基于一般运输问题求解的编程实践

周佳豪 PB19051035

背景介绍

- 人们在从事生产活动中， 不可避免地要进行物资调运工作。 如某时期内将生产基地的煤、 钢铁、 粮食等各类物资， 分别运到需要这些物资的地区， 根据各地的生产量和需要量及各地之间的运输费用， 如何制定一个运输方案， 使总的运输费用最小。 这样的问题称为**运输问题**。
- 运输问题是一种特殊的线性规划问题， 它们的约束方程组的系数矩阵具有特殊的结构， 有可能找到比单纯形法更为简便的求解方法。
- 运输问题的特征：
  - 每一个出发地都有一定的供应量配送到目的地， 每一个目的地都有需要一定的需求量， 接收从出发地发出的产品。
  - 需求假设：
    每一个出发地都有一个固定的供应量， 所有的供应量都必须配送到目的地。 与之相类似， 每一个目的地都有一个固定的需求量， 整个需求量都必须由出发地满足， 即：**总供应量=总需求量**
  - 可行解特性：
    当且仅当供应量的总和等于需求量的总和时， 运输问题才有可行解。
  - 成本假设：
    从任何一个出发地到任何一个目的地的货物配送成本和所配送的数量成线性比例关系， 因此这个成本就等于配送的单位成本乘以所配送的数量。
  - 整数解性质：
    只要它的供应量和需求量都是整数， 任何有可行解的运输问题必然有所有决策变量都是整数的最优解。 因此， 没有必要加上所有变量都是整数的约束条件。

实验目的

- 本次实验希望通过编程来解决一般的运输问题，即向程序中输入**生产地数目**、**销地数目**、**每个生产地对应的生产量**、**每个销地对应的需求量**、**从某个生产地到某个某个消费地的单价运输费用**，程序会自动计算最优运输解，并求出所需费用的最小值。
- 在上条基础上也实现了产销不平衡问题的求解，即总产量和总需量不相等条件下运输问题的求解。

实验环境

本实验是用C++编写，在一般的C++环境下都能正确运行。

实验内容

理论知识

- 运输问题的数学模型：
  - 已知有m个生产地点A_i,i=1,2,…,m。 可供应某种物资， 其供应量(产量)分别为a_i， i=1,2,…,m，有n个销地B_j， j=1,2,…,n， 其需求量分别为b_j，j=1,2,…,n， 从A_i到B_j运输单位物资的运价(单价)为c_{ij}。这些数据可汇总于产销平衡表和单位运价表中，如图：

产销平衡表

产地 \ 销地	1,2,⋯,n	产量
1		a_1
2		a_2
⋮		⋮
m		a_m
销量	$b_1,b_2,\cdots,b_n$	

单位运价表

产地 \ 销地	1,2,⋯,n			
1	c_{11}	c_{12}	⋯	c_{1n}
2	c_{21}	c_{22}	⋯	c_{2n}
⋮	⋮	⋮	⋱	⋮
m	c_{m1}	c_{m2}	⋯	c_{mn}

  - 若用x_{ij}表示从A_i到B_j的运量， 那么在产销平衡的条件下， 要求得总运费最小的调运方案的数学模型为：

$$\min z=\sum_{i=1}^{m}\sum_{j=1}^{n}c_{ij}x_{ij}$$

$$\sum_{i=1}^{m}x_{ij}=b_j,j=1,2,\cdots,n$$

$$\sum_{j=1}^{n}x_{ij}=a_i,i=1,2,\cdots,m$$

$$x_{ij}\geqslant 0$$

$$\begin{array}{c}
\begin{matrix} x_{11} & x_{12} & \cdots & x_{1n} & x_{21} & x_{22} & \cdots & x_{2n} & \cdots & x_{m1} & x_{m2} & \cdots & x_{mn} \end{matrix} \\
\left[\begin{matrix}
1 & 1 & \cdots & 1 & & & & & & & & & \\
 & & & & 1 & 1 & \cdots & 1 & & & & & \\
 & & & & & & & & \ddots & & & & \\
 & & & & & & & & & 1 & 1 & \cdots & 1 \\
1 & & & & 1 & & & & & 1 & & & \\
 & 1 & & & & 1 & & & & & 1 & & \\
 & & \ddots & & & & \ddots & & & & & \ddots & \\
 & & & 1 & & & & 1 & & & & & 1
\end{matrix}\right]
\begin{matrix} \} \text{m行} \\ \\ \\ \\ \} \text{n行} \end{matrix}
\end{array}$$

    - 它包含m× n个变量， (m+n)个约束方程， 其系数矩阵结构比较松散且特殊。
    - 该系数矩阵中对应于变量xij的系数向量Pij， 其分量中除第i个和第m+j个为1以外， 其余的都为零。 即:$P_{ij}=(0,\ldots,1,0,\ldots,0,1,0,\ldots,0)^T=e_i+e_{m+j}$
    - 对于产销平衡的运输问题， 由于有以下关系式存在：$\sum_{j=1}^{n}b_j=\sum_{i=1}^{m}(\sum_{j=1}^{n}x_{ij})=\sum_{j=1}^{n}(\sum_{i=1}^{m}x_{ij})=\sum_{i=1}^{m}a_i$,所以模型最多只有 m+n-1个独立的约束方程， 即系数矩阵秩≤m+n-1
    - 运输问题存在可行解，因此必存在有限最优解
      - 由于总供求=总需求，显然：

$$x_{ij} = \frac{a_i b_j}{\sum_{i=1}^{m} a_i}$$

构成了一组可行解

- 又由于 $0 \le x_{ij} \le \min(a_i, b_j)$ 有界， 因而存在最优解

- 求解一般运输问题的过程：
  - **表上作业法是单纯形法在求解运输问题时的一种简化方法，其实质是单纯形法， 也称为运输问题单纯形法。**
  - **表上作业法的步骤如下：**
    1. 找出初始基可行解。 在有（m× n） 格的产销平衡表按一定规则，给出m+n-1个数字,称为数字格,即初始基变量的值；
    2. 求各非基变量的检验数。 在表上计算空格的检验数， 判别是否为最优解，是则停止运算，否则转到下一步；
    3. 确定换入变量和换出变量， 找出新的基可行解—在表上用闭回路法调整；
    4. 重复上述步骤2和3直到得到最优解。
  - **表上作业法求解步骤的具体实现：**
    - 确定初始基可行解的常用方法有三种： 西北角法，最小元素法和伏格尔法。本实验采用的是西北角法，故在此只说明西北角法的基本思想：从产销平衡表的西北角开始，顺序选定基变量,具体流程为：
      1. 初始化：选择西北角变量 $x_{ij} = x_{11}$ 为基变量
      2. 确定基变量的值：$x_{ij} = \min(a_i, b_j)$。 如果 x_{ij} 为最后一个元素，停止。否则，进入下一步。
      3. 更新产销平衡表： $a_i \leftarrow a_i - x_{ij}$ ， $b_j \leftarrow b_j - x_{ij}$
      4. 修改产销平衡表：划去 $a_i = 0$ 的A_i行或 $b_j = 0$的B_j列
      5. 选择下一个基变量 ： 如果 $a_i = 0$，$x_{ij} \leftarrow x_{i+1,j}$ （ 下移一行)。否则$x_{ij} \leftarrow x_{i,j+1}$ （ 右移一列)
      6. 重复2-5
    - 最优解的判别一般是通过检验数的计算。计算非基变量的检验数$o_{ij}=c_{ij}-C_B B^{-1} P_{ij}$,i,j=1,2,3...,因运输问题要求目标函数最小（即总花费最小） ，故当所有的检验数均>=0时，该解为最优解。求检验数的方法主要包括**闭回路法**和**位势法**两种。本次实验采用的是位势法，故在此只介绍位势法的原理：
      - 位势法是根据对偶理论推导出来的一种方法
      - 设产地约束和销地约束对应的对偶变量分别为u_i和v_j,则有以下公式：

$$\min z=\sum_{i=1}^{m}\sum_{j=1}^{n}c_{ij}x_{ij}$$

$$\begin{cases}\sum_{i=1}^{m}x_{ij}=b_j, j=1,2,\cdots,n\\ \sum_{j=1}^{n}x_{ij}=a_i, i=1,2,\cdots,m\\ x_{ij}\geq 0\end{cases}$$

$$\Longrightarrow$$

$$\max \omega=\sum_{i=1}^{m}a_iu_i+\sum_{j=1}^{n}b_jv_j$$

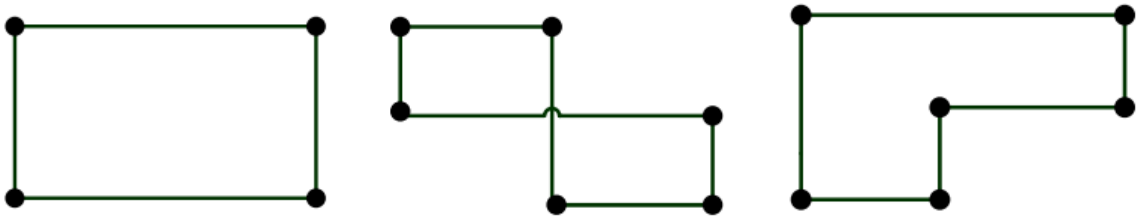
$$u_i+v_j\leqslant c_{ij},\ (i=1,2,\cdots,m;j=1,2,\cdots,n)$$

u_i,v_j无约束

- 设 $u_1,u_2,...u_m$，$v_1,v_2,...,v_n$是对应产销平衡的运输问题的m+n个约束条件的对偶变量。
- B是含有一个人工变量 x_a的(m+n)x(m+n)初始基矩阵
- 当用西北角法给出初始基可行解时，有m+n-1个数字格，即m+n-1个基变量
- 从线性规划的对偶理论可知：$C_BB^{-1}=(u_1,u_2,...,u_m;v_1,v_2,...,v_n)$
- 每个决策变量X_{ij}的系数列向量为：$P_{ij}=e_i+e_{m+j}$
- 所以$C_BB^{-1}P_{ij}=u_i+v_j$
- 于是检验数$o_{ij}=c_{ij}-(u_i+v_j)$
- 由单纯形法，基变量对应的检验数为0，可得：$c_{ij}-(u_i+v_j)=0$，i，j∈B
- 上述方程组有m+n个变量，只有m+n-1个约束方程，故存在自由变量
- 为简单起见，首先设$u_1=0$,可以求解，得到一组u_i，v_j的值
- 求出非基变量检验数：$o_{ij}=c_{ij}-(u_i+v_j)$,i,j∈N
- 称u_i，v_j为运输问题的对偶解，又称为**位势**
- 根据自变量u_1取值的不同，可以得到无穷多组解，但不同解对应的检验数都是相同的，是唯一的

- 在计算非基变量的检验数时，一般选其中最小的负检验数，以它对应的空格为调入格， 即以它对应的非基变量为换入变量。再利用闭回路调整法找出换出变量，具体步骤为：从一空格（即将换入的非基变量） 出发找一条闭回路：**沿着纵向或横向行进， 遇到适当数字格（基变量） 时90度转弯， 继续行进， 直到回到起始空格为止。**

  关于闭回路的性质：
  - 集合中的变量称为闭回路的顶点， 相邻两个变量的连线为闭回路的边
  - 闭回路的顶点间用水平或垂直线段连接起来， 组成一条封闭的回路。
  - 闭回路的顶点数一定是偶数。



  - 将闭回路定义中的向量组写成列向量的形式并分别乘以正负号线性组合后等于零,即：

$$P_{i_1 j_1} - P_{i_1, j_2} + P_{i_2, j_2} - \cdots - P_{i_s, j_1} = 0$$

- 从每一空格出发一定存在和可以找到唯一的闭回路。因(m+n—1)个数字格(基变量)对应的系数向量是一个基。任一空格(非基变量)对应的系数向量是这个基的线性组合。
- **闭回路的经济解释：** 以用西北角法求得的初始解为例：可以从任一空格出发， 如（A_1,B_1） ，若让A_1的产品调运1吨给B_1，为了保持产销平衡，就要依次做调整：在（A_1,B_3）处减少1吨， （ A_2,B_3） 处增加1吨， （ A_2,B_1） 处减少1吨，即构成了以（A_1,B_1）空格为起点，其它为数字格的闭回路。如图所示：

销地 / 产地	B_1	B_2	B_3	B_4	产量
A_1	(+1)		4(-1)	3	7
A_2	3(-1)		1(+1)		4
A_3		6		3	9
销量	3	6	5	6	

产地 / 销地	B_1	B_2	B_3	B_4
A_1	3	11	3	10
A_2	1	9	2	8
A_3	7	4	10	5

- **闭回路调整法步骤：**选择闭回路上具有 (-1) 的数字格中的最小者，即 θ = min(4,3) = 3。对应的基变量为换出变量。然后按照闭回路上的正负号， 加入和减去此值， 得到调整方案， 并根据调整后的方案重新计算检验数。

- 可能会出现的一些问题：
  - 无穷多最优解：即某个非基变量（空格） 的检验数为 0 时，该问题有无穷多最优解，但从这些解得到的最小花费是一样的
  - 退化：当确定初始解的各供需关系时， 若 A_i 处的余量等于 B_j 处的需量， 在产销平衡表的 (i,j) 格填入一个数后， 需要在单位运价表上同时划去 A_i行和 B_j列。为了使产销平衡表上有 m+n-1 个数字格（基变量） ， 这时需要添加一个 0，0的位置可在对应同时划去的那行或那列的任一空格处 。
- 关于产销不平衡问题的解决：
  - 实际问题中产销往往是不平衡的， 就需要把产销不平衡的问题化成产销平衡的问题。
  - 当产大于销时， 只要增加一个假想的销地 j=n+1 (实际上是储存)， 该销地总需要量为**总产量-总销量**，在单位运价表中从各产地到假想销地的单位运价为 0， 就转化为一个产销平衡的运输问题.
  - 当销大于产时， 可以在产销平衡表中增加一个假想的产地 i=m+1， 该地产量为:**总销量-总产量**，在单位运价表上令从该假想产地到各销地的运价为0 ， 同样可以转化为一个产销平衡的运输问题

编程实现

- 代码中定义了几个大数组用于储存工作

```
 //代表共有m个生产地点，n个销地
 int m = 0, n = 0;
 //prodecer[i]代表第i的生产地点的生产量
 //consumer[i]代表第i个消费地点的需求量
int producer[100], consumer[100];
//c[i][j]表示从生产地i到销地j的运输费用
int c[100][100] = {0};
//base[i][j]表示从生产地i到销地j的运输量
int base[100][100] = {0};
//is_base[i][j]表示该点是否为基变量
int is_base[100][100] = {0};
```

- 西北角法求初始基可行解

> 主要采用递归的方法，并将基本量的is_base置为1

```
void northwest(int i, int j)
{
  if (i >= 1 && i <= m && j >= 1 && j <= n)
  {
      is_base[i][j] = 1;
      base[i][j] = min(producer[i], consumer[j]);
      producer[i] = producer[i] - base[i][j];
      consumer[j] = consumer[j] - base[i][j];
      if (producer[i] == 0 && consumer[j] != 0)
      {
          northwest(i + 1, j);
      }
      else if (producer[i] != 0 && consumer[j] == 0)
      {
          northwest(i, j + 1);
      }
      else if (producer[i] == 0 && consumer[j] == 0)
      {
          //防止出现退化
          base[i + 1][j] = 0;
          is_base[i + 1][j] = 1;
          northwest(i + 1, j + 1);
      }
  }
}
```

- 位势法验证是否为最优解

> 传入的变量i，j用于记录换入变量
> 由于位势法需要解方程，即对数组u，v进行赋值，故设置了u_begin，v_begin两个数组用于记录数组u,v是否已经被赋值

> 程序是顺序求解方程c[i][j]=u[i]+v[j],故可能会出现u[i]和v[j]的值都不知道的情况，即u_begin[i]和v_begin[j]均为0，这时需要先跳过该方程，故一次for循环遍历并不能全部求解方程组，故需要使用while(1)循环来保证方程组的全部求解。在确定方程组全部求解之后，用break跳过while循环。
> 求解上述全部方程后，即可求得非基变量检验数，然后判断无穷多最优解，唯一最优解，不是最优解三种情况。

```
//位势法求解，找到换入变量
int u[100], v[100];
//代表是否已经被赋值
int u_begin[100], v_begin[100];
//检验数
int o[100][100];
//若不是最优解，则返回值为值最小的检验数
//若是最优解：无穷多最优解返回1，唯一最优解返回0
int position(int *i, int *j)
{
    int a, b;
    int start = 0;
    memset(u, 0, sizeof(u));
    memset(v, 0, sizeof(v));
    memset(u_begin, 0, sizeof(u_begin));
    memset(v_begin, 0, sizeof(v_begin));
    memset(o, 0, sizeof(o));
    //令基变量xij的检验数全为0，求得相应的u[i],v[j]
    while (1)
    {
        for (a = 1; a <= m; a++)
            for (b = 1; b <= n; b++)
            {
                if (is_base[a][b] == 1)
                {
                    if (start == 0)
                    {
                        start = 1;
                        u[a] = 0;
                        u_begin[a] = 1;
                    }
                    if (u_begin[a] == 1 && v_begin[b] == 0)
                    {
                        v[b] = c[a][b] - u[a];
                        v_begin[b] = 1;
                    }
                    if (u_begin[a] == 0 && v_begin[b] == 1)
                    {
                        u[a] = c[a][b] - v[b];
                        u_begin[a] = 1;
                    }
                }
            }
        //判断是否每个u[i],v[j]都已正确赋值
        for (a = 1; a <= m; a++)
        {
            if (start == 0)
                break;
            for (b = 1; b <= n; b++)
            {
                if (is_base[a][b] == 1)
```

```
            {
                if (u_begin[a] == 0 || v_begin[a] == 0)
                {
                    //表示还未完全赋值
                    start = 0;
                    break;
                }
            }
        }
    }
    if (start == 1)
        break;
}
//求得所有变量的检验数
for (a = 1; a <= m; a++)
    for (b = 1; b <= n; b++)
    {
        o[a][b] = c[a][b] - u[a] - v[b];
    }
//检查检验数的值是否正确
for (a = 1; a <= m; a++)
    cout << "u[" << a << "]=" << u[a] << endl;
for (b = 1; b <= n; b++)
    cout << "v[" << b << "]=" << v[b] << endl;
for (a = 1; a <= m; a++)
    for (b = 1; b <= n; b++)
    {
        cout << "o[" << a << "][" << b << "]=" << o[a][b] << endl;
        //o[a][b] = c[a][b] - u[a] - v[b];
    }
//返回检验数最小的i，j
int temp_a, temp_b, k;
k = o[1][1];
for (a = 1; a <= m; a++)
    for (b = 1; b <= n; b++)
    {
        if (o[a][b] < k)
        {
            k = o[a][b];
            temp_b = b;
            temp_a = a;
        }
    }
if (k == 0)
{
    for (a = 1; a <= m; a++)
        for (b = 1; b <= n; b++)
        {
            if (is_base[a][b] == 0 && o[a][b] == 0)
            {
                cout << "There are endless Optimal solutions" << endl;
```

```
                    *i = 0;
                    *j = 0;
                    return 1;
                }
            }
        cout << "There is only one Optimal solution" << endl;
        return 0;
    }
    if (k < 0)
    {
        *i = temp_a;
        *j = temp_b;
    }
    return k;
}
```

- 当位势法得出不是最优解时需要使用闭回路调整法改变基变量

> 由于闭回路的寻找需要遍历每个点的条件邻居（条件邻居和普通的邻居不一样，因为闭回路找到一个点后需要旋转90度，故该邻居只能是一个方向上的邻居，即横向或纵向），故使用DFS搜索，即深度优先搜索算法，故设置一个简易的栈用于储存基变量的位置
> 由于需要得知整个闭回路上的基变量位置，故设置一个结构体point，里面储存指向该节点的父节点。结构体数组point与栈共同使用，来查找并储存闭回路

```
typedef struct stack
{
    int i;
    int j;
};

const int maxStackSize = 100000;
const int M = 1000000;
stack s[maxStackSize];
int tail = 0;
void initial_stack()
{
    tail = 0;
    for (int a = 0; a < maxStackSize; a++)
    {
        s[a].i = 0;
        s[a].j = 0;
    }
    tail = 0;
}
int is_empty_stack()
{
    if (tail == 0)
        return 1;
    return 0;
}
void push_stack(int x, int y)
{
    s[tail].i = x;
    s[tail].j = y;
    tail++;
}
void pop_stack(int *x, int *y)
{
    *x = s[tail - 1].i;
    *y = s[tail - 1].j;
    tail--;
}
typedef struct point
{
    int x;
    int y;
    int visited;
    int parent;    //指向该点的父节点
    int direction; //该点指向的方向
};
point p[maxStackSize];

//闭回路调整法找到换出变量,并进行基变量互换
//换入变量为Xij
//定义上下方向为1，左右方向为0
```

```
//返回值为闭回路终点对应的point结构体数组的下标
int closed_loop(int i, int j)
{
    cout << "i=" << i << ", j=" << j << endl;
    int a, b;
    int parents, k, w, flag;
    int timeT = 0;
    is_base[i][j] = 1;
    //运用DFS搜索找到一个闭回路
    //并对闭回路上的所有基变量进行下一步操作
    initial_stack();
    push_stack(i, j);
    p[0].x = i;
    p[0].y = j;
    p[0].visited = 0;
    p[0].parent = -1;
    p[0].direction = -1;

    int arr = 1;
    //timeT的使用是为了避免无限循环
    //maxStackSize可认为是一个比较大的数
    while (is_empty_stack() != 1 && timeT++ < maxStackSize)
    {
        pop_stack(&a, &b);
        cout << "pop stack!" << endl;
        cout << "a=" << a << ",b=" << b << endl;
        //flag用于判断该点是否已经被访问
        //若已访问，则拒绝该点的邻居入栈
        //每个点的操作顺序是先入栈，再出栈，访问该点，条件邻居入栈
        //由于是闭回路，故初始点可指向上下，左右任一个方向，这里规定只能初始点的左右邻居入栈，即初始点可
        for (k = 0, flag = 0; k < arr; k++)
        {
            if (p[k].x == a && p[k].y == b)
            {
                if (p[k].visited == 1)
                    flag = 1;
                else
                {
                    p[k].visited = 1;
                    parents = k;
                }
            }
        }
        if (flag == 0)
        {
            //该点的邻居入栈
            cout << "aid is :" << endl;
            cout << "a=" << p[parents].x << ",b=" << p[parents].y << endl;
            cout << "visited = " << p[parents].visited << endl;
            cout << "direction = " << p[parents].direction << endl;
            if (p[parents].direction == 0)
```

```
            {
                //上下方向的邻居入栈
                for (int c = 1; c <= m; c++)
                {
                    if (is_base[c][p[parents].y] == 1)
                    {
                        if (c == p[parents].x)
                            continue;
                        push_stack(c, p[parents].y);
                        cout << "push stack!" << endl;
                        cout << "a=" << c << ",b=" << p[parents].y << endl;
                        p[arr].x = c;
                        p[arr].y = p[parents].y;
                        p[arr].visited = 0;
                        p[arr].direction = 1;
                        p[arr].parent = parents; //parent为数组的下标
                        if (c == i && p[parents].y == j)
                            return arr;
                        arr++;
                    }
                }
            }
            else
            {
                //左右方向的邻居入栈
                for (int c = 1; c <= n; c++)
                {
                    if (is_base[p[parents].x][c] == 1)
                    {
                        if (c == p[parents].y)
                            continue;
                        push_stack(p[parents].x, c);
                        cout << "push stack!" << endl;
                        cout << "a=" << p[parents].x << ",b=" << c << endl;
                        p[arr].x = p[parents].x;
                        p[arr].y = c;
                        p[arr].visited = 0;
                        p[arr].direction = 0;
                        p[arr].parent = parents;
                        if (p[parents].x == i && c == j)
                            return arr;
                        arr++;
                    }
                }
            }
        }
    }

    return 0;
}
```

- 由于位势法和闭回路调整法是成对出现的（即只要位势法求得的不是最优解就需要使用闭回路调整法来更换基变量，再使用位势法判断是否为最优解），故在main函数里使用while循环，直至位势法判断解为最优解。

> 遍历整个闭回路需要使用结构体point中的parent成员变量
> 其中函数closed_loop()返回值为闭回路终点对应的point结构体数组的下标
> 当解不是最优解时，需要使用闭回路法调整，即对基变量进行+1，-1操作，由于闭回路上+1，-1总是依次出现，故使用一个变量s，用s是否能整除2来判断该变量是+1还是-1。
> 找到-1基变量中的最小值（即换出变量），使该点的is_base变为0，使每个-1的基变量减去这个值，每个+1的基变量以及换入变量加上这个值，并使换入变量的is_base变为1.

```
//位势法求最优解，找到换入变量
    int o = position(&i, &j);
    while (o < 0)
    {
        int arr = closed_loop(i, j);
        temp = arr;
        if (arr == 0)
            cout << "error!" << endl;
        else
        {
            //找到-1项中数值最小的基变量
            s = 1;
            t = maxStackSize;
            while (temp != 0)
            {
                if (s % 2 == 0)
                {
                    if (base[p[temp].x][p[temp].y] < t)
                    {
                        t = base[p[temp].x][p[temp].y];
                        temp_x = p[temp].x;
                        temp_y = p[temp].y;
                    }
                }
                s++;
                temp = p[temp].parent;
            }
            cout << "temp_x=" << temp_x << ",temp_y=" << temp_y << endl;
            is_base[temp_x][temp_y] = 0;
            temp = arr;
            s = 1;
            while (temp != 0)
            {
                if (s % 2 == 0)

                    base[p[temp].x][p[temp].y] -= t;
                else
                    base[p[temp].x][p[temp].y] += t;
                s++;
                temp = p[temp].parent;
            }
        }
        o = position(&i, &j);
    }
```

实验结果

- 本次实验准备四组测试数据，前三组均为运筹学课本的课后习题，第四组为网上查找，答案已验证是正确的
  - **test1（产销平衡）**

产地＼销地	B_1	B_2	B_3	B_4	产量
A_1					7
A_2					4
A_3					9
销量	3	6	5	6	

产地＼销地	B_1	B_2	B_3	B_4
A_1	3	11	3	10
A_2	1	9	2	8
A_3	7	4	10	5

输出结果：

```
There are endless Optimal solutions
solution is :
The transportation form producer 1 to consumer 1 is 2
The transportation form producer 1 to consumer 3 is 5
The transportation form producer 2 to consumer 1 is 1
The transportation form producer 2 to consumer 4 is 3
The transportation form producer 3 to consumer 2 is 6
The transportation form producer 3 to consumer 4 is 3
The minimum total cost is 85
```

可知有无穷多最优解，这里只给出了一个

  - **test2（产大于销）**

产地＼销地	甲	乙	丙	丁	戊	产量
1	10	20	5	9	10	5
2	2	10	8	30	6	6
3	1	20	7	10	4	2
4	8	6	3	7	5	9
销量	4	4	6	2	4	

```
There are endless Optimal solutions
solution is :
The transportation form producer 1 to consumer 3 is 3
The transportation form producer 2 to consumer 1 is 4
The transportation form producer 2 to consumer 5 is 2
The transportation form producer 3 to consumer 5 is 2
The transportation form producer 4 to consumer 2 is 4
The transportation form producer 4 to consumer 3 is 3
The transportation form producer 4 to consumer 4 is 2
The minimum total cost is 90
```

可知有无穷多最优解，这里只给出了一个

- **test3（产大于销）**

产地＼销地	甲	乙	丙	丁	戊	产量
1	10	18	29	13	22	100
2	13	M	21	14	16	120
3	0	6	11	3	M	140
4	9	11	23	18	19	80
5	24	28	36	30	34	60
销量	100	120	100	60	80	

```
There are endless Optimal solutions
solution is :
The transportation form producer 1 to consumer 1 is 100
The transportation form producer 2 to consumer 3 is 40
The transportation form producer 2 to consumer 5 is 80
The transportation form producer 3 to consumer 1 is 0
The transportation form producer 3 to consumer 2 is 20
The transportation form producer 3 to consumer 3 is 60
The transportation form producer 3 to consumer 4 is 60
The transportation form producer 4 to consumer 2 is 80
The transportation form producer 5 to consumer 2 is 20
The minimum total cost is 5520
```

可知有无穷多最优解，这里只给出了一个

- **test4（销大于产）**

输入：

```
Input the total of producer:3
Input the total of consumer:4
Input the production of producer 1 :7
Input the production of producer 2 :4
Input the production of producer 3 :9
Input the consumption of consumer 1 :30
Input the consumption of consumer 2 :6
Input the consumption of consumer 3 :5
Input the consumption of consumer 4 :6
sum_produce = 20,sum_consume = 47
Input the cost from producer 1 to consumer 1 :3
Input the cost from producer 1 to consumer 2 :11
Input the cost from producer 1 to consumer 3 :3
Input the cost from producer 1 to consumer 4 :10
Input the cost from producer 2 to consumer 1 :1
Input the cost from producer 2 to consumer 2 :9
Input the cost from producer 2 to consumer 3 :2
Input the cost from producer 2 to consumer 4 :8
Input the cost from producer 3 to consumer 1 :7
Input the cost from producer 3 to consumer 2 :4
Input the cost from producer 3 to consumer 3 :10
Input the cost from producer 3 to consumer 4 :5
```

输出：

```
There are endless Optimal solutions
solution is :
The transportation form producer 1 to consumer 1 is 7
The transportation form producer 2 to consumer 1 is 4
The transportation form producer 3 to consumer 2 is 6
The transportation form producer 3 to consumer 4 is 3
The minimum total cost is 64
```

可知有无穷多最优解，这里只给出了一个

总结

- 本次实验使用编程求解了本人运筹学课上学到的运输问题，在做这章作业的时候计算量特别大，一道题能写十页纸张，在此实验课上用编程求解了该问题，收获巨大。个人感觉这部分最难的是闭回路的查找与储存，这里我运用了数据结构学到的DFS算法以及使用结构体储存闭回路，在这里特别感谢孙广中老师在课堂上讲述的编程思想。

参考资料和文献

[1].清华大学运筹学教材编写组.运筹学第4版.清华大学出版,1982.
[2].清华大学运筹学教材编写组.运筹学解题指导.清华大学出版社,2006.
[3].顾乃杰,黄章进.运筹学基础（运输问题）.合肥：中国科学技术大学，2021.