电子信息学院

实验报告书

**课程名： 算法设计与分析**

**授课老师：** 连志刚

**完成日期：2019年12月10日**

# 目录

[目录 1](#_Toc26882187)

[实验一 4](#_Toc26882188)

[实验目标 4](#_Toc26882189)

[实验 4](#_Toc26882190)

[1. 自己电脑安装VS或VC、或Dev-C++ 4](#_Toc26882191)

[2. 三种方法实现两个数最大公约数 4](#_Toc26882192)

[3. 输出n个数的最大公约数 6](#_Toc26882193)

[4. 输出两个数的最小公倍数 7](#_Toc26882194)

[5. 计算n！ 7](#_Toc26882195)

[作业 8](#_Toc26882196)

[1. 输出1000内质数的程序 8](#_Toc26882197)

[2. 求解鸡兔同笼问题 9](#_Toc26882198)

[实验二 10](#_Toc26882199)

[实验目标 10](#_Toc26882200)

[实验 10](#_Toc26882201)

[1. 输出斐波那契数列 10](#_Toc26882202)

[2. 输出一个整数划分 10](#_Toc26882203)

[3. 汉诺塔移动次数递归设计 11](#_Toc26882204)

[4. 求两个矩阵乘积 11](#_Toc26882205)

[5. 输出可找开钱的排序数量 13](#_Toc26882206)

[6. 实现排列A(n,m)递归程序设计 14](#_Toc26882207)

[7. 实现组合C(n,m)递归程序设计 15](#_Toc26882208)

[作业 15](#_Toc26882209)

[1. 随机产生数组和一个整数，查找整数位置 16](#_Toc26882210)

[2. 快速排序法排序输出一个数组. 16](#_Toc26882211)

[实验三 18](#_Toc26882212)

[实验目标 18](#_Toc26882213)

[实验 18](#_Toc26882214)

[1. 输出2^k循环赛日程表 18](#_Toc26882215)

[2. 最长公共子序列 19](#_Toc26882216)

[3. 寻找两个或多个已知字符串最长的子串 20](#_Toc26882217)

[4. 动态规划法实现0-1背包问题 23](#_Toc26882218)

[实验四 25](#_Toc26882219)

[实验目标 25](#_Toc26882220)

[实验 25](#_Toc26882221)

[1. 安排活动算法程序 25](#_Toc26882222)

[2. 可拆分背包问题 26](#_Toc26882223)

[3. 单源最短路径 28](#_Toc26882224)

[4. 多机调度问题 30](#_Toc26882225)

[作业 31](#_Toc26882226)

[1. Prim算法构造最小生成树 31](#_Toc26882227)

[2. 最优装载问题 33](#_Toc26882228)

[实验五 35](#_Toc26882229)

[实验目标 35](#_Toc26882230)

[实验 35](#_Toc26882231)

[1. 批处理作业调度问题 35](#_Toc26882232)

[2. N皇后问题 37](#_Toc26882233)

[3. TSP售货员问题 39](#_Toc26882234)

[4. 连续邮资问题 41](#_Toc26882235)

[作业 43](#_Toc26882236)

[1. 图着色问题 43](#_Toc26882237)

[2. 装载问题 45](#_Toc26882238)

[实验六 48](#_Toc26882239)

[实验目标 48](#_Toc26882240)

[实验 48](#_Toc26882241)

[1. 单源最短路径问题 48](#_Toc26882242)

[2. 装载问题 51](#_Toc26882243)

[3. 0-1背包问题 53](#_Toc26882244)

[4. TSP问题 56](#_Toc26882245)

[作业 61](#_Toc26882246)

[1. Flowshop问题 61](#_Toc26882247)

[实验七 65](#_Toc26882248)

[实验目标 65](#_Toc26882249)

[实验 66](#_Toc26882250)

[1. 模拟π的值 66](#_Toc26882251)

[2. 蒙特卡罗算法计算定积分 66](#_Toc26882252)

[3. 概率算法设计n皇后问题程序 67](#_Toc26882253)

[4. 检验某数是否为素数 69](#_Toc26882254)

[作业 71](#_Toc26882255)

[1. 牛顿法求解方程 71](#_Toc26882256)

[2. 利用二分法求解方程 72](#_Toc26882257)

[实验八 74](#_Toc26882258)

[作业 74](#_Toc26882259)

输入[字符串产生新的数组 74](#_Toc26882260)

# 实验一

实验目标

了解程序运行运算来确定时间复杂度的评价，掌握事前分析中的程序步分析算法、掌握递归与分治策略，能用递归与分治方法解决实际问题，如最大公约数等。

实验

1. **自己电脑安装VS或VC、或Dev-C++（已完成）**
2. **三种方法实现两个数最大公约数（via ）**
3. 枚举法

源代码：

#include<stdio.h>

int main() {

    int a, b, t;

    printf("请输入两个数求它们的最大公约数:");

    scanf("%d %d", &a, &b);

    if (a <= 0 || b <= 0) {

        printf("输入有误，请重新输入");

        return 0;

    }

    if (a < b) {        //将较小的数的值放在b中

        t = a;

        a = b;

        b = t;

    }

    for (t = b; t > 0; t--)

        if (a % t == 0 && b % t == 0)

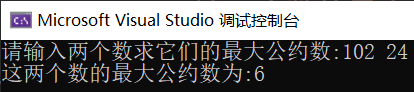
            break;

    printf("这两个数的最大公约数为:%d\n", t);

    return 0;

}

运行截图：



1. 辗转相除法

源代码：

#include<stdio.h>

int main() {

    int a, b, temp;

    printf("请输入两个数求它们的最大公约数:");

    scanf("%d %d", &a, &b);

    if (a < b) {            //将较小的数保存在b中

        temp = a;

        a = b;

        b = temp;

    }

    while (a % b != 0) {    //反复取余，并把取余所得的值赋给比较小的数

        temp = a % b;

        a = b;

        b = temp;

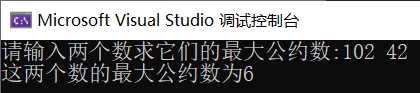
    }

    printf("这两个数的最大公约数为%d\n", b);

    return 0;

}

运行截图：



1. 更相减损法

源代码：

#include<stdio.h>

void swap(int &a,int &b) {                  //交换a,b

    int temp;

    temp = a;

    a = b;

    b = temp;

}

int main(void) {

    int a, b, t;

    printf("请输入两个数求它们的最大公约数:");

    scanf("%d %d", &a, &b);

    while (a != b) {

        if (a < b)          //保证a>b

            swap(a, b);

        t = a - b;          //反复使用差和减数，用他们两个中大的减去小的

        a = b;

        b = t;

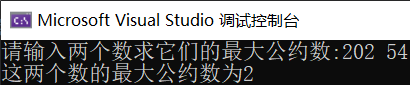
    }

    printf("这两个数的最大公约数为%d\n", b);

    return 0;

}

运行截图：



1. **输出n个数的最大公约数（via ）**

源代码：（省略求最大公因数的函数）

#include<iostream>

#include<vector>

using namespace std;

int gcd(int a, int b) {

    int temp;

    if (a < b) {

        temp = a;

        a = b;

        b = temp;

    }

    while (a % b != 0) {

        temp = a % b;

        a = b;

        b = temp;

    }

    return b;

}

int main(void) {

    vector<int> numSeries; //一个int型数组

    cout << "请输入一组整数，以“#”终止：" << endl;

    int a;

    while (cin >> a) //存放输入的数

        numSeries.push\_back(a);

    int finGCD = numSeries[0];

    for (int i = 1; i != numSeries.size(); i++)

        finGCD = gcd(finGCD, numSeries[i]);

//遍历数组中的数，先求前两个数的最大公约数，

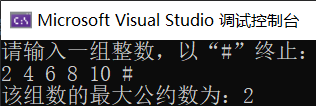
//再将所得到的数与第三个数求最大公约数，以此类推直到最后一个数

    cout << "该组数的最大公约数为：" << finGCD << endl;

    return 0;

}

运行截图：



1. **输出两个数的最小公倍数（via ）**

源代码：（省略求最大公因数的函数）

#include<stdio.h>

int gcd(int a, int b) {

    int temp;

    if (a < b) {

        temp = a;

        a = b;

        b = temp;

    }

    while (a % b != 0) {

        temp = a % b;

        a = b;

        b = temp;

    }

    return b;

}

int main(void) {

    int a, b;

    printf("输入两个数: \n");

    scanf("%d, %d", &a, &b);

    int j = a \* b / gcd(a,b);

//用两个数的乘积除以他们的最大公因数就是他们的最小公倍数

    printf("最小公倍数为: %d", j);

    return 0;

}

运行截图：



1. **计算n！（via ）**

#include<stdio.h>

int main() {

    int n, s = 1;

    scanf("%d", &n);        /\*输入一个数\*/

    if (n == 0 || n == 1)   /\*如果n是0或1， 它的阶乘为1\*/

        s = 1;

    else

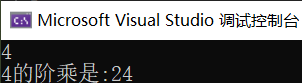
        for (int i = 1; i <= n; i++)

            s \*= i;

    printf("%d的阶乘是:%d", n, s);

}

运行截图：



作业

1. **输出1000内质数的程序（via ）**

源代码：

#include<iostream>

#include<vector>

using namespace std;

//数学原理：如果一个数不能被小于它的所有质数整除，则这个数是质数

int main() {

    vector<int> primeNumber;

//新建一个数组primeNumber作为质数表

    primeNumber.push\_back(2);

//2为第一个质数，把它先放入质数表

    for (int i = 3; i <= 1000; i += 2)

//循环变量i即为待验证数，从3开始到1000进行验证，因为除了2的偶数必定不是质数，因此i+=2跳过偶数

        for (int j = primeNumber.size() - 1; j >= 0; j--) {

//对于每一个待验证的数，遍历已有的质数表进行求余验证

            if (i % primeNumber[j] == 0)

//如果验证时该数整除了质数表中的某个数，则该数是合数，跳过

                break;

            else if (j == 0)

//如果一直遍历完质数表，该数没能整除任何数，说明它是质数

                primeNumber.push\_back(i);

//将找到的质数插入质数表

        }

    int count = 0;

    for (int i = 0; i != primeNumber.size(); i++) {

//遍历得到的质数表，输出所得到的质数

        cout << primeNumber[i] << "\t" ;

        count++;

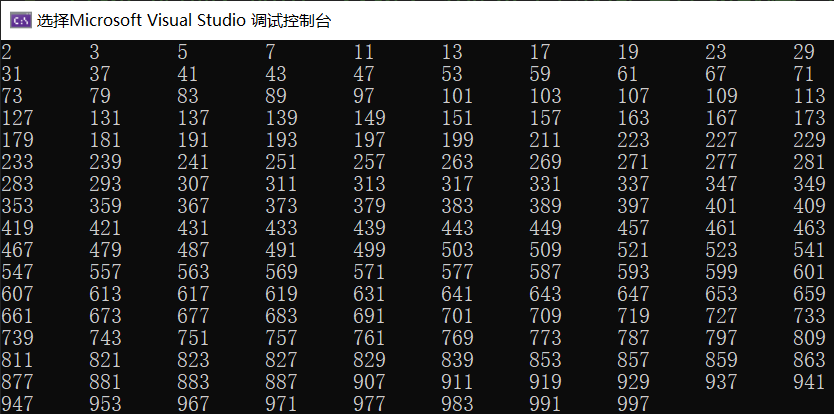
        if (count % 10 == 0)

            cout << "\n";

    }

}

运行截图：



1. **求解鸡兔同笼问题（via ）**

源代码：

#include<iostream>

#include<vector>

using namespace std;

/\*数学原理：让兔子和鸡同时抬起两只脚，这样笼子里的脚就减少了总头数×2只，

由于鸡只有2只脚，所以笼子里只剩下兔子的两只脚，再÷2就是兔子数。\*/

int main() {

    cout << "请分别输入头和脚的数量:";

    int numOfHead, numOfFoot;

    cin >> numOfHead >> numOfFoot;

    int numOfRabbit, numOfHen;

    numOfRabbit = (numOfFoot - numOfHead \* 2) / 2;

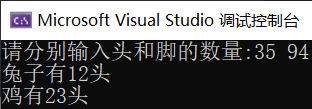
    numOfHen = numOfHead - numOfRabbit;

    cout << "兔子有" << numOfRabbit << "头" << endl;

    cout << "鸡有" << numOfHen << "头" << endl;

}

运行截图：



# 实验二

实验目标

了解程序运行运算来确定时间复杂度的评价，掌握事前分析中的程序步分析算法、掌握递归与分治策略，能用递归与分治方法解决实际问题，如汉诺塔等。

实验

1. **输出斐波那契数列（via ）**

源代码：

#include<stdio.h>

int fibonacci(int n) {

    if (n == 1 || n == 2)

        return 1;

    return fibonacci(n - 1) + fibonacci(n - 2);

}

int main() {

    int n, i;

    printf("请输入n的值:");

    scanf("%d", &n);

    for (i = 1; i <= n; i++)

        printf("%d ", fibonacci(i));

}

运行截图：



1. **输出一个整数划分（via ）**

源代码：

#include <stdio.h>

int q(int n, int m) {

if ((n < 1) || (m < 1))

return 0;

if ((n == 1) || (m == 1))

return 1;

if (n < m)

return q(n, n);

if (n == m)

return q(n, m - 1) + 1;

return q(n, m - 1) + q(n - m, m);

}

int main() {

int a, b;

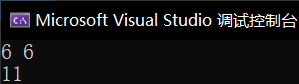
scanf("%d %d", &a, &b);

printf("%d", q(a, b));

return 0;

}

运行截图：



1. **汉诺塔移动次数递归设计（via ）**

源代码：

#include <stdio.h>

double g(int n) {

double s;

if (n == 1)

s = 1; //初始条件

else

s = 2 \* g(n - 1) + 1; //实施递归

return s;

}

int main() {

int m;

printf("请输入盘片数m:");

scanf("%d", &m);

printf("%d盘的移动次数为:%.0f\n", m, g(m));

}

运行截图：



1. **求两个矩阵乘积（via ）**

源代码：

#include<iostream>

#include<vector>

using namespace std;

//输入矩阵函数

void Input(vector<vector<int> >& matrix,int x,int y) {

    int temp;

    for (int i = 0; i < x; i++) {

        matrix.push\_back(vector<int>());

        for (int j = 0; j < y; j++) {

            cin >> temp;

            matrix[i].push\_back(temp);

        }

    }

}

//矩阵相加函数，传入矩阵matrixA和matrixB，返回矩阵matrixC

vector<vector<int> > MatrixMulti(vector<vector<int> >& matrixA, vector<vector<int> >& matrixB) {

    vector<vector<int> > matrixC;

    int temp;

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

//每进行一次外层循环，matrixC中插入新的一行

        matrixC.push\_back(vector<int>());

        for (int j = 0; j < matrixA.size(); j++) {

//每进行一次中层循环，在当前行中加入0

            matrixC[i].push\_back(0);

            for (int k = 0; k < matrixA[0].size(); k++)

                matrixC[i][j] += matrixA[i][k] \* matrixB[k][j];

        }

    }

    return matrixC;

}

int main(void) {

    vector<vector<int> > matrixA, matrixB, matrixC;

    int A\_x, A\_y, B\_x, B\_y;

    cout << "请输入两个矩阵：" << endl << "请输入第一个矩阵的行数和列数（格式：x y）：" << endl;

    cin >> A\_x >> A\_y;

    cout << "请输入第二个矩阵的行数和列数（格式：x y）：" << endl;

    cin >> B\_x >> B\_y;

    if (A\_y != B\_x) {

        cout << "两个矩阵无法相乘，请重新输入！";

        return 0;

    }

    cout << "请输入第一个矩阵：（格式：以空格分隔，每输入完一行敲一次回车）" << endl;

    Input(matrixA,A\_x,A\_y);

    cout << "请输入第二个矩阵：（格式：以空格分隔，每输入完一行敲一次回车）" << endl;

    Input(matrixB,B\_x,B\_y);

    matrixC = MatrixMulti(matrixA, matrixB);

    cout << "这两个矩阵相乘的结果是：" << endl;

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

        for (int j = 0; j < matrixC[i].size(); j++)

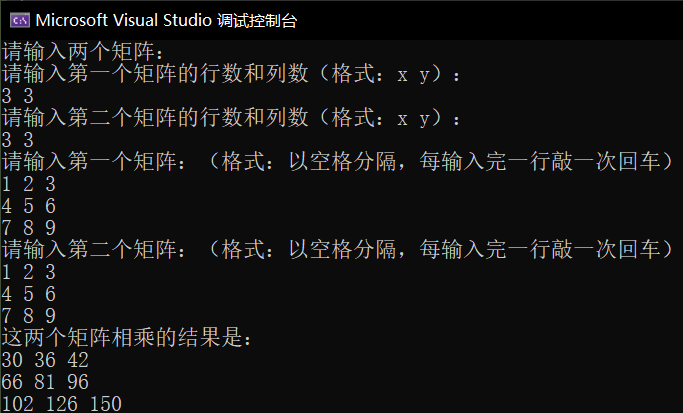
            cout << matrixC[i][j] << " ";

        cout << endl;

    }

}

运行截图：



1. **输出m+n个分别手持50元与100元买票可找开钱的排序数量（via ）**

源代码：

#include<stdio.h>

int s(int m,int n){

    int y = 0;

    if(n==0)

        y = 1;

    else if(m<n)

        y = 0;

    else

        y = s(m - 1, n) + s(m, n - 1);

    return y;

}

int main(void){

    int m, n;

    printf("请输入手持50元和手持100元的人的数量：\n");

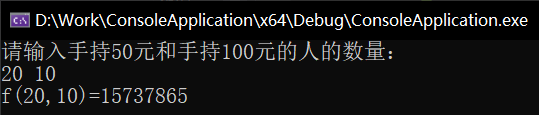
    scanf("%d %d", &m, &n);

    printf("f(%d,%d)=%d", m, n, s(m, n));

    return 0;

}

运行截图：



1. **实现排列A(n,m)递归程序设计：p(n,m)=n(n-1)(n-2)……(n-m+1)= n!/(n-m)!(规定0!=1)** **（via ）**

源代码：

#include<iostream>

using namespace std;

//求阶乘

int factorial(int a) {

    if (a == 0)

        return 1;

    else

        return a\*factorial(a - 1);

}

//根据公式，P(n,m)=n!/(n-m)!

int Pmt(int n, int m) {

    return factorial(n) / factorial(n - m);

}

int main(void) {

    int n, m;

    cout << "请输入两个数n,m：" << endl;

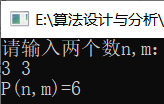
    cin >> n >> m;

    cout << "P(n,m)=" << Pmt(m, n) << endl;

    return 0;

}

运行截图：



1. **实现组合C(n,m)递归程序设计：C(n,m)=P(n,m)/P(m,m) =n!/m!(n-m)!** **（via ）**

源代码：

#include<iostream>

using namespace std;

int factorial(int a) {

    if (a == 0)

        return 1;

    else

        return a\*factorial(a - 1);

}

int Pmt(int n, int m) {

    return factorial(n) / factorial(n - m);

}

//根据公式，C(n,m)=P(n,m)/P(m,m)

int Cbt(int n, int m) {

    return Pmt(n, m) / Pmt(m, m);

}

int main(void) {

    int n, m;

    cout << "请输入两个数n,m：" << endl;

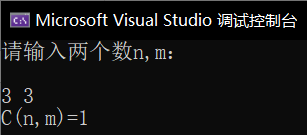
    cin >> n >> m;

    cout << "C(n,m)=" << Cbt(m, n) << endl;

    return 0;

}

运行截图：



作业

1. **随机产生一个某整数范围内的数组，随机产生一个该范围内整数，并查找该数在数组中的位置（via ）**

源代码：

#include<iostream>

#include<cstdlib>

#include<vector>

#include<ctime>

using namespace std;

int find(vector<int> a, int num) {

    for (auto i = a.begin(); i != a.end(); i++) {

        if (\*i == num)

            return i - a.begin();

    }

    return -1;

}

int main(void) {

    vector<int> intArray;

    srand((int)time(0));

    for (int i = 0; i != 100; i++)

        intArray.push\_back(rand() % 100);

    cout << "数组中的数为：" << endl;

    for (auto i = intArray.begin(); i != intArray.end(); i++)

        cout << \*i << '\t';

    cout << endl << "请输入要查找的数:";

    int num;

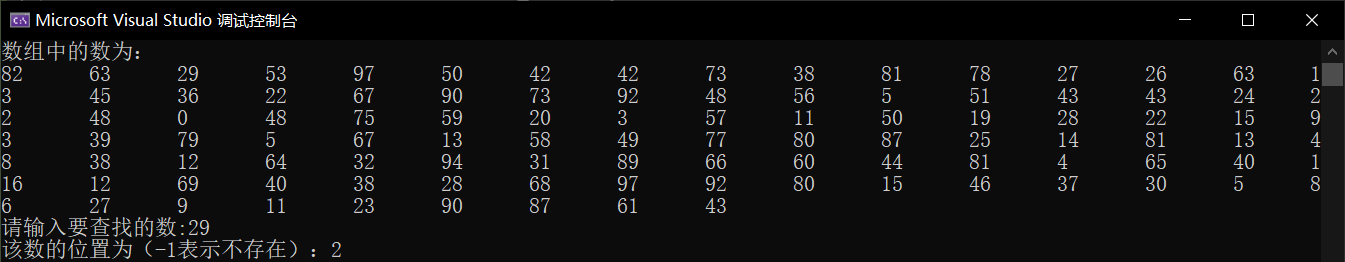
    cin >> num;

    cout << "该数的位置为（-1表示不存在）：" << find(intArray, num) << endl;

    return 0;

}

运行截图：



1. **随机产生一个数组，并用快速排序法排序输出。（via ）.**

源代码：

#include<iostream>

#include<cstdlib>

#include<vector>

#include<ctime>

using namespace std;

void quickSort(vector<int>& arr, int low, int high) {

    if (high <= low)

        return;

    int i = low;

    int j = high + 1;

    int key = arr[low];

    while (true) {

        while (arr[++i] < key) {

            if (i == high) {

                break;

            }

        }

        while (arr[--j] > key)

        {

            if (j == low) {

                break;

            }

        }

        if (i >= j)

            break;

        int temp = arr[i];

        arr[i] = arr[j];

        arr[j] = temp;

    }

    int temp = arr[low];

    arr[low] = arr[j];

    arr[j] = temp;

    quickSort(arr, low, j - 1);

    quickSort(arr, j + 1, high);

}

int main(void) {

    vector<int> intArray;

    srand((int)time(0));

    for (int i = 0; i != 10; i++)

        intArray.push\_back(rand() % 100);

    cout << "排序前的数组为：" << endl;

    for (auto i = intArray.begin(); i != intArray.end(); i++)

        cout << \*i << '\t';

    cout << endl;

    quickSort(intArray, 0, intArray.size() - 1);

    cout << "排序后的数组为：" << endl;

    for (auto i = intArray.begin(); i != intArray.end(); i++)

        cout << \*i << '\t';

    return 0;

}

运行截图：



**实验三**

实验目标

了解程序运行运算来确定时间复杂度的评价，掌握事前分析中的程序步分析算法、掌握动态规划算法，能用动态规算算法解决实际问题，如最长公共子序列等。

**实验**

1. **输出2^k循环赛日程表（via ）**

源代码：

#include <cstdio>

#include <math.h>

int a[1029][1029];

int m;

int main(void) {

    printf("请输入k的值：");

    scanf("%d", &m);

    int n = (int)pow(2,m), k = 1, half = 1;

    a[0][0] = 1;

    while (k <= m) {

        for (int i = 0; i < half; i++) {

            for (int j = 0; j < half; j++) {

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

            }

        }

        for (int i = 0; i < half; i++) {

            for (int j = 0; j < half; j++) {

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

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

            }

        }

        half \*= 2;

        k++;

    }

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            printf("%d", a[i][j]);

            if (j != n - 1) {

                printf("");

            }

        }

        printf("\n");

    }

    return 0;

}

运行截图：



1. **最长公共子序列（via ）**

源代码：

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int N = 1010;

char x[N], y[N];

int dp[N][N];

int b[N][N];

void Print(int i, int j) {

    if (i == 0 || j == 0) {

        return;

    }

    if (b[i][j] == 1) {

        Print(i - 1, j - 1);

        printf("%c", x[i - 1]);

    }

    else if (b[i][j] == 2) {

        Print(i - 1, j);

    }

    else if (b[i][j] == 3){

        Print(i, j - 1);

    }

}

int main(void) {

    int lena, lenb, i, j;

    scanf("%s%s", x, y);

    memset(dp, 0, sizeof(dp));

    memset(b, 0, sizeof(b));

    lena = strlen(x);

    lenb = strlen(y);

    for (i = 1; i <= lena; i++) {

        for (j = 1; j <= lenb; j++) {

            if (x[i - 1] == y[j - 1]) {

                dp[i][j] = dp[i - 1][j - 1] + 1;

                b[i][j] = 1;///来自于左上方

            }

            else {

                if (dp[i - 1][j] > dp[i][j - 1]) {

                    dp[i][j] = dp[i - 1][j];

                    b[i][j] = 2;///来自于左方

                }

                else {

                    dp[i][j] = dp[i][j - 1];

                    b[i][j] = 3;///来自于上方

                }

            }

        }

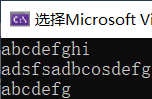
    }

    Print(lena, lenb);

    return 0;

}

运行截图：



1. **寻找两个或多个已知字符串最长的子串（via ）**

源代码：

#include<iostream>

#include<vector>

#include<string>

using namespace std;

//按照特定字符分割函数

vector<string> splitstr(const string& str, char tag) {

    vector<string>  li;

    string subStr;

    for (size\_t i = 0; i < str.length(); i++) {

        if (tag == str[i]) {

            if (!subStr.empty()) {

                li.push\_back(subStr);

                subStr.clear();

            }

        }

        else {

            subStr.push\_back(str[i]);

        }

    }

    if (!subStr.empty()) {

        li.push\_back(subStr);

    }

    return li;

}

//查找最长公共子串

string findLongestCommonSubstring(const string& s1, const string& s2) {

    if (s1.length() == 0 || s2.length() == 0) {

        return "";

    }

    string temp = "";

    for (int i = 0; i < s1.length(); i++) {

        for (int j = i, k = 0; j < s1.length() && k < s2.length(); j++, k++)

            if (s1[j] == s2[k])

                temp.push\_back(s1[j]);

        if(temp[temp.length()-1]!='\0')

            temp.push\_back(' ');

    }

    if(temp[temp.length()-1] != '\0')

        temp.push\_back(' ');

    for (int i = 0; i < s2.length(); i++) {

        for (int j = i, k = 0; j < s2.length() && k < s1.length(); j++, k++)

            if (s1[j] == s2[k])

                temp.push\_back(s2[j]);

        if (temp[temp.length()-1] != '\0')

            temp.push\_back(' ');

    }

    if (temp == "")

        return "";

    for (auto i = temp.begin(); i != temp.end(); i++) {

        if (\*i != ' ') {

            if (i == temp.end() - 1)

                return temp;

            else

                continue;

        }

        else

            break;

    }

    auto result = splitstr(temp, ' ');

    int maxlength = 0;

    for (auto i = result.begin(); i != result.end(); i++)

        if ((\*i).length() > maxlength)

            maxlength = (\*i).length();

    for (auto i = result.begin(); i != result.end(); i++)

        if ((\*i).length() == maxlength)

            return \*i;

}

int main(void) {

    int count = 0;

    cout << "请输入字符串数量：";

    cin >> count;

    if (count < 2) {

        cout << "字符串数量不可以少于2" << endl;

        return 0;

    }

    string s1 = "", s2 = "";

    cout << "请输入第1个字符串：" << endl;

    cin >> s1;

    count --;

    int count2 = 2;

    while (count) {

        cout << "请输入第" << count2 << "个字符串：" << endl;

        s2 = "";

        cin >> s2;

        s1 = findLongestCommonSubstring(s1, s2);

        if (s1 == "") {

            cout << "无最长公共子串";

            return 0;

        }

        count--;

        count2++;

    }

    cout << "最长公共子串为：" << s1;

    return 0;

}

运行截图：



1. **动态规划法实现0-1背包问题（via ）**

源代码：

#include<stdio.h>

#define N 10

#define Q 100

int V[N + 1][Q + 1];

int knapsack(int n, int w[], int v[], int C) {

    int i, j;

    for (j = 0; j <= C; j++)

        V[0][j] = 0;

    for (i = 0; i <= n; i++)

        V[i][0] = 0;

    for (i = 1; i <= n; i++)

        for (j = 1; j <= C; j++) {

            if (j < w[i - 1])

                V[i][j] = V[i - 1][j];

            else if (V[i - 1][j] > V[i - 1][j - w[i - 1]] + v[i - 1])

                V[i][j] = V[i - 1][j];

            else

                V[i][j] = V[i - 1][j - w[i - 1]] + v[i - 1];

        }

    return V[n][C];

}

int main() {

    int w[N], v[N], C, maxvalue;

    int n, i;

    printf("请输入物体的个数:\n");

    scanf("%d", &n);

    printf("请输入物体的重量:\n");

    for (i = 0; i < n; i++)

        scanf("%d", &w[i]);

    printf("请输入物体的价值:\n");

    for (i = 0; i < n; i++)

        scanf("%d", &v[i]);

    printf("请输入背包的容量:\n");

    scanf("%d", &C);

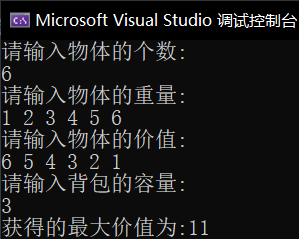
    maxvalue = knapsack(n, w, v, C);

    printf("获得的最大价值为:%d\n", maxvalue);

    return 0;

}

运行截图：



# 实验四

实验目标

理解贪心算法的思想，原理，掌握贪心算法解决实际问题，如活动安排，单元最短路径搜寻、多机调度安排等。

实验

1. **安排活动算法程序（via ）**

源代码：

#include<stdio.h>

void sort(int s[], int f[], int n) {

    int a, b;

    int i, j;

    for (i = 0; i < n; i++) {

        for (j = i + 1; j < n; j++) {

            if (f[i] > f[j]) {

                a = f[i]; f[i] = f[j]; f[j] = a;

                b = s[i]; s[i] = s[j]; s[j] = b;

            }

        }

    }

}

int activemanage(int s[], int f[], bool a[], int n) {

    a[0] = 1;

    int i;

    int j = 1, count = 1;

    for (i = 1; i < n; i++) {

        if (s[i] >= f[j]) {

            a[i] = 1;

            j = i;

            count++;

        }

        else a[i] = 0;

    }

    return count;

}

int main(void) {

    int i, n;

    int p;

    int s[100], f[100];

    bool a[100];

    printf("输入节目数：\n");

    scanf("%d", &n);

    printf("请依次输入节目的开始和结束时间\n");

    for (i = 0; i < n; i++) {

        scanf("%d %d", &s[i], &f[i]);

    }

    sort(s, f, n);

    p = activemanage(s, f, a, n);

    printf("安排的节目个数为:%d\n", p);

    printf("节目的选取情况为(0表示不选 1表示选取):\n");

    for (i = 0; i < n; i++)

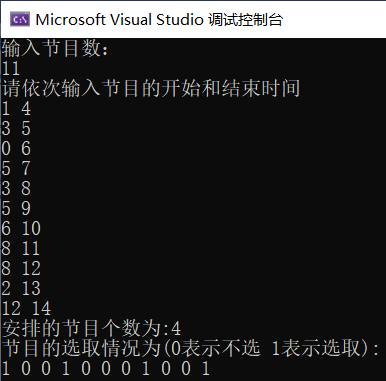
        printf("%d ", a[i]);

    printf("\n");

    return 0;

}

运行截图：



1. **可拆分背包问题（via ）**

源代码：

#include <iostream>

#include <algorithm>

#include<iomanip>

using namespace std;

//做一个排序，需要将性价比由高到低排序，排序的过程中重量和（价值）要对应上

typedef struct {

    double aver;

    double w;

    double v;

}Knapsack;

bool cmp(Knapsack a, Knapsack b) {

    return a.aver > b.aver;

}

int main() {

    Knapsack arrays[1009];

    int n;

    double m;

    double V = 0;

    cout << "请输入物品数量n和背包容量m" << endl;

    cin >> n >> m;

    cout << "请依次输入物品的价值" << endl;

    for (int i = 0; i < n; i++)

        cin >> arrays[i].v;

    cout << "请依次输入物品的重量" << endl;

    for (int i = 0; i < n; i++)

        cin >> arrays[i].w;

    //求性价比

    for (int i = 0; i < n; i++) {

        arrays[i].aver = arrays[i].v / arrays[i].w;

        //cout << arrays[i].aver << endl;

    }

    //性价比排序

    sort(arrays, arrays + n, cmp);

    int sum = 0;

    for (int i = 0; i < n; i++) {//当背包能装下所有物品时，直接输出所有的物品价值之和

        sum += arrays[i].w;

    }

    if (sum < m) {

        for (int j = 0; j < n; j++)

            V += arrays[j].v;

        cout << setiosflags(ios::fixed) << setprecision(3) << V << endl;

        return 0;

    }

    //应该由性价比的顺序，通过容量，选择装入的物品

    for (int i = 0; i < n; i++) {

        if (arrays[i].w <= m) {

            V = V + arrays[i].v;

            m = m - arrays[i].w;

        }

        else {//直接将剩余的m加入即可

            V = V + m \* arrays[i].aver;

            m = 0;

        }

        if (m == 0)

            break;

    }

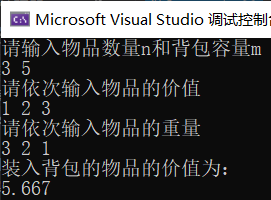
    cout << "装入背包的物品的价值为：" << endl;

    cout << setiosflags(ios::fixed) << setprecision(3) << V << endl;

    return 0;

}

运行截图：



1. **单源最短路径（via ）**

源代码：

#include <iostream>

#include <cstdio>

#define MAX 1000000

using namespace std;

int arcs[10][10];   //邻接矩阵

int D[10];          //保存最短路径长度

int p[10][10];      //路径

int final[10];      //若final[i] = 1则说明 顶点vi已在集合S中

int n = 0;          //顶点个数

int v0 = 0;         //源点

int v, w;

void ShortestPath\_DIJ() {

    for (v = 0; v < n; v++) {//初始化

        final[v] = 0; D[v] = arcs[v0][v];

        for (w = 0; w < n; w++)

            p[v][w] = 0;    //设空路径

        if (D[v] < MAX) { p[v][v0] = 1; p[v][v] = 1; }

    }

    D[v0] = 0; final[v0] = 0; //初始化 v0顶点属于集合S

    //开始主循环 每次求得v0到某个顶点v的最短路径 并加v到集合S中

    for (int i = 1; i < n; i++) {

        int min = MAX;

        for (w = 0; w < n; w++) {

            //选点

            if (!final[w]) {//如果w顶点在V-S中

                //选出当前V-S中与S有关联边且当前离V0最近的点

                if (D[w] < min) { v = w; min = D[w]; }

            }

        }

        final[v] = 1;               //选出该点后加入到合集S中

        for (w = 0; w < n; w++){    //更新当前最短路径和距离

            /\*v为当前刚选入集合S中的点

            以点V为中间点，看d0v+dvw是否小于D[w]，如果小于则更新

            \*/

            if (!final[w] && (min + arcs[v][w] < D[w])) {

                D[w] = min + arcs[v][w];

                p[w][w] = 1;

            }

        }

    }

}

int main() {

    cout << "请输入有向图的顶点个数：";

    cin >> n;

    cout << "请输入有向图的邻接矩阵：" << endl;

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            cin >> arcs[i][j];

        }

    }

    ShortestPath\_DIJ();

    cout << "最短路径为：" << endl;

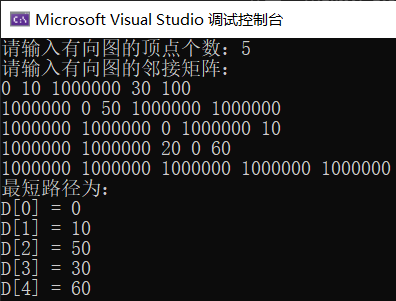
    for (int i = 0; i < n; i++)

        printf("D[%d] = %d\n", i, D[i]);

    return 0;

}

运行截图：



1. **多机调度问题（via ）**

源代码：

#include<iostream>

#include<algorithm>

#include<string.h>

using namespace std;

int speed[10010];//作业时间

int mintime[101];//存放添加作业后完成的时间

bool cmp(const int& x, const int& y) {

    return x > y;//比较函数，求较大值

}

int main() {

    int m, n;

    memset(speed, 0, sizeof(speed));//将函数清空

    memset(mintime, 0, sizeof(mintime));

    cout << "请输入作业数n和机器数m：" << endl;

    cin >> n >> m;

    cout << "请输入每个作业所需的处理时间:" << endl;

    for (int i = 0; i < n; i++) {

        cin >> speed[i];

    }

    sort(speed, speed + n, cmp);//对speed数组进行降序排序

    for (int i = 0; i < n; ++i) {

        \*min\_element(mintime, mintime + m) += speed[i];//m是机器数，依次对m台机器添中最小的加speed

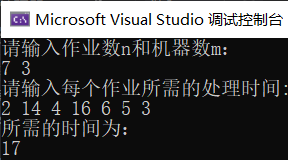
    }

    cout << "所需的时间为：" << endl << \*max\_element(mintime, mintime + m) << endl;//求mintime数组中的最大值（所有作业都已完成的时间）

    return 0;

}

运行截图：



作业

1. **Prim算法构造最小生成树（via ）**

源代码：

#include<stdio.h>

#include<stdlib.h>

#define MAXLEN 10

#define INF 9999999

typedef struct {

    char vexs[MAXLEN];

    int edges[MAXLEN][MAXLEN];

    int n, e;

}MGraph;

typedef struct {

    int lowcost;

    char vex;

}TempArray;

void creatMGraph(MGraph\* G) {

    int i, j, k;

    char ch1, ch2;

    printf("请输入顶点数和边数（格式：x,y）：\n");

    scanf("%d,%d", &(G->n), &(G->e));

    printf("请输入顶点信息（用大写英文字母表示，以空格为间隔）：\n");

    getchar();

    for (i = 0; i < G->n; i++) {

        G->vexs[i] = getchar();

        if (G->vexs[i] == ' ')

            i--;

    }

    for (i = 0; i < G->n; i++)

        for (j = 0; j < G->n; j++)

            G->edges[i][j] = 0;

    printf("请输入该图的邻接矩阵（以空格为间隔，用-1表示两个顶点无直接通路）：\n");

    for (i = 0; i < G->n; i++)

        for (j = 0; j < G->n; j++) {

            scanf("%d", &(G->edges[i][j]));

            if (G->edges[i][j] == -1)

                G->edges[i][j] = INF;

        }

}

void outputMST(MGraph\* G, int u) {

    int i, j, min, k;

    TempArray closedge[MAXLEN];

    for (i = 0; i <= G->n; i++) {

        closedge[i].vex = u + 'A';

        closedge[i].lowcost = G->edges[u][i];

    }

    for (i = 1; i < G->n; i++) {

        min = INF;

        for (j = 0; j < G->n; j++)

            if (closedge[j].lowcost != 0 && closedge[j].lowcost < min) {

                min = closedge[j].lowcost;

                k = j;

            }

        printf("边(%c,%c)，权:%d\n", closedge[k].vex, k + 'A', min);

        closedge[k].lowcost = 0;

        for (j = 0; j < G->n; j++)

            if (G->edges[k][j] < closedge[j].lowcost && G->edges[k][j] != 0) {

                closedge[j].lowcost = G->edges[k][j];

                closedge[j].vex = k + 'A';

            }

    }

}

int main(void) {

    int u;

    MGraph G, \* g;

    g = &G;

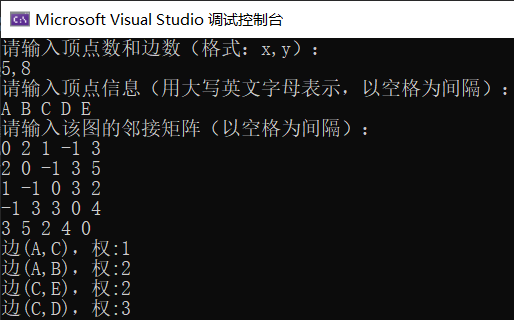
    creatMGraph(g);

    outputMST(g, 0);

    return 0;

}

运行截图：



1. **最优装载问题（via ）**

源代码：

#include<iostream>

#include<algorithm>

#define MAXN 1000005

using namespace std;

int w[MAXN];            //每件物品的重量

int main() {

    int c, n;           //c为载重量，n为物品数

    int sum = 0;        //装入物品的数量

    int tmp = 0;        //装入物品的重量

    cout << "请输入载重量和物品数：";

    cin >> c >> n;

    cout << "请输入物品的重量：";

    for (int i = 1; i <= n; ++i)

        cin >> w[i];

    sort(w + 1, w + 1 + n);

    for (int i = 1; i <= n; ++i){

        tmp += w[i];

        if (tmp <= c)

            ++sum;

        else

            break;

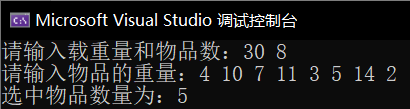
    }

    cout << "选中物品数量为：" << sum << endl;

    return 0;

}

运行截图：



# 实验五

实验目标

理解回溯算法的思想，原理，掌握回溯算法解决实际问题的方法，如回溯算法如何解决装载问题、连续邮资问题等。

实验

1. **批处理作业调度问题（via ）**

源代码：

#include<iostream>

using namespace std;

//当前作业调度————其中一种排列顺序

int x[100];

//当前最优作业调度

int bestx[100];

//各作业所需的处理时间，M[j][i]代表第j个作业在第i台机器上的处理时间

int m[100][100];

//机器1完成处理时间

int f1 = 0;

//机器2完成处理时间

int f2 = 0;

//完成时间和

int cf = 0;

//当前最优值，即最优的处理时间和

int bestf = 10000;

//作业数

int n;

//t用来指示到达的层数（第几步，从0开始），同时也指示当前执行完第几个任务/作业

void Backtrack(int t) {

    int tempf, j;

    //到达叶子结点，搜索到最底部

    if (t > n) {

        if (cf < bestf) {

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

                //更新最优调度序列

                bestx[i] = x[i];

            //更新最优目标值

            bestf = cf;

        }

    }

    //非叶子结点

    else {

        //j用来指示选择了哪个任务/作业（也就是执行顺序）

        for (j = t; j <= n; j++) {

            //选择第x[j]个任务在机器１上执行，作为当前的任务

            f1 += m[x[j]][1];

            //保存上一个作业在机器2的完成时间

            tempf = f2;

            //保存当前作业在机器2的完成时间

            f2 = (f1 > f2 ? f1 : f2) + m[x[j]][2];

            //在机器2上的完成时间和

            cf += f2;

            //如果该作业处理完之后，总时间已经超过最优时间，就直接回溯。

            //剪枝函数

            if (cf < bestf) {//总时间小于最优时间

                //交换两个作业的位置，把选择出的原来在x[j]位置上的任务调到当前执行的位置x[t]

                swap(x[t], x[j]);

                //深度搜索解空间树，进入下一层

                Backtrack(t + 1);

                //进行回溯，还原，执行该层的下一个任务,如果是叶子节点返回上一层

                swap(x[t], x[j]);

            }

            //回溯需要还原各个值

            f1 -= m[x[j]][1];

            cf -= f2;

            f2 = tempf;

        }

    }

}

int main() {

    int i, j;

    cout << "请输入作业数：";

    cin >> n;

    cout << "请输入在各机器上的处理时间" << endl;

    //ｉ从１开始

    for (i = 1; i <= 2; i++)

        for (j = 1; j <= n; j++)

            //第j个作业,第i台机器的时间值

            cin >> m[j][i];

    for (i = 1; i <= n; i++)

        //初始化当前作业调度的一种排列顺序

        x[i] = i;

    Backtrack(1);

    cout << "调度作业顺序：" << endl;

    for (i = 1; i <= n; i++)

        cout << bestx[i] << ' ';

    cout << endl;

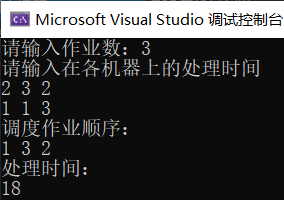
    cout << "处理时间：" << endl;

    cout << bestf;

    return 0;

}

运行截图：



1. **N皇后问题（via ）**

源代码：

#include<iostream>

#include<vector>

using namespace std;

//输出函数，用于打印所有的解

void outputResult(const vector<vector<int> >& result) {

    if (result.size() == 0) {

        cout << "无解" << endl;

        return;

    }

    for (auto i = result.begin(); i != result.end(); i++) {

        for (auto j = (\*i).begin(); j != (\*i).end(); j++) {

            for (int k = 0; k < (\*i).size(); k++) {

                if (k == \*j)

                    cout << 'Q';

                else

                    cout << 'B';

            }

            cout << endl;

        }

        cout << endl;

    }

    cout << "解的个数为：" << (result.size());

}

//检查当前位置是否可行,包括列检查和对角线检查，

//对角线检查原理：|x1-x2|！=|y1-y2|（x,y表示行号和列号）

bool checkPosition(const vector<int>& queenInfo, const int& X, const int& Y) {

    for (int i = 0; i < X; i++)

        if ((queenInfo[i] == Y) || (abs(X - i) == abs(Y - queenInfo[i])))

            return false;

    return true;

}

//求解函数

vector<vector<int> > findSolution(const int& qn) {

    //二维数组，用于存放解，每一行表示一种解

    vector<vector<int> >allResult;

    /\*根据皇后个数定义数组，下标表示皇后所在行数，值表示皇后所在列数，

    qn表示将数组初始化为qn个int默认值，即qn个0；相当于一个栈\*/

    vector<int> thisResult(qn);

    int i = 0; //定义栈指针

    //进行反复回溯遍历（一行必定只有一个皇后），直到第一个皇后没有位置放

    while(thisResult[0] < qn){

        //当前皇后在当前行每一列进行尝试

        while(thisResult[i] < qn){

            //如果当前位置可行，跳出该内循环，尝试下一个皇后

            if (checkPosition(thisResult, i, thisResult[i]))

                break;

            //不可行则移动到下一位置

            thisResult[i]++;

        }

        /\*如果当前行皇后放在“第n+1格”，说明该皇后在当前行没有位置可以放，则对前一个皇后进行操作；如果是第一个皇后没地方放则表示已经遍历结束应结束程序，这段程序最终能让循环结束\*/

        if (thisResult[i] == qn && i != 0) {

            //让当前行皇后回到原位

            thisResult[i] = 0;

            //出栈（即回溯到前一行），让前一行的皇后挪到下一列

            thisResult[--i]++;

            continue;

        }

        //如果最后一个皇后也放置成功了，说明当前解为有效解

        if (i == qn - 1) {

            //把当前解存入解集

            allResult.push\_back(vector<int>(thisResult));

            //如果最后一个皇后并未遍历完所有列

            if (thisResult[i] < qn) {

                //将最后一个皇后向前移动，检查是否还有其他的解

                thisResult[i]++;

                //后续没有皇后，跳过入栈，让最后一个皇后重新进入内循环检查

                continue;

            }

        }

        //新的皇后入栈

        i++;

    }

    return allResult;

}

int main(void) {

    int queenNum;

    cout << "请输入皇后数量：";

    cin >> queenNum;

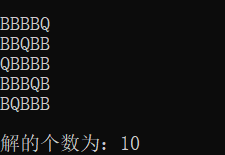
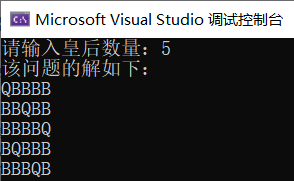
    cout << "该问题的解如下：" << endl;

    outputResult(findSolution(queenNum));

    return 0;

}

运行截图：



1. **TSP售货员问题（via ）**

源代码：

#include<iostream>

#include<climits>

using namespace std;

//定义INF为整型的上限

constexpr int INF = INT\_MAX;

//n为解空间树的层数/城市数，cc为当前所走的路程，bestc为当前的最优值

int n, cc = 0, bestc = INF;

//g为图的邻接矩阵

int\*\* g;

//x为城市的数组，bestx存放最优值对应的路线

int\* x, \* bestx;

void travel(int t) {

    //t到达第n层，即到达叶子结点，已经形成一个完整的解

    if (t == n) {

        /\*城市x[t-1]可以到达城市x[t]，并且城市x[t]可以回到城市1，

        且此时所走的路程cc加上x[t - 1]与x[t]的距离和x[t]与1的距离小于当前最优值bestc\*/

        if (g[x[t - 1]][x[t]] != INF && g[x[t]][1] != INF &&

            (cc + g[x[t - 1]][x[t]] + g[x[t]][1] < bestc || bestc == INF)) {

            //记录最优解

            for (int i = 0; i < n + 1; i++)

                bestx[i] = x[i];

            //记录最优值

            bestc = cc + g[x[t - 1]][x[t]] + g[x[t]][1];

        }

        return;

    }

    for (int i = t; i < n; i++) {

        /\*城市x[t-1]能达到城市x[i]即这两个城市间有边，

        并当前所走的路程cc加上这两个城市的距离没有比当前最优值bestc大\*/

        if (g[x[t - 1]][x[i]] != INF && (cc + g[x[t - 1]][x[i]] < bestc

            || bestc == INF)) {

            //交换

            swap(x[i], x[t]);

            //修改所走的路程

            cc += g[x[t - 1]][x[t]];

            //进入下一层递归

            travel(t + 1);

            //恢复原来的值

            cc -= g[x[t - 1]][x[t]];

            swap(x[i], x[t]);

        }

    }

}

void output() {

    cout << "最短的路程为：" << bestc << endl;

    cout << "所经过的路线为：" << endl;

    for (int i = 1; i < n + 1; i++)

        cout << bestx[i] <<' ';

    //因为要回到起点，所以要额外再输出一次起点

    cout << bestx[1];

}

int main() {

    cout << "请输入城市数：";

    cin >> n;

    g = new int\* [n + 1];

    x = new int[n + 1];

    bestx = new int[n + 1];

    //将所有的数组初始化

    for (int i = 0; i < n + 1; i++) {

        g[i] = new int[n + 1];

        x[i] = i;

        //把邻接矩阵初始化为INF

        for (int j = 0; j < n + 1; j++)

            g[i][j] = INF;

    }

    //图的邻接矩阵，方便起见数组下标全部从1开始

    cout << "请输入图的邻接矩阵(-1表示城市无通路)：" << endl;

    for (int i = 1; i <= n; i++)

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

            int temp;

            cin >> temp;

            if (temp == -1)

                g[i][j] = INF;

            else

                g[i][j] = temp;

        }

    //起点t=1已经由题目给出，因此t从2开始

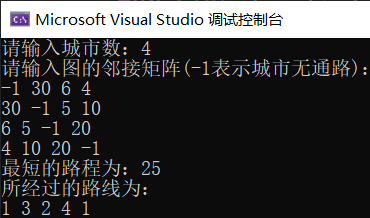
    travel(2);

    output();

    return 0;

}

运行截图：



1. **连续邮资问题（via ）**

源代码：

#include <iostream>

#define NUM 10

#define LEN 10000

using namespace std;

int x[NUM];

int cnt = 0;//当前邮票种类

int knd = 0;//邮票种数

int lim = 0;//最大张数

int max = 0;//邮资最大值

int C[NUM][LEN] = {};

int findMax() {

    int j = 1;

    while (C[cnt - 1][j]) {

        if (j < x[cnt] || C[cnt - 1][j] <= C[cnt][j - x[cnt]] + 1)

            C[cnt][j] = C[cnt - 1][j];

        else

            C[cnt][j] = C[cnt][j - x[cnt]] + 1;

        j++;

    }

    while (true) {

        int tmp = INT\_MAX;

        for (int i = 1; i <= cnt; i++) {

            if (tmp > C[cnt][j - x[i]] + 1)

                tmp = C[cnt][j - x[i]] + 1;

        }

        if (tmp == INT\_MAX || tmp > lim)

            break;

        else

            C[cnt][j] = tmp;

        j++;

    }

    C[cnt][j] = 0;

    return j - 1;

}

void dfs() {

    if (cnt == knd) {

        if (x[cnt] \* lim < max)

            return;

        int tmp = findMax();

        if (tmp > max)

            max = tmp;

    }

    else {

        int tmp = findMax();

        for (int i = tmp + 1; i >= x[cnt] + 1; i--) {

            x[++cnt] = i;

            dfs();

            cnt--;

        }

    }

}

int main() {

    x[1] = 1;

    cnt = 1;

    cout << "请输入邮票种数和最多允许贴的邮票数：";

    cin >> knd >> lim;

    cout << "请输入邮票种类：";

    for (int i = 1; i <= knd; i++)

        cin >> x[i];

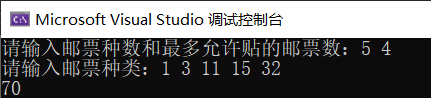
    dfs();

    cout << max;

    return 0;

}

运行截图：



作业

1. **图着色问题（via ）**

源代码：

#include<iostream>

using namespace std;

//v顶点数， e边数， graph图的邻接矩阵

int v, e, graph[100][100];

//c颜色数  color当前边的颜色

int c, color[100];

//着色方法的数目

int sum = 0;

//判断当前位置的颜色是否和相邻位置颜色重复

bool ok(int cur) {

    for (int i = 1; i <= v; i++) {

        //如果坐标(cur,i)相邻 且 cur的颜色和i的颜色相同

        if (graph[cur][i] && color[cur] == color[i]) {

            return false;

        }

    }

    return true;

}

void backtrace(int cur) {

    if (cur > v) {

        sum++;

    }

    else {

        //分别为cur位置尝试第i中颜色

        for (int i = 1; i <= c; i++) {

            //表示cur位置图上第i种颜色

            color[cur] = i;

            if (ok(cur)) {

                backtrace(cur + 1);

            }

            color[cur] = 0;

        }

    }

}

int main() {

    cout << "请输入顶点数和颜色数量：";

    cin >> v >> c;

    cout << "请输入图的邻接矩阵：" << endl;

    //输入邻接矩阵

    for (int i = 1; i <= v; i++) {

        for (int j = 1; j <= v; j++) {

            cin >> graph[i][j];

        }

    }

    backtrace(1);

    cout << "该图的色数为：" <<sum;

    return 0;

}

运行截图：



1. **装载问题（via ）**

源代码：

#include <cstdio>

using namespace std;

int n; //集装箱数

int cw; // 当前载重量, current weight

int bestw; //最优载重重量

int r;  //剩余集装箱重量

int c1; //第一艘轮船的载重量

int c2; //第二艘轮船的载重量

int x[100]; //当前解

int bestx[100]; //当前最优解

int w[100]; //集装箱重量数组

void OutPut() {

    int restweight = 0;

    for (int i = 1; i <= n; ++i)

        if (bestx[i] == 0)

            restweight += w[i];

    if (restweight > c2)

        printf("不能装入\n");

    else {

        printf("船1装入的货物为:");

        for (int i = 1; i <= n; ++i)

            if (bestx[i] == 1)

                printf(" %d", i);

        printf("\n船2装入的货物为:");

        for (int i = 1; i <= n; ++i)

            if (bestx[i] != 1)

                printf(" %d", i);

    }

}

void BackTrack(int i) {

    if (i > n) {

        if (cw > bestw) {

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

                bestx[i] = x[i];

            bestw = cw;

        }

        return;

    }

    r -= w[i];

    //约束条件

    if (cw + w[i] <= c1) {

        cw += w[i];

        x[i] = 1;

        BackTrack(i + 1);

        x[i] = 0;

        cw -= w[i];

    }

    //限界函数

    if (cw + r > bestw) {

        x[i] = 0;

        BackTrack(i + 1);

    }

    r += w[i];

}

void Initialize() {

    bestw = 0;

    r = 0;

    cw = 0;

    for (int i = 1; i <= n; ++i)

        r += w[i];

}

void InPut() {

    printf("请输入货物数量：");

    scanf\_s("%d", &n);

    printf("请输入两艘船的容量：");

    scanf\_s("%d %d", &c1, &c2);

    printf("请依次输入货物重量：");

    for (int i = 1; i <= n; ++i)

        scanf\_s("%d", &w[i]);

}

int main() {

    InPut();

    Initialize();

    BackTrack(1);

    OutPut();

    return 0;

}

运行截图：



# 实验六

实验目标

理解掌握分支限界法思想，原理，能够设计程序解决实际问题，如分支限界法如何解决单源最短路径问题、装载问题等。

实验

1. **单源最短路径问题（via ）**

源代码：

#include <iostream>

#include<queue>

#include<climits>

using namespace std;

typedef struct ArcCell {

    //保存权值

    int adj;

    //存储最短路径长度

    int info;

}ArcCell, AdjMaxtrix[100][100];

typedef struct {

    int data;

    int length;

}VerType;

typedef struct {

    //顶点向量

    VerType vexs[100];

    AdjMaxtrix arcs;

    //顶点数

    int vexnum;

    //弧数

    int arcnum;

}Graph;

Graph G;

queue<int> q;

void CreateGraph() {

    int m, n, t;

    printf("输入顶点数和弧数:");

    scanf\_s("%d%d", &G.vexnum, &G.arcnum);

    printf("输入顶点:");

    for (int i = 1; i <= G.vexnum; i++) {

        scanf\_s("%d", &G.vexs[i].data);

        G.vexs[i].length = INT\_MAX;

    }

    for (int i = 1; i <= G.vexnum; i++)

        for (int j = 1; j <= G.vexnum; j++) {

            G.arcs[i][j].adj = 0;

        }

    printf("输入弧及权重:\n");

    for (int i = 1; i <= G.arcnum; i++) {

        scanf\_s("%d%d%d", &m, &n, &t);

        G.arcs[m][n].adj = 1;

        G.arcs[m][n].info = t;

    }

}

int NextAdj(int v, int w) {

    for (int i = w + 1; i <= G.vexnum; i++)

        if (G.arcs[v][i].adj)

            return i;

    //返回0表示未找到

    return 0;

}

void ShortestPaths(int v) {

    //从首个节点开始访问

    int k = 0;

    int t;

    G.vexs[v].length = 0;

    q.push(G.vexs[v].data);

    while (!q.empty()) {

        t = q.front();

        k = NextAdj(t, k);

        while (k != 0) {

            //减枝操作

            if (G.vexs[t].length + G.arcs[t][k].info <= G.vexs[k].length) {

                G.vexs[k].length = G.vexs[t].length + G.arcs[t][k].info;

                q.push(G.vexs[k].data);

            }

            k = NextAdj(t, k);

        }

        q.pop();

    }

}

void Print() {

    cout << "从源点出发到各个点的最短路径为：" << endl;

    for (int i = 1; i <= G.vexnum; i++) {

        cout << "1→" << G.vexs[i].data << ":";

        if(G.vexs[i].length==INT\_MAX)

            cout << "∞" << endl;

        else

            cout << G.vexs[i].length << endl;

    }

}

int main() {

    CreateGraph();

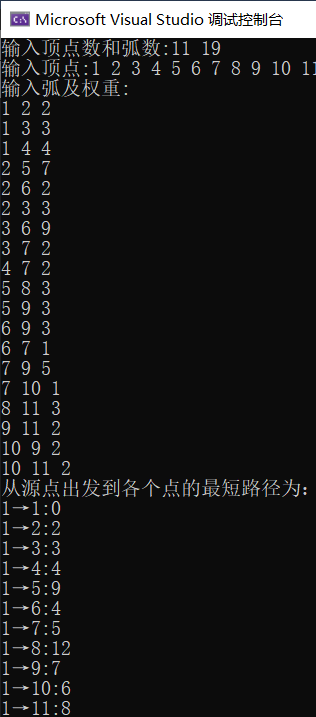
    ShortestPaths(1);

    Print();

    return 0;

}

运行截图：



1. **装载问题（via ）**

源代码：

#include<queue>

#include<iostream>

using namespace std;

typedef struct QNode {

    QNode\* parent;

    int lchild;

    int weight;

}QNode;

int n;

int c;

int bestw;

int w[100];

int bestx[100];

void InPut() {

    cout << "请输入货物数量和船的装载量：";

    cin >> n >> c;

    cout << "请依次输入货物重量：";

    for (int i = 1; i <= n; ++i)

        cin >> w[i];

}

/\*QNode \*&bestE 的原因是 首先bestE是个地址，

其次引用为了赋值使用, 后边for循环中用到\*/

void EnQueue(queue<QNode\*>& q, int wt, int i, QNode\* E, QNode\*& bestE, int ch) {

    if (i == n) {

        if (wt == bestw) {

            bestE = E;

            bestx[n] = ch;

            return;

        }

    }

    QNode\* b;

    b = new QNode;

    b->weight = wt;

    b->lchild = ch;

    b->parent = E;

    q.push(b);

}

void MaxLoading() {

    queue<QNode\*>q;

    q.push(0);

    int i = 1;

    int Ew = 0, r = 0;

    bestw = 0;

    for (int j = 2; j <= n; ++j)

        r += w[j];

    /\*结束while循环后，bestE指向最优解的叶子节点，

    然后通过bestE->parent找到装入了哪些物品。\*/

    QNode\* E, \* bestE;

    //E这里作为一个中间量，连接parent和child

    E = new QNode;

    //赋0是因为树的根的值是0，while刚开始的时候其代表root

    E = 0;

    while (true) {

        int wt = Ew + w[i];

        if (wt <= c) {

            if (wt > bestw)

                bestw = wt;

            EnQueue(q, wt, i, E, bestE, 1);

        }

        if (Ew + r >= bestw) {

            EnQueue(q, Ew, i, E, bestE, 0);

        }

        E = q.front();

        q.pop();

        if (!E) {

            if (q.empty())

                break;

            q.push(0);

            E = q.front();

            q.pop();

            i++;

            r -= w[i];

        }

        //更新最新节点的值

        Ew = E->weight;

    }

    for (int j = n - 1; j > 0; --j) {

        bestx[j] = bestE->lchild;

        bestE = bestE->parent;

    }

}

void OutPut() {

    printf("最优装载量为 %d\n", bestw);

    printf("装载的物品为 \n");

    for (int i = 1; i <= n; ++i)

        if (bestx[i] == 1)

            printf("%d ", i);

}

int main() {

    InPut();

    MaxLoading();

    OutPut();

    return 0;

}

运行截图：



1. **0-1背包问题（via ）**

源代码：

#include<iostream>

#include<vector>

#include<queue>

using namespace std;

struct Res {

    int weight, value;

    bool used = false;

};

struct Bags {

    vector< Res\*> bags;

    int nweigt, nvalue = 0, ub;

    Bags operator+(Res\*& res) {

        Bags bag = \*this;

        bag.bags.push\_back(res);

        bag.nweigt -= res->weight;  bag.nvalue += res->value;

        return bag;

    }

};

int main() {

    int W;

    cout << "请输入背包容量：";

    cin >> W;

    cout << "请输入物品数量：";

    int n;

    cin >> n;

    cout << "请输入物品的重量与价值（格式：w v）：" << endl;

    vector<Res> thing;// = { {4, 40}, { 7,42 }, { 5,25 }, { 3,12 } };

    int temp1,temp2;

    for (int i = 0; i < n; i++) {

        cin >> temp1 >> temp2;

        thing.push\_back(Res());

        thing[i].weight = temp1;

        thing[i].value = temp2;

    }

    vector<Res\*> res;

    vector<Bags> result;

    for (auto& it : thing)

        res.push\_back(&it);

    //优先排序

    sort(res.begin(), res.end(), [](Res\*& a, Res\*& b) {return a->value / a->weight > b->value / b->weight; });

    Bags start;

    int up = W \* res.front()->value / res.front()->weight, down = 0;

    //贪心获取下界

    for (int i = 0, w = W; i < res.size(); i++) {

        if (res[i]->weight < w) {

            down += res[i]->value;

            w -= res[i]->weight;

        }

    }

    start.ub = up;  start.nweigt = W;

    int size, count = 0;

    queue<Bags> q;

    q.push(start);

    while (!q.empty() && count < res.size()) {

        //记录每层的节点数，进行层处理

        size = q.size();

        for (int i = 0; i < size; i++) {

            Bags right = q.front();

            if (right.nweigt - res[count]->weight >= 0) {

                //左节点

                Bags left = right + res[count];

                //无法访问最后一层的count+1，值为0

                left.ub = left.nvalue + (count + 1 < res.size() ? left.nweigt \* (res[count + 1]->value / res[count + 1]->weight) : 0);

                q.push(left);

            }

            //修改右结点

            right.ub = right.nvalue + (count + 1 < res.size() ? right.nweigt \* (res[count + 1]->value / res[count + 1]->weight) : 0);

            if (right.ub > down)

                q.push(right);

            else

                result.push\_back(right);

            q.pop();

        }

        count++;

    }

    Bags\* minBag = nullptr;

    //在表中寻找最优方案

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

        if (minBag == nullptr || result[i].nvalue > minBag->nvalue)

            minBag = &result[i];

    }

    for (auto& it : minBag->bags)

        it->used = true;

    cout << "最大价值为：" << minBag->nvalue << endl;

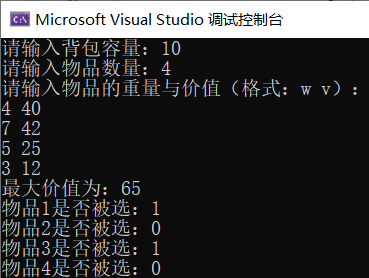
    for (int i = 0; i < res.size(); i++)

        cout << "物品" << i + 1 << "是否被选：" << thing[i].used << endl;

    return 0;

}

运行截图：



1. **TSP问题（via ）**

源代码：

#include<iostream>

#include<algorithm>

#include<cstdio>

#include<queue>

#include<limits>

const int INF = INT\_MAX;

const int MAX\_N = 22;

using namespace std;

//n\*n的一个矩阵

int n;

//最少3个点，最多MAX\_N个点

int cost[MAX\_N][MAX\_N];

struct Node {

    //标记哪些点走了

    bool visited[MAX\_N];

    //第一个点

    int s;

    //第一个点的邻接点

    int s\_p;

    //最后一个点

    int e;

    //最后一个点的邻接点

    int e\_p;

    //走过的点数

    int k;

    //经过路径的距离

    int sumv;

    //目标函数的值（目标结果）

    int lb;

    bool operator <(const Node& p)const {

        //目标函数值小的先出队列

        return p.lb < lb;

    }

};

//创建一个优先队列

priority\_queue<Node> pq;

//下界和上界

int low, up;

//在dfs过程中搜索过

bool dfs\_visited[MAX\_N];

//确定上界,利用dfs（属于贪心算法），贪心法的结果是一个大于实际值的估测结果

//当前节点，目标节点，已经消耗的路径

int dfs(int u, int k, int l) {

    //如果已经检查了n个节点，则直接返回路径消耗+第n个节点回归起点的消耗

    if (k == n) return l + cost[u][1];

    int minlen = INF, p;

    for (int i = 1; i <= n; i++) {

        //取与所有点的连边中最小的边

        if (!dfs\_visited[i] && minlen > cost[u][i]) {

            //找出对于每一个节点，其可达节点中最近的节点

            minlen = cost[u][i];

            p = i;

        }

    }

    //以p为下一个节点继续搜索

    dfs\_visited[p] = true;

    return dfs(p, k + 1, l + minlen);

}

void get\_up() {

    //以第一个点作为起点

    dfs\_visited[1] = true;

    up = dfs(1, 1, 0);

}

//用这种简单粗暴的方法获取必定小于结果的一个值

void get\_low() {

    //取每行最小值之和作为下界

    low = 0;

    for (int i = 1; i <= n; i++) {

        //创建一个等同于map的临时数组，可用memcpy

        int tmpA[MAX\_N];

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

            tmpA[j] = cost[i][j];

        }

        //对临时的数组进行排序

        sort(tmpA + 1, tmpA + 1 + n);

        low += tmpA[1];

    }

}

int get\_lb(Node p) {

    //路径上的点的距离的二倍

    int ret = p.sumv \* 2;

    //起点和终点连出来的边

    int min1 = INF, min2 = INF;

    for (int i = 1; i <= n; i++)

        if (!p.visited[i] && min1 > cost[i][p.s])

            min1 = cost[i][p.s];

    ret += min1;

    for (int i = 1; i <= n; i++)

        if (!p.visited[i] && min2 > cost[p.e][i])

            min2 = cost[p.e][i];

    ret += min2;

    for (int i = 1; i <= n; i++)

        if (!p.visited[i]) {

            min1 = min2 = INF;

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

                if (min1 > cost[i][j])

                    min1 = cost[i][j];

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

                if (min2 > cost[j][i])

                    min2 = cost[j][i];

            ret += min1 + min2;

        }

    return (ret + 1) / 2;

}

int solve() {

    //贪心法确定上界

    get\_up();

    //取每行最小的边之和作为下界

    //cout << up << endl;//test

    get\_low();

    //设置初始点,默认从1开始

    Node star;

    star.s = 1;//起点为1

    star.e = 1;//终点为1

    star.k = 1;//走过了1个点

    for (int i = 1; i <= n; i++) {

        star.visited[i] = false;

    }

    star.visited[1] = true;

    //经过的路径距离初始化

    star.sumv = 0;

    //让目标值先等于下界

    star.lb = low;

    //ret为问题的解

    int ret = INF;

    //将起点加入队列

    pq.push(star);

    while (pq.size()) {

        Node tmp = pq.top(); pq.pop();

        //如果已经走过了n-1个点

        if (tmp.k == n - 1) {

            //找最后一个没有走的点

            int p;

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

                if (!tmp.visited[i]) {

                    //让没有走的那个点为最后点能走的点

                    p = i;

                    break;

                }

            //已消耗+回到开始消耗+走到P的消耗

            int ans = tmp.sumv + cost[p][tmp.s] + cost[tmp.e][p];

            //如果当前的路径和比所有的目标函数值都小则跳出

            if (ans <= tmp.lb) {

                ret = min(ans, ret);

                break;

            }

            //否则继续求其他可能的路径和，并更新上界

            else {

                //上界更新为更接近目标的ans值

                up = min(up, ans);

                ret = min(ret, ans);

                continue;

            }

        }

        //当前点可以向下扩展的点入优先级队列

        Node next;

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

            if (!tmp.visited[i]) {

                //沿着tmp走到next，起点不变

                next.s = tmp.s;

                //更新路径和

                next.sumv = tmp.sumv + cost[tmp.e][i];

                //更新最后一个点

                next.e = i;

                //更新走过的顶点数

                next.k = tmp.k + 1;

                //tmp经过的点也是next经过的点

                for (int j = 1; j <= n; j++) next.visited[j] = tmp.visited[j];

                //自然也要更新当前点

                next.visited[i] = true;

                //求目标函数

                next.lb = get\_lb(next);

                //如果大于上界就不加入队列

                if (next.lb > up) continue;

                //否则加入队列

                pq.push(next);

            }

        }

    }

    return ret;

}

int main() {

    cout << "请输入城市数量：";

    cin >> n;

    cout << "请输入城市的邻接矩阵（-1表示无法到达）：" << endl;

    for (int i = 1; i <= n; i++)

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

            cin >> cost[i][j];

            if (i == j || cost[i][j] == -1)

                cost[i][j] = INF;

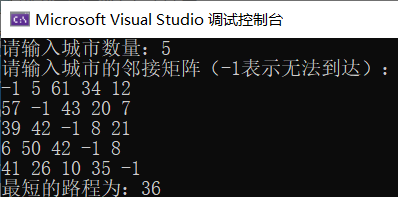
        }

    cout << "最短的路程为：" << solve();

    return 0;

}

运行截图：



作业

1. **Flowshop问题（via ）**

源代码：

#include <iostream>

#include<queue>

using namespace std;

const int MAX = 100;

const int MACHINE = 2;

int n;

int M[MAX][MACHINE];

int b[MAX][MACHINE];

int a[MAX][MACHINE];

int y[MAX][MACHINE];

int bestx[MAX];

int bestc;

struct Node{

    int s;

    int f1;

    int f2;

    int sf2;

    int bb;

    int\* x;

    bool operator < (const Node& node) const{

        return bb > node.bb;

    }

};

priority\_queue<Node> pq;

void initNode(Node& node, int n){

    node.x = new int[n];

    int i;

    for (i = 0; i < n; i++)

        node.x[i] = i;

    node.s = 0;

    node.f1 = 0;

    node.f2 = 0;

    node.sf2 = 0;

    node.bb = 0;

}

void newNode(Node& node, Node E, int Ef1, int Ef2, int Ebb, int n){

    node.x = new int[n];

    int i;

    for (i = 0; i < n; i++)

        node.x[i] = E.x[i];

    node.f1 = Ef1;

    node.f2 = Ef2;

    node.sf2 = E.sf2 + Ef2;

    node.bb = Ebb;

    node.s = E.s + 1;

}

void sort(){

    int\* c = new int[n];

    int i, j, k;

    for (j = 0; j < 2; j++){

        for (i = 0; i < n; i++){

            b[i][j] = M[i][j];

            c[i] = i;

        }

        for (i = 0; i < n - 1; i++)

            for (k = n - 1; k > i; k--)

                if (b[k][j] < b[k - 1][j]){

                    swap(b[k][j], b[k - 1][j]);

                    swap(c[k], c[k - 1]);

                }

        for (i = 0; i < n; i++)

            a[c[i]][j] = i;

    }

    delete[]c;

}

int bound(Node E, int& f1, int& f2, int y[MAX][MACHINE]){

    int k, j;

    for (k = 0; k < n; k++)

        for (j = 0; j < 2; j++)

            y[k][j] = 0;

    for (k = 0; k <= E.s; k++)

        for (j = 0; j < 2; j++)

            y[a[E.x[k]][j]][j] = 1;

    f1 = E.f1 + M[E.x[E.s]][0];

    f2 = ((f1 > E.f2) ? f1 : E.f2) + M[E.x[E.s]][1];

    int sf2 = E.sf2 + f2;

    int s1 = 0, s2 = 0;

    int k1 = n - E.s, k2 = n - E.s;

    int f3 = f2;

    for (j = 0; j < n; j++)

        if (!y[j][0]){

            k1--;

            if (k1 == n - E.s - 1)

                f3 = (f2 > f1 + b[j][0]) ? f2 : f1 + b[j][0];

            s1 += f1 + k1 \* b[j][0];

        }

    for (j = 0; j < n; j++)

        if (!y[j][1]){

            k2--;

            s1 += b[j][1];

            s2 += f3 + k2 \* b[j][1];

        }

    return sf2 + (s1 > s2 ? s1 : s2);

}

int flowShop(){

    sort();

    Node E;

    initNode(E, n);

    bestc = 1e6;

    while (E.s <= n){

        if (E.s == n){

            if (E.sf2 < bestc){

                bestc = E.sf2;

                int i;

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

                    bestx[i] = E.x[i];

                delete[]E.x;

            }

        }

        else{

            int i;

            for (i = E.s; i < n; i++){

                swap(E.x[E.s], E.x[i]);

                int f1, f2;

                int bb = bound(E, f1, f2, y);

                if (bb < bestc){

                    Node node;

                    newNode(node, E, f1, f2, bb, n);

                    pq.push(node);

                }

                swap(E.x[E.s], E.x[i]);

            }

            delete[]E.x;

        }

        if (pq.empty())

            break;

        E = pq.top();

        pq.pop();

    }

    return bestc;

}

void init(int n1, int M1[3][2]){

    n = n1;

    int i, j;

    for (i = 0; i < n; i++)

        for (j = 0; j < 2; j++)

            M[i][j] = M1[i][j];

}

int main(){

    int n1;

    int M1[MAX][MACHINE];

    cout << "请输入作业数量：";

    cin >> n1;

    cout << "请依次输入每个作业在两台机器上的作业时间（格式：t1 t2）：" << endl;

    for (int i = 0; i < n1; i++)

        for (int j = 0; j < 2; j++)

            cin >> M1[i][j];

    init(n1, M1);

    int best = flowShop();

    printf("最少完成时间和：%d\n", best);

    printf("最优调度：\n");

    for (int i = 0; i < n; i++)

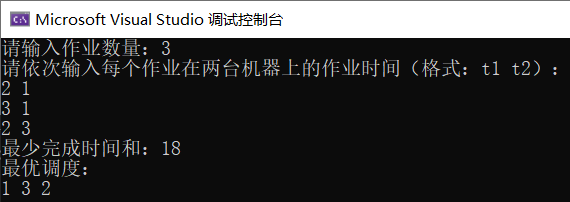
        printf("%d ", bestx[i] + 1);

    printf("\n");

    return 0;

}

运行截图：



# 实验七

实验目标

理解概率算法的思想，原理，掌握概率算法模拟PI，定积分的算法设计及程序。

实验

1. **设计一个概率算法及程序，模拟π的值（via ）**

源代码：

#include<iostream>

#include<cstdlib>

#include<ctime>

using namespace std;

double calPi(long long k) {

    double x;

    double y;

    double m = 0;

    int seed = time(0);

    srand(seed);

    for (long long i = 0; i <= k; i++) {

        x = rand() % 1000;

        y = rand() % 1000;

        if ((x \* x + y \* y) <= 1000000)

            m++;

    }

    return 4 \* m / k;

}

int main(void) {

    cout << "请输入循环次数：";

    long long k;

    cin >> k;

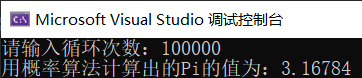
    auto myPi = calPi(k);

    cout << "用概率算法计算出的Pi的值为：" << myPi;

    return 0;

}

运行截图：



1. **设计蒙特卡罗算法计算定积分（via ）**

源代码：

#include<iostream>

#include<cmath>

#include<ctime>

using namespace std;

double darts(long n) {

    long count = 0;

    time\_t t;

    srand((unsigned)time(&t));

    for (long i = 1; i <= n; i++) {

        double x = rand() / (RAND\_MAX + 1.0);

        double y = rand() / (RAND\_MAX + 1.0);

        if (y <= x \* cos(x))

            count++;

    }

    return (double)count / n;

}

int main() {

    cout << "请输入投点数量：";

    long point;

    cin >> point;

    cout << "随机投点法积分值" << darts(point);

    return 0;

}

运行截图：



1. **用概率算法设计n皇后问题程序（via ）**

源代码：

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define N 20

int q[N];

int num = 0;

//输出找到的解

void dispasolution(int n) {

    printf("第%d次尝试找到一个解：", num + 1);

    for (int i = 1; i <= n; i++) {

        printf("(%d,%d)", i, q[i]);

    }

    printf("\n");

}

int randa(int a, int b) {

    //产生一个随机数，用于表示随机放置的位置

    return rand() % (b - a + 1) + a;

}

bool place(int i, int j) {

    //第一个皇后可以随意放置

    if (i == 1) return true;

    int k = 1;

    while (k < i) {

        if ((q[k] == j) || (abs(q[k] - j) == abs(i - k)))

            return false;

        k++;

    }

    return true;

}

bool queen(int i, int n) {

    int count, j;

    //当最后一个皇后放置好以后，

    if (i > n) {

        dispasolution(n);

        return true;

    }

    else {

        count = 0;

        while (count <= n) {

            //随机产生一个数用于表示第i个皇后试探的位置

            j = randa(1, n);

            count++;

            //判断随机放置的位置是否满足条件

            if (place(i, j))

                break;

        }

        /\*规定当产生n个随机数还是不能找到合适位置，

        就认为没有 （不考虑这行每个位置是否都找过）\*/

        if (count > n) {

            return false;

        }

        q[i] = j;

        //递归  放置下一个皇后

        queen(i + 1, n);

    }

}

int main(void) {

    int n;

    printf("输入皇后数目（<20）:");

    scanf\_s("%d", &n);

    printf("%d皇后问题求解如下:\n", n);

    srand((unsigned)time(NULL));

    while (num < 10) {

        if (queen(1, n))

            break;

        num++;

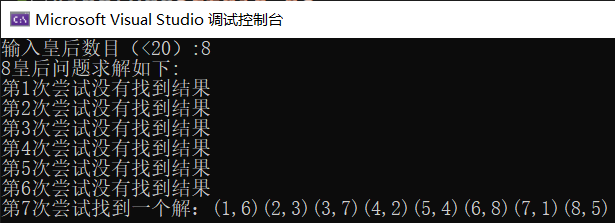
        printf("第%d次尝试没有找到结果\n", num);

    }

    return 0;

}

运行截图：



1. **设计程序，检验某数是否为素数（via ）**

源代码：

#include<iostream>

#include<stdlib.h>

using namespace std;

typedef long long ll;

//快速积取模,求（a\*b\*c\*..........）% c1

ll mm(ll a, ll b, ll mod) {

    ll ans = 0;

    while (b) {

        if (b & 1) ans = (ans + a) % mod;

        a = (a + a) % mod;

        b >>= 1;

    }

    return ans;

}

//快速幂取模,用来计算(a^b)%c

ll f(ll x, ll n, ll mod) {

    ll ans = 1;

    while (n) {

        if (n & 1) ans = mm(ans, x, mod);

        x = mm(x, x, mod);

        n >>= 1;

    }

    return ans;

}

//Miller-Rabin素数检测算法

bool miller\_rabin(ll n) {

    if (n == 2)

        return true;

    else if (n == 1 || n % 2 == 0)

        return false;

    ll u = n - 1, t = 0;

    //n-1=u\*2^t

    while (u % 2 == 0) {

        u /= 2;

        t++;

    }

    for (int i = 0; i < 10; i++) {

        //随机选取一个底数a

        ll a = rand() % (n - 1) + 1;

        //计算(a^(n-1))%n=(a^(u\*2^t))%n

        //先计算(a^u)%n

        ll x = f(a, u, n);

        //再经过t次循环计算得到 ( (a^u)^(2^t) )%n

        for (int j = 1; j <= t; j++) {

            ll y = mm(x, x, n);

            //如果不满足二次探测定理，则不是素数

            if (y == 1 && x != 1 && x != n - 1)

                return false;

            x = y;

        }

        //如果不满足费马小定理，则不是素数

        if (x != 1) return false;

    }

    //是素数

    return true;

}

int main() {

    int t;

    ll n;

    cout << "请输入素数搜索范围：";

    cin >> t;

    for (int i = 3; i <= t; i++)

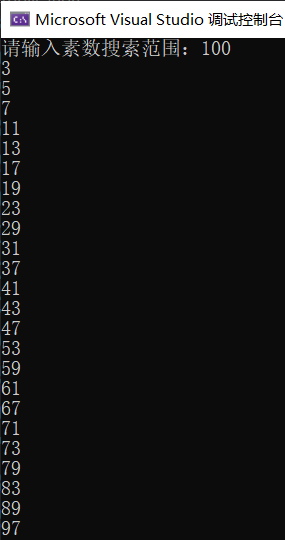
        if (miller\_rabin(i))

            cout << i << endl;

    return 0;

}

运行截图：



作业

1. **牛顿法求解方程（小数点后保留两位）（via ）**

源代码：

#include<iostream>

#include <iomanip>

#include<cmath>

using namespace std;

double diedai(double x) {

    while (abs(pow(x, 4) - 3 \* pow(x, 3) + 1.5 \* pow(x, 2) - 4) > 0.000001) {

        x = x - (pow(x, 4) - 3 \* pow(x, 3) + 1.5 \* pow(x, 2) - 4) / (4 \* pow(x, 3) - 9 \* pow(x, 2) + 3 \* x);

    }

    return x;

}

int main() {

    double x = 10000.0;

    x = diedai(x);

    cout <<"方程x^4-3x^3+1.5x^2-4=0的解为："<< setprecision(2)<< x;

    return 0;

}

运行截图：



1. **利用二分法求解方程：（小数点后保留两位）（via ）**

源代码：

#include <stdio.h>

#include <math.h>

//允许的最大误差，float最多能表示小数点后6位，注意不要超过表示范围。

const float DEVIATION = 1e-5;

//解所在的可能区间

float x\_left = -3.00;

float x\_right = 6.00;

/\*

\*\* 函数表达式，返回函数值

\*/

float my\_function(float x\_value) {

    return pow(x\_value, 4) + 3 \* pow(x\_value, 3) + 1.5 \* pow(x\_value, 2) - 4;

}

/\*

\*\* 判断两个数是否同号

\*\* 同号返回1，异号返回0

\*/

int is\_same\_symbol(float x, float y) {

    return ((x > 0 && y > 0) || (x < 0 && y < 0)) ? 1 : 0;

}

int main(void) {

    float x\_middle = (x\_left + x\_right) / 2;

    static int count\_num = 0;

    while (fabs(my\_function(x\_middle)) > DEVIATION) {

        printf("x%d = %f...\n", count\_num, x\_middle);

        count\_num++;

        if (is\_same\_symbol(my\_function(x\_middle), my\_function(x\_right)) == 1) {

            //迭代下一次判断的区间范围

            x\_right = x\_middle;

            x\_middle = (x\_middle + x\_left) / 2;

            continue;

        }

        else {

            x\_left = x\_middle;

            x\_middle = (x\_middle + x\_right) / 2;

            continue;

        }

    }

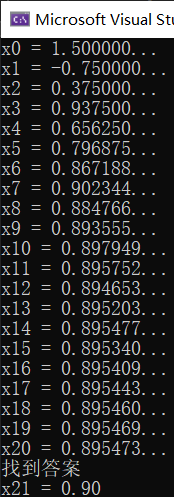
    printf("找到答案\n");

    printf("x%d = %.2f", count\_num, x\_middle);

    return 0;

}

运行截图：



# 实验八

作业

**找出任意数码字符串中，每个数码前面比它小的数码个数组成新的数组**

源代码：

#include<stdio.h>

int main(void) {

    int i, j, n, arr2[100] = { 0 };

    char arr[100] = { '\n' };

    printf("请输入一个数组：");

    gets\_s(arr);

    printf("请输入数组长度：");

    scanf\_s("%d", &n);

    for (i = 0; i < n; i++)

        for (j = 0; j < i; j++)

            if (arr[i] > arr[j])

                arr2[i] ++;

    printf("原数组为");

    for (i = 0; i < n; i++) {

        printf("%c", arr[i]);

    }

    printf("\n新数组为：");

    for (i = 0; i < n; i++) {

        printf("%d", arr2[i]);

    }

    return 0;

}

运行截图：

