return 0;

    }

    if (S == T)

    {

        puts("两个单词相同！");

        return 0;

    }

    if (s[S.size()].find(S) == s[S.size()].end())

    {

        puts("起始单词不存在！");

        return 0;

    }

    if (s[T.size()].find(T) == s[T.size()].end())

    {

        puts("终止单词不存在！");

        return 0;

    }

    M[S] = ++n; F[n] = 0; L[n] = -1;

    FM[n] = S;

    Q.push(n);

    while (!Q.empty())

    {

        int u = Q.front();

        Q.pop();

        std::string us = FM[u];

    //  std::cout << "F[" << us << "]=" << F[u] << std::endl;

        if (us == T)

            break;

        for (int i = 0; i < us.size(); ++i)

        {

            char CH = us[i];

            std::string str = us;

            for (char ch = 32; ch < 127; ++ch)

            {

                if (ch == CH)

                    continue;

                str[i] = ch;

            //  if (str == T) printf("%d!\n", us.size());

                if (s[us.size()].find(str) != s[us.size()].end())

                {

                    if (!M.count(str))

                    {

                        M[str] = ++n;

                        F[n] = F[u] + 1;

                        L[n] = u;

                        FM[n] = str;

                        Q.push(n);

                    }

                }

            }

        }

    }

    if (!M.count(T))

    {

        puts("词梯不存在！");

        return 0;

    }

    int id = M[T];

    int u = id;

    for (int i = 1; i <= F[id]; ++i)

    {

        A[i] = u;

        u = L[u];

    }

    puts("词梯为：");

    std::cout << S;

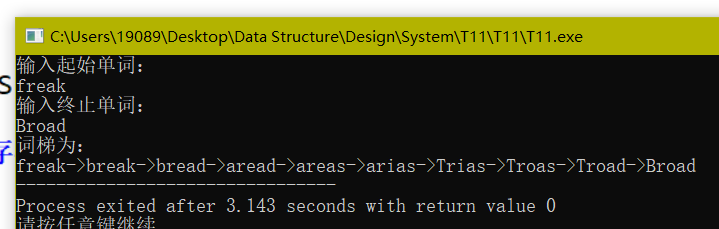
    for (int i = F[id]; i; --i)

        std::cout << "->" << FM[A[i]];

    return 0;

}

**2.4.4 测试结果**



**2.4.5 时间复杂度与改进**

O(nlogn\*d)，d为有效字符数量

但实测需要时间较长。

改进：换个编译器。实测用dev-c++跑的很快。

**2.5 城市巡游赛**

**2.5.1 数据结构**

本题使用图。

**2.5.2 算法思想**

首先对于原图，枚举每个点对(X,Y)，用LCA判断点对是否满足X能到达Y的要求，若满足则依据此建立新图。对于新图，使用Tarjan算法进行缩点，最后使用拓扑排序跑遍dp即可。

**2.5.3 关键代码**

int LCA(int u, int v)

{

    if (dep[u] > dep[v])

        std::swap(u, v);

    int hu = dep[u], hv = dep[v];

    int tu = u, tv = v;

    for (int det = hv - hu, i = 0; det; det >>= 1, ++i)

    {

        if (det & 1)

            tv = F[tv][i];

    }

    if (tu == tv)

        return tu;

    for (int i = MAX\_I; ~i; --i)

    {

        if (F[tu][i] == F[tv][i])

            continue;

        tu = F[tu][i];

        tv = F[tv][i];

    }

    return F[tu][0];

}

void Tarjan(Graph &G, int u,std::stack<int> &Stack)

{

    int v;

    Low[u] = DFN[u] = ++Index;

    Stack.push(u);

    Instack[u] = true;

    for (ArcNode\* k = G.V[u].Head; k != NULL; k = k->next)

    {

        v = k->point;

        if (!DFN[v])

        {

            Tarjan(G, v, Stack);

            if (Low[u] > Low[v])

                Low[u] = Low[v];

        }

        else if (Instack[v] && Low[u] > DFN[v])

            Low[u] = DFN[v];

    }

    if (Low[u] == DFN[u])

    {

        ++scc;

        do

        {

            v = Stack.top();

            Stack.pop();

            Instack[v] = false;

            Bel[v] = scc;

            ++Siz[scc];

        } while (v != u);

    }

}

int main()

{

    std::cin >> n;

    CreateGraph(T, n);

    CreateGraph(G, n);

    for (int i = 1; i < n; ++i)

    {

        int u, v;

        std::cin >> u >> v;

        AddEdge(T, u, v);

        AddEdge(T, v, u);

    }

    for (int i = 1; i <= n; ++i)

        std::cin >> T.V[i].A;

    DFS(T, 1, 0);

    for(int i = 1; i <= n; ++i)

        for (int j = 1; j <= n; ++j)

        {

            if (i == j)

                continue;

            int length = dep[i] + dep[j] - 2 \* dep[LCA(i, j)];

            if (length == T.V[i].A)

                AddEdge(G, i, j);

        }

    std::stack<int> S;

    for (int i = 1; i <= n; ++i)

        if (!DFN[i])

            Tarjan(G, i, S);

    CreateGraph(IG, scc);

    for (int i = 1; i <= n; ++i)

    {

        for (ArcNode \*k = G.V[i].Head; k != NULL; k = k->next)

        {

            int v = k->point;

            if (Bel[i] == Bel[v])

                continue;

            AddEdge(IG, Bel[i], Bel[v]);

            ++ind[Bel[v]];

        }

    }

    std::queue<int> Q;

    for (int i = 1; i <= scc; ++i)

        if (!ind[i])

            Q.push(i);

    while (!Q.empty())

    {

        int u = Q.front();

        Q.pop();

        dp[u] += Siz[u];

        for (ArcNode \*k = IG.V[u].Head; k != NULL; k = k->next)

        {

            int v = k->point;

            dp[v] = std::max(dp[v], dp[u]);

            --ind[v];

            if (!ind[v])

                Q.push(v);

        }

        ans = std::max(ans, dp[u]);

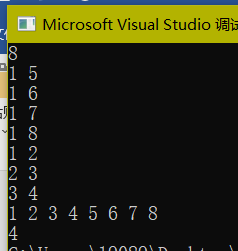
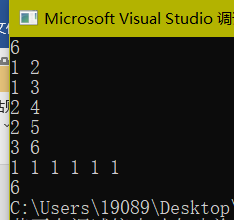
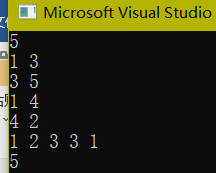
    }

    std::cout << ans;

    return 0;

}

**2.5.4 测试结果**



**2.5.5 时间复杂度与改进**

O(n²logn)

改进：其实完全不需要LCA，对每个点暴力DFS的话可以优化到 O(n²)。

**2.6 平衡二叉树操作的演示**

**2.6.1 数据结构**

本题使用AVL平衡二叉树和栈。

**2.6.2 算法思想**

首先，对于结点的插入与删除，遵守二叉查找树的性质。

其次，注意到一条规律：每次对平衡树的调整只存在于调整的结点与根之间的路径上。因此我们可以使用栈来存储路径上的结点，然后回溯进行调整即可。

具体调整内容有4个：LL型，RR型，LR型，RL型。根据不同的平衡因子情况进行调整。注意，调整完可能父子关系发生变化，最后可能需要手动连接。

**2.6.3 关键代码**

bool Insert(AVLTree &ptr, int &key)

{

    if (ptr == NULL)

    {

        ptr = (AVLNode \*)malloc(sizeof(AVLNode));

        ptr->key = key;

        ptr->bf = 0;

        ptr->left = ptr->right = NULL;

        return true;

    }

    AVLNode \*pr = NULL, \*p = ptr, \*q;

    std::stack<AVLNode \*> S;

    while (p != NULL)

    {

        if (key == p->key)

            return false;

        S.push(p);

        p = (key < p->key) ? p->left : p->right;

    }

    p = (AVLNode \*)malloc(sizeof(AVLNode));

    p->key = key; p->bf = 0;

    p->left = p->right = NULL;

    if (!S.size())

        ptr = p;

    else if (key < S.top()->key)

        S.top()->left = p;

    else

        S.top()->right = p;

    while (!S.empty())

    {

        pr = S.top(); S.pop();

        if (p == pr->left)

            ++pr->bf;

        else

            --pr->bf;

        if (!pr->bf)

            break;

        if (pr->bf == 1 || pr->bf == -1)

            p = pr;

        else

        {

            int d = (pr->bf < 0) ? -1 : 1;

            if (p->bf == d)

                if (d == 1)

                    RotateLL(pr);

                else

                    RotateRR(pr);

            else

                if (d == 1)

                    RotateLR(pr);

                else

                    RotateRL(pr);

            break;

        }

    }

    if (S.empty())

        ptr = pr;

    else //重新链接

    {

        q = S.top();

        if (q->key > pr->key)

            q->left = pr;

        else

            q->right = pr;

    }

    return true;

}

bool Remove(AVLTree &ptr, int &key)

{

    AVLNode \*pr = NULL, \*p = ptr, \*q, \*gr = NULL;

    std::stack<AVLNode \*> S;

    int d, dd = 0;

    while (p != NULL)

    {

        if (key == p->key)

            break;

        pr = p;

        S.push(pr);

        p = (key < p->key) ? p->left : p->right;

    }

    if (p == NULL)

        return false;

    if (p->left != NULL && p->right != NULL)

    {

        pr = p;

        S.push(pr);

        q = p->left;

        while (q->right != NULL)

        {

            pr = q;

            S.push(pr);

            q = q->right;

        }

        p->key = q->key;

        p = q;

    }

    if (p->left != NULL)

        q = p->left;

    else

        q = p->right;

    if (pr == NULL) //网络上的代码大部分在这个地方有问题好像

    {

        ptr = q;

        return true;

    }

    else

        if (pr->left == p)

            pr->left = q;

        else

            pr->right = q;

    while (!S.empty())

    {

        pr = S.top();

        S.pop();

        if (pr->left == q)

            --pr->bf;

        else

            ++pr->bf;

        if (pr->left == NULL && pr->right == NULL)

            pr->bf = 0;

        if (!S.empty())

            gr = S.top(),

            dd = (gr->left == pr) ? 1 : -1;

        else

            dd = 0;

        if (pr->bf == 1 || pr->bf == -1)

            break;

        if (pr->bf != 0)

        {

            if (pr->bf == 2)

            {

                d = 1;

                q = pr->left;

            }

            else

            {

                d = -1;

                q = pr->right;

            }

            if (q->bf == 0)

            {

                if (d == 1)

                {

                    RotateLL(pr);

                    pr->bf = -1;

                    pr->right->bf = 1;

                }

                else

                {

                    RotateRR(pr);

                    pr->bf = 1;

                    pr->left->bf = -1;

                }

                break;

            }

            if (q->bf == d)

            {

                if (d == 1)

                    RotateLL(pr);

                else

                    RotateRR(pr);

            }

            else

            {

                if (d == 1)

                    RotateLR(pr);

                else

                    RotateRL(pr);

            }

            if (dd == 1)

                gr->left = pr;

            else if (dd == -1)

                gr->right = pr;

        }

        q = pr;

    }

    if (S.empty())

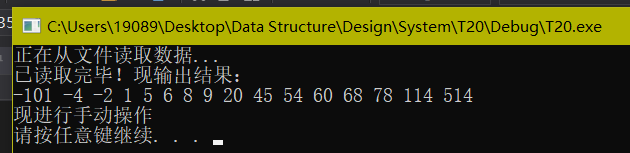
        ptr = pr;

    free(p);

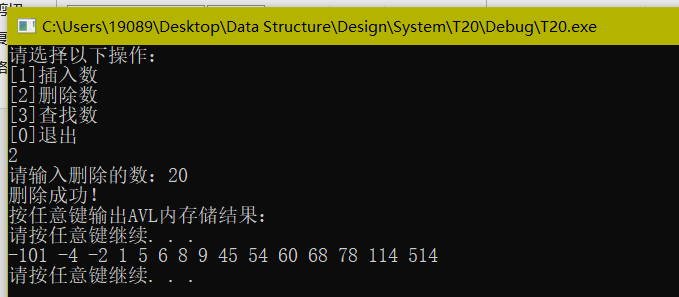
    return true;

}

**2.6.4 测试结果**







**2.6.5 时间复杂度与改进**

任何操作皆为O(logn)

改进：用其它平衡树，比如splay，代码短，效率也高，支持操作也多，但比较难理解。

3、结束语

**总体完成情况：**

除了第20题AVL平衡树参考了网上资料外（吐槽，大部分人的代码都还有bug），其余皆为个人独自编写。大部分功能均已实现，且部分题目分有头文件，模块清晰。部分地方有注释，增加可读性。程序健壮性尚可，除CSP题等已给出具体数据范围的题目，能处理大部分异常情况。

**代码量统计：**

T1：196+70+199+57+52=574

T2：367

T3：34+56=90

T4：374+49=423

T5：175

T6：382

T7：76

T8：116

T9：198

T11：109

T16：149+42=191

T20：74+271=345

总共3046行。

**心得体会：**

除了某些较难题（如通信系统管理、AVL树等），其余题目难度皆在可接受的范围内。身为ACM校队选手，这些题目用个人实力足以应付。因此这次课程设计带给我的最大收获并不在于算法，而在于代码模块化的编写。由于工程性较强，在使用较多STL之后代码量依旧较大，这时候，通过编写头文件，编写各个类的参数以及接口显得尤为重要。

不仅如此，在一些题目（如表达式计算、Huffman编码等），存在许许多多的特殊边界情况需要考虑，考验我的思维的严密性以及代码的鲁棒性。因此在这方面，我也收获颇多。

本次课设的不足之处在于，由于个人时间安排问题，使用一些UI界面（如QT界面）来支持一些UI操作的计划不得不废止，故在所有的测试结果中依旧使用命令行来呈现。因此美观性不足。