**北京邮电大学软件学院**

**2022-2023学年第一学期实验报告**

**课程名称：** 算法分析与设计

**项目名称：** 实验项目三——分支限界法

**项目完成人：**

**姓名：** 王宇涵 **学号：** 2020211730

**指导教师：** 李朝晖

**日 期： 2023年 4 月 20 日**

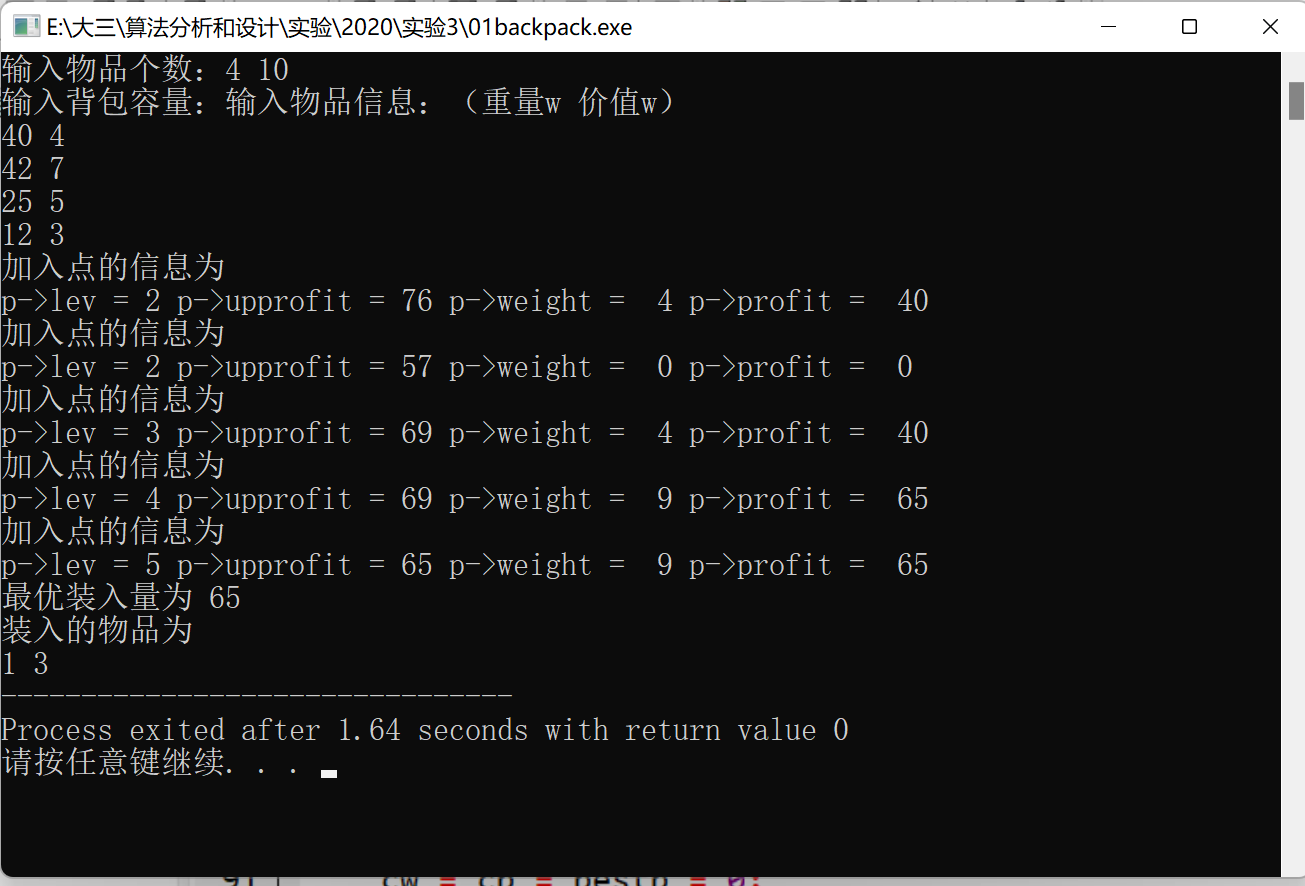
1. **实验目的**
   1. 深刻理解并掌握分支限界法的设计思想,比较与分支限界法的不同之处。
   2. 提高应用分支限界法设计算法的技能。
   3. 理解这样一个观点：好的限界函数不仅计算简单，还要保证最优解在搜索空间中，更重要的是能在搜索的早期对超出目标函数界的结点进行丢弃，减少搜索空间，从而尽快找到问题的最优解。
2. **实验内容**
3. (必做)应用分支限界法解决0-1背包问题；√
4. (必做)应用分支限界法解决旅行商售货员问题；√
5. (必做)对所设计的算法进行时间复杂性分析；√
6. (选作)实现八数码问题; √
7. **实验环境**

Windows 10 Dev-C++

1. **实验结果**

**第1题**

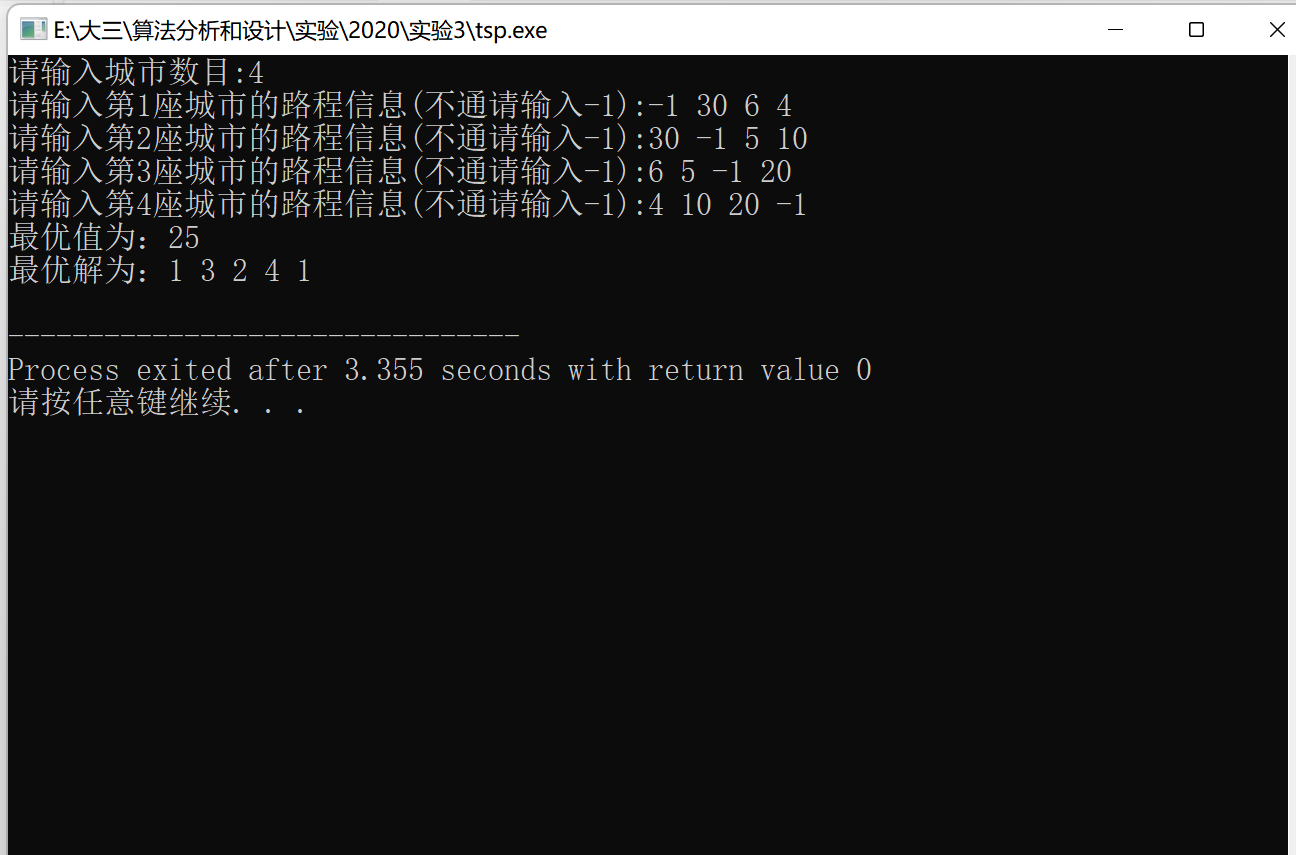
测试1：

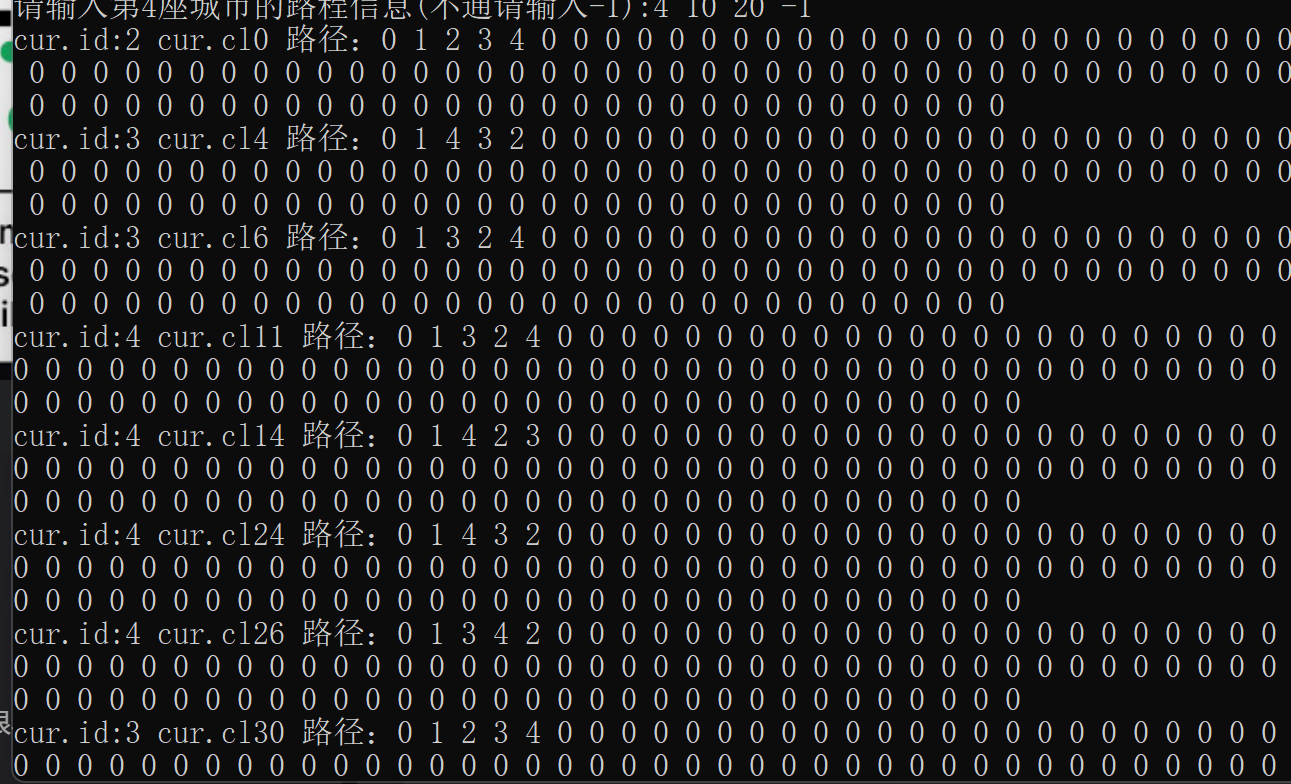


测试2



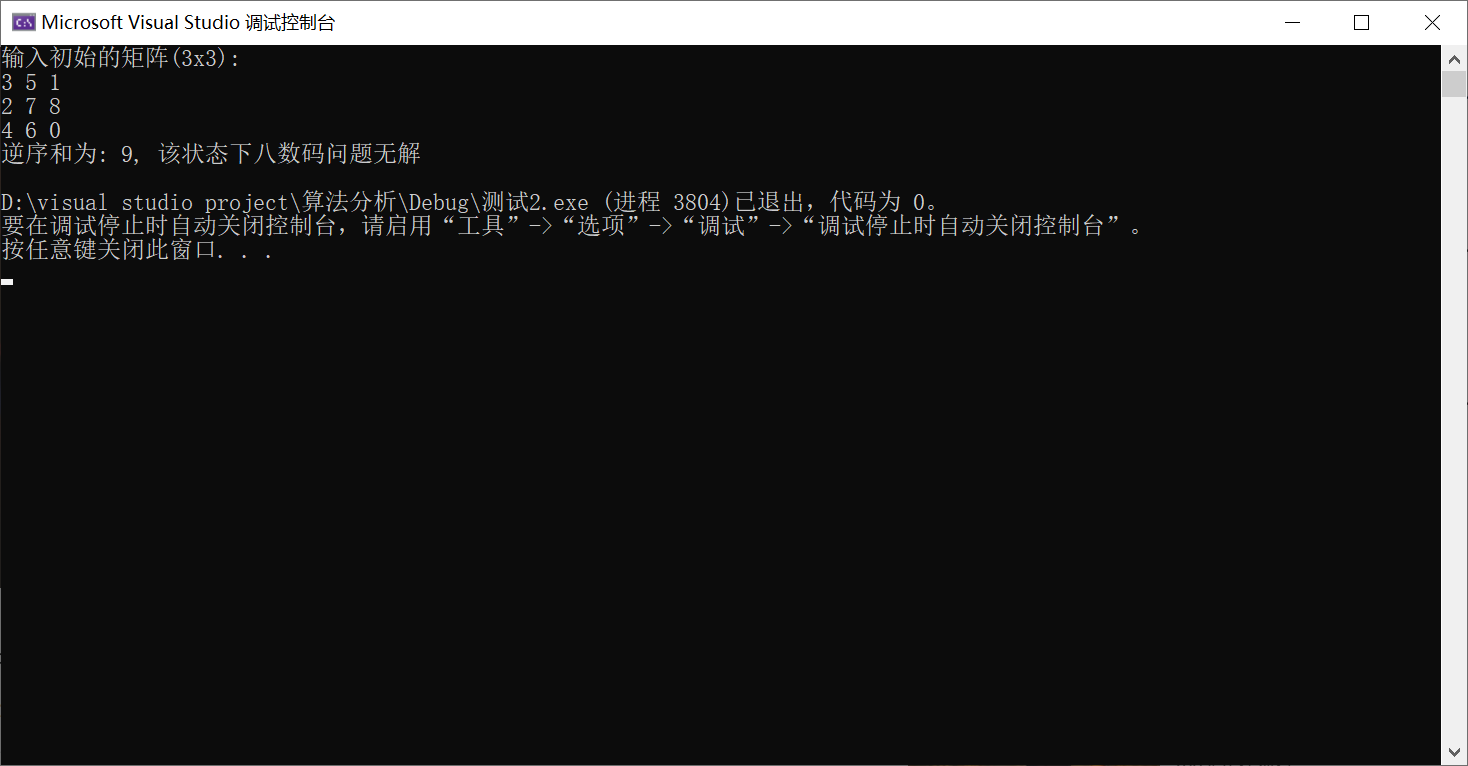
**第2题**

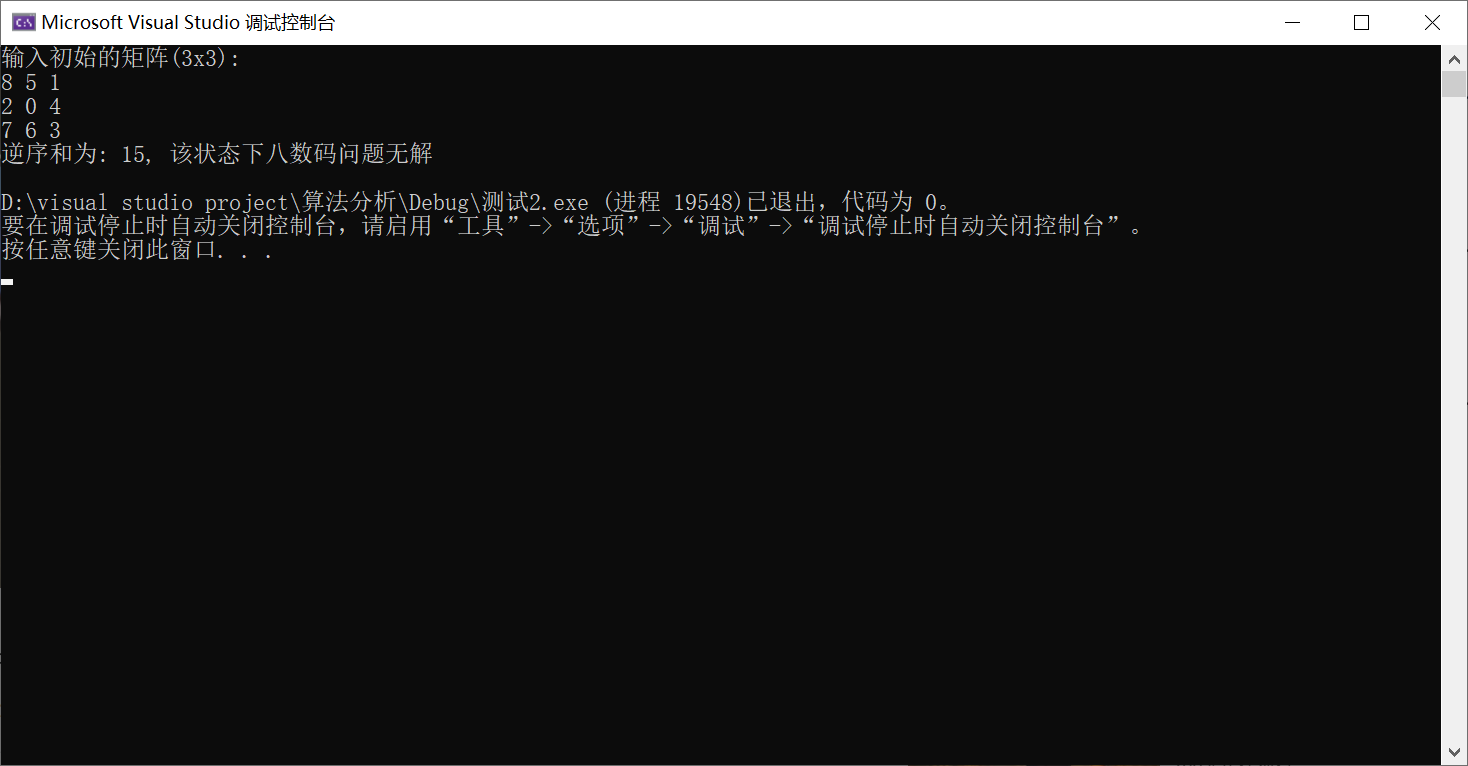




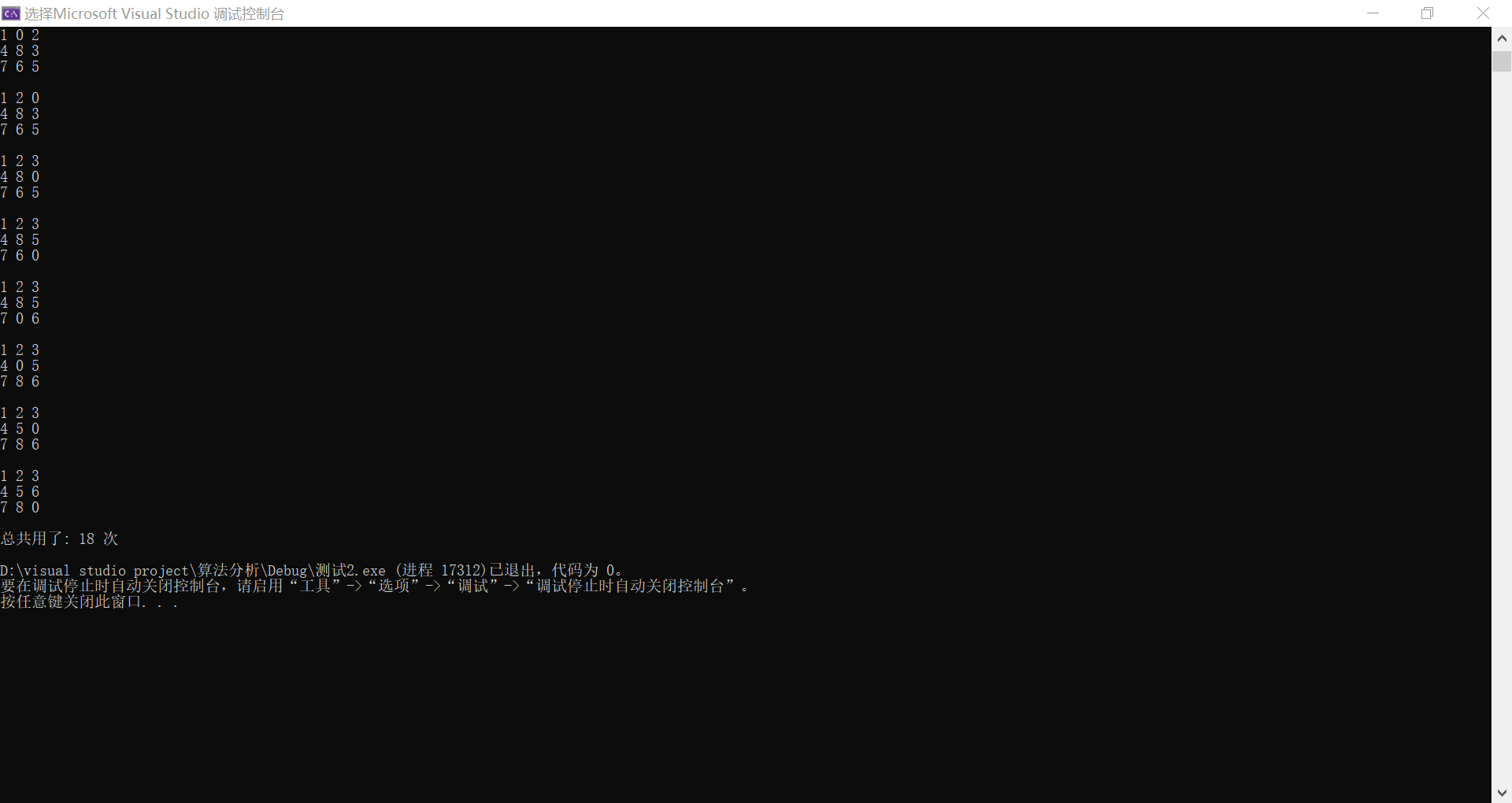
**第3题**

剪枝1：

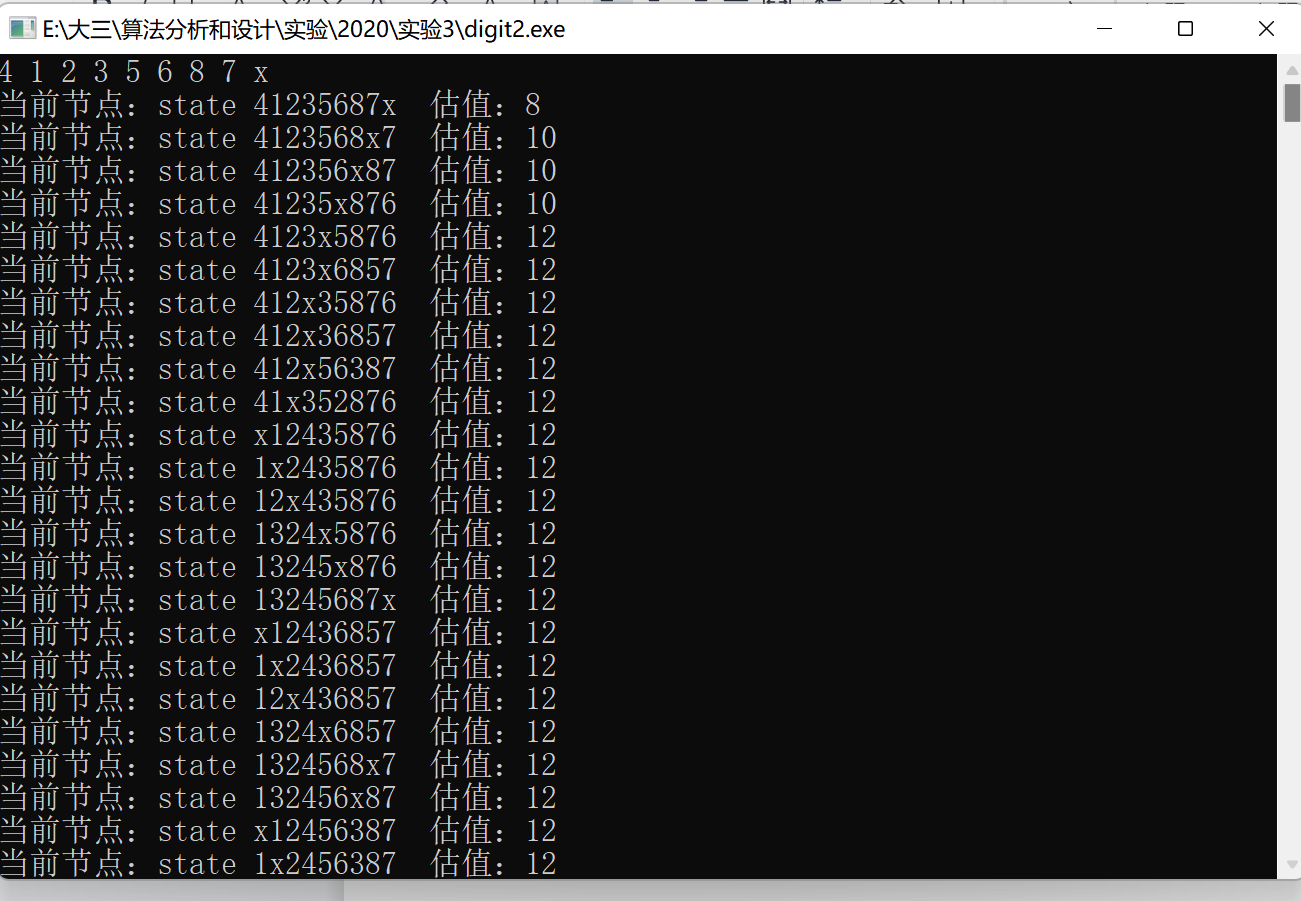


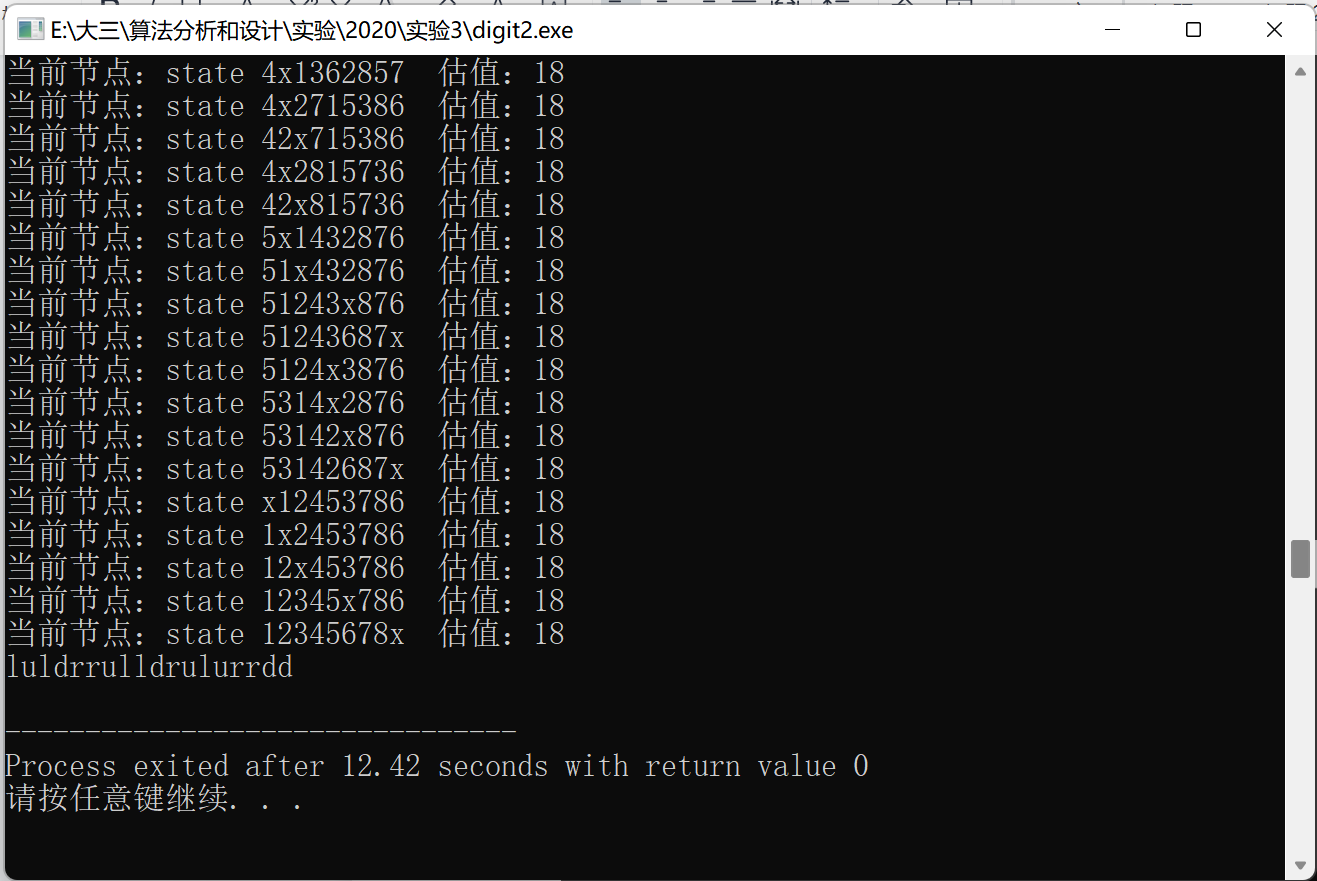


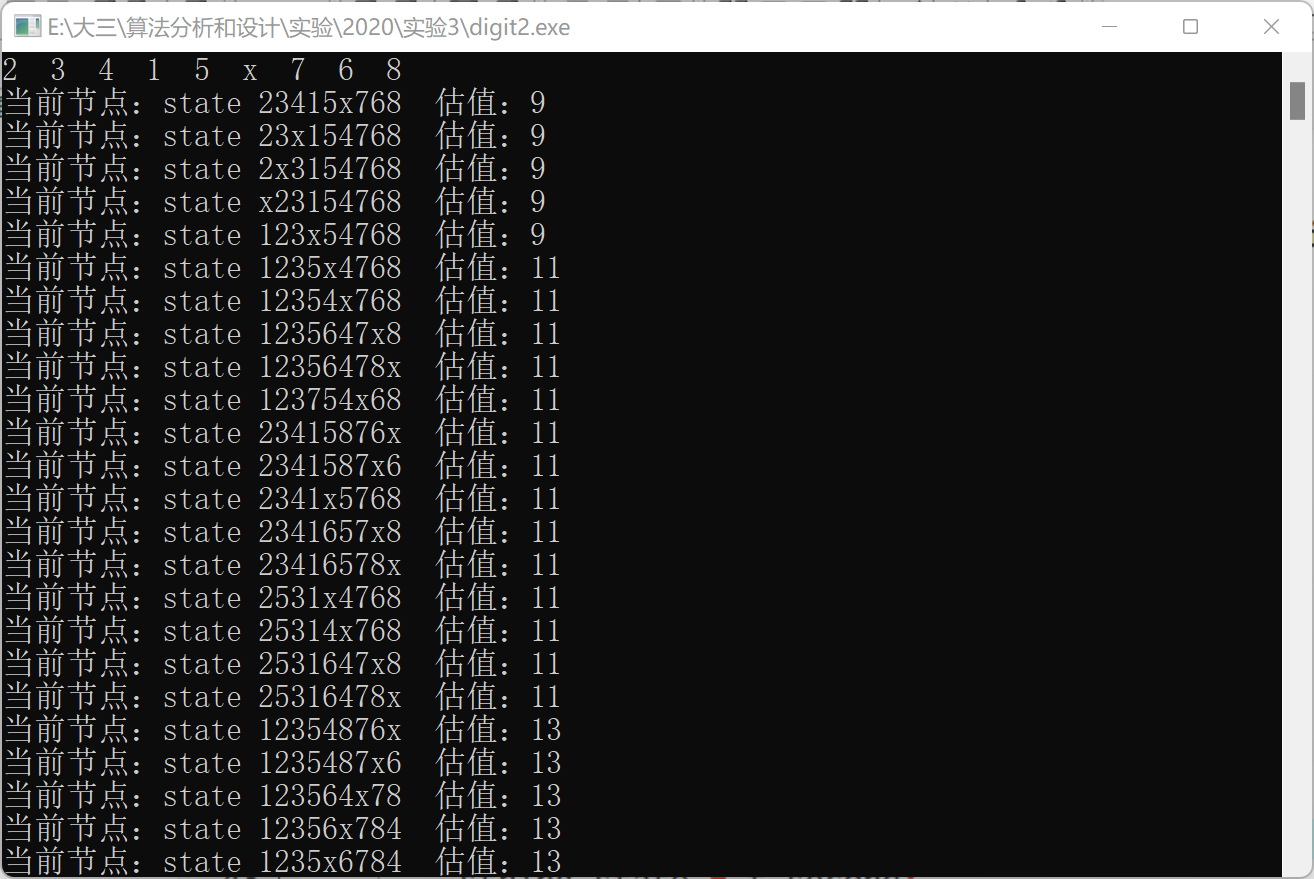


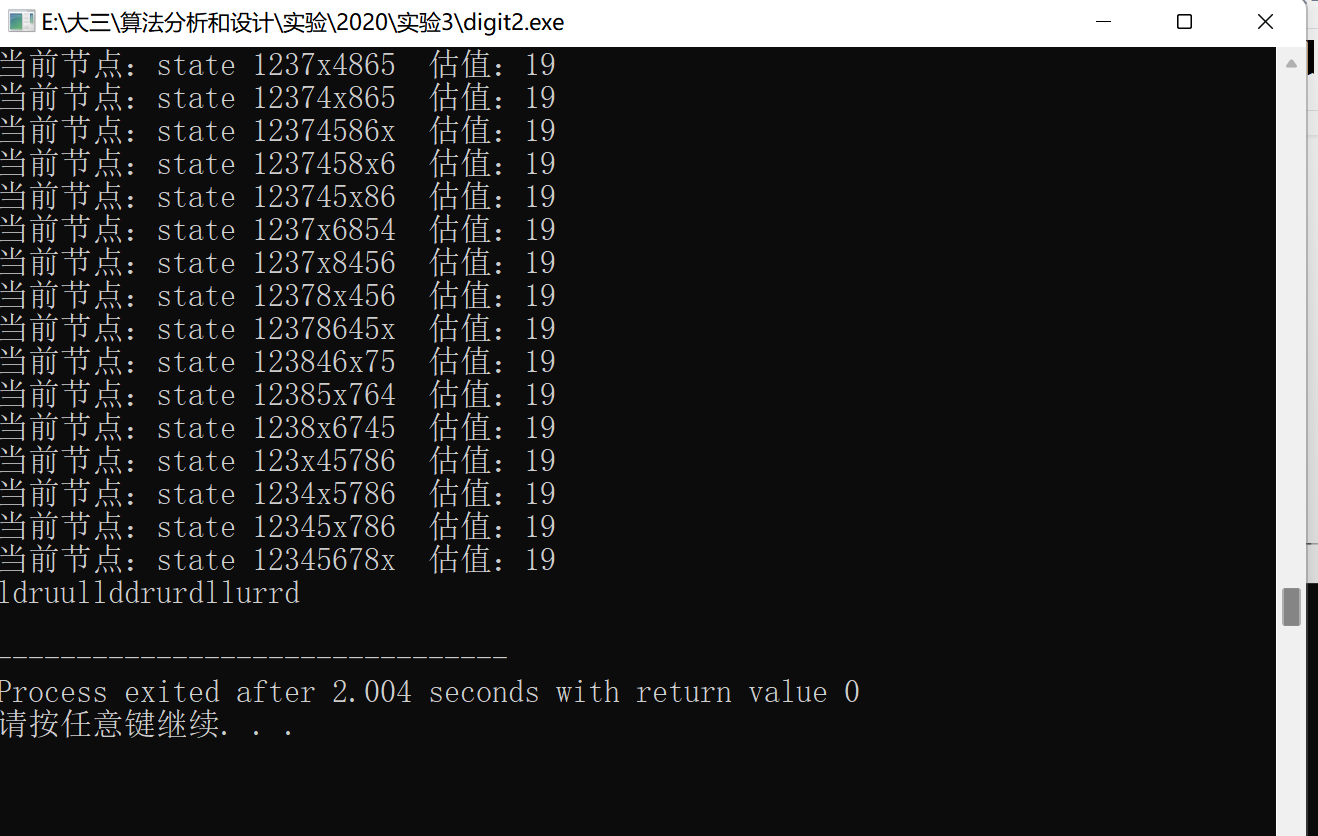


剪枝2：









完整

2 3 4 1 5 x 7 6 8

当前节点：state 23415x768 估值：9

当前节点：state 23x154768 估值：9

当前节点：state 2x3154768 估值：9

当前节点：state x23154768 估值：9

当前节点：state 123x54768 估值：9

当前节点：state 1235x4768 估值：11

当前节点：state 12354x768 估值：11

当前节点：state 1235647x8 估值：11

当前节点：state 12356478x 估值：11

当前节点：state 123754x68 估值：11

当前节点：state 23415876x 估值：11

当前节点：state 2341587x6 估值：11

当前节点：state 2341x5768 估值：11

当前节点：state 2341657x8 估值：11

当前节点：state 23416578x 估值：11

当前节点：state 2531x4768 估值：11

当前节点：state 25314x768 估值：11

当前节点：state 2531647x8 估值：11

当前节点：state 25316478x 估值：11

当前节点：state 12354876x 估值：13

当前节点：state 1235487x6 估值：13

当前节点：state 123564x78 估值：13

当前节点：state 12356x784 估值：13

当前节点：state 1235x6784 估值：13

当前节点：state 1237546x8 估值：13

当前节点：state 12375468x 估值：13

当前节点：state 123x56784 估值：13

当前节点：state 12x543768 估值：13

当前节点：state 1x3524768 估值：13

当前节点：state 234158x76 估值：13

当前节点：state 234165x78 估值：13

当前节点：state 23416x785 估值：13

当前节点：state 2341x6785 估值：13

当前节点：state 2341x8756 估值：13

当前节点：state 23418x756 估值：13

当前节点：state 23418675x 估值：13

当前节点：state 234x15768 估值：13

当前节点：state 23x164785 估值：13

当前节点：state 23x184756 估值：13

当前节点：state 25314876x 估值：13

当前节点：state 2531487x6 估值：13

当前节点：state 253164x78 估值：13

当前节点：state 25316x784 估值：13

当前节点：state 2531x6784 估值：13

当前节点：state 253x14768 估值：13

当前节点：state 25x143768 估值：13

当前节点：state 2x3156784 估值：13

当前节点：state 2x3164785 估值：13

当前节点：state 2x3184756 估值：13

当前节点：state 2x4135768 估值：13

当前节点：state 24x135768 估值：13

当前节点：state x23156784 估值：13

当前节点：state x23164785 估值：13

当前节点：state 123x64785 估值：13

当前节点：state x23184756 估值：13

当前节点：state 123x84756 估值：13

当前节点：state x24135768 估值：13

当前节点：state 124x35768 估值：13

当前节点：state 123548x76 估值：15

当前节点：state 1235867x4 估值：15

当前节点：state 12358674x 估值：15

当前节点：state 1235x8746 估值：15

当前节点：state 12358x746 估值：15

当前节点：state 1236x4785 估值：15

当前节点：state 12364x785 估值：15

当前节点：state 12364578x 估值：15

当前节点：state 123756x84 估值：15

当前节点：state 12375x684 估值：15

当前节点：state 123764x85 估值：15

当前节点：state 123784x56 估值：15

当前节点：state 1237x4658 估值：15

当前节点：state 12374x658 估值：15

当前节点：state 1238x4756 估值：15

当前节点：state 12384x756 估值：15

当前节点：state 12384675x 估值：15

当前节点：state 1238547x6 估值：15

当前节点：state 123x58746 估值：15

当前节点：state 123x64578 估值：15

当前节点：state 1243x5768 估值：15

当前节点：state 12435x768 估值：15

当前节点：state 1243657x8 估值：15

当前节点：state 12436578x 估值：15

当前节点：state 124735x68 估值：15

当前节点：state 12x354768 估值：15

当前节点：state 12x563784 估值：15

当前节点：state 13x524768 估值：15

当前节点：state 1x2543768 估值：15

当前节点：state 1x3526784 估值：15

当前节点：state 2341867x5 估值：15

当前节点：state 234715x68 估值：15

当前节点：state 234x16785 估值：15

当前节点：state 234x18756 估值：15

当前节点：state 234x58176 估值：15

当前节点：state 234x65178 估值：15

当前节点：state 23x156784 估值：15

当前节点：state 24513x768 估值：15

当前节点：state 2451x3768 估值：15

当前节点：state 2451637x8 估值：15

当前节点：state 24516378x 估值：15

当前节点：state 253148x76 估值：15

当前节点：state 2531867x4 估值：15

当前节点：state 25318674x 估值：15

当前节点：state 2531x8746 估值：15

当前节点：state 25318x746 估值：15

当前节点：state 253714x68 估值：15

当前节点：state 253x16784 估值：15

当前节点：state 253x64178 估值：15

当前节点：state 25x163784 估值：15

当前节点：state 2631x4785 估值：15

当前节点：state 26314x785 估值：15

当前节点：state 26314578x 估值：15

当前节点：state 2831x4756 估值：15

当前节点：state 28314x756 估值：15

当前节点：state 28314675x 估值：15

当前节点：state 2831547x6 估值：15

当前节点：state 2x3158746 估值：15

当前节点：state 2x4136785 估值：15

当前节点：state 24x136785 估值：15

当前节点：state 2x4138756 估值：15

当前节点：state 24x138756 估值：15

当前节点：state 2x5143768 估值：15

当前节点：state x13524768 估值：15

当前节点：state x23156784 估值：15

当前节点：state x23158746 估值：15

当前节点：state x24136785 估值：15

当前节点：state 124x36785 估值：15

当前节点：state x24138756 估值：15

当前节点：state 124x38756 估值：15

当前节点：state x25143768 估值：15

当前节点：state 125x43768 估值：15

当前节点：state 1254x3768 估值：15

当前节点：state 1254637x8 估值：15

当前节点：state 12546378x 估值：15

当前节点：state x34215768 估值：15

当前节点：state x53214768 估值：15

当前节点：state 123586x74 估值：17

当前节点：state 12358x746 估值：17

当前节点：state 1236457x8 估值：17

当前节点：state 1236847x5 估值：17

当前节点：state 12368475x 估值：17

当前节点：state 1236x4578 估值：17

当前节点：state 12364x578 估值：17

当前节点：state 12374865x 估值：17

当前节点：state 1237568x4 估值：17

当前节点：state 12375684x 估值：17

当前节点：state 123758x46 估值：17

当前节点：state 1237584x6 估值：17

当前节点：state 1237648x5 估值：17

当前节点：state 12376485x 估值：17

当前节点：state 1237845x6 估值：17

当前节点：state 1237x4586 估值：17

当前节点：state 12374x586 估值：17

当前节点：state 12374658x 估值：17

当前节点：state 1237x5684 估值：17

当前节点：state 1238467x5 估值：17

当前节点：state 12385476x 估值：17

当前节点：state 123854x76 估值：17

当前节点：state 123x48576 估值：17

当前节点：state 1234x8576 估值：17

当前节点：state 12348x576 估值：17

当前节点：state 12348657x 估值：17

当前节点：state 123x54876 估值：17

当前节点：state 123x74658 估值：17

当前节点：state 123674x58 估值：17

当前节点：state 12435876x 估值：17

当前节点：state 1243587x6 估值：17

当前节点：state 124365x78 估值：17

当前节点：state 12436x785 估值：17

当前节点：state 1243x6785 估值：17

当前节点：state 1243x8756 估值：17

当前节点：state 12438x756 估值：17

当前节点：state 12438675x 估值：17

当前节点：state 1247356x8 估值：17

当前节点：state 12473568x 估值：17

当前节点：state 124736x85 估值：17

当前节点：state 124738x56 估值：17

当前节点：state 12543x768 估值：17

当前节点：state 125463x78 估值：17

当前节点：state 12546x783 估值：17

当前节点：state 1254x6783 估值：17

当前节点：state 125743x68 估值：17

当前节点：state 12x364785 估值：17

当前节点：state 12x384756 估值：17

当前节点：state 12x435768 估值：17

当前节点：state 12x465783 估值：17

当前节点：state 12x583746 估值：17

当前节点：state 12x643785 估值：17

当前节点：state 12x743658 估值：17

当前节点：state 12x753684 估值：17

当前节点：state 12x843756 估值：17

当前节点：state 13452x768 估值：17

当前节点：state 13x526784 估值：17

当前节点：state 1425x3768 估值：17

当前节点：state 1425637x8 估值：17

当前节点：state 14256378x 估值：17

当前节点：state 142x53768 估值：17

当前节点：state 1x2354768 估值：17

当前节点：state 1x2563784 估值：17

当前节点：state 1x3528746 估值：17

当前节点：state 1x3624785 估值：17

当前节点：state 1x3724658 估值：17

当前节点：state 1x3824756 估值：17

当前节点：state 1x4325768 估值：17

当前节点：state 14x325768 估值：17

当前节点：state 1x5423768 估值：17

当前节点：state 15x423768 估值：17

当前节点：state 15342x768 估值：17

当前节点：state 234186x75 估值：17

当前节点：state 2345x8176 估值：17

当前节点：state 23458x176 估值：17

当前节点：state 23458617x 估值：17

当前节点：state 2346x5178 估值：17

当前节点：state 23465x178 估值：17

当前节点：state 2347156x8 估值：17

当前节点：state 23471568x 估值：17

当前节点：state 234716x85 估值：17

当前节点：state 234718x56 估值：17

当前节点：state 23615x784 估值：17

当前节点：state 23615478x 估值：17

当前节点：state 23x158746 估值：17

当前节点：state 23x584176 估值：17

当前节点：state 23x654178 估值：17

当前节点：state 24513876x 估值：17

当前节点：state 2451387x6 估值：17

当前节点：state 245163x78 估值：17

当前节点：state 24516x783 估值：17

当前节点：state 2451x6783 估值：17

当前节点：state 245x13768 估值：17

当前节点：state 24613x785 估值：17

当前节点：state 24613578x 估值：17

当前节点：state 2461x3785 估值：17

当前节点：state 24813x756 估值：17

当前节点：state 24813675x 估值：17

当前节点：state 2481x3756 估值：17

当前节点：state 2481537x6 估值：17

当前节点：state 24x165783 估值：17

当前节点：state 253186x74 估值：17

当前节点：state 25318x746 估值：17

当前节点：state 2536x4178 估值：17

当前节点：state 25364x178 估值：17

当前节点：state 2537146x8 估值：17

当前节点：state 25371468x 估值：17

当前节点：state 253716x84 估值：17

当前节点：state 253x18746 估值：17

当前节点：state 253x48176 估值：17

当前节点：state 2534x8176 估值：17

当前节点：state 25348x176 估值：17

当前节点：state 25348617x 估值：17

当前节点：state 25x183746 估值：17

当前节点：state 2631457x8 估值：17

当前节点：state 2631847x5 估值：17

当前节点：state 26318475x 估值：17

当前节点：state 263x14785 估值：17

当前节点：state 26x143785 估值：17

当前节点：state 2831467x5 估值：17

当前节点：state 28315476x 估值：17

当前节点：state 283154x76 估值：17

当前节点：state 283x14756 估值：17

当前节点：state 28x143756 估值：17

当前节点：state 2x3458176 估值：17

当前节点：state 2x3584176 估值：17

当前节点：state 2x3654178 估值：17

当前节点：state 2x5146783 估值：17

当前节点：state 25x146783 估值：17

当前节点：state 2x5163784 估值：17

当前节点：state 2x6143785 估值：17

当前节点：state 2x8143756 估值：17

当前节点：state 3x4215768 估值：17

当前节点：state 3142x5768 估值：17

当前节点：state 31425x768 估值：17

当前节点：state 3142657x8 估值：17

当前节点：state 31426578x 估值：17

当前节点：state 314x25768 估值：17

当前节点：state 31x254768 估值：17

当前节点：state 34x215768 估值：17

当前节点：state 513x24768 估值：17

当前节点：state 5x3214768 估值：17

当前节点：state 5132x4768 估值：17

当前节点：state 51324x768 估值：17

当前节点：state 5132647x8 估值：17

当前节点：state 51326478x 估值：17

当前节点：state x12543768 估值：17

当前节点：state x13526784 估值：17

当前节点：state x23158746 估值：17

当前节点：state x23164578 估值：17

当前节点：state x23458176 估值：17

当前节点：state x23584176 估值：17

当前节点：state x23654178 估值：17

当前节点：state x25146783 估值：17

当前节点：state 125x46783 估值：17

当前节点：state x25163784 估值：17

当前节点：state 125x63784 估值：17

当前节点：state x26143785 估值：17

当前节点：state 126x43785 估值：17

当前节点：state 1264x3785 估值：17

当前节点：state x28143756 估值：17

当前节点：state 128x43756 估值：17

当前节点：state 1284x3756 估值：17

当前节点：state 1284537x6 估值：17

当前节点：state x34216785 估值：17

当前节点：state x34218756 估值：17

当前节点：state x34258176 估值：17

当前节点：state x34265178 估值：17

当前节点：state x53216784 估值：17

当前节点：state x53264178 估值：17

当前节点：state 1234785x6 估值：19

当前节点：state 123478x56 估值：19

当前节点：state 1234865x7 估值：19

当前节点：state 123486x57 估值：19

当前节点：state 1234x6587 估值：19

当前节点：state 1235x4876 估值：19

当前节点：state 12354x876 估值：19

当前节点：state 12354687x 估值：19

当前节点：state 123645x78 估值：19

当前节点：state 12364857x 估值：19

当前节点：state 1236745x8 估值：19

当前节点：state 12367458x 估值：19

当前节点：state 123684x75 估值：19

当前节点：state 12368x754 估值：19

当前节点：state 1236x5748 估值：19

当前节点：state 12365x748 估值：19

当前节点：state 1237465x8 估值：19

当前节点：state 123746x58 估值：19

当前节点：state 1237486x5 估值：19

当前节点：state 123748x65 估值：19

当前节点：state 12375846x 估值：19

当前节点：state 12375x468 估值：19

当前节点：state 12375x846 估值：19

当前节点：state 12376x854 估值：19

当前节点：state 12378456x 估值：19

当前节点：state 1237856x4 估值：19

当前节点：state 12378564x 估值：19

当前节点：state 123785x64 估值：19

当前节点：state 1237x4865 估值：19

当前节点：state 12374x865 估值：19

当前节点：state 12374586x 估值：19

当前节点：state 1237458x6 估值：19

当前节点：state 123745x86 估值：19

当前节点：state 1237x6854 估值：19

当前节点：state 1237x8456 估值：19

当前节点：state 12378x456 估值：19

当前节点：state 12378645x 估值：19

当前节点：state 123846x75 估值：19

当前节点：state 12385x764 估值：19

当前节点：state 1238x6745 估值：19

当前节点：state 123x45786 估值：19

当前节点：state 1234x5786 估值：19

当前节点：state 12345x786 估值：19

当前节点：state 12345678x 估值：19

ldruullddrurdllurrd

1. **附录**

**项目1：0—1 背包问题**

【整体思路】

给定n种物品和一个背包。物品i的重量是wi，其价值为vi，背包的容量为c。

应如何选择装入背包的物品，使得装入背包中物品的总价值最大?

在选择装入背包的物品时，对每种物品i只有2种选择，即装入背包或不装入背包。不能将物品i装入背包多次，也不能只装入部分的物品i。

 首先，要对输入数据进行预处理，将各物品依其单位重量价值从大到小进行排列。

在实现时，由Bound计算当前结点处的上界。在解空间树的当前扩展结点处，仅当要进入右子树时才计算右子树的上界Bound，以判断是否将右子树剪。进入左子树时不需要计算上界，因为其上界与其父节点上界相同。

在优先队列分支限界法中，结点的优先级定义为：以结点的价值上界作为优先级（由bound函数计算出）

【问题分析】

算法首先根据基于可行结点相应的子树最大价值上界优先级，从堆中选择一个节点（根节点）作为当前可扩展结点。

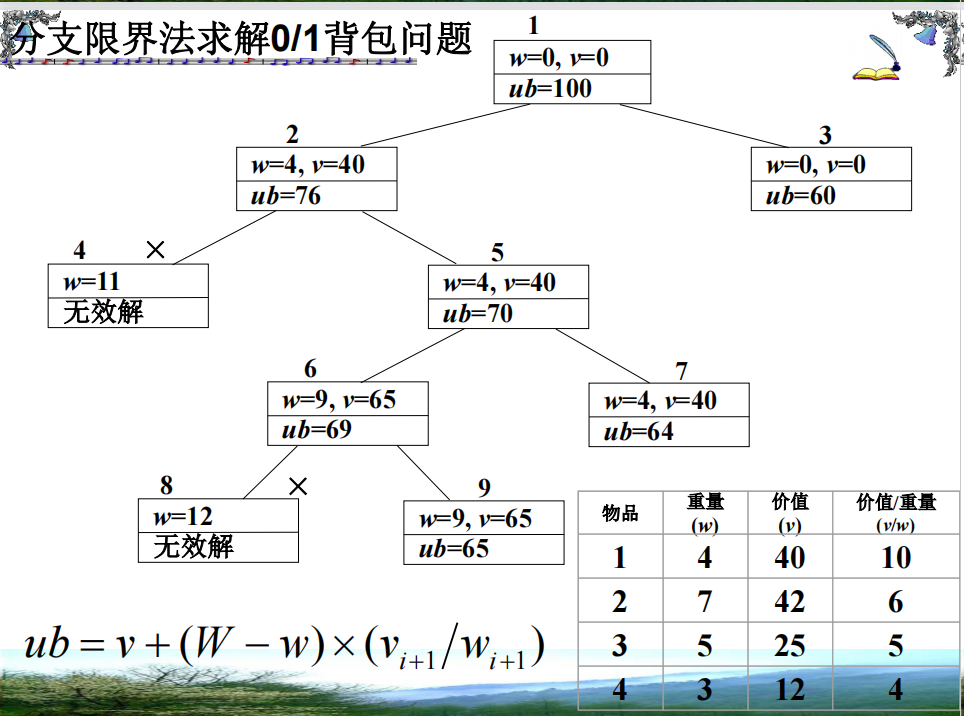
检查当前扩展结点的左儿子结点的可行性。

如果左儿子结点是可行结点，则将它加入到子集树和活结点优先队列中。

当前扩展结点的右儿子结点一定是可行结点，仅当右儿子结点满足上界函数约束时,才将它加入子集树和活结点优先队列。

当扩展到叶节点时，算法结束，叶子节点对应的解即为问题的最优值。

上界计算  
   先装入物品1，剩余的背包容量为6，只能装入物品2的6/7(即42\*(6/7)=36)。 即上界为40+6\*6=76



【算法设计】

1.定义类goods和qnode，分别表示物品和搜索节点。

2.输入物品信息，并按照单位重量价值从大到小排序。

3.计算上限函数Bound，用来估算当前节点可能达到的最大价值。Bound(i)表示处理到第i个物品时，剩余容量完全装满的背包所能获得的最大价值。

4.建立搜索队列，用来存储搜索过程中生成的节点，并按照节点的上限价值从大到小排序。

5.初始化搜索队列，并进入循环搜索过程：

（1）从队列中取出一个节点，计算当前节点的上限价值和下一层节点的上限价值；

（2）如果当前节点的价值已经超过当前最优解，则将该节点的子节点加入队列；

（3）如果当前节点的上限价值大于当前最优解，则将该节点的子节点加入队列。

6.当搜索队列为空时，搜索结束。回溯最优解，并输出结果。

整个算法的核心思想是利用上限函数Bound和分支定界策略，尽可能剪枝不必要的搜索路径，从而提高求解效率。

限界函数为：

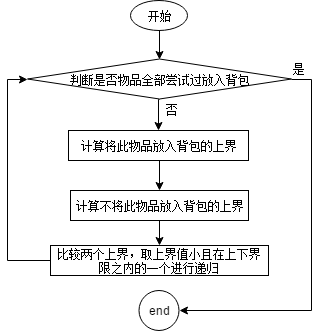


对于本题目来说：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 重量 | 价值 | 价值/重量 |
| 物体1 | 4 | 40 | 10 |
| 物体2 | 7 | 42 | 6 |
| 物体3 | 5 | 25 | 5 |
| 物体4 | 3 | 12 | 4 |

贪心法求得近似解为(1, 0, 0, 0)，获得的价值为40，由界限函数求得的上界为ub=W×(v1/w1 )=10×10=100。因此得到评估函数的界[40, 100]，这也作为了代码中的上下界。

设计图



【核心代码及解释】

void backpack()
  
{
  
 priority\_queue<qnode \*, vector<qnode \*>, cmp > q; // 大顶堆
  
 qnode \*pa = NULL;
  
 cw = cp = bestp = 0;
  
 int i = 1;
  
 int up = Bound(1); //Bound(i)函数计算的是i还未处理时候的上限值
  
 while(i != n + 1)
  
 {
  
 int wt = cw + obj[i].weight;
  
 if(wt <= c)
  
 {
  
 if(bestp < cp + obj[i].price)
  
 bestp = cp + obj[i].price;
  
 addnode(q, pa, up, cw + obj[i].weight, cp + obj[i].price, i, 1);
  
 }
  
 up = Bound(i + 1); //注意这里 up != up - obj[i].price而且 up >= up - obj[i].price
  
 if(up >= bestp) //注意这里必须是大于等于
  
 {
  
 addnode(q, pa, up, cw, cp, i, 0);
  
 }
  
 pa = q.top();
  
 q.pop();
  
 cw = pa->weight;
  
 cp = pa->profit;
  
 up = pa->upprofit;
  
 i = pa->lev;
  
 }
  
 for(int j = n; j > 0; --j)
  
 {
  
 bestx[obj[pa->lev - 1].id] = pa->lchild;
  
 pa = pa->parent;
  
 }
  
}

以下是代码的具体解释：

1. cw = cp = bestp = 0;

初始化背包当前重量cw、当前价值cp、最优价值bestp均为0。

1. int up = Bound(1);

计算处理第1个物品时的上限价值，即Bound(1)。

1. while(i != n + 1)

循环遍历每个物品，直到处理完所有物品。

1. int wt = cw + obj[i].weight;

计算加入第i个物品后的背包重量wt。

1. if(wt <= c)

如果加入第i个物品后的重量小于等于背包容量c，则执行以下操作：

* 如果加入第i个物品后的总价值cp + obj[i].price大于当前最优价值bestp，则将bestp更新为cp + obj[i].price。
* 添加一个新的结点到优先队列q中，表示选择第i个物品，即调用addnode函数。其中，pa表示当前结点的父结点，up表示当前处理的结点的上界，cw + obj[i].weight表示加入第i个物品后的背包重量，cp + obj[i].price表示加入第i个物品后的总价值，i表示当前处理到的物品序号，1表示选择第i个物品。

1. up = Bound(i + 1);

计算处理第i+1个物品时的上限价值，即Bound(i+1)，以便后面判断是否可以剪枝。

1. if(up >= bestp)

如果处理第i+1个物品时的上限价值up大于等于当前最优价值bestp，则执行以下操作：

* 添加一个新的结点到优先队列q中，表示不选择第i个物品，即调用addnode函数。其中，pa表示当前结点的父结点，up表示当前处理的结点的上界，cw表示不选择第i个物品时的背包重量，cp表示不选择第i个物品时的总价值，i表示当前处理到的物品序号，0表示不选择第i个物品。

1. pa = q.top(); q.pop();

从优先队列q中取出最优结点pa，并将其从优先队列中删除。

1. cw = pa->weight; cp = pa->profit; up = pa->upprofit; i = pa->lev;

更新当前处理的结点的背包重量、总价值、上界和物品序号。

1. for(int j = n; j > 0; --j)

从最后一个物品往前遍历，将每个物品的选择状态记录下来，即记录最优解中每个物品是否被选中，记录在bestx数组中，其中bestx[i]表示第i个物品是否被选中（1表示选中，0表示未选中）。这里使用pa指向最优解节点的父节点，从父节点一直往上遍历，直到根节点，记录每个节点的选中状态。由于pa最后指向的是最优解节点的父节点，因此从n开始往前遍历，直到第一个物品，保证bestx数组的下标与物品序号对应。

int Bound(int i)
  
{
  
 int tmp\_cleft = c - cw;
  
 int tmp\_cp = cp;
  
 while(tmp\_cleft >= obj[i].weight && i <= n)
  
 {
  
 tmp\_cleft -= obj[i].weight;
  
 tmp\_cp += obj[i].price;
  
 i++;
  
 }
  
 if(i <= n)
  
 {
  
 tmp\_cp += tmp\_cleft \* obj[i].d;
  
 }
  
 return tmp\_cp;
  
}

该函数用于计算在处理第i个物品之前，背包还剩余容量所能装载的物品的最大价值上界。

首先，定义一个临时变量tmp\_cleft表示背包剩余容量，初始值为c - cw，即当前背包剩余容量。

定义另一个临时变量tmp\_cp表示当前背包中已经装载的物品的总价值，初始值为cp，即当前背包已经装载的物品总价值。

然后，用循环遍历从第i个物品开始往后的每个物品，如果当前背包剩余容量可以装载该物品，就将该物品的价值加入tmp\_cp中，同时将tmp\_cleft减去该物品的重量，即tmp\_cleft -= obj[i].weight。

循环结束后，如果还有物品未处理完，就使用一种贪心策略来计算剩余容量所能装载的最大价值上界。即假设从第i+1个物品开始，将每个物品的单位价值按照从大到小排序，然后依次选取单位价值高的物品，直到背包装满为止。假设选到第j个物品时背包已经装满，那么剩余容量所能装载的物品价值为(c - cw) \* obj[j].d，其中obj[j].d表示第j个物品的单位价值。

最后返回tmp\_cp，即为剩余容量所能装载的最大价值上界。

void addnode(priority\_queue<qnode \*, vector<qnode \*>, cmp> &q, qnode \*E, int up, int wt, int curp, int i, int ch)
  
{
  
 qnode \*p = new qnode;
  
 p->parent = E;
  
 p->lchild = ch;
  
 p->weight = wt;
  
 p->upprofit = up;
  
 p->profit = curp;
  
 p->lev = i + 1;
  
 q.push(p);
  
 cout << "加入点的信息为 " << endl;
  
 cout << "p->lev = " << p->lev << " p->upprofit = " << p->upprofit << " p->weight = " << p->weight << " p->profit = " << p->profit << endl;
  
}

这个函数是用来往优先队列中添加结点的。优先队列是一种能够自动按照元素的优先级排序的数据结构。这里使用了一个自定义的结构体 qnode，来表示搜索树的结点。其中 upprofit 表示当前结点的上界价值，weight 表示当前结点的背包重量，profit 表示当前结点的总价值，lev 表示当前结点对应的物品序号（即处理到第几个物品了）。

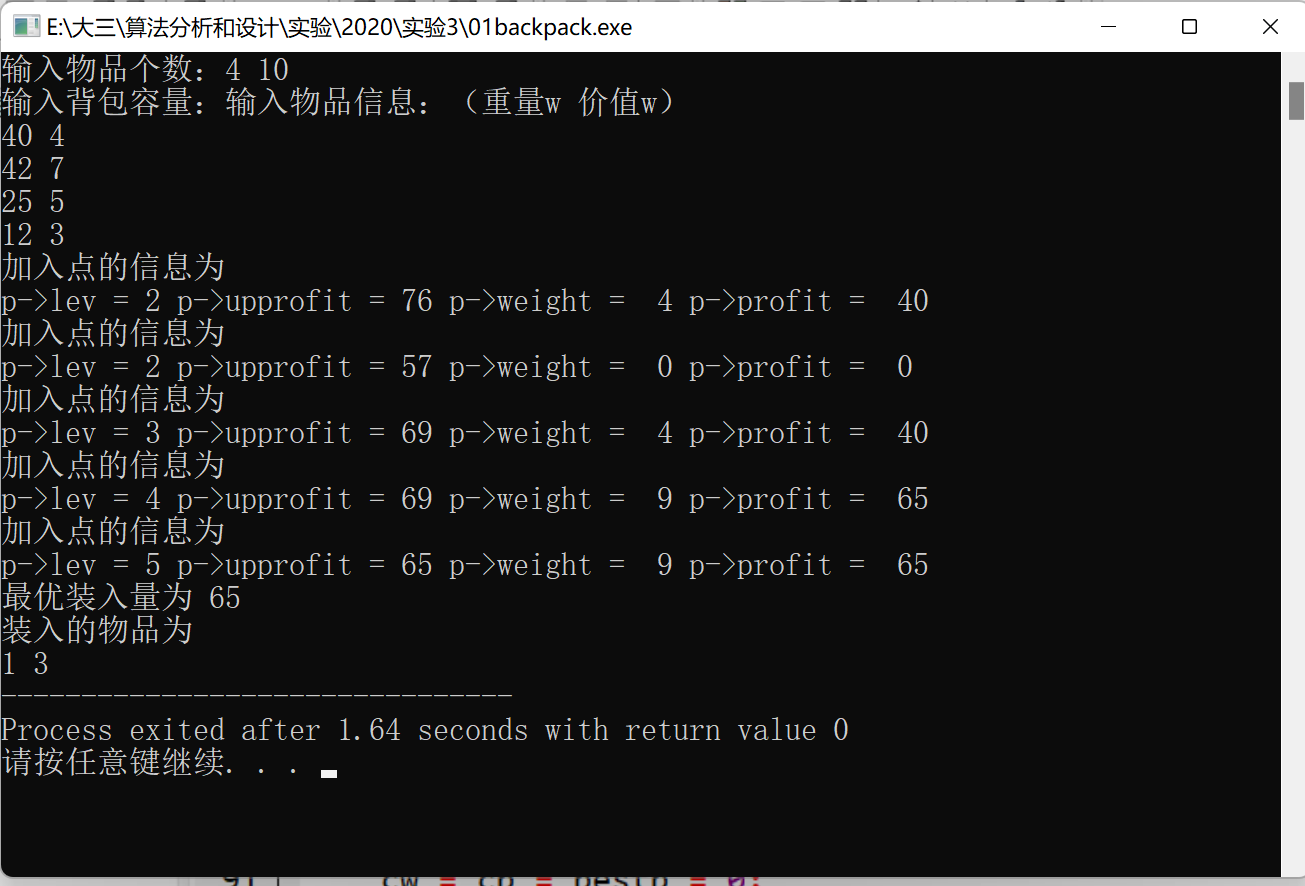
这个函数的输入参数包括当前优先队列 q、当前结点的父结点 E、当前结点的上界价值 up、当前结点的背包重量 wt、当前结点的总价值 curp、当前结点对应的物品序号 i，以及当前结点的左子结点是否被选中（0 表示未选中，1 表示选中）。

这个函数首先创建一个新的结点 p，并将其各项属性初始化为输入参数。然后将其加入到优先队列中，使其自动按照上界价值排序。

最后，这个函数还会输出刚刚加入的结点的信息，方便调试。

【实验结果】

测试1：



测试2



时间复杂度分析：O(2^nlogn)

**项目2：旅行商售货员问题的分支限界算法**

【整体思路】

TSP问题是指旅行家要旅行n个城市，要求各个城市经历且仅经历一次然后回到出发城市，并要求所走的路程最短，若要采用分支界限的方法求解，因为在每条路径上，每个城市都有两条邻接边，一条是进入这个城市的，另一条是离开这个城市的，所以可以找出与每个城市连接的两条最小的路径来代表进入和离开的路程，在每次选择下一个地点后，将其路程值替换掉原来的值中的较大的一个，以此来设计界限。

【问题分析】

解旅行售货员问题的优先队列式分支限界法用优先队列存储活结点表。

活结点m在优先队列中的优先级定义为：活结点m对应的子树费用下界lcost。

lcost=cc+rcost，其中，cc为当前结点费用，rcost为当前顶点最小出边费用加上剩余所有顶点的最小出边费用和。

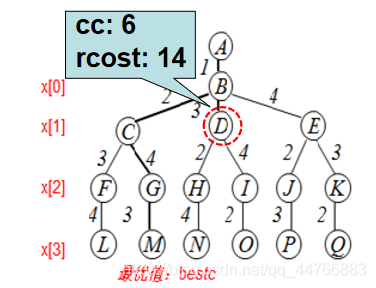
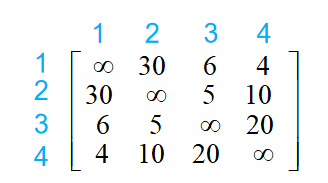
优先队列中优先级最大的活结点成为下一个扩展结点。

排列树中叶结点所相应的载重量与其优先级（下界值）相同，即：叶结点所相应的回路的费用(bestc)等于子树费用下界lcost的值。

与子集树的讨论相似，实现对排列树搜索的优先队列式分支限界法也可以有两种不同的实现方式：(旅行售货员问题采用第一种实现方式。)

用优先队列来存储活结点。优先队列中每个活结点都存储从根到该活结点的相应路径。

用优先队列来存储活结点，并同时存储当前已构造出的部分排列树。优先队列中的活结点不必存储从根到该活结点的相应路径，该路径必要时从存储的部分排列树中获得。



令Minout[i]代表顶点i的最小出边费用。图中：

   Minout[1]=4                 Minout[2]=5

   Minout[3]=5                 Minout[4]=4

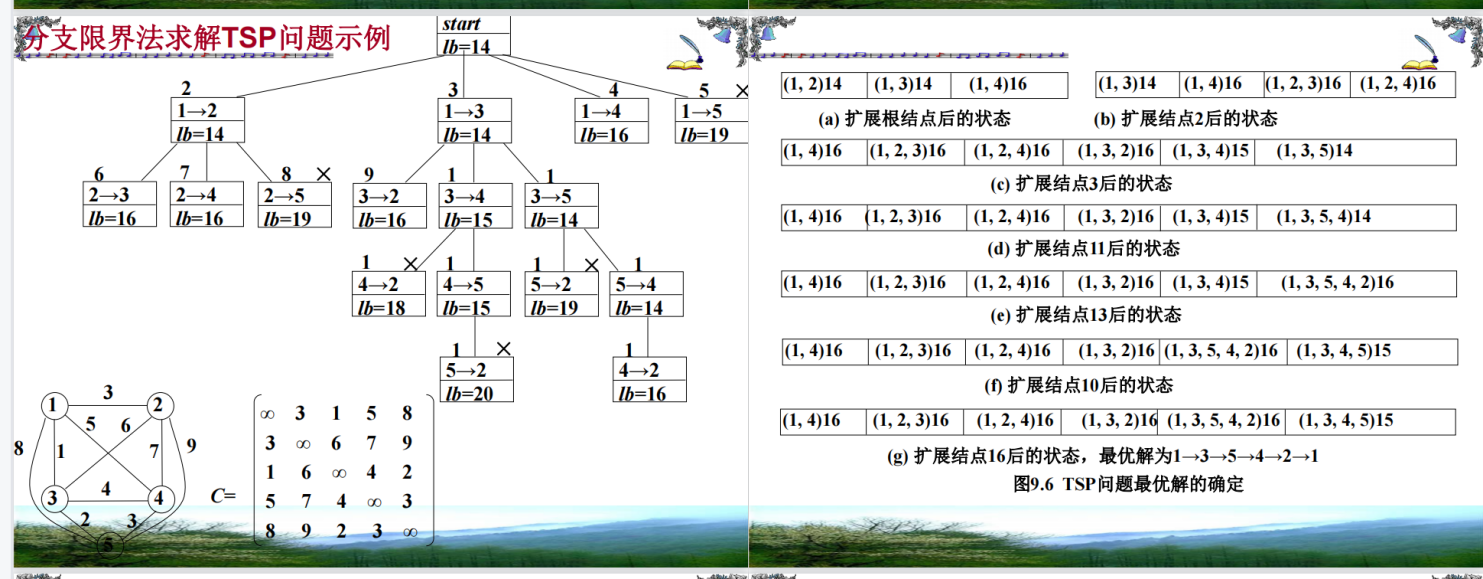
如，在扩展顶点D处：

 rcost=Minout[3]+Minout[2]+Minout[4]=14

  其中:Minout[3]:当前顶点最小出边

     Minout[2],Minout[4]:剩余所有顶点最小出边

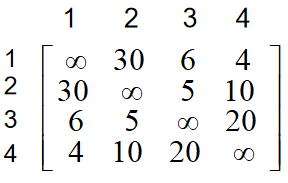
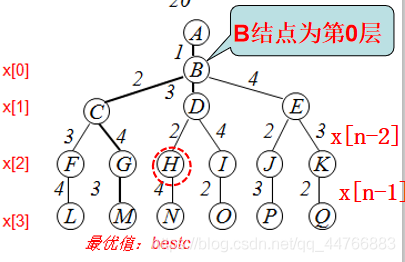
【算法设计】



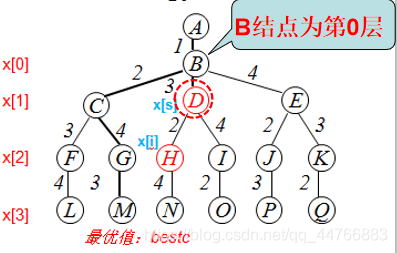
基于优先队列式分支限界的旅行售货员问题求解算法，采用限界函数lcost ，作为优先级，不断调整搜索方向，选择最有可能取得最优解的子树优先搜索；同时，根据限界函数lcost进行剪枝，剪掉不包含最优解的分支。

1. 算法的while循环完成对排列树内部结点的扩展。对于当前扩展结点，算法分两种情况进行处理。  
   令s表示结点在排列树中的层次（节点B为第0层）。考虑排列树层次s=n-2的情形，此时当前扩展结点是排列树中某个叶结点的父结点。  
    (1).检测图G是否存在一条从顶点x[n-2]到顶点x[n-1]的边和一条从顶点x[n-1]到顶点1的边。  
    (2). 如果这两条边都存在，则找到一条旅行员售货回路。  
    (3).此时，算法还需要判断这条回路的费用是否优于已找到的当前最优回路的费用bestc。  
    (4). 如果是，则必须更新当前最优值bestc和当前最优解bestx。

if(cc + a[x[n-2]][x[n-1]] + a[x[n-1]][1] < bestc )
  
{ // 找到更优的旅行路径
  
 for (int j = 0; j <= n-1; j++) bestx[j] = x[j];
  
 bestc = cc + a[x[n-2]][x[n-1]] + a[x[n-1]][1];
  
}

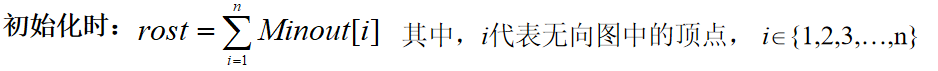


1. 令x记录当前解，s表示结点在排列树中的层次（结点B为第0层）。当扩展结点所处的层次s<n-2时，算法依次产生当前扩展结点的所有儿子结点。  
    (1). 由于当前扩展结点所相应的路径是x[0: s]，该扩展结点可行儿子结点是从剩余顶点x[s+1: n-1]中选取的顶点x[i]，且(x[s]，x[i])是所给有向图G中的一条边。  
    (2).对于当前扩展结点的每一个可行儿子结点，计算出其前缀(x[0:s]，x[i])的费用cc和相应的下界lcost。  
    (3).当lcost<bestc时，将这个可行儿子结点插入到活结点优先队列中,否则,将剪去以该儿子结点为根的子树。

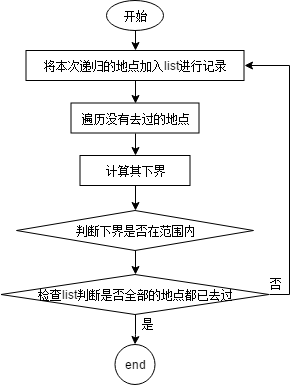


1. 令x记录当前解，s表示结点在排列树中的层次（结点B为第0层）。当扩展结点所处的层次s<n-2时，算法依次产生当前扩展结点的所有儿子结点。  
    (1). 由于当前扩展结点所相应的路径是x[0: s]，该扩展结点可行儿子结点是从剩余顶点x[s+1: n-1]中选取的顶点x[i]，且(x[s]，x[i])是所给有向图G中的一条边。  
    (2).对于当前扩展结点的每一个可行儿子结点，计算出其前缀(x[0:s]，x[i])的费用cc和相应的下界lcost。  
    (3).当lcost<bestc时，将这个可行儿子结点插入到活结点优先队列中,否则,将剪去以该儿子结点为根的子树。

下界：lcost=cc+rcost  
Minout[i]：顶点i的最小费用出边  
初始化时：cc=0, s=0, x[0]=1, x[1: n-1]=(2,3,…, n)



初始化时:bestc为较大数值



【核心代码及解释】

priority\_queue<node,vector<node>,cmp> q;

创建一个最小堆，每次将距离最小的节点放在队首。

node next\_node(0,2);

初始化一个节点next\_node，起点为1，下一个要遍历的城市为2。

for(int i=1; i<=n; ++i)
  
next\_node.x[i]=i;

将next\_node节点的路径赋值为1~n的顺序。

q.push(next\_node);  
将初始节点next\_node推入最小堆q中。

cur=q.top();
  
q.pop();
  
t=cur.id;

从最小堆q中取出距离最小的节点，将其赋值给当前节点cur，同时将其从最小堆中删除，取出当前节点cur的下一个要遍历的城市t。

if(t==n)
  
{
  
if(m[cur.x[t-1]][cur.x[t]]!=INF&&m[cur.x[t]][1]!=INF)
  
{
  
if(cur.cl+m[cur.x[t-1]][cur.x[t]]+m[cur.x[t]][1]<bestl)
  
{
  
bestl=cur.cl+m[cur.x[t-1]][cur.x[t]]+m[cur.x[t]][1];
  
for(int i=1; i<=n; ++i)
  
bestx[i]=cur.x[i];
  
}
  
}
  
continue;
  
}

如果当前节点已经遍历了所有的城市，则判断是否满足约束条件，即从最后一个城市到1之间是否存在路径，并且路径的长度是否小于当前的最优路径长度bestl。如果满足条件，则更新bestl和最优路径bestx，然后继续循环。

if(cur.cl>=bestl)
  
{
  
continue;
  
}

如果当前节点的距离cur.cl已经大于或等于当前的最优路径长度bestl，则直接跳过该节点的遍历。

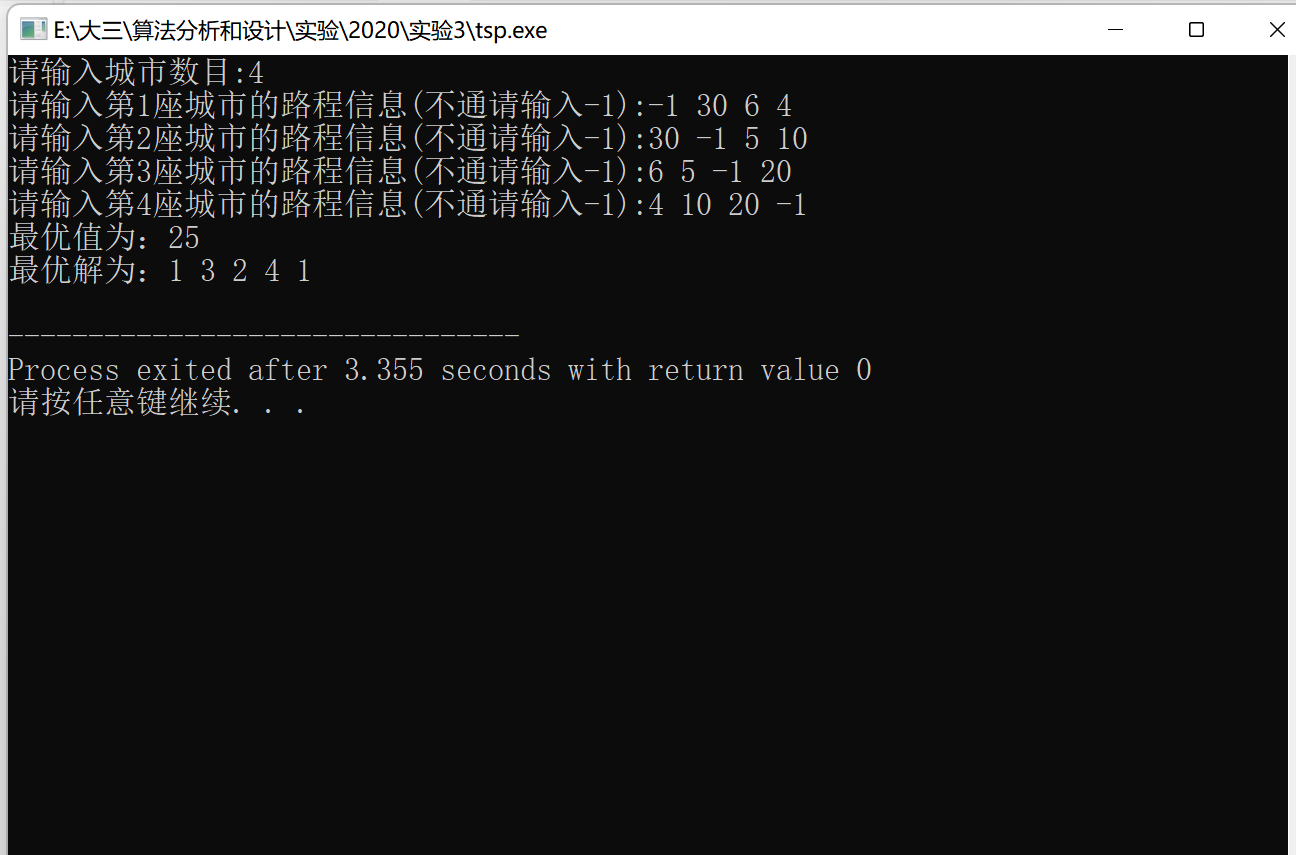
next\_node = node(cur.cl + m[cur.x[t - 1]][cur.x[j]], t + 1);
  
for (int k = 1; k <= n; ++k)
  
 next\_node.x[k] = cur.x[k];
  
swap(next\_node.x[t], next\_node.x[j]);
  
q.push(next\_node);

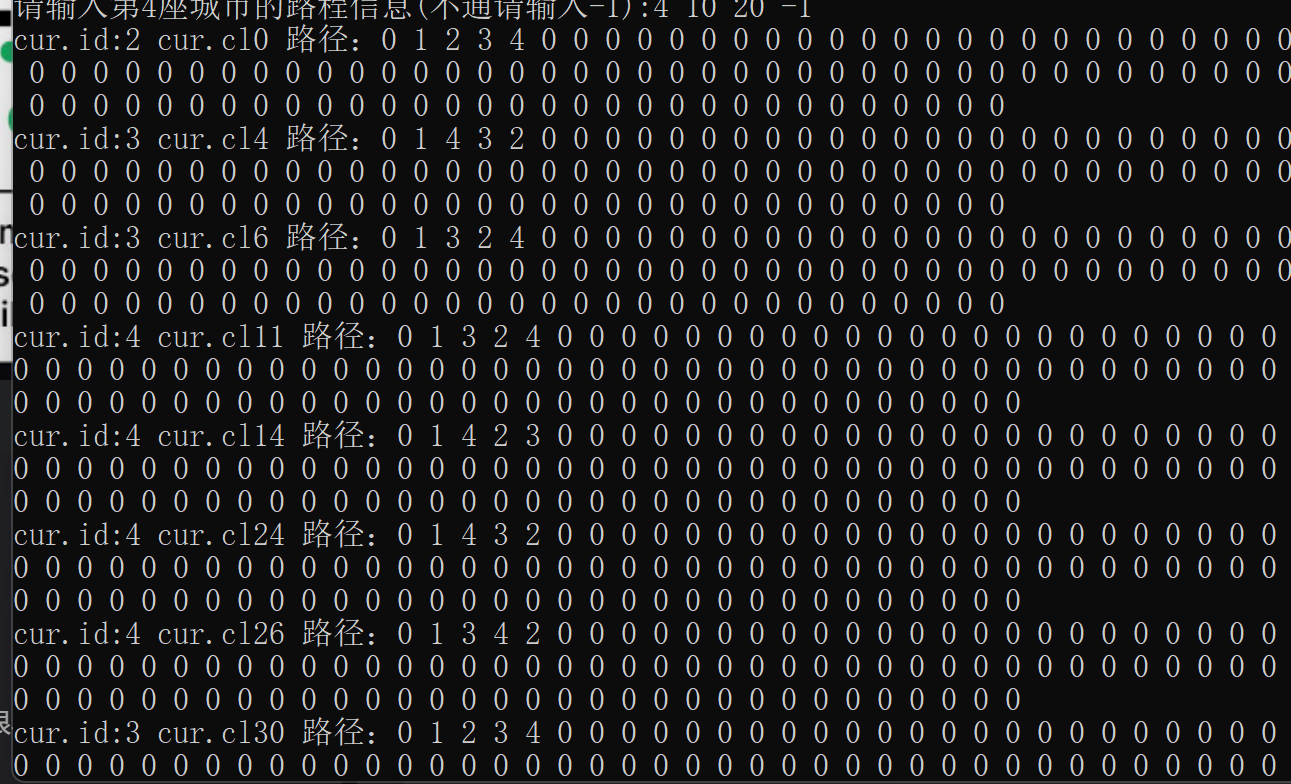
如果当前节点的距离cur.cl小于当前的最优路径长度bestl，则遍历下一个城市，判断是否满足约束条件，即当前城市和上一个城市之间是否存在路径，并且路径的长度是否小于当前的最优路径长度bestl。如果满足条件，则创建一个新的节点next\_node，并将其加入最小堆q中。在创建新节点时，需要将当前节点的路径复制给新节点，并交换新节点路径中下一个要遍历的城市和当前遍历的城市，以便继续搜索下一级节点

for (int k = 1; k <= n; ++k)
  
 next\_node.x[k] = cur.x[k];

然后，将新节点路径中 t 和 j 所对应的城市进行交换.最后，将新节点加入到最小堆 q 中，以便继续搜索下一级节点。

【实验结果】





时间复杂度分析：

时间复杂度的主要来源是最小堆的维护和节点的扩展。在最坏情况下，每个节点都需要扩展到最多n-1个子节点，因此节点的扩展次数不超过n!-1，而每次扩展时需要将子节点加入最小堆中，堆的大小不超过n!-1，因此堆的维护复杂度为O(log(n!))。由于n!的阶乘是指数级别的，因此总时间复杂度为O(n!\*log(n!))，非常高昂，只适用于较小的问题规模。(没有剪枝的情况下)

**项目3：八数码问题**

【整体思路】

八数码问题就是在一个3×3的九宫格棋盘上，分别将标有数字1~8的八个棋子摆放其中，摆放时要求棋子不能重叠。允许空格周围的某一个棋子向空格移动。在使用分支界限函数求解时需要一个评估函数用来决定棋子下一次应该移动的方向，计算节点所在解空间树的深度和其与目标方阵中数字不相同的个数作为评估函数是一种可行的方式，也是本算法中才采用的评估函数。

在一个 3×3 的网格中，1∼8 这 8 个数字和一个 X 恰好不重不漏地分布在这 3×3 的网格中。

例如：

1 2 3

X 4 6

7 5 8

在游戏过程中，可以把 X 与其上、下、左、右四个方向之一的数字交换（如果存在）。

我们的目的是通过交换，使得网格变为如下排列（称为正确排列）：

1 2 3

4 5 6

7 8 X

例如，示例中图形就可以通过让 X 先后与右、下、右三个方向的数字交换成功得到正确排列。

交换过程如下：

1 2 3 1 2 3 1 2 3 1 2 3

X 4 6 4 X 6 4 5 6 4 5 6

7 5 8 7 5 8 7 X 8 7 8 X

把 X 与上下左右方向数字交换的行动记录为 u、d、l、r。

现在，给你一个初始网格，请你通过最少的移动次数，得到正确排列。

输入格式

输入占一行，将 3×3 的初始网格描绘出来。

例如，如果初始网格如下所示：

1 2 3

x 4 6

7 5 8

则输入为：1 2 3 x 4 6 7 5 8

输出格式

输出占一行，包含一个字符串，表示得到正确排列的完整行动记录。

如果答案不唯一，输出任意一种合法方案即可。

如果不存在解决方案，则输出 unsolvable。

输入样例：

2 3 4 1 5 x 7 6 8

输出样例

ullddrurdllurdruldr

【问题分析】

关于启发式（评估函数）

启发式的一些性质

首先A\*算法中用f(n) = g(n) + h(n) 其中g是到达n结点的路径代价（即起点到n的距离）

评估函数h一定要满足可采纳性：h(n) < d(n),d(n)指从n到终点的实际代价。

还有一个略强于可采纳性的性质：一致性（单调性），只作用于图搜索中使用A\*算法。

一致性的定义是：对于任一结点n有：n通过任一路径a到达n’，h(n) <= h(n’) + c(n,a,n’)(n通过a到达n’的路径长度)

由一致性我们很容易可以得到：

在任何路径上f的值都是非递减的：

上述式子左右各加上g(n)有

g(n)+h(n)=f(n)

g(n)+c(n,a,n’)+h(n’) = g(n’)+h(n’) = f(n’)

从而有f(n) <= f(n’)

若结点n被出队，则此时已经找到了到达n的最优路径

采用反证法：若n被出队时还没有找到n的最优路径，那么n的最优路径上的某一点n’一定在优先队列中，

由f在任何路径上的非递减性，f(n’) <= f’(n)(指n经过最优路径的f值) < f(n)(指n经过当前路径的f)，

因为对于n点f(n) = g(n) + h(n) n经过任一路径到达n都不会改变h(n),因此经过最优路径的f值一定小于其他的。

那么此时n’的f一定会先被出队，矛盾。

要证明Manhattan heuristic的一致性也很简单：

每一步最多减少1的曼哈顿距离，而每一步的路径长度都是1，因此一定满足上述不等式。

更多启发式

设计启发式的思想

在八数码问题中可以用构造松弛问题的方式来得到评估函数。

首先，原问题中对于状态改变的限制如下：数字可以从方格A移动到方格B，当且仅当

1) A与B水平或竖直相邻

2) B是空的

我们可以减少操作的限制，生成两个子问题：

1）数字可以从方格A移动到方格B，当且仅当A与B水平或竖直相邻

2）数字可以从方格A移动到方格B，当且仅当B是空的

由1)我们可以得到评估函数：Manhattan heuristic，即曼哈顿距离之和

由2)我们可以得到评估函数：Gaschnig heuristic，本函数可以以模拟的方式求出

【算法设计】

剪枝法1：

首先需要输入初始矩阵和你想要的终止矩阵(默认终止矩阵是按照顺序, 即0,1,2,3,4,5,6,7,8的顺序).

之后需要计算一下初始矩阵的逆序和, 如果初始矩阵和终止矩阵的逆序和奇偶性不同, 则这种情况八数码问题无解.

八数码问题的有解无解的结论：  
一个状态表示成一维的形式，求出除0之外所有数字的逆序数之和，也就是每个数字前面比它大的数字的个数的和，称为这个状态的逆序。  
若两个状态的逆序奇偶性相同，则可相互到达，否则不可相互到达。

首先建立一个根节点, 之后对这个根节点放入函数进行求解(分支限界法)

对于根节点, 计算它的cost值, 即该状态矩阵与最后想要的矩阵中对应位置元素不同的数量,之后把根节点装入优先队列best,之后对于这个深度优先队列best, 如果它不为空, 则从队列中取出第一个节点, 该节点的cost值+layer(它所在的层数)的和在该优先队列中最小, 如果该节点的cost值为0, 则我们找到了答案, 我们可以打印结果, 并且在遍历该答案节点的所有子树的过程中记录下来所用的步骤数; 如果该节点不是答案, 则我们需要把当前情况下所对应的能产生的四种接下来的情况塞进优先队列.之后继续循环(即best不为空)

剪枝法2：

A\*算法

做法：

引入一个估值函数，用来估计某个点到达终点的距离。

记f是估值函数，g是真实值，那么f(state) <= g(state)，越接近越好（当估值是0时，类似于Dijkstra算法）

记dist是从起点到state状态的步数；

利用的是优先队列，排序依据是dist[state] + f(state)

证明：

反证法：

假设终点第一次出堆时不是最小值，那么意味着dist[end] > dist优

那么说明堆中存在一个最优路径中的某个点（起码起点在路径上），记该点为u，

dist优 = dist[u] + g(u) >= dist[u] + f(u)

-> dist[end] > dist优 >= dist[u] + f(u),说明优先队列中存在一个比出堆元素更小的值，这就矛盾了。

所以说终点第一次出堆时就是最优的。

应用的环境：

1、有解（无解时，仍然会把所有空间搜索，会比一般的bfs慢，因为优先队列的操作是logn的）

2、边权非负，如果是负数，那么终点的估值有可能是负无穷，终点可能会直接出堆

性质：

除了终点以外的其他点无法在出堆或者如堆的时候确定距离，只能保证终点出堆时是最优的可以。

【核心代码及解释】

剪枝1：

Node\* addnode(int newMatrix[N][N], int x, int y, int newX, int newY, int layer, Node\* parent)
  
{
  
 Node\* node = new Node();
  
 node->parent = parent;
  
 memcpy(node->matrix, newMatrix, sizeof(node->matrix));
  
 swap(node->matrix[x][y], node->matrix[newX][newY]);
  
 node->cost = 1000;
  
 node->layer = layer;
  
 node->x = newX;
  
 node->y = newY;
  
 return node;
  
}
  
  
int getf(int start[N][N], int end[N][N])
  
{
  
 int count = 0;
  
 for (int i = 0; i < N; i++)
  
 {
  
 for (int j = 0; j < N; j++)
  
 {
  
 if (start[i][j] != end[i][j])
  
 {
  
 count++;
  
 }
  
 }
  
 }
  
 return count;
  
}
  
  
bool isSafe(int x, int y)
  
{
  
 return (x >= 0 && x < N&& y >= 0 && y < N);
  
}
  
  
void bfs(int start[3][3], int end[3][3], int x, int y)
  
{
  
 priority\_queue<Node\*, std::vector<Node\*>, comp> best;
  
 Node\* root = addnode(start, x, y, x, y, 0, NULL);
  
 root->cost = getf(start, end);
  
 best.push(root);
  
 while (!best.empty())
  
 {
  
 Node\* min = best.top();
  
 best.pop();
  
 // 找到了最优解
  
 if (min->cost == 0)
  
 {
  
 //cout << "ddd" << endl;
  
 output(min);
  
 cout << "总共用了: " << total << " 次" << endl;
  
 return;
  
 }
  
 // 现在要把它上下左右塞进去
  
 for (int i = 0; i < 4; i++)
  
 {
  
 if (isSafe(min->x + dirx[i], min->y + diry[i]))
  
 {
  
 Node\* son = addnode(min->matrix, min->x, min->y, min->x + dirx[i], min->y + diry[i], min->layer + 1, min);
  
 son->cost = getf(son->matrix, end);
  
 best.push(son);
  
 }
  
 }
  
 }
  
}

* start：表示八数码问题的起始状态，是一个3x3的矩阵。
* end：表示八数码问题的目标状态，同样是一个3x3的矩阵。
* x和y：表示起始状态中数字0所在的位置，即空格所在的位置。
* best：是一个优先队列，其中的元素是指向Node结构体的指针，这些指针按照优先级排序。
* root：是根节点，它是一个Node结构体指针，代表了初始状态。
* min：表示队列中优先级最高的节点。
* i：循环变量，表示上下左右四个方向。

函数的主要逻辑是：

* 创建一个根节点，并将其放入优先队列中。
* 不断从优先队列中取出优先级最高的节点。
* 如果该节点表示目标状态，输出结果并结束。
* 如果不是目标状态，则遍历该节点的上下左右四个方向，并生成子节点。
* 将新生成的子节点加入到优先队列中，优先级由cost决定，cost是从当前状态到目标状态的估价函数，通过getf函数计算出来。

剪枝2：

string ed = "12345678x";
  
int dx[4] = {-1 , 0 , 1 , 0} , dy[4] = {0 , 1 , 0 , -1};
  
char op[] = "urdl";
  
  
int f(string state)//求估值函数,这里是曼哈顿距离
  
{
  
 int res = 0;
  
 for(int i = 0 ; i < 9 ; i++)
  
 {
  
 if(state[i] != 'x')
  
 {
  
 int t = state[i] - '1';
  
 res += abs(t / 3 - i / 3) + abs(t % 3 - i % 3);
  
 }
  
 }
  
 return res;
  
}
  
  
string bfs(string start)
  
{
  
 heap.push({f(start) , start});
  
 dist[start] = 0;
  
  
 while(heap.size())
  
 {
  
 auto t = heap.top();
  
 heap.pop();
  
 cout<<"当前节点：state "<<t.second<<" 估值：" <<t.first<<endl;
  
 string state = t.second;
  
 int step = dist[state];//记录到达state的实际距离
  
 if(state == ed) break;//如果到达终点就break
  
  
 int k = state.find('x');
  
 int x = k / 3 , y = k % 3;
  
  
 string source = state;//因为在下面state会变，所以留一个备份
  
 for (int i = 0; i < 4; i ++ )
  
 {
  
 int a = x + dx[i], b = y + dy[i];
  
 if (a >= 0 && a < 3 && b >= 0 && b < 3)
  
 {
  
 swap(state[x \* 3 + y], state[a \* 3 + b]);
  
 if (!dist.count(state) || dist[state] > step + 1)
  
 {
  
 dist[state] = step + 1;
  
 pre[state] = {source, op[i]};
  
 heap.push({dist[state] + f(state), state});
  
 }
  
 swap(state[x \* 3 + y], state[a \* 3 + b]);//因为要多次交换，所以要恢复现场
  
 }
  
 }
  
  
 }
  
  
 string res;
  
 while(ed != start)
  
 {
  
 res += pre[ed].second;
  
 ed = pre[ed].first;
  
 }
  
 reverse(res.begin() , res.end());
  
 return res;
  
}

首先定义了一个估值函数f，这里使用的是曼哈顿距离，用于估算当前状态到达目标状态的距离。然后使用一个priority\_queue来存储状态，每次取出估值最小的状态进行扩展。同时使用一个unordered\_map来存储状态到达起点的实际距离和之前的状态及操作。

在bfs函数中，首先将起点状态加入队列，并将其距离设置为0。然后每次取出估值最小的状态进行扩展，如果扩展到的状态之前没有到达过，或者当前到达状态的距离更小，则更新状态距离和之前状态，并将其加入队列。当到达终点状态时，跳出循环。

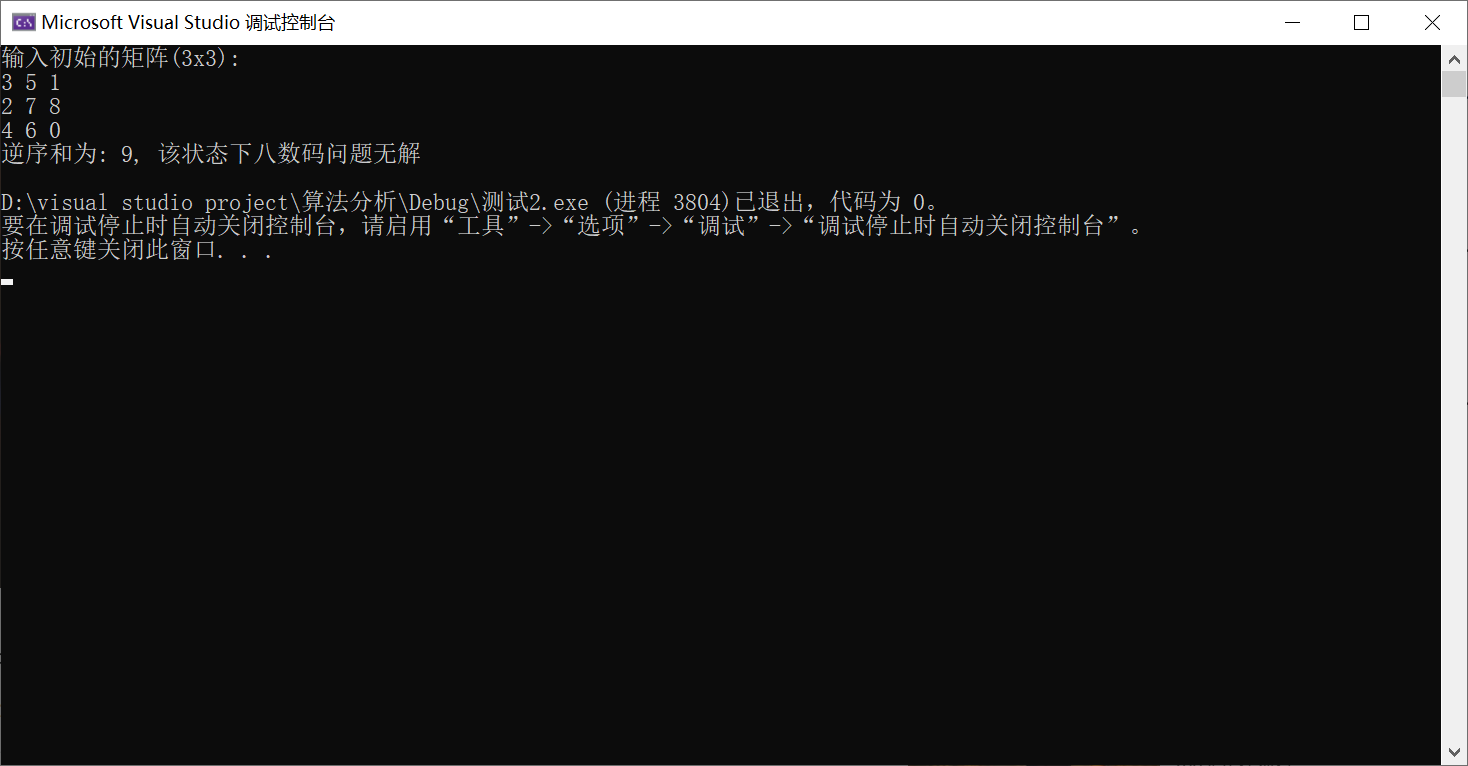
最后，从终点状态开始遍历之前的状态及其操作，得到最终的解法。

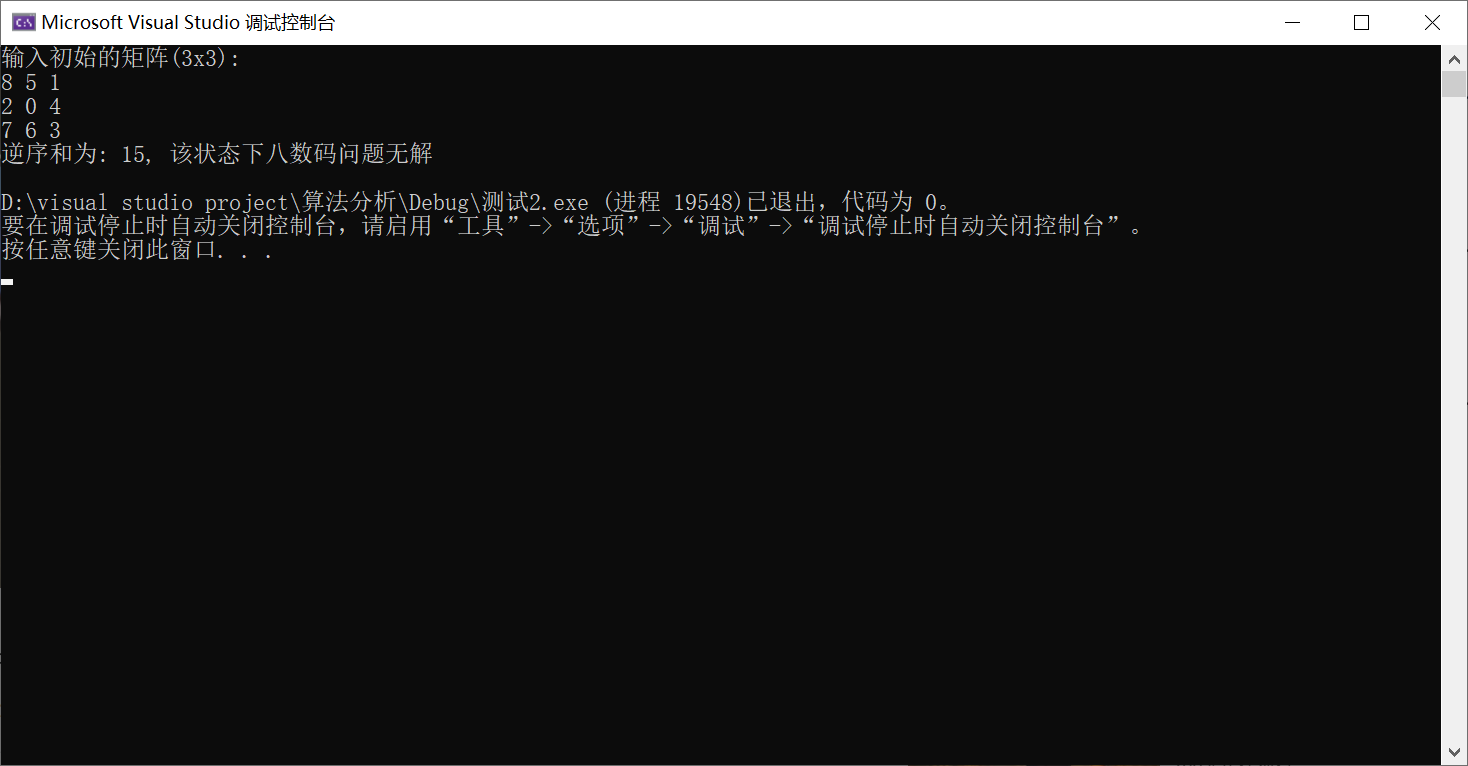
值得注意的是，由于交换状态时需要修改字符串，因此在交换后需要恢复现场，以便进行下一次交换。

其他与剪枝一类似。

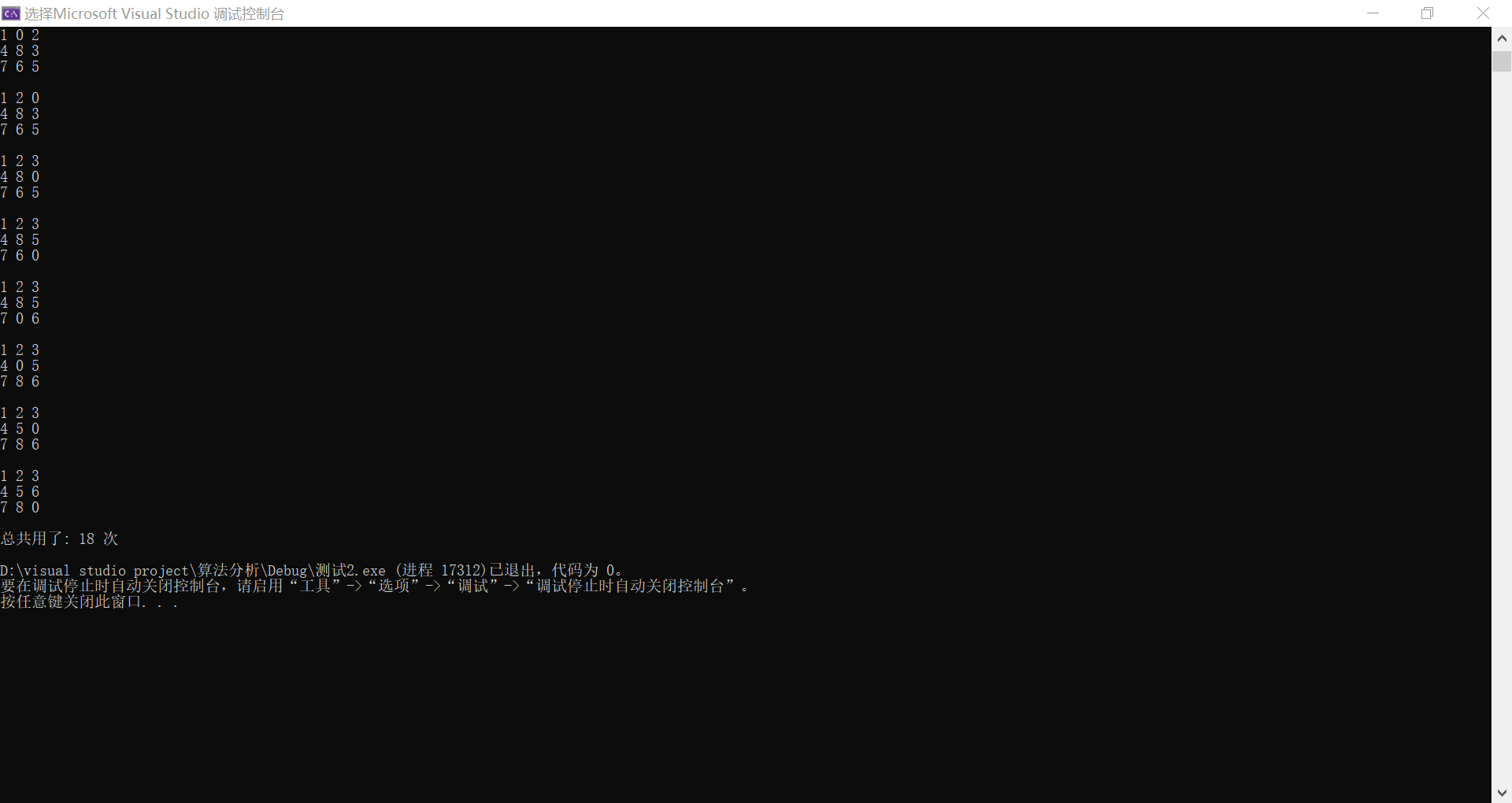
【实验结果】

剪枝1：

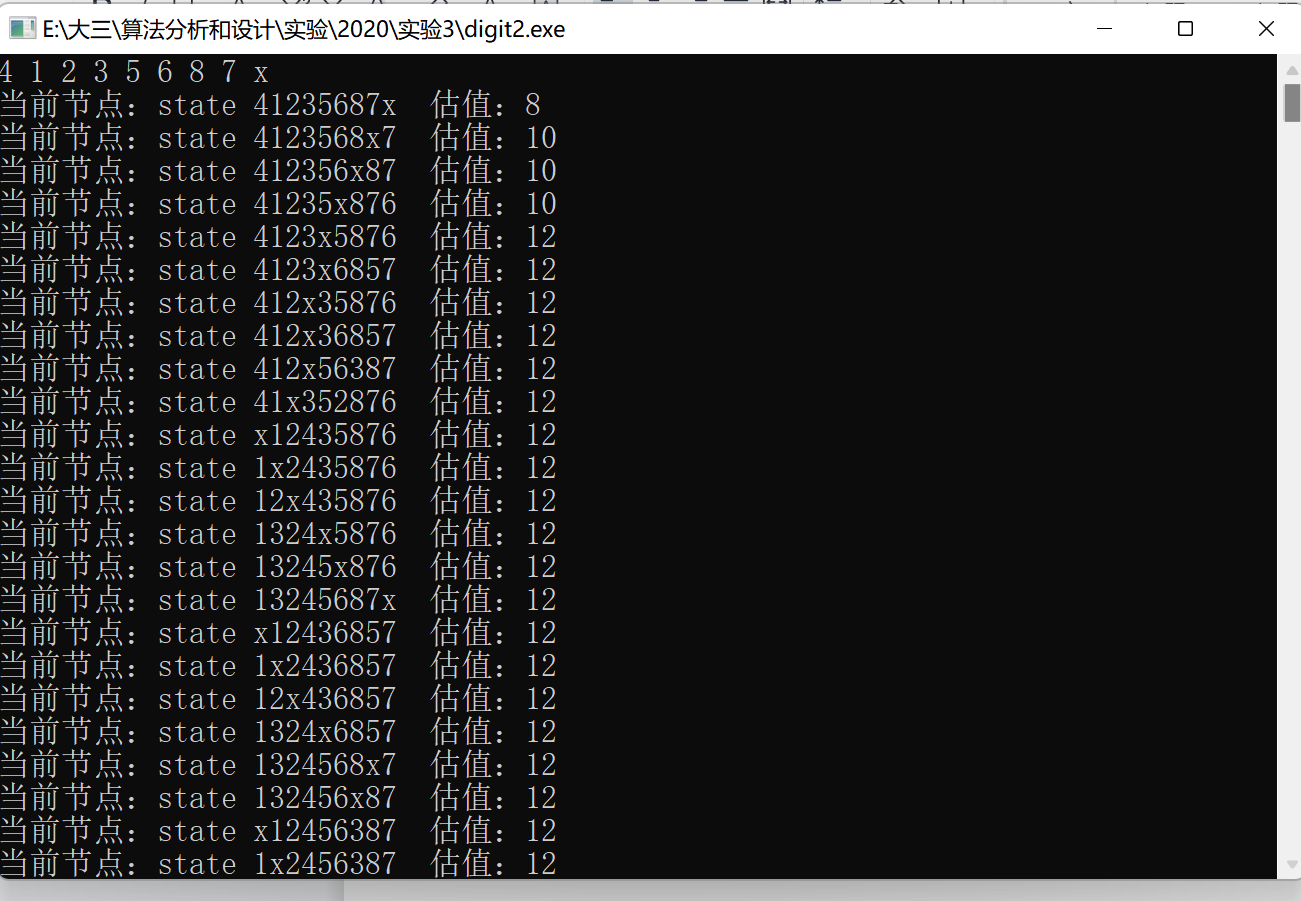


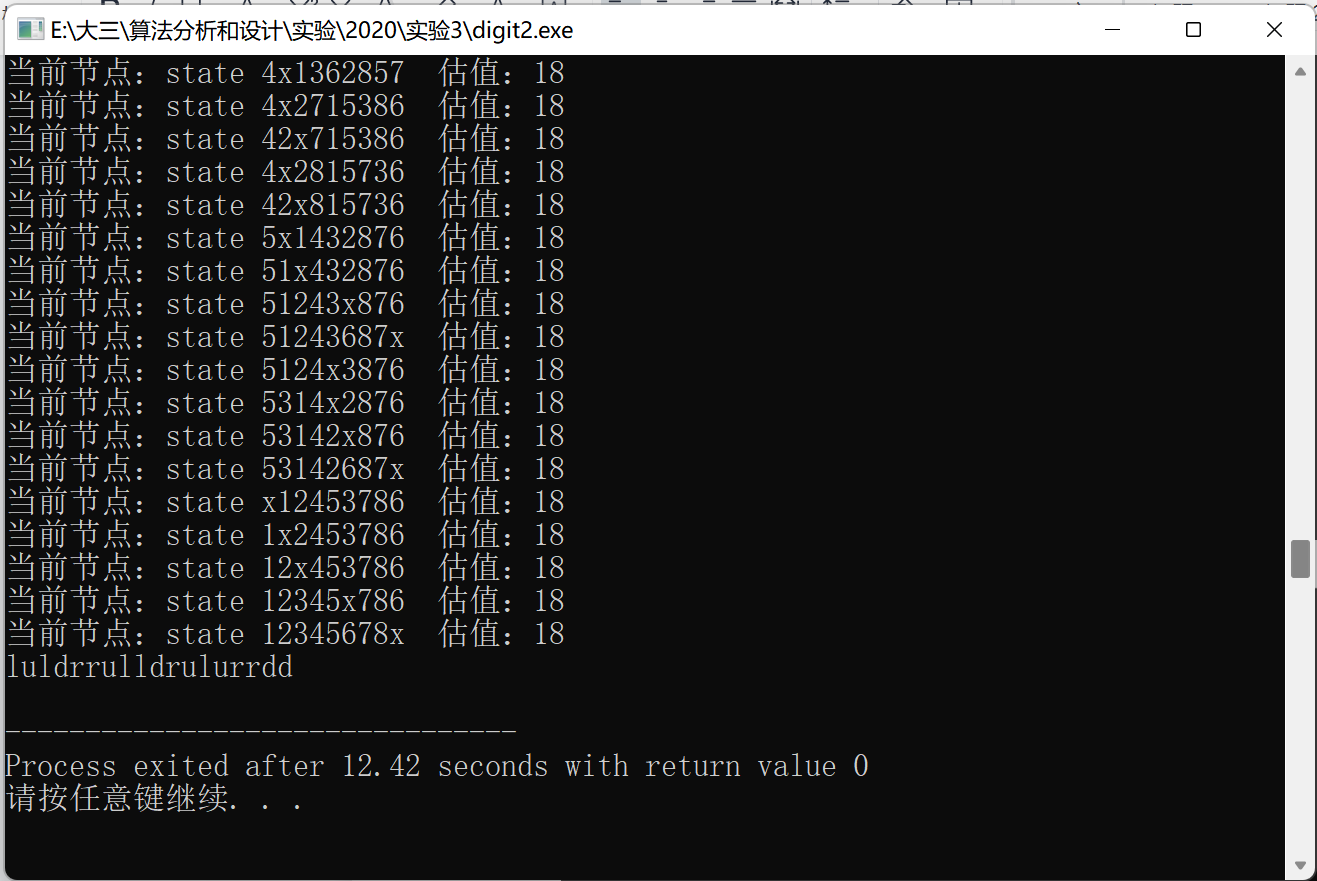


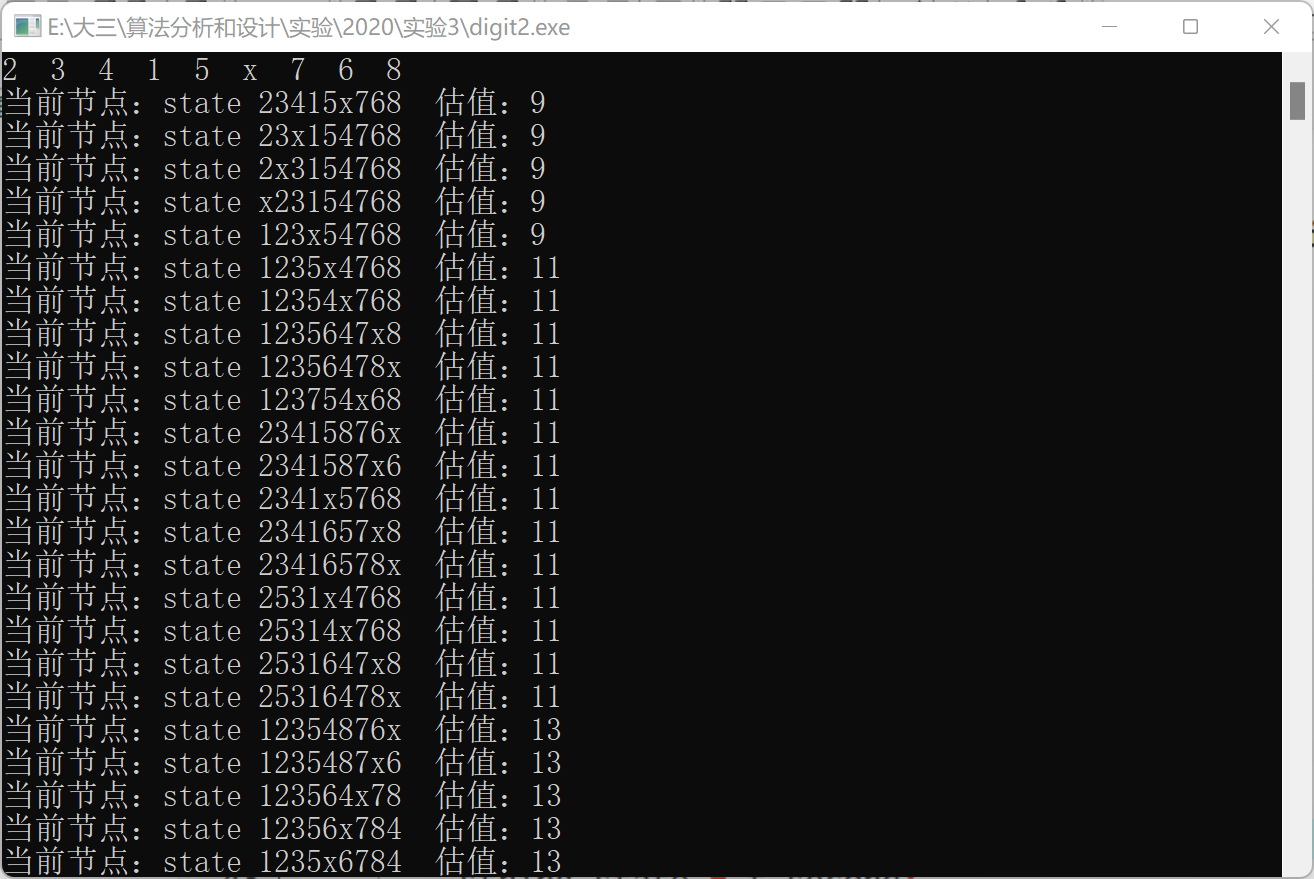


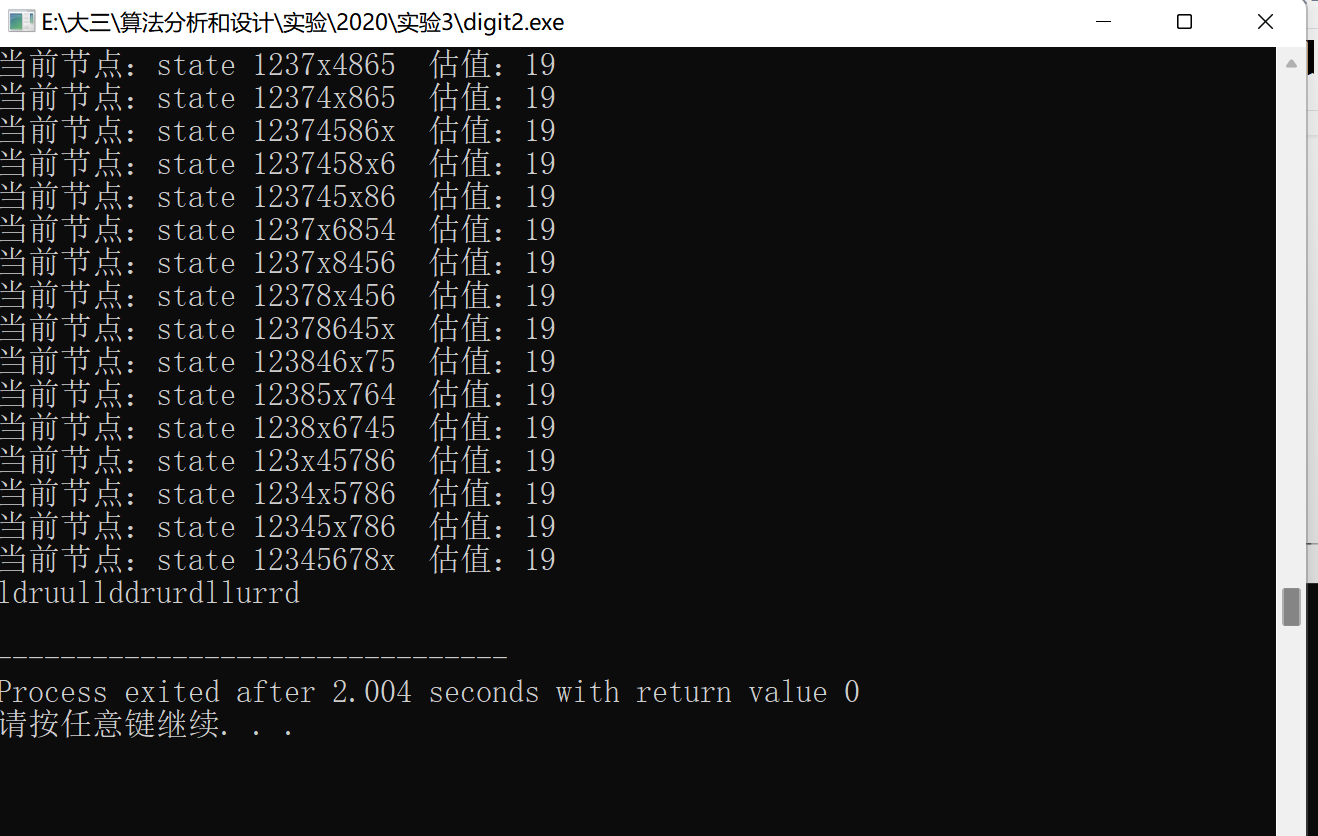


剪枝2：









完整

2 3 4 1 5 x 7 6 8

当前节点：state 23415x768 估值：9

当前节点：state 23x154768 估值：9

当前节点：state 2x3154768 估值：9

当前节点：state x23154768 估值：9

当前节点：state 123x54768 估值：9

当前节点：state 1235x4768 估值：11

当前节点：state 12354x768 估值：11

当前节点：state 1235647x8 估值：11

当前节点：state 12356478x 估值：11

当前节点：state 123754x68 估值：11

当前节点：state 23415876x 估值：11

当前节点：state 2341587x6 估值：11

当前节点：state 2341x5768 估值：11

当前节点：state 2341657x8 估值：11

当前节点：state 23416578x 估值：11

当前节点：state 2531x4768 估值：11

当前节点：state 25314x768 估值：11

当前节点：state 2531647x8 估值：11

当前节点：state 25316478x 估值：11

当前节点：state 12354876x 估值：13

当前节点：state 1235487x6 估值：13

当前节点：state 123564x78 估值：13

当前节点：state 12356x784 估值：13

当前节点：state 1235x6784 估值：13

当前节点：state 1237546x8 估值：13

当前节点：state 12375468x 估值：13

当前节点：state 123x56784 估值：13

当前节点：state 12x543768 估值：13

当前节点：state 1x3524768 估值：13

当前节点：state 234158x76 估值：13

当前节点：state 234165x78 估值：13

当前节点：state 23416x785 估值：13

当前节点：state 2341x6785 估值：13

当前节点：state 2341x8756 估值：13

当前节点：state 23418x756 估值：13

当前节点：state 23418675x 估值：13

当前节点：state 234x15768 估值：13

当前节点：state 23x164785 估值：13

当前节点：state 23x184756 估值：13

当前节点：state 25314876x 估值：13

当前节点：state 2531487x6 估值：13

当前节点：state 253164x78 估值：13

当前节点：state 25316x784 估值：13

当前节点：state 2531x6784 估值：13

当前节点：state 253x14768 估值：13

当前节点：state 25x143768 估值：13

当前节点：state 2x3156784 估值：13

当前节点：state 2x3164785 估值：13

当前节点：state 2x3184756 估值：13

当前节点：state 2x4135768 估值：13

当前节点：state 24x135768 估值：13

当前节点：state x23156784 估值：13

当前节点：state x23164785 估值：13

当前节点：state 123x64785 估值：13

当前节点：state x23184756 估值：13

当前节点：state 123x84756 估值：13

当前节点：state x24135768 估值：13

当前节点：state 124x35768 估值：13

当前节点：state 123548x76 估值：15

当前节点：state 1235867x4 估值：15

当前节点：state 12358674x 估值：15

当前节点：state 1235x8746 估值：15

当前节点：state 12358x746 估值：15

当前节点：state 1236x4785 估值：15

当前节点：state 12364x785 估值：15

当前节点：state 12364578x 估值：15

当前节点：state 123756x84 估值：15

当前节点：state 12375x684 估值：15

当前节点：state 123764x85 估值：15

当前节点：state 123784x56 估值：15

当前节点：state 1237x4658 估值：15

当前节点：state 12374x658 估值：15

当前节点：state 1238x4756 估值：15

当前节点：state 12384x756 估值：15

当前节点：state 12384675x 估值：15

当前节点：state 1238547x6 估值：15

当前节点：state 123x58746 估值：15

当前节点：state 123x64578 估值：15

当前节点：state 1243x5768 估值：15

当前节点：state 12435x768 估值：15

当前节点：state 1243657x8 估值：15

当前节点：state 12436578x 估值：15

当前节点：state 124735x68 估值：15

当前节点：state 12x354768 估值：15

当前节点：state 12x563784 估值：15

当前节点：state 13x524768 估值：15

当前节点：state 1x2543768 估值：15

当前节点：state 1x3526784 估值：15

当前节点：state 2341867x5 估值：15

当前节点：state 234715x68 估值：15

当前节点：state 234x16785 估值：15

当前节点：state 234x18756 估值：15

当前节点：state 234x58176 估值：15

当前节点：state 234x65178 估值：15

当前节点：state 23x156784 估值：15

当前节点：state 24513x768 估值：15

当前节点：state 2451x3768 估值：15

当前节点：state 2451637x8 估值：15

当前节点：state 24516378x 估值：15

当前节点：state 253148x76 估值：15

当前节点：state 2531867x4 估值：15

当前节点：state 25318674x 估值：15

当前节点：state 2531x8746 估值：15

当前节点：state 25318x746 估值：15

当前节点：state 253714x68 估值：15

当前节点：state 253x16784 估值：15

当前节点：state 253x64178 估值：15

当前节点：state 25x163784 估值：15

当前节点：state 2631x4785 估值：15

当前节点：state 26314x785 估值：15

当前节点：state 26314578x 估值：15

当前节点：state 2831x4756 估值：15

当前节点：state 28314x756 估值：15

当前节点：state 28314675x 估值：15

当前节点：state 2831547x6 估值：15

当前节点：state 2x3158746 估值：15

当前节点：state 2x4136785 估值：15

当前节点：state 24x136785 估值：15

当前节点：state 2x4138756 估值：15

当前节点：state 24x138756 估值：15

当前节点：state 2x5143768 估值：15

当前节点：state x13524768 估值：15

当前节点：state x23156784 估值：15

当前节点：state x23158746 估值：15

当前节点：state x24136785 估值：15

当前节点：state 124x36785 估值：15

当前节点：state x24138756 估值：15

当前节点：state 124x38756 估值：15

当前节点：state x25143768 估值：15

当前节点：state 125x43768 估值：15

当前节点：state 1254x3768 估值：15

当前节点：state 1254637x8 估值：15

当前节点：state 12546378x 估值：15

当前节点：state x34215768 估值：15

当前节点：state x53214768 估值：15

当前节点：state 123586x74 估值：17

当前节点：state 12358x746 估值：17

当前节点：state 1236457x8 估值：17

当前节点：state 1236847x5 估值：17

当前节点：state 12368475x 估值：17

当前节点：state 1236x4578 估值：17

当前节点：state 12364x578 估值：17

当前节点：state 12374865x 估值：17

当前节点：state 1237568x4 估值：17

当前节点：state 12375684x 估值：17

当前节点：state 123758x46 估值：17

当前节点：state 1237584x6 估值：17

当前节点：state 1237648x5 估值：17

当前节点：state 12376485x 估值：17

当前节点：state 1237845x6 估值：17

当前节点：state 1237x4586 估值：17

当前节点：state 12374x586 估值：17

当前节点：state 12374658x 估值：17

当前节点：state 1237x5684 估值：17

当前节点：state 1238467x5 估值：17

当前节点：state 12385476x 估值：17

当前节点：state 123854x76 估值：17

当前节点：state 123x48576 估值：17

当前节点：state 1234x8576 估值：17

当前节点：state 12348x576 估值：17

当前节点：state 12348657x 估值：17

当前节点：state 123x54876 估值：17

当前节点：state 123x74658 估值：17

当前节点：state 123674x58 估值：17

当前节点：state 12435876x 估值：17

当前节点：state 1243587x6 估值：17

当前节点：state 124365x78 估值：17

当前节点：state 12436x785 估值：17

当前节点：state 1243x6785 估值：17

当前节点：state 1243x8756 估值：17

当前节点：state 12438x756 估值：17

当前节点：state 12438675x 估值：17

当前节点：state 1247356x8 估值：17

当前节点：state 12473568x 估值：17

当前节点：state 124736x85 估值：17

当前节点：state 124738x56 估值：17

当前节点：state 12543x768 估值：17

当前节点：state 125463x78 估值：17

当前节点：state 12546x783 估值：17

当前节点：state 1254x6783 估值：17

当前节点：state 125743x68 估值：17

当前节点：state 12x364785 估值：17

当前节点：state 12x384756 估值：17

当前节点：state 12x435768 估值：17

当前节点：state 12x465783 估值：17

当前节点：state 12x583746 估值：17

当前节点：state 12x643785 估值：17

当前节点：state 12x743658 估值：17

当前节点：state 12x753684 估值：17

当前节点：state 12x843756 估值：17

当前节点：state 13452x768 估值：17

当前节点：state 13x526784 估值：17

当前节点：state 1425x3768 估值：17

当前节点：state 1425637x8 估值：17

当前节点：state 14256378x 估值：17

当前节点：state 142x53768 估值：17

当前节点：state 1x2354768 估值：17

当前节点：state 1x2563784 估值：17

当前节点：state 1x3528746 估值：17

当前节点：state 1x3624785 估值：17

当前节点：state 1x3724658 估值：17

当前节点：state 1x3824756 估值：17

当前节点：state 1x4325768 估值：17

当前节点：state 14x325768 估值：17

当前节点：state 1x5423768 估值：17

当前节点：state 15x423768 估值：17

当前节点：state 15342x768 估值：17

当前节点：state 234186x75 估值：17

当前节点：state 2345x8176 估值：17

当前节点：state 23458x176 估值：17

当前节点：state 23458617x 估值：17

当前节点：state 2346x5178 估值：17

当前节点：state 23465x178 估值：17

当前节点：state 2347156x8 估值：17

当前节点：state 23471568x 估值：17

当前节点：state 234716x85 估值：17

当前节点：state 234718x56 估值：17

当前节点：state 23615x784 估值：17

当前节点：state 23615478x 估值：17

当前节点：state 23x158746 估值：17

当前节点：state 23x584176 估值：17

当前节点：state 23x654178 估值：17

当前节点：state 24513876x 估值：17

当前节点：state 2451387x6 估值：17

当前节点：state 245163x78 估值：17

当前节点：state 24516x783 估值：17

当前节点：state 2451x6783 估值：17

当前节点：state 245x13768 估值：17

当前节点：state 24613x785 估值：17

当前节点：state 24613578x 估值：17

当前节点：state 2461x3785 估值：17

当前节点：state 24813x756 估值：17

当前节点：state 24813675x 估值：17

当前节点：state 2481x3756 估值：17

当前节点：state 2481537x6 估值：17

当前节点：state 24x165783 估值：17

当前节点：state 253186x74 估值：17

当前节点：state 25318x746 估值：17

当前节点：state 2536x4178 估值：17

当前节点：state 25364x178 估值：17

当前节点：state 2537146x8 估值：17

当前节点：state 25371468x 估值：17

当前节点：state 253716x84 估值：17

当前节点：state 253x18746 估值：17

当前节点：state 253x48176 估值：17

当前节点：state 2534x8176 估值：17

当前节点：state 25348x176 估值：17

当前节点：state 25348617x 估值：17

当前节点：state 25x183746 估值：17

当前节点：state 2631457x8 估值：17

当前节点：state 2631847x5 估值：17

当前节点：state 26318475x 估值：17

当前节点：state 263x14785 估值：17

当前节点：state 26x143785 估值：17

当前节点：state 2831467x5 估值：17

当前节点：state 28315476x 估值：17

当前节点：state 283154x76 估值：17

当前节点：state 283x14756 估值：17

当前节点：state 28x143756 估值：17

当前节点：state 2x3458176 估值：17

当前节点：state 2x3584176 估值：17

当前节点：state 2x3654178 估值：17

当前节点：state 2x5146783 估值：17

当前节点：state 25x146783 估值：17

当前节点：state 2x5163784 估值：17

当前节点：state 2x6143785 估值：17

当前节点：state 2x8143756 估值：17

当前节点：state 3x4215768 估值：17

当前节点：state 3142x5768 估值：17

当前节点：state 31425x768 估值：17

当前节点：state 3142657x8 估值：17

当前节点：state 31426578x 估值：17

当前节点：state 314x25768 估值：17

当前节点：state 31x254768 估值：17

当前节点：state 34x215768 估值：17

当前节点：state 513x24768 估值：17

当前节点：state 5x3214768 估值：17

当前节点：state 5132x4768 估值：17

当前节点：state 51324x768 估值：17

当前节点：state 5132647x8 估值：17

当前节点：state 51326478x 估值：17

当前节点：state x12543768 估值：17

当前节点：state x13526784 估值：17

当前节点：state x23158746 估值：17

当前节点：state x23164578 估值：17

当前节点：state x23458176 估值：17

当前节点：state x23584176 估值：17

当前节点：state x23654178 估值：17

当前节点：state x25146783 估值：17

当前节点：state 125x46783 估值：17

当前节点：state x25163784 估值：17

当前节点：state 125x63784 估值：17

当前节点：state x26143785 估值：17

当前节点：state 126x43785 估值：17

当前节点：state 1264x3785 估值：17

当前节点：state x28143756 估值：17

当前节点：state 128x43756 估值：17

当前节点：state 1284x3756 估值：17

当前节点：state 1284537x6 估值：17

当前节点：state x34216785 估值：17

当前节点：state x34218756 估值：17

当前节点：state x34258176 估值：17

当前节点：state x34265178 估值：17

当前节点：state x53216784 估值：17

当前节点：state x53264178 估值：17

当前节点：state 1234785x6 估值：19

当前节点：state 123478x56 估值：19

当前节点：state 1234865x7 估值：19

当前节点：state 123486x57 估值：19

当前节点：state 1234x6587 估值：19

当前节点：state 1235x4876 估值：19

当前节点：state 12354x876 估值：19

当前节点：state 12354687x 估值：19

当前节点：state 123645x78 估值：19

当前节点：state 12364857x 估值：19

当前节点：state 1236745x8 估值：19

当前节点：state 12367458x 估值：19

当前节点：state 123684x75 估值：19

当前节点：state 12368x754 估值：19

当前节点：state 1236x5748 估值：19

当前节点：state 12365x748 估值：19

当前节点：state 1237465x8 估值：19

当前节点：state 123746x58 估值：19

当前节点：state 1237486x5 估值：19

当前节点：state 123748x65 估值：19

当前节点：state 12375846x 估值：19

当前节点：state 12375x468 估值：19

当前节点：state 12375x846 估值：19

当前节点：state 12376x854 估值：19

当前节点：state 12378456x 估值：19

当前节点：state 1237856x4 估值：19

当前节点：state 12378564x 估值：19

当前节点：state 123785x64 估值：19

当前节点：state 1237x4865 估值：19

当前节点：state 12374x865 估值：19

当前节点：state 12374586x 估值：19

当前节点：state 1237458x6 估值：19

当前节点：state 123745x86 估值：19

当前节点：state 1237x6854 估值：19

当前节点：state 1237x8456 估值：19

当前节点：state 12378x456 估值：19

当前节点：state 12378645x 估值：19

当前节点：state 123846x75 估值：19

当前节点：state 12385x764 估值：19

当前节点：state 1238x6745 估值：19

当前节点：state 123x45786 估值：19

当前节点：state 1234x5786 估值：19

当前节点：state 12345x786 估值：19

当前节点：state 12345678x 估值：19

ldruullddrurdllurrd

**源代码**

### 项目1

#include <bits/stdc++.h>
  
using namespace std;
  
class goods
  
{
  
public:
  
 int id;
  
 int weight;
  
 int price;
  
 float d;
  
};
  
class qnode
  
{
  
public:
  
 qnode \*parent;
  
 int lchild;
  
 int upprofit;
  
 int profit;
  
 int weight;
  
 int lev;
  
};
  
struct cmp
  
{
  
 bool operator()(qnode \*&a, qnode \*&b) const
  
 {
  
 return a->upprofit < b->upprofit;
  
 }
  
};
  
bool cmp2(const goods &a, const goods &b)
  
{
  
 return a.d >= b.d;
  
}
  
int n;
  
int c;
  
int cw;
  
int cp;
  
int bestp;
  
goods obj[100];
  
int bestx[100];
  
void input()
  
{
  
 cout << "输入物品个数：";
  
 cin >> n;
  
 cout << "输入背包容量：";
  
 cin >> c;
  
 cout << "输入物品信息：（重量w 价值w）" << endl;
  
 for(int i = 1; i <= n; ++i)
  
 {
  
 scanf("%d %d", &obj[i].price, &obj[i].weight);
  
 obj[i].id = i;
  
 obj[i].d = 1.0 \* obj[i].price / obj[i].weight;
  
 }
  
  
 sort(obj + 1, obj + n + 1, cmp2);
  
// for(int i = 1; i <= n; ++i)
  
// cout << obj[i].d << " ";
  
// cout << endl << "input Complete" << endl;
  
}
  
int Bound(int i)
  
{
  
 int tmp\_cleft = c - cw;
  
 int tmp\_cp = cp;
  
 while(tmp\_cleft >= obj[i].weight && i <= n)
  
 {
  
 tmp\_cleft -= obj[i].weight;
  
 tmp\_cp += obj[i].price;
  
 i++;
  
 }
  
 if(i <= n)
  
 {
  
 tmp\_cp += tmp\_cleft \* obj[i].d;
  
 }
  
 return tmp\_cp;
  
}
  
void addnode(priority\_queue<qnode \*, vector<qnode \*>, cmp> &q, qnode \*E, int up, int wt, int curp, int i, int ch)
  
{
  
 qnode \*p = new qnode;
  
 p->parent = E;
  
 p->lchild = ch;
  
 p->weight = wt;
  
 p->upprofit = up;
  
 p->profit = curp;
  
 p->lev = i + 1;
  
 q.push(p);
  
 cout << "加入点的信息为 " << endl;
  
 cout << "p->lev = " << p->lev << " p->upprofit = " << p->upprofit << " p->weight = " << p->weight << " p->profit = " << p->profit << endl;
  
}
  
void backpack()
  
{
  
 priority\_queue<qnode \*, vector<qnode \*>, cmp > q; // 大顶堆
  
 qnode \*pa = NULL;
  
 cw = cp = bestp = 0;
  
 int i = 1;
  
 int up = Bound(1); //Bound(i)函数计算的是i还未处理时候的上限值
  
 while(i != n + 1)
  
 {
  
 int wt = cw + obj[i].weight;
  
 if(wt <= c)
  
 {
  
 if(bestp < cp + obj[i].price)
  
 bestp = cp + obj[i].price;
  
 addnode(q, pa, up, cw + obj[i].weight, cp + obj[i].price, i, 1);
  
 }
  
 up = Bound(i + 1); //注意这里 up != up - obj[i].price而且 up >= up - obj[i].price
  
 if(up >= bestp) //注意这里必须是大于等于
  
 {
  
 addnode(q, pa, up, cw, cp, i, 0);
  
 }
  
 pa = q.top();
  
 q.pop();
  
 cw = pa->weight;
  
 cp = pa->profit;
  
 up = pa->upprofit;
  
 i = pa->lev;
  
 }
  
 for(int j = n; j > 0; --j)
  
 {
  
 bestx[obj[pa->lev - 1].id] = pa->lchild;
  
 pa = pa->parent;
  
 }
  
}
  
void output()
  
{
  
 printf("最优装入量为 %d\n", bestp);
  
 printf("装入的物品为 \n");
  
 for(int i = 1; i <= n; ++i)
  
 if(bestx[i] == 1)
  
 printf("%d ", i);
  
}
  
int main()
  
{
  
 input();
  
 backpack();
  
 output();
  
}
  
//4 10
  
//40 4
  
//42 7
  
//25 5
  
//12 3

### 项目2

#include<bits/stdc++.h>
  
#define debug(a) cout<<#a<<"="<<a<<endl
  
#define ll long long
  
#define INF 1e7
  
using namespace std;
  
  
int m[100][100];//存储路径，从1开始
  
int bestx[100];//最优解路径
  
int bestl;//最优解长度
  
int n;//城市数目
  
  
//排列树的节点定义
  
struct node
  
{
  
 int cl;//当前走过的路径长度
  
 int id;//处理的第几个城市
  
 int x[100];//记录当前路径，下标从1开始
  
  
 node() {}
  
 node(int cl\_,int id\_)
  
 {
  
 cl=cl\_;
  
 id=id\_;
  
 memset(x,0,sizeof(x));
  
 }
  
};
  
  
  
  
//用于构建最小堆
  
struct cmp
  
{
  
 //当前路径长度短的优先级高
  
 bool operator()(node n1, node n2)
  
 {
  
 return n1.cl > n2.cl;
  
 }
  
};
  
  
void bfs()
  
{
  
 //选用最小堆
  
 priority\_queue<node,vector<node>,cmp> q;
  
 //创建一个节点，从该节点开始，因为1是固定位，其实是从1开始探索
  
 node next\_node(0,2);
  
 //初始化解向量
  
 for(int i=1; i<=n; ++i)
  
 next\_node.x[i]=i;
  
 q.push(next\_node);
  
 node cur;//当前节点，也就是活节点
  
 int t;
  
 while(!q.empty())
  
 {
  
 cur=q.top();
  
 q.pop();
  
 t=cur.id;
  
 cout<<"cur.id:"<<cur.id<<" cur.cl"<<cur.cl<<" 路径：" ;
  
 for(auto m:cur.x)
  
 {
  
 cout<<m<<" ";
  
 }
  
 cout<<endl;
  
 //处理到倒数第二个城市
  
 if(t==n)
  
 {
  
 //满足约束条件，有路径
  
 //检测图G是否存在一条从顶点x[n-1]到顶点x[n]的边和一条从顶点x[n]到顶点1的边
  
 //这里和前面的理论部分不同，因为这里是从1开始，而不是0开始
  
 if(m[cur.x[t-1]][cur.x[t]]!=INF&&m[cur.x[t]][1]!=INF)
  
 {
  
 if(cur.cl+m[cur.x[t-1]][cur.x[t]]+m[cur.x[t]][1]<bestl)
  
 {
  
 //更新最优解和最优路径
  
 bestl=cur.cl+m[cur.x[t-1]][cur.x[t]]+m[cur.x[t]][1];
  
 for(int i=1; i<=n; ++i)
  
 bestx[i]=cur.x[i];
  
 }
  
 }
  
 continue;
  
 }
  
 //大于等于最优路径，没必要继续探索了，从下一个节点开始
  
 if(cur.cl>=bestl)
  
 continue;
  
 //从当前节点开始探索t-1 -> t,t+1,t+2...
  
 for(int j=t; j<=n; ++j)
  
 {
  
 //满足约束条件和限界条件
  
 if(m[cur.x[t-1]][cur.x[j]]!=INF&&cur.cl+m[cur.x[t-1]][cur.x[j]]<bestl)
  
 {
  
 next\_node=node(cur.cl+m[cur.x[t-1]][cur.x[j]],t+1);
  
 //如果找到了一个下级节点，那么该节点到现在为止和同级的节点路径相同，除了当前这一级的不同
  
 for(int k=1; k<=n; ++k)
  
 next\_node.x[k]=cur.x[k];
  
 swap(next\_node.x[t],next\_node.x[j]);
  
 q.push(next\_node);
  
 }
  
 }
  
 }
  
}
  
  
int main()
  
{
  
 cout<<"请输入城市数目:";
  
 cin>>n;
  
  
 //先进行初始化
  
 int i,j;
  
 for(i=1; i<=n; ++i)
  
 for(j=1; j<=n; ++j)
  
 m[i][j]=INF;
  
 memset(bestx,0,n);
  
 bestl=INF;
  
  
 for(int i=1;i<=n;i++)
  
 {
  
 cout << "请输入第" << i << "座城市的路程信息(不通请输入-1):";
  
 for (int j = 1; j <= n; j++)
  
 {
  
 int next\_node;
  
 cin>>next\_node;
  
 if(next\_node==-1)
  
 {
  
 continue;
  
 }
  
 m[i][j] = next\_node;
  
 }
  
 }
  
 //进行探索
  
 bfs();
  
 cout<<"最优值为："<<bestl<<endl;
  
 cout<<"最优解为：";
  
 for(int i=1; i<=n; ++i)
  
 cout<<bestx[i]<<" ";
  
 cout<<bestx[1]<<endl;
  
 return 0;
  
}

### 项目3

//剪枝1
  
#include <bits/stdc++.h>
  
using namespace std;
  
#define N 3
  
  
int dirx[] = { 1,0,-1,0 };
  
int diry[] = { 0,-1,0,1 };
  
int total = -1;
  
  
class Node
  
{
  
public:
  
 Node\* parent;
  
 int matrix[N][N];
  
 int x, y;
  
 int cost;
  
 int layer;
  
};
  
  
Node\* addnode(int newMatrix[N][N], int x, int y, int newX, int newY, int layer, Node\* parent)
  
{
  
 Node\* node = new Node();
  
 node->parent = parent;
  
 memcpy(node->matrix, newMatrix, sizeof(node->matrix));
  
 swap(node->matrix[x][y], node->matrix[newX][newY]);
  
 node->cost = 1000;
  
 node->layer = layer;
  
 node->x = newX;
  
 node->y = newY;
  
 return node;
  
}
  
  
int getf(int start[N][N], int end[N][N])
  
{
  
 int count = 0;
  
 for (int i = 0; i < N; i++)
  
 {
  
 for (int j = 0; j < N; j++)
  
 {
  
 if (start[i][j] != end[i][j])
  
 {
  
 count++;
  
 }
  
 }
  
 }
  
 return count;
  
}
  
  
bool isSafe(int x, int y)
  
{
  
 return (x >= 0 && x < N&& y >= 0 && y < N);
  
}
  
  
// 用来比较的
  
struct comp
  
{
  
 bool operator()(const Node\* first, const Node\* second) const
  
 {
  
 return (first->cost + first->layer) > (second->cost + second->layer);
  
 }
  
};
  
  
void printm(int matrix[N][N])
  
{
  
 for (int i = 0; i < N; i++)
  
 {
  
 for (int j = 0; j < N; j++)
  
 {
  
 cout << matrix[i][j] << " ";
  
 }
  
 cout << endl;
  
 }
  
}
  
  
// 打印每一步
  
void output(Node\* node)
  
{
  
 if (node == NULL)
  
 {
  
 return;
  
 }
  
 total++;
  
 output(node->parent);
  
 printm(node->matrix);
  
 cout << endl;
  
}
  
  
void bfs(int start[3][3], int end[3][3], int x, int y)
  
{
  
 priority\_queue<Node\*, std::vector<Node\*>, comp> best;
  
 Node\* root = addnode(start, x, y, x, y, 0, NULL);
  
 root->cost = getf(start, end);
  
 best.push(root);
  
 while (!best.empty())
  
 {
  
 Node\* min = best.top();
  
 best.pop();
  
 // 找到了最优解
  
 if (min->cost == 0)
  
 {
  
 //cout << "ddd" << endl;
  
 output(min);
  
 cout << "总共用了: " << total << " 次" << endl;
  
 return;
  
 }
  
 // 现在要把它上下左右塞进去
  
 for (int i = 0; i < 4; i++)
  
 {
  
 if (isSafe(min->x + dirx[i], min->y + diry[i]))
  
 {
  
 Node\* son = addnode(min->matrix, min->x, min->y, min->x + dirx[i], min->y + diry[i], min->layer + 1, min);
  
 son->cost = getf(son->matrix, end);
  
 best.push(son);
  
 }
  
 }
  
 }
  
}
  
  
int main()
  
{
  
  
 int start[N][N];
  
 int x = 0;
  
 int y = 0;
  
  
 cout << "输入初始的矩阵(3x3): " << endl;
  
 for (int i = 0; i < N; i++)
  
 {
  
 for (int j = 0; j < N; j++)
  
  
 {
  
 cin>>start[i][j];
  
 }
  
 }
  
  
 int judge[N\*N];
  
 for (int i = 0; i < N; i++)
  
 {
  
 for (int j = 0; j < N; j++)
  
 {
  
 judge[i \* N + j] = start[i][j];
  
 }
  
 }
  
  
 int count = 0;
  
 for (int i = 0; i < N \* N; i++)
  
 {
  
 if (judge[i] != 0)
  
 {
  
 for (int j = 0; j < i; j++)
  
 {
  
 if (judge[j] > judge[i])
  
 {
  
 count++;
  
 }
  
 }
  
 }
  
 }
  
 if (count % 2 == 1)
  
 {
  
 cout << "逆序和为: " << count << ", 该状态下八数码问题无解" << endl;
  
 return 0;
  
 }
  
  
 cout << "==============================开始解题==============================" << endl;
  
  
 for (int i = 0; i < N; i++)
  
 {
  
 for (int j = 0; j < N; j++)
  
 {
  
 if (start[i][j] == 0)
  
 {
  
 x = i;
  
 y = j;
  
 }
  
 }
  
 }
  
  
 int final[N][N] =
  
 {
  
 {1,2,3},
  
 {4,5,6},
  
 {7,8,0}
  
 };
  
  
 bfs(start, final, x,y);
  
  
 return 0;
  
}
  
//2
  
#include <bits/stdc++.h>
  
using namespace std;
  
  
typedef pair<int , string> PIS;
  
  
unordered\_map<string , int> dist;
  
unordered\_map<string , pair<string , char>> pre;
  
priority\_queue<PIS , vector<PIS> , greater<PIS>> heap;
  
string ed = "12345678x";
  
int dx[4] = {-1 , 0 , 1 , 0} , dy[4] = {0 , 1 , 0 , -1};
  
char op[] = "urdl";
  
  
int f(string state)//求估值函数,这里是曼哈顿距离
  
{
  
 int res = 0;
  
 for(int i = 0 ; i < 9 ; i++)
  
 {
  
 if(state[i] != 'x')
  
 {
  
 int t = state[i] - '1';
  
 res += abs(t / 3 - i / 3) + abs(t % 3 - i % 3);
  
 }
  
 }
  
 return res;
  
}
  
  
string bfs(string start)
  
{
  
 heap.push({f(start) , start});
  
 dist[start] = 0;
  
  
 while(heap.size())
  
 {
  
 auto t = heap.top();
  
 heap.pop();
  
 cout<<"当前节点：state "<<t.second<<" 估值：" <<t.first<<endl;
  
 string state = t.second;
  
 int step = dist[state];//记录到达state的实际距离
  
 if(state == ed) break;//如果到达终点就break
  
  
 int k = state.find('x');
  
 int x = k / 3 , y = k % 3;
  
  
 string source = state;//因为在下面state会变，所以留一个备份
  
 for (int i = 0; i < 4; i ++ )
  
 {
  
 int a = x + dx[i], b = y + dy[i];
  
 if (a >= 0 && a < 3 && b >= 0 && b < 3)
  
 {
  
 swap(state[x \* 3 + y], state[a \* 3 + b]);
  
 if (!dist.count(state) || dist[state] > step + 1)
  
 {
  
 dist[state] = step + 1;
  
 pre[state] = {source, op[i]};
  
 heap.push({dist[state] + f(state), state});
  
 }
  
 swap(state[x \* 3 + y], state[a \* 3 + b]);//因为要多次交换，所以要恢复现场
  
 }
  
 }
  
  
 }
  
  
 string res;
  
 while(ed != start)
  
 {
  
 res += pre[ed].second;
  
 ed = pre[ed].first;
  
 }
  
 reverse(res.begin() , res.end());
  
 return res;
  
}
  
  
int main()
  
{
  
 string start , seq;
  
 for(int i = 0 ; i < 9 ; i++)
  
 {
  
 char c;
  
 cin >> c;
  
 start += c;
  
 if(c != 'x') seq += c;
  
 }
  
  
 int cnt = 0;
  
 for(int i = 0 ; i < 8 ; i ++)
  
 for(int j = i + 1 ; j < 8 ; j++)
  
 if(seq[i] > seq[j])
  
 cnt++;
  
  
 if(cnt % 2) puts("unsolvable");
  
 else cout << bfs(start) << endl;
  
  
 return 0;
  
}

**实验心得**

作为一种解决优化问题的算法，分支限界法非常有用。在实验中，我学习了如何使用分支限界法解决背包问题。在算法的实现过程中，需要一些重要的步骤，如界函数的计算和节点的加入和弹出等。通过实验，我更加深入地理解了这些步骤的意义。

此外，我还了解到了如何使用优先队列来维护待处理的节点集合，并在加入节点时，使用节点的上限值作为优先级来确定节点在队列中的位置。这种方法能够使得队列按照优先级从高到低依次弹出节点，从而保证算法的效率。

1. 分支限界法和回溯法的不同之处在于分支限界法用的是宽度优先的思想, 回溯法用的是深度优先的思想
2. 分支限界法的关键是对每一个节点计算它的目标函数值, 以及维护全局的low和up值, 还有一个全局的优先队列
3. low和up值一个是可行解, 一个是贪心求得的最优解, 答案应该位于[low,up]之间, 一般来说, 一开始求得的可行解在算法中用来剪枝, 并且该值可以被更新(一般用不是最终解的叶子节点所携带的值来更新该值).
4. 好的限界函数确实可以简化很多操作,如果一开始的限界函数就恰好是最终答案, 那么就再好不过了.一般用贪心法可以求得还不错的限界函数.

总的来说，这次实验让我更加深入地了解了分支限界法，并学习了如何使用优先队列来实现这个算法。这对我的算法学习和提高都有很大的帮助。