小组成员：成员1 成员2 成员3

目 录

[第2章 线性表 3](#_Toc127888826)

[2.1 顺序表基本操作的实现 3](#_Toc127888827)

[2.2 单链表基本操作的实现 4](#_Toc127888828)

[2.3 单链表原地逆置 4](#_Toc127888829)

[2.4 单链表排序 4](#_Toc127888830)

[2.5 两个有序表合并 5](#_Toc127888831)

[2.6 利用单链表求两个集合的交集 5](#_Toc127888832)

[2.7 利用单链表判断两个集合是否相等 5](#_Toc127888833)

[2.8 删除单链表中的重复元素 6](#_Toc127888834)

[2.9 使用单链表实现一元多项式相加 6](#_Toc127888835)

[第3章 栈和队列 6](#_Toc127888836)

[3.1 顺序栈基本操作的实现 6](#_Toc127888837)

[3.2 链栈基本操作的实现 7](#_Toc127888838)

[3.3 循环队列基本操作的实现 7](#_Toc127888839)

[3.4 链队列的基本操作的实现 7](#_Toc127888840)

[3.5 Hanio塔问题 8](#_Toc127888841)

[3.6 利用顺序栈实现进制转换 8](#_Toc127888842)

[3.7 表达式括号匹配问题 8](#_Toc127888843)

[3.8 后缀表达式求值问题 9](#_Toc127888844)

[3.9 中缀表达式求值问题 9](#_Toc127888845)

[3.10 中缀表达式转换成后缀表达式问题 9](#_Toc127888846)

[第4章 字符串和多维数组 10](#_Toc127888847)

[4.1 字符串的BF匹配算法的实现 10](#_Toc127888848)

[4.2 字符串的KMP匹配算法的实现 11](#_Toc127888849)

[4.3 对称矩阵的压缩存储 13](#_Toc127888850)

[4.4 上三角矩阵的压缩存储 16](#_Toc127888851)

[4.5 下三角矩阵的压缩存储 18](#_Toc127888852)

[4.6 三对角矩阵的压缩存储 21](#_Toc127888853)

[4.7 稀疏矩阵的压缩存储 24](#_Toc127888854)

[4.8 约瑟夫环问题 27](#_Toc127888855)

[4.9 求解矩阵的马鞍点 29](#_Toc127888856)

[4.10 螺旋方阵问题 32](#_Toc127888857)

[4.11 幻方问题 34](#_Toc127888858)

[第5章 树和二叉树 38](#_Toc127888859)

[5.1 二叉链表基本操作的实现 38](#_Toc127888860)

[5.2 二叉树遍历的非递归实现 38](#_Toc127888861)

[5.3 求二叉树的最小深度 39](#_Toc127888862)

[5.4 判断二叉树是否是完全二叉树 39](#_Toc127888863)

[5.5 判断二叉树的结构是否对称 39](#_Toc127888864)

[5.6 判断二叉树是否对称 40](#_Toc127888865)

[5.7 求二叉树中叶子结点个数及结点个数 40](#_Toc127888866)

[5.8 中序线索二叉树的构造 40](#_Toc127888867)

[5.9 哈夫曼算法的实现 41](#_Toc127888868)

[5.10 由前序遍历序列和中序遍历序列构造二叉树 41](#_Toc127888869)

[5.11 由后序遍历序列和中序遍历序列构造二叉树 41](#_Toc127888870)

[5.12 求二叉树第k层的结点个数和叶子结点个数 41](#_Toc127888871)

[5.13 打印二叉树第k层的结点和叶子结点 42](#_Toc127888872)

[5.14 求二叉树最大的结点距离 42](#_Toc127888873)

[第6章 图 43](#_Toc127888874)

[6.1 邻接矩阵存储无向图的实现 43](#_Toc127888875)

[6.2 邻接表存储有向图的实现 43](#_Toc127888876)

[6.3 Prim算法的实现 43](#_Toc127888877)

[6.4 Kruscal算法的实现 44](#_Toc127888878)

[6.5 Dijkstra算法的实现 44](#_Toc127888879)

[6.6 Floyd算法的实现 45](#_Toc127888880)

[6.7 拓扑排序的实现 45](#_Toc127888881)

[6.8 选址问题 46](#_Toc127888882)

[6.9 七巧板涂色问题 46](#_Toc127888883)

[6.10 五叉路口交通灯问题 46](#_Toc127888884)

[6.11 农夫过河问题 46](#_Toc127888885)

[第7章 查找 47](#_Toc127888886)

[7.1 线性查找的实现 47](#_Toc127888887)

[7.2 折半查找的实现 47](#_Toc127888888)

[7.3 二叉排序树基本操作的实现 47](#_Toc127888889)

[7.4 散列查找的实现 47](#_Toc127888890)

[第8章 排序 47](#_Toc127888891)

[8.1 直接插入排序的实现 47](#_Toc127888892)

[8.2 Shell排序的实现 48](#_Toc127888893)

[8.3 冒泡排序的实现 48](#_Toc127888894)

[8.4 快速排序的实现 48](#_Toc127888895)

[8.5 简单选择排序的实现 48](#_Toc127888896)

[8.6 堆排序的实现 48](#_Toc127888897)

[8.7 非递归实现二路归并排序 48](#_Toc127888898)

说明：

（1）实训从教材第2章线性表开始；

（2）实训项目的实训数据可参考实训教材，例如树和二叉树、图等，考虑到部分同学没有教材，本模板中已对实验数据做了详细说明。

（3）如有必要使用多组实训数据的，则使用多组，例如求解马鞍点，分有一个马鞍点，没有马鞍点，有多个马鞍点等多种情况，鼓励使用多组结果不同的实训数据；

（4）实验运行截图需要更改为白底黑字，不要使用黑底白字，否则截图不清楚；

**请不要使用本模板中的实验结果截图，**需要自己做。

（5）最终提交实训报告时，可将此说明删掉；请务必注意不要修改本实训报告的默认样式，特别是关于代码的，代码不要任何样式（不要颜色，不要加粗，不要斜体等），只需要注意缩进规范即可；

（6）实训报告中的第3部分，遇到的问题和解决的办法在实训中需要如实记录，这是比较重要的部分，认真记录的小组会得到较好的成绩；

（7）凡是本报告中二级标题中没有相应说明的，参照教材的要求；

（8）所有参与运算操作的数据尽量输入，不要设置固定值。

（9）实训每个小组有实训答辩，最后上交的材料有实训报告、实训答辩ppt和代码（要求去掉exe等调试和运行程序）

# 第2章 线性表

## 2.1 顺序表基本操作的实现

说明：实现SeqList类模板，基本操作包括无参构造函数、有参构造函数、求顺序表的表长、按位查找、按值查找、插入、删除、遍历等，并在主函数中对各种操作进行验证。

1. 实验代码

/\*在此补充代码，如果有多个文件，则分多个文件\*/

（1）SeqList.h

……

（2）SeqList.cpp

……

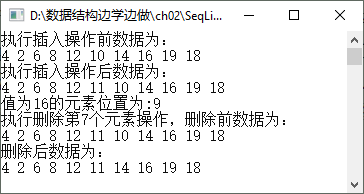
（3）SeqListMain.cpp

……

2. 实验运行结果

/\*在此给出运行截图，给出实验数据，如果实验数据是图或树，还需要画出图或树的结构\*/

实验数据可参考下图：



3. 遇到的问题和解决的办法

/\*在此如实记录解决此问题中遇到的问题及解决方法\*/

（1）……

（2）……

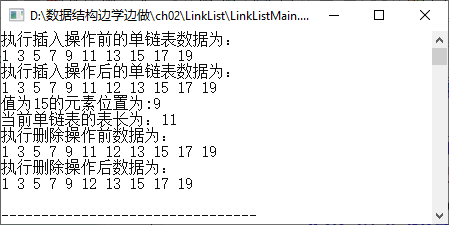
（3）……

## 2.2 单链表基本操作的实现

说明：实现LinkList类模板，基本操作包括无参构造函数、有参构造函数、析构函数，求单链表的表长，按位查找，按值查找、插入元素、删除元素、遍历整个单链表等，并在主函数中对各操作进行验证，本项目需要使用多文件实现。

/\*此后的每一个实验项目都参照2.1的格式\*/

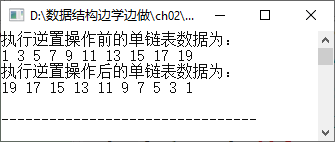
实验数据可参考下图：



## 2.3 单链表原地逆置

说明：借助于LinkList类模板，实现单链表的构造、原地逆置、分别遍历逆置前后的单链表。原地逆置指的是不能申请新的结点空间。

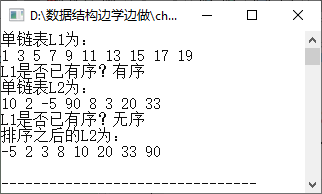
实验数据可参考下图：



## 2.4 单链表排序

说明：借助于LinkList类模板，实现单链表的构造、用直接插入法对单链表进行排序、遍历排序前后的单链表。

实验数据可参考下图：



## 2.5 两个有序表合并

说明：借助于LinkList类模板，构造两个有序表，将两个有序表合并成一个有序表，分别遍历两个有序表，再遍历合并后的有序表。

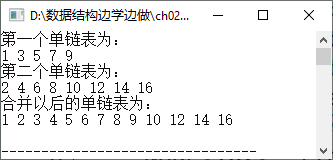
例如：

两个有序表：

1，3，5，7，9

2，4，6，8，10，12

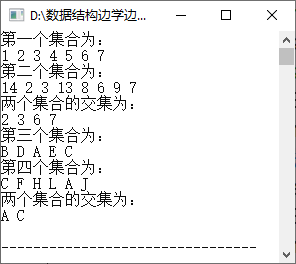
实验数据可参考下图：



## 2.6 利用单链表求两个集合的交集

说明：借助于LinkList类模板，构造两个单链表分别存储两个集合，求两个单链表都包含的元素结点，遍历两个单链表，输出两个单链表都包含的元素结点。

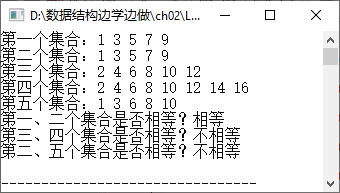
实验数据可参考下图：



## 2.7 利用单链表判断两个集合是否相等

说明：借助于LinkList类模板，构造两个单链表分别存储两个集合，判断两个单链表是否相等。可先将单链表按元素结点值排序，然后再判断所包含的元素值是否相等。分别遍历两个单链表，给出相等及不相等的实验数据。

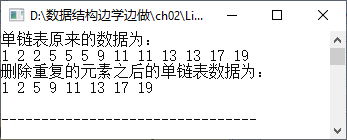
实验数据可参考下图：



## 2.8 删除单链表中的重复元素

说明：借助于LinkList类模板，构造单链表，将单链表中的所有重复元素删除，有多个值相等的，只保留一个，分别遍历删除前后的单链表。构造单链表的值需要输入。

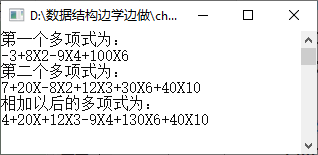
实验数据可参考下图：



## 2.9 使用单链表实现一元多项式相加

说明：借助于LinkList类模板，实现两个一元多项式相加。使用单链表分别存储一元多项式，将两个一元多项式相加，分别输出两个一元多项式及合并以后的多项式。所有数据需要输入。需要注意一元多项式的输出格式，例如：A(x)=12X6-7X3-8X2+100，多项式中的指数可放到变量后，最高的指数项前的+号省略，其它项的+号不能省略。

实验数据可参考下图：

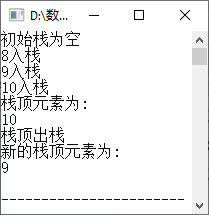


# 第3章 栈和队列

## 3.1 顺序栈基本操作的实现

说明：实现SeqStack类模板，基本操作包括构造函数、取栈顶、出栈、入栈、判空、判满，并在主函数中验证。

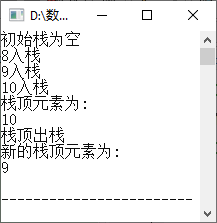
实验数据可参考下图：



## 3.2 链栈基本操作的实现

说明：实现LinkStack类模板，基本操作包括构造函数、析构函数、取栈顶、出栈、入栈、判空，并在主函数中验证。

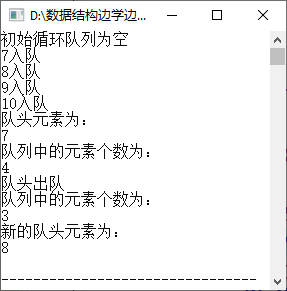
实验数据可参考下图：



## 3.3 循环队列基本操作的实现

说明：实现CirQueue类模板，基本操作包括构造函数、入队、出队、取队头、判空、判满，并在主函数中验证。

实验数据可参考下图：



## 3.4 链队列的基本操作的实现

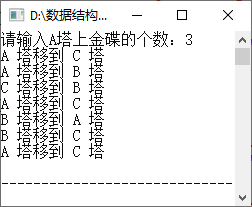
说明：实现LinkQueue类模板，基本操作包括构造函数、析构函数、入队、出队、取队头、判空，并在主函数中验证。

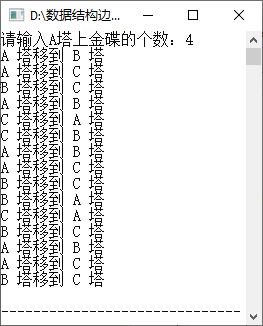
实验数据与循环队列类似。

## 3.5 Hanio塔问题

说明：递归实现Hanio塔问题，对于n个盘子，给出详细的解决方案，盘子的个数n=3，4，5分别作为测试数据。

实验数据可分别参考下列各图：

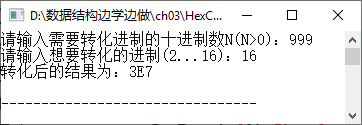




## 3.6 利用顺序栈实现进制转换

说明：利用SeqStack类模板，将一个非负的十进制数转换成2-16进制之间的数。输入十进制数，以及转换以后的进制，例如十进制数200，转换成16进制数。需要考虑转换为大于10进制的数时，大于等于10的数如何转化成A-F，例如转换成16进制时，如何将10-15对应转换为A-F。

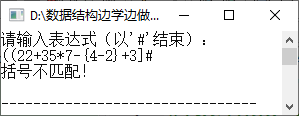
实验数据可参考下图：

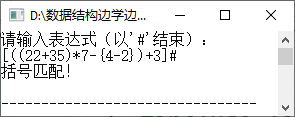


## 3.7 表达式括号匹配问题

说明：利用SeqStack类模板，实现中缀表达式中括号的匹配问题。假设表达式中的括号只包括(、[、{、}、]、)三种括号，输入表达式，给出匹配或不匹配的结果。

实验数据可分别参考下列各图：





## 3.8 后缀表达式求值问题

说明：利用SeqStack类模板，实现后缀表达式求值。假设表达式中只包括+、-、\*、/四种运算。分别考虑操作数为一位整数，操作数为多位整数的不同情况。输入后缀表达式，输出表达式结果。

实验数据可参考：

89#60#-12#8#-\*

输出结果为：

116

## 3.9 中缀表达式求值问题

说明：利用栈求解中缀表达式，假设表达式中只包括+、-、\*、/四种运算。操作数为一位整数或者多位整数。输入中缀表达式，输出表达式结果。难点在于判断各种运算符的优先级次序。

实验数据可参考：

(89-60)\*(12-8)#

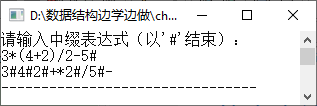
输出结果：

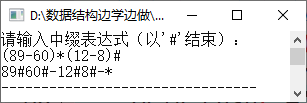
116

## 3.10 中缀表达式转换成后缀表达式问题

说明：利用栈将中缀表达式转换成后缀表达式，假设表达式中只包括+、-、\*、/四种运算。操作数为一位整数或者多位整数。输入中缀表达式，输出其对应的后缀表达式。

实验数据可分别参考下列各图：





# 第4章 字符串和多维数组

## 4.1 字符串的BF匹配算法的实现

说明：使用字符数组存储字符串S和T，输入S和T，使用BF算法进行模式匹配，输出T在S中第一次出现的逻辑位置。

### 4.1.1 代码

#include<iostream>

using namespace std;

int BF(char s[], char t[]);

int main()

{

char TargetString[1005], PatternString[1005];

cout << "Input the target string." << endl;

cin >> TargetString;

cout << "Input the pattern string." << endl;

cin >> PatternString;

int ans = BF(TargetString, PatternString);

cout << "The first occurrence of " << PatternString << " in " << TargetString << " is " << ans << endl;

cout << "By 李冠润" << endl;

return 0;

}

int BF(char s[], char t[])

{

int i = 0, j = 0;

while (s[i] != '\0' && t[j] != '\0')

{

if(s[i] == t[j])

{

i++;

j++;

}

else

{

i = i - j + 1;

j = 0;

}

}

if(t[j] == '\0')

return i - j + 1;

else

return 0;

}

### 4.1.2 实验结果

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

## 4.2 字符串的KMP匹配算法的实现

说明：使用字符数组存储字符串S和T，输入S和T，使用KMP算法进行模式匹配，输出T在S中第一次出现的逻辑位置。

### 4.2.1 代码

#include <iostream>

#include <cstring>

using namespace std;

void GetNext(string T, int next[]);

int KMP(string s, string t);

int main()

{

string TargetString, PatternString;

cout << "Input the target string." << endl;

cin >> TargetString;

cout << "Input the pattern string." << endl;

cin >> PatternString;

int ans = 0;

ans = KMP(TargetString, PatternString);

if(ans)

cout << "The first occurrence of " << PatternString << " in " << TargetString << " is " << ans << endl;

else

cout << "No " << PatternString << " in " << TargetString << endl;

cout << "By 李冠润" << endl;

return 0;

}

void GetNext(string T, int next[])

{

int j = 0, k = -1;

next[0] = -1;

while (j < (int)T.length())

{

if (k == -1 || T[j] == T[k])

{

j++, k++;

if (T[j] == T[k])

next[j] = next[k];

else

next[j] = k;

}

else

k = next[k];

}

}

int KMP(string s, string t)

{

int Next[1005], i = 0, j = 0;

GetNext(t, Next);

while (i < (int)s.length() && j < (int)t.length())

{

if (j == -1 || s[i] == t[j])

i++, j++;

else

j = Next[j];

}

if (j >= (int)t.length())

return i - j + 1;

else

return 0;

}

### 4.2.2 实验结果

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

## 4.3 对称矩阵的压缩存储

说明：随机生成nxn阶对称矩阵，将对称矩阵按行优先压缩存下三角的方式压缩存储到一维数组中，根据矩阵的行列下标在一维数组中存取元素。

### 4.3.1 代码

#include<iostream>

#include<cstdlib>

#include<ctime>

using namespace std;

#define MAX 105

class Matrix

{

private:

int A[MAX][MAX];

int SA[MAX \* (MAX + 1) / 2];

public:

int n;

Matrix(int n);

void compressMatrix();

int getElement(int i, int j);

void PrintMatrix();

};

Matrix::Matrix(int n)

{

this->n = n;

int a = 100;

int b = 500;

srand((unsigned)time(NULL));

for (int i = 0; i < n; i++)

for (int j = 0; j <= i; j++)

{

A[i][j] = (rand() % (b - a)) + a;

A[j][i] = A[i][j];

}

}

void Matrix::compressMatrix()

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j <= i; j++)

{

SA[i \* (i + 1) / 2 + j] = A[i][j];

}

}

}

int Matrix::getElement(int i, int j)

{

if (i < 0 || i >= n || j < 0 || j >= n)

throw "参数非法";

if (i >= j)

return SA[i \* (i + 1) / 2 + j];

else

return SA[j \* (j + 1) / 2 + i];

}

void Matrix::PrintMatrix()

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout.width(3);

cout << A[i][j] << " ";

}

cout << endl;

}

}

int main()

{

int n;

cout << "Input the number of rows and columns of the matrix" << endl;

cin >> n;

Matrix A(n);

cout << "The constructed random matrix is as follows" << endl;

A.PrintMatrix();

A.compressMatrix();

cout << "Input the coordinates to query" << endl;

int x, y;

cin >> x >> y;

int ans;

ans = A.getElement(x - 1, y - 1);

cout << "The value of " << x << "rows and " << y << " columns is " << ans << endl;

cout << "By 李冠润" << endl;

}

### 4.3.2 实验结果

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

## 4.4 上三角矩阵的压缩存储

说明：随机生成nxn阶上三角矩阵，将上三角矩阵按行优先压缩存储的方式存储到一维数组中，根据矩阵的行列下标在一维数组中存取元素。

### 4.4.1 代码

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

#define MAX 105

class Matrix

{

private:

int A[MAX][MAX];

int SA[MAX \* (MAX + 1) / 2];

public:

int n;

Matrix(int n);

void compressMatrix();

int getElement(int i, int j);

void PrintMatrix();

};

Matrix::Matrix(int n)

{

this->n = n;

int a = 100;

int b = 500;

srand((unsigned)time(NULL));

int c = (rand() % (b - a)) + a;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < i; j++)

A[i][j] = c;

for (int j = i; j < n; j++)

A[i][j] = (rand() % (b - a)) + a;

}

}

void Matrix::compressMatrix()

{

for (int i = 0; i < n; i++)

for (int j = i; j < n; j++)

SA[i \* (2 \* n - i + 1) / 2 + j - 1] = A[i][j];

SA[n \* (n + 1) / 2] = A[n - 1][0];

}

int Matrix::getElement(int i, int j)

{

if (i < 0 || i >= n || j < 0 || j >= n)

throw "参数非法";

if (i <= j)

return SA[i \* (2 \* n - i + 1) / 2 + j - i];

else

return SA[n \* (n + 1) / 2];

}

void Matrix::PrintMatrix()

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout.width(3);

cout << A[i][j] << " ";

}

cout << endl;

}

}

int main()

{

int n;

cout << "Input the number of rows and columns of the matrix" << endl;

cin >> n;

Matrix A(n);

cout << "The constructed random matrix is as follows" << endl;

A.PrintMatrix();

A.compressMatrix();

cout << "Input the coordinates to query" << endl;

int x, y;

cin >> x >> y;

int ans;

ans = A.getElement(x - 1, y - 1);

cout << "The value of " << x << "rows and " << y << " columns is " << ans << endl;

cout << "By 李冠润" << endl;

}

### 4.4.2 实验结果

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

## 4.5 下三角矩阵的压缩存储

说明：随机生成nxn阶下三角矩阵，将下三角矩阵按行优先压缩存储的方式存储到一维数组中，根据矩阵的行列下标在一维数组中存取元素。

### 4.5.1 代码

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

#define MAX 105

class Matrix

{

private:

int A[MAX][MAX];

int SA[MAX \* (MAX + 1) / 2];

public:

int n;

Matrix(int n);

void compressMatrix();

int getElement(int i, int j);

void PrintMatrix();

};

Matrix::Matrix(int n)

{

this->n = n;

int a = 10;

int b = 50;

srand((unsigned)time(NULL));

int c = (rand() % (b - a)) + a;

for (int i = 0; i < n; i++)

{

for (int j = 0; j <= i; j++)

A[i][j] = (rand() % (b - a)) + a;

for (int j = i + 1; j < n; j++)

A[i][j] = c;

}

}

void Matrix::compressMatrix()

{

for (int i = 0; i < n; i++)

for (int j = 0; j <= i; j++)

SA[i \* (i + 1) / 2 + j] = A[i][j];

SA[n \* (n + 1) / 2] = A[0][n - 1];

}

int Matrix::getElement(int i, int j)

{

if (i < 0 || i >= n || j < 0 || j >= n)

throw "参数非法";

if (i >= j)

return SA[i \* (i + 1) / 2 + j];

else

return SA[n \* (n + 1) / 2];

}

void Matrix::PrintMatrix()

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout.width(3);

cout << A[i][j] << " ";

}

cout << endl;

}

}

int main()

{

int n;

cout << "Input the number of rows and columns of the matrix" << endl;

cin >> n;

Matrix A(n);

cout << "The constructed random matrix is as follows" << endl;

A.PrintMatrix();

A.compressMatrix();

cout << "Input the coordinates to query" << endl;

int x, y;

cin >> x >> y;

int ans;

ans = A.getElement(x - 1, y - 1);

cout << "The value of " << x << "rows and " << y << " columns is " << ans << endl;

}

### 4.5.2 实验结果

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

## 4.6 三对角矩阵的压缩存储

说明：随机生成nxn阶三对角矩阵，将三对角矩阵按行优先压缩存储的方式存储到一维数组中，根据矩阵的行列下标在一维数组中存取元素。

### 4.6.1 代码

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

#define MAX 105

class Matrix

{

private:

int A[MAX][MAX];

int SA[MAX \* (MAX + 1) / 2];

public:

int n;

Matrix(int n);

void compressMatrix();

int getElement(int i, int j);

void PrintMatrix();

};

Matrix::Matrix(int n)

{

this->n = n;

int a = 10;

int b = 50;

srand((unsigned)time(NULL));

int i, j;

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

if (abs(i - j) <= 1)

A[i][j] = (rand() % (b - a)) + a;

else

A[i][j] = 0;

}

}

}

void Matrix::compressMatrix()

{

int k = 0;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

{

if (A[i][j] != 0)

{

SA[k++] = A[i][j];

}

}

}

int Matrix::getElement(int i, int j)

{

if (i < 0 || i >= n || j < 0 || j >= n)

throw "参数非法";

if (abs(i - j) <= 1)

return SA[2 \* i + j];

else

return 0;

}

void Matrix::PrintMatrix()

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout.width(3);

cout << A[i][j] << " ";

}

cout << endl;

}

}

int main()

{

int n;

cout << "Input the number of rows and columns of the matrix" << endl;

cin >> n;

Matrix A(n);

cout << "The constructed random matrix is as follows" << endl;

A.PrintMatrix();

A.compressMatrix();

cout << "Input the coordinates to query" << endl;

int x, y;

cin >> x >> y;

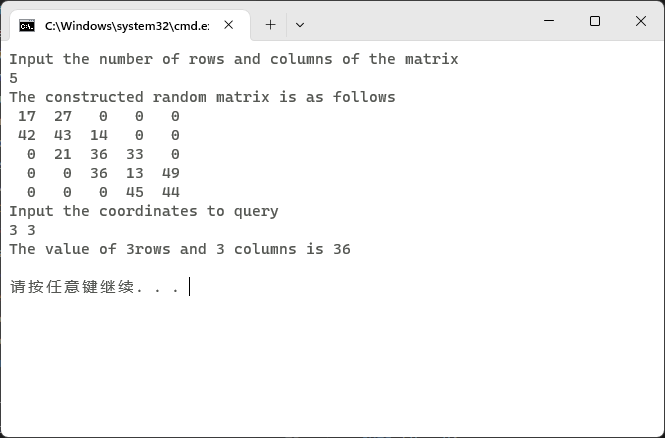
int ans;

ans = A.getElement(x - 1, y - 1);

cout << "The value of " << x << "rows and " << y << " columns is " << ans << endl;

}

### 4.6.2 实验结果



## 4.7 稀疏矩阵的压缩存储

说明：输入nxm阶的稀疏矩阵A，将稀疏矩阵按行优先压缩存储到三元组顺序表中，按照矩阵的行列下标在三元组顺序表中进行元素的存取。分别使用普通转置算法和快速转置算法求矩阵A的转置矩阵B，分别输出A和B的三元组顺序表。

### 4.7.1 代码

#include<iostream>

#define MAX 10005

using namespace std;

class Triple

{

public:

int x, y;

int value;

};

class tripleTable

{

private:

Triple data[MAX];

int maxx, maxy, num;

public:

void inputTripleTable();

void printTripleTable();

tripleTable transposeTripleTable();

tripleTable fastTransposeTripleTable();

};

void tripleTable::inputTripleTable()

{

cin >> maxx >> maxy >> num;

for (int i = 1; i <= num; i++)

{

cin >> this->data[i].x >> this->data[i].y >> this->data[i].value;

}

}

void tripleTable::printTripleTable()

{

cout << "The triplet table you entered is " << endl;

for (int i = 1; i <= this->num; i++)

{

cout << this->data[i].x << " " << this->data[i].y << " " << this->data[i].value << endl;

}

}

tripleTable tripleTable::transposeTripleTable()

{

tripleTable after;

after.maxx = this->maxy;

after.maxy = this->maxx;

after.num = this->num;

if(after.num)

{

int q = 1;

for (int i = 1; i <= this->maxy; i++)

for (int j = 1; j <= this->maxx; j++)

if (this->data[j].y == i)

{

after.data[q].x = this->data[j].y;

after.data[q].y = this->data[j].x;

after.data[q].value = this->data[i].value;

q++;

}

}

return after;

}

tripleTable tripleTable::fastTransposeTripleTable()

{

tripleTable after;

after.maxx = this->maxy;

after.maxy = this->maxx;

after.num = this->num;

int cnum[MAX];

int cpot[MAX];

if (after.num)

{

for (int i = 1; i <= this->maxy; i++)

cnum[i] = 0;

for (int i = 1; i <= this->num; i++)

cnum[this->data[i].y]++;

cpot[1] = 1;

for (int i = 2; i <= this->maxx; i++)

cpot[i] = cpot[i - 1] + cnum[i - 1];

for (int i = 1; i <= this->num; i++)

{

int col = this->data[i].y;

int q = cpot[col];

after.data[q].x = this->data[i].y;

after.data[q].y = this->data[i].x;

after.data[q].value = this->data[i].value;

cpot[col]++;

}

}

return after;

}

int main()

{

tripleTable a;

a.inputTripleTable();

a.printTripleTable();

tripleTable transposedTripleTable;

// transposedTripleTable = a.fastTransposeTripleTable();

transposedTripleTable = a.transposeTripleTable();

transposedTripleTable.printTripleTable();

cout << "By 李冠润" << endl;

}

### 4.7.2 实验数据：

6 7 8

1 2 12

1 3 9

3 1 -3

3 6 14

4 3 24

5 2 18

6 1 15

6 4 -7

说明：

第一行三个数分别为稀疏矩阵的行数maxx、列数maxy、非零元素数num；

接下来的num行，每行分别是一个三元组

### 4.7.3 实验结果

图形用户界面

描述已自动生成

## 4.8 约瑟夫环问题

说明：输入n和密码m，使用一维数组实现约瑟夫环问题，输出序列值。

实验数据可参考下图：

### 4.8.1 实验代码

#include<iostream>

using namespace std;

#define MAX 1005

int main()

{

int n, m;

cin >> n >> m;

int r[MAX];

int i = 0, k = 0, j = 0;

for (i = 0; i < n; i++)

r[i] = i + 1;

i = 0;

cout << "出列的顺序为" << endl;

while(i < n)

{

if(r[i] != 0)

k++;

if(k == m)

{

k = 0;

cout << r[i] << " ";

r[i] = 0;

j++;

}

if(j == n)

break;

i = (i + 1) % n;

}

cout << endl;

cout << "By 李冠润" << endl;

return 0;

}

### 4.8.2 实验数据

9 5

输入的两个数分别为总人数n和密码m。

### 4.8.3 实验结果

图形用户界面, 应用程序, Word

描述已自动生成

## 4.9 求解矩阵的马鞍点

说明：输入nxm阶的矩阵，输出矩阵的马鞍点。注意需要分别使用不存在马鞍点、存在一个马鞍点、存在多个马鞍点三组测试数据，

实验数据可分别参考下列各图：

### 4.9.1 代码

#include <iostream>

using namespace std;

#define MAXN 105

int main()

{

int Matrix[MAXN][MAXN];

int m, n, minData, flag = 0;

cin >> m >> n;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cin >> Matrix[i][j];

}

}

for (int i = 0; i < m; i++)

{

minData = Matrix[i][0];

for (int j = 0; j < n; j++)

{

if (Matrix[i][j] < minData)

minData = Matrix[i][j];

}

for (int j = 0; j < n; j++)

{

if (Matrix[i][j] == minData)

{

int k;

for (k = 0; k < m; k++)

if (Matrix[k][j] > minData)

break;

if (k == m)

{

cout << "The saddle point is " << Matrix[i][j] << " of " << i + 1 << " row and " << j + 1 << " columns" << endl;

flag = 1;

}

}

}

}

if (!flag)

cout << "This matrix has no saddle point" << endl;

cout << "By 李冠润" << endl;

}

### 4.9.2 实验数据与结果

#### 4.9.2.1 数据1-只有一个鞍点

4 4

1 2 3 4

3 4 5 6

5 6 7 8

6 7 8 9

第一行两个数分别为矩阵的行数m和列数n。

接下来的m行，每行n个数，代表矩阵的m\*n个元素。

图形用户界面, 文本, 应用程序, Word

描述已自动生成

#### 4.9.2.2 数据2-不存在鞍点

4 4

1 2 3 4

3 4 5 6

4 5 6 7

9 8 7 6

第一行两个数分别为矩阵的行数m和列数n。

接下来的m行，每行n个数，代表矩阵的m\*n个元素。

图形用户界面, 应用程序, Word

描述已自动生成

#### 4.9.2.3 数据3-有多个鞍点

4 4

1 2 3 4

3 4 5 2

5 5 8 5

4 1 3 3

第一行两个数分别为矩阵的行数m和列数n。

接下来的m行，每行n个数，代表矩阵的m\*n个元素。

图形用户界面, 文本, 应用程序

描述已自动生成

## 4.10 螺旋方阵问题

说明：生成任意nxn阶的螺旋方阵并输出。

### 4.10.1 代码

#include<iostream>

using namespace std;

int calculate(int n, int i, int j);

int main()

{

int n;

cout << "Enter the number of rows and columns of the matrix" << endl;

cin >> n;

cout << "The spiral square matrix of order "<< n << " is" << endl;

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

{

cout.width(5);

cout << calculate(n - 1, i, j);

}

cout << endl;

}

cout << "By 李冠润" << endl;

}

int calculate(int n, int i, int j)

{

int k = 0;

int mini = i < n - i ? i : n - i;

int minj = j < n - j ? j : n - j;

int min = mini < minj ? mini : minj;

int h;

for (h = 0; h < min; ++h)

{

k += (n - 2 \* h) \* 4;

}

if (i == min)

{

k += j - min + 1;

}

else if (j == n - min)

{

k += (n - 2 \* min) + (i - min) + 1;

}

else if (i == n - min)

{

k += (n - 2 \* min) \* 2 + (n - min - j) + 1;

}

else if (j == min)

{

k += (n - 2 \* min) \* 3 + (n - min - i) + 1;

}

return k;

}

### 4.10.2 实验结果

表格

描述已自动生成

## 4.11 幻方问题

说明：生成奇数阶幻方，并输出；生成双偶阶幻方，并输出。

实验数据可分别参考下列各图：

### 4.11.1 生成双偶阶幻方

#include <iostream>

#include <iomanip>

using namespace std;

#define MAXN 105

int judge(int i, int j)

{

int flag = 0;

if (i == j)

return 1;

else if (i > j && (i - j) % 4 == 0)

return 1;

else if (j > i && (j - i) % 4 == 0)

return 1;

else if ((i + j + 1) % 4 == 0)

return 1;

else

return 0;

}

void MagicSquresDoubleEven(int a[][MAXN], int n)

{

int i, j;

int k = 1;

for (i = 0; i < n; i++)

for (j = 0; j < n; j++)

a[i][j] = k++;

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

if (judge(i, j) == 1)

a[i][j] = n \* n + 1 - a[i][j];

}

}

}

void PrintMagicSquares(int a[][MAXN], int n)

{

int i, j;

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

cout << setw(5) << a[i][j];

}

cout << endl;

}

}

int main()

{

int a[MAXN][MAXN];

int i, j;

int n;

cout << "Enter the order of magic square" << endl;

cin >> n;

MagicSquresDoubleEven(a, n);

cout << "The magic square of order " << n << " is" << endl;

PrintMagicSquares(a, n);

cout << "By 李冠润" << endl;

return 0;

}

图形用户界面, 表格

中度可信度描述已自动生成

### 4.11.2 生成奇数阶幻方

#include <iostream>

#include <iomanip>

#define MAXN 100

using namespace std;

void MagicSquresOdd(int a[][MAXN], int n)

{

int i, j, k;

i = 0;

j = n / 2;

a[i][j] = 1;

for (k = 2; k <= n \* n; k++)

{

int ti, tj;

ti = i;

tj = j;

i = (i - 1 + n) % n;

j = (j - 1 + n) % n;

if (a[i][j] > 0)

{

i = (ti + 1) % n;

j = tj;

}

a[i][j] = k;

}

}

void PrintMagicSquares(int a[][MAXN], int n)

{

int i, j;

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

cout << setw(5) << a[i][j];

}

cout << endl;

}

}

int main()

{

int a[MAXN][MAXN];

int i, j;

int n;

cout << "Enter the order of magic square" << endl;

cin >> n;

for (i = 0; i < n; i++)

for (j = 0; j < n; j++)

a[i][j] = 0;

MagicSquresOdd(a, n);

cout << "The magic square of order " << n << " is" << endl;

PrintMagicSquares(a, n);

cout << "By 李冠润" << endl;

return 0;

}

图形用户界面, 应用程序

描述已自动生成

# 第5章 树和二叉树

提示：本章所有的实训项目都需要输入，大部分是需要输入结点序列构造二叉链表。

5.10，5.11需要输入序列，类似于字符数组。

## 5.1 二叉链表基本操作的实现

说明：实现类模板BiTree，基本操作包括构造函数、递归前序遍历、递归中序遍历、递归后序遍历、层序遍历，并在主函数中验证，测试数据参照教材图5-19，即下图。



## 5.2 二叉树遍历的非递归实现

说明：实现类模板BiTree，基本操作包括构造函数、非递归前序遍历、非递归中序遍历、非递归后序遍历，并在主函数中验证，测试数据参照教材图5-19，即下图。



## 5.3 求二叉树的最小深度

说明：利用类模板BiTree，实现二叉链表的构造、求二叉树的最小深度，并在主函数中验证，测试数据参照教材图5-13，即下图。



## 5.4 判断二叉树是否是完全二叉树

说明：利用类模板BiTree，实现二叉链表的构造，判断二叉树是否是完全二叉树，并在主函数中进行验证。要求分别使用教材图5-19所示的二叉树和教材图5-32所示的二叉树作为测试数据，即下列各图。





## 5.5 判断二叉树的结构是否对称

说明：利用类模板BiTree，实现二叉链表的构造，并判断二叉树是否结构对称，并在主函数中进行验证。要求分别使用教材图5-34和图5-35所示的二叉树作为测试数据，即下列各图。





## 5.6 判断二叉树是否对称

说明：利用类模板BiTree，实现二叉链表的构造，并判断二叉树是否是对称的，并在主函数中进行验证。要求分别使用教材图5-38和图5-39所示的二叉树作为测试数据，即下列各图。





## 5.7 求二叉树中叶子结点个数及结点个数

说明：利用类模板BiTree，实现二叉链表的构造，并计算二叉树中叶子结点的个数及结点个数，并在主函数中输出。要求使用教材图5-13所示的二叉树作为测试数据，即下图。



## 5.8 中序线索二叉树的构造

说明：利用InThrBiTree类模板，需要依次完成以下工作：

（1）声明和定义InThrBiTree类模板；

（2）构造初始的未线索化的中序线索二叉链表；

（3）完成求已知结点p的后继结点的算法；

（4）完成对中序线索二叉树进行非递归的中序遍历的算法；

（5）在主函数中对遍历算法进行验证，输出中序遍历序列。

测试数据参考下图。



## 5.9 哈夫曼算法的实现

说明：使用三叉链表的静态链表模板存储二叉树的各个结点，实现哈夫曼算法。以教材图5-27的数据为例，即下图。



## 5.10 由前序遍历序列和中序遍历序列构造二叉树

说明：输入二叉树的前序遍历序列和中序遍历序列，构造二叉树，并对构造的二叉树进行前序、中序、后序遍历。要求使用教材图5-51所示的二叉树作为测试数据，即下图。



## 5.11 由后序遍历序列和中序遍历序列构造二叉树

说明：输入二叉树的后序遍历序列和中序遍历序列，构造二叉树，并对构造的二叉树进行前序、中序、后序遍历。要求使用教材图5-51所示的二叉树作为测试数据，即下图。



## 5.12 求二叉树第k层的结点个数和叶子结点个数

说明：利用BiTree类模板

实验数据参考下图。



可以求第3层的结点个数和叶子结点个数。

## 5.13 打印二叉树第k层的结点和叶子结点

实验数据参考下图。



可以打印第3层的结点和叶子结点。

## 5.14 求二叉树最大的结点距离

实验数据可分别参考以下各图。







# 第6章 图

图这一章全部需要输入，图的类型包括：有向图，无向图，有向网，无向网，根据具体情况决定。

图需要输入图的顶点个数，边数，边依附的顶点以及可能的权值。

## 6.1 邻接矩阵存储无向图的实现

说明：实现类模板MGraph，基本操作包括构造函数、深度优先遍历、广度优先遍历，并验证。

实验数据参考下图。



## 6.2 邻接表存储有向图的实现

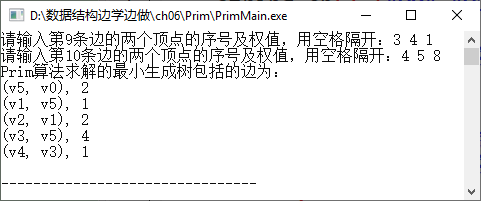
说明：实现类模板ALGraph，基本操作包括构造函数、深度优先遍历、广度优先遍历，并验证。**教材P180的图6-21有误，实验数据参照的图如下图所示。**



## 6.3 Prim算法的实现

说明：实现Prim算法，对于给定的连通网，输出最小生成树中的各边，参照数据以及预期的结果分别如下图所示。

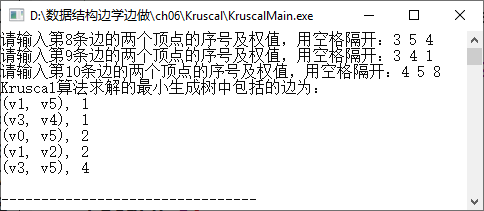




## 6.4 Kruscal算法的实现

说明：实现Kruscal算法，对于给定的连通网，输出最小生成树中的各边，参照数据以及预期的结果分别如下图所示。



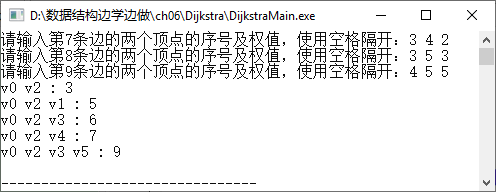


## 6.5 Dijkstra算法的实现

说明：实现Dijkstra算法，求源点V0到其它顶点的最短路径长度。

实验数据及预期结果如图所示。



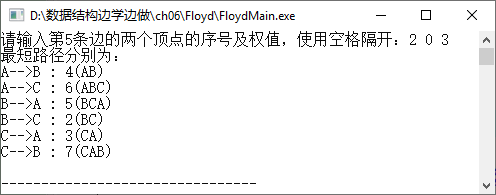


## 6.6 Floyd算法的实现

说明：实现Floyd算法，对于给定的有向网，求每一对顶点之间的最短路径长度。实验数据参照**教材P193的图6-42，教材上的图有误，正确的如下图所示。**



预期的结果如下图所示。



## 6.7 拓扑排序的实现

说明：对于有向图，求解拓扑序列，不存在拓扑序列的，给出说明。

参考实验数据如图所示。



注意：输出的拓扑序列不唯一。

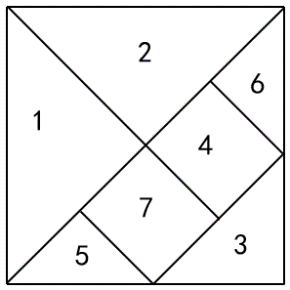
## 6.8 选址问题

说明：图中的四个顶点为四个小区，有向边上的权值表示小区之间的距离，现在要在四个小区中选择一个小区新建一所活动中心。怎么选取合适？



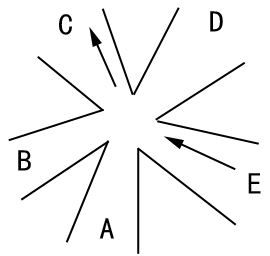
## 6.9 七巧板涂色问题

说明：对于如下图所示的七巧板涂色，要求相邻的板颜色不同，并且总的颜色数尽可能少。



## 6.10 五叉路口交通灯问题

说明：对于如下图所示的五叉路口，设计交通灯方案，总体可行的方案较多，只选择其中的六种输出即可。

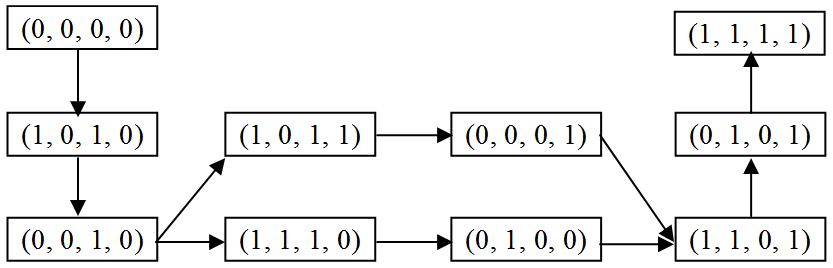


说明：不能从C到其它路口，C路口为单行道，只出不进。即和C路口相关的可通行路线包括BC、AC、EC、DC，但是C到其它的路口是不通的。

## 6.11 农夫过河问题

说明：一位农夫带着一只狼、一只羊和一棵白菜，身处河的南岸，他要把这些东西全部运到河的北岸。他面前只有一条小船，船只能容下他和一件物品，只有农夫自己才能撑船。如果农夫在场，则狼不能吃羊，羊不能吃白菜；如果农夫不在场，则狼会吃羊，羊会吃白菜。所以在任何情况下，农夫不能留下狼和羊单独离开，也不能留下羊和白菜单独离开，按要求设计过河方案。

**说明：教材P208图6-64有误，正确的如下图所示。**



# 第7章 查找

输入可有可无，如果没有输入，需要在main()函数中设置好数据。

## 7.1 线性查找的实现

可随机生成查找序列

## 7.2 折半查找的实现

可随机生成递增序列

## 7.3 二叉排序树基本操作的实现

说明：实现类BiSortTree，基本操作包括构造函数、插入元素、查找、中序遍历，并验证（选做二叉排序树中结点的删除）。

请自行设计实验数据

## 7.4 散列查找的实现

请自行设计实验数据。

# 第8章 排序

输入数据可有可无，如果没有输入数据，需要在main()函数中设置好。

也可以采用随机数生成待排序序列。

## 8.1 直接插入排序的实现

## 8.2 Shell排序的实现

## 8.3 冒泡排序的实现

## 8.4 快速排序的实现

## 8.5 简单选择排序的实现

## 8.6 堆排序的实现

## 8.7 非递归实现二路归并排序