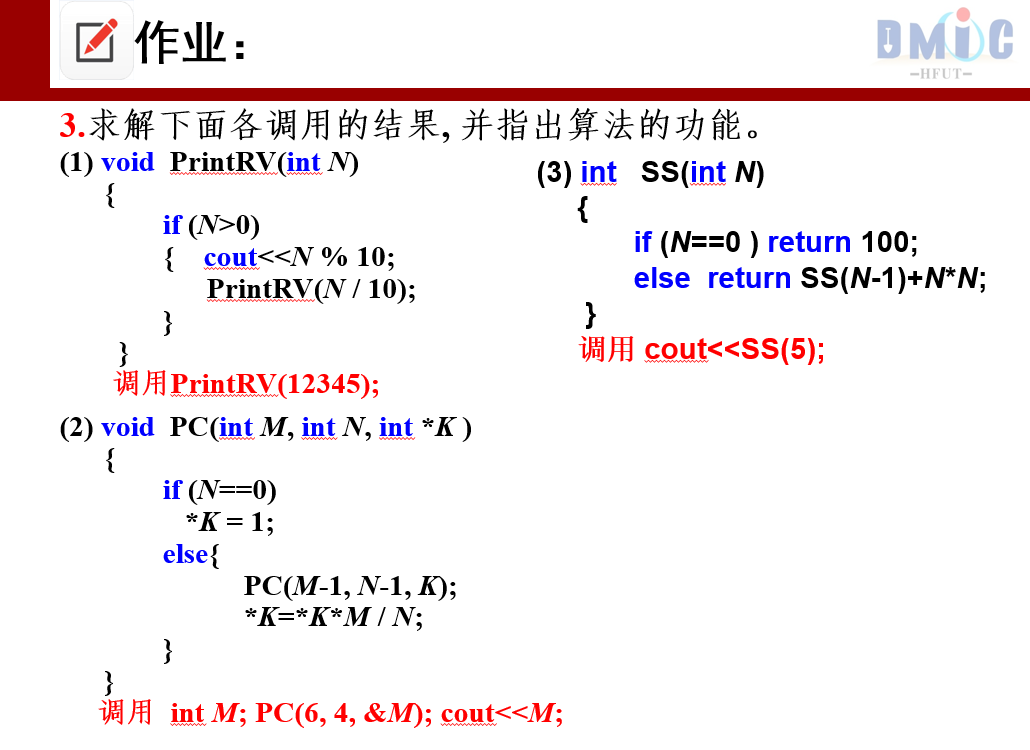
**2022212153 陈祥烨 第七章作业**

**第三题**

**题目要求：**

****

（1）

调用结果：54321

算法功能：倒序输出正整数

（2）

调用结果：15

算法功能：M\*(M-1)\*…\*(M-(M-N-1)) / (1\*2\*…\*N)

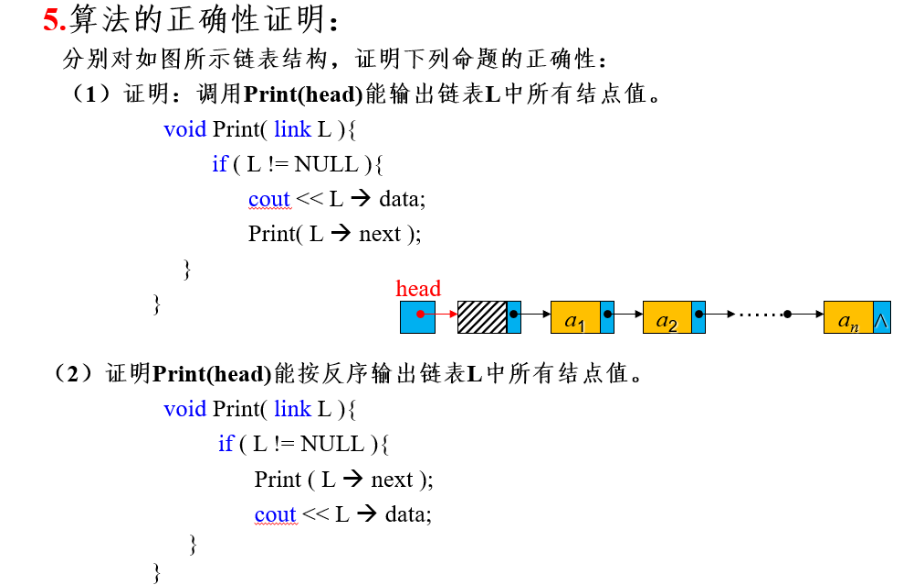
（3）

调用结果：155

算法功能：N\*N+(N-1)\*(N-1)+…+2\*2+1\*1+100

**第五题**

**题目要求：**



（1）

①当L = NULL 时，不再输出，结束程序，符合题意；

②当L != NULL 时，输出该点的数据（data），并指向下一个结点，再次进入程序，循环往复；有多少个结点，输出多少个数据；

综上所述，算法正确。

（2）

①当L = NULL 时，不再寻找尾结点（即尾结点已经找到），结束程序，符合题意；

②当L != NULL 时，找后面的节点，下一个结点的数据输出结束后，再输出本结点，如此循环，实现倒序输出；有多少个结点输出多少个数据；

综上所述，算法正确。

#include<iostream>

using namespace std;

struct Link

{

int data;

Link\* next;

};

void Print01(Link\* L)

{

if (L != NULL)

{

cout << L->data;

Print01(L->next);

}

}

void Print02(Link\* L)

{

if (L != NULL)

{

Print02(L->next);

cout << L->data;

}

}

int main()

{

Link\* head = new Link;

head->next = NULL;

Link\* temp = head;

for (int i = 1; i <= 5; i++)

{

Link\* node = new Link;

node->data = i;

node->next = NULL;

temp->next = node;

temp = node;

}

Print01(head);

cout << endl;

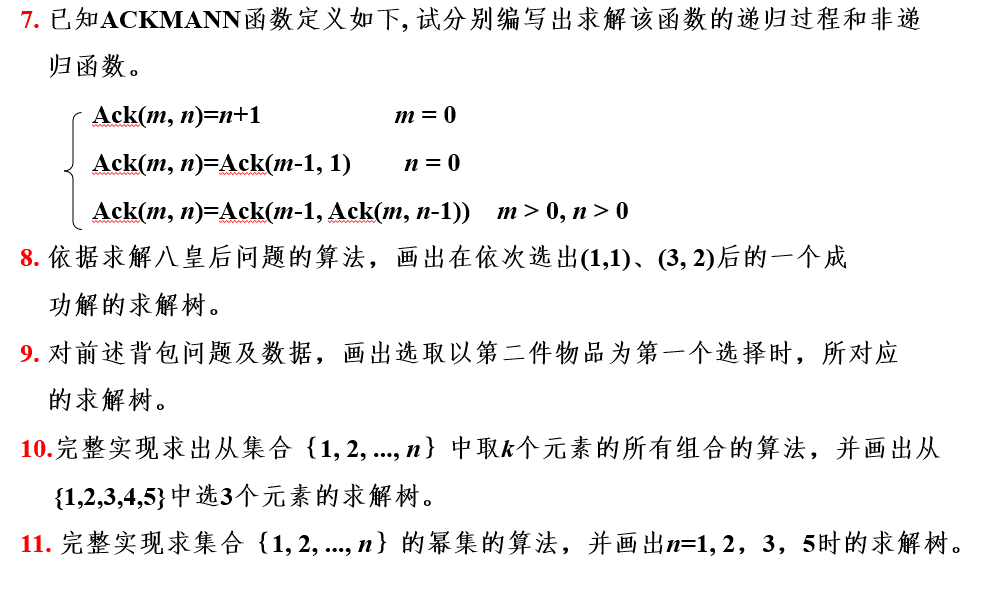
Print02(head);

}



**第七题**

**题目要求：**



**（1）递归函数：**

#include<iostream>

using namespace std;

int Ack(int m, int n)

{

if (m == 0)

return n + 1;

else if (n == 0)

return Ack(m - 1, 1);

else

return Ack(m - 1, Ack(m, n - 1));

}

void test()

{

int m, n;

while (1)

{

cout << "请输入 m 和 n：";

cin >> m >> n;

if (m!= 0||n!=0)

cout << "Ack( " << m << ", " << n << " ) = " << Ack(m, n) << endl;

else

{

cout << "Ack( " << m << ", " << n << " ) = " << Ack(m, n) << endl;

break;

}

}

}

int main()

{

test();

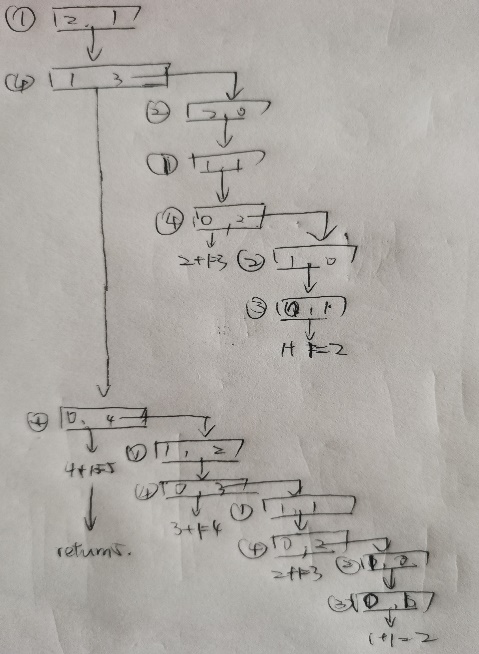
return 0;

}



PS: m = 4，n = 2时，程序就崩掉了…

**（2）非递归函数：**

**思路：**

三个条件m = 0，n = 0，m > 0且n > 0设为③，②，①。

①中程序的运行相较于其他两个条件的程序特殊,①中程序需要递归，需要存储当前值m，再进行后续操作，并满足后进先出的顺序。根据以上条件，选择链栈来存储m。

将①的程序设为④，如右图，④时入栈m-1，n--，运用goto语句达到递归效果。②n = 1；m--；③到达结束条件，取出上个入栈的m并更新当前m，且n++。

整个程序结束条件为栈空且m==0

**源码：**

Ack非递归.cpp

#include<iostream>

#include"Stack.h"

using namespace std;

int Ack2(int m, int n)

{

Stack stack;

do

{

L1://递归入口

if (m > 0 && n > 0)

{

stack.push(m-1);

n--;

goto L1;

}

if (n == 0 && m > 0)

{

n = 1;

m--;

goto L1;

}

if (m == 0)

{

m = stack.get\_top()->\_m;

n++;

stack.pop();

goto L1;

}

} while (!stack.Empty() && m == 0);

return n;

//Ack(2, 1) = Ack(1, Ack(2, 0))

// = Ack(1, Ack(1, 1))

// = Ack(1, Ack(0, Ack(1, 0)))

// = Ack(1, Ack(0, Ack(0, 1)))

// = Ack(1, Ack(0, 2))

// = Ack(1, 3)

// = Ack(0, Ack(1, 2))

// = Ack(0, Ack(0, Ack(1, 1)))

// = Ack(0, Ack(0, Ack(0, Ack(1, 0))))

// = Ack(0, Ack(0, Ack(0, Ack(0, 1))))

// = Ack(0, Ack(0, Ack(0, 2)))

// = Ack(0, Ack(0, 3))

// = Ack(0, 4)

// = 5

}

int Ack1(int m, int n)

{

if (m == 0)

return n + 1;

else if (n == 0)

return Ack1(m - 1, 1);

else

return Ack1(m - 1, Ack1(m, n - 1));

}

void test01()

{

int m, n;

while (1)

{

cout << "请输入 m 和 n：";

cin >> m >> n;

if (m != 0 || n != 0)

{

cout << "Ack1( " << m << ", " << n << " ) = " << Ack1(m, n) << "\t";

cout << "Ack2( " << m << ", " << n << " ) = " << Ack2(m, n) << endl;

}

else

{

cout << "Ack1( " << m << ", " << n << " ) = " << Ack1(m, n) << "\t";

cout << "Ack2( " << m << ", " << n << " ) = " << Ack2(m, n) << endl;

break;

}

}

}

int main()

{

test01();

return 0;

}

Stack.h

#pragma once

#include<iostream>

using namespace std;

struct Node

{

int \_m;

Node\* \_next;

};

class Stack

{

public:

Stack();

bool Empty() const; //判断栈空

Node\* get\_top() const; //得到栈顶指针

void push(const int m); //入栈

void pop(); //出栈

private:

Node\* \_top;

int \_count;

};

Stack::Stack()

{

Node\* top = new Node;

top->\_next = NULL;

\_top = top;

\_count = 0;

}

Node\* Stack::get\_top() const //得到栈顶指针

{

return \_top;

}

bool Stack::Empty() const //判断栈空

{

return \_count == 0;

}

void Stack::push(const int m) //入栈

{

Node\* node = new Node;

node->\_m = m;

node->\_next = \_top;

\_top = node;

\_count++;

}

void Stack::pop() //出栈

{

if (\_top->\_next = NULL)return;

Node\* temp = \_top;

\_top = \_top->\_next;

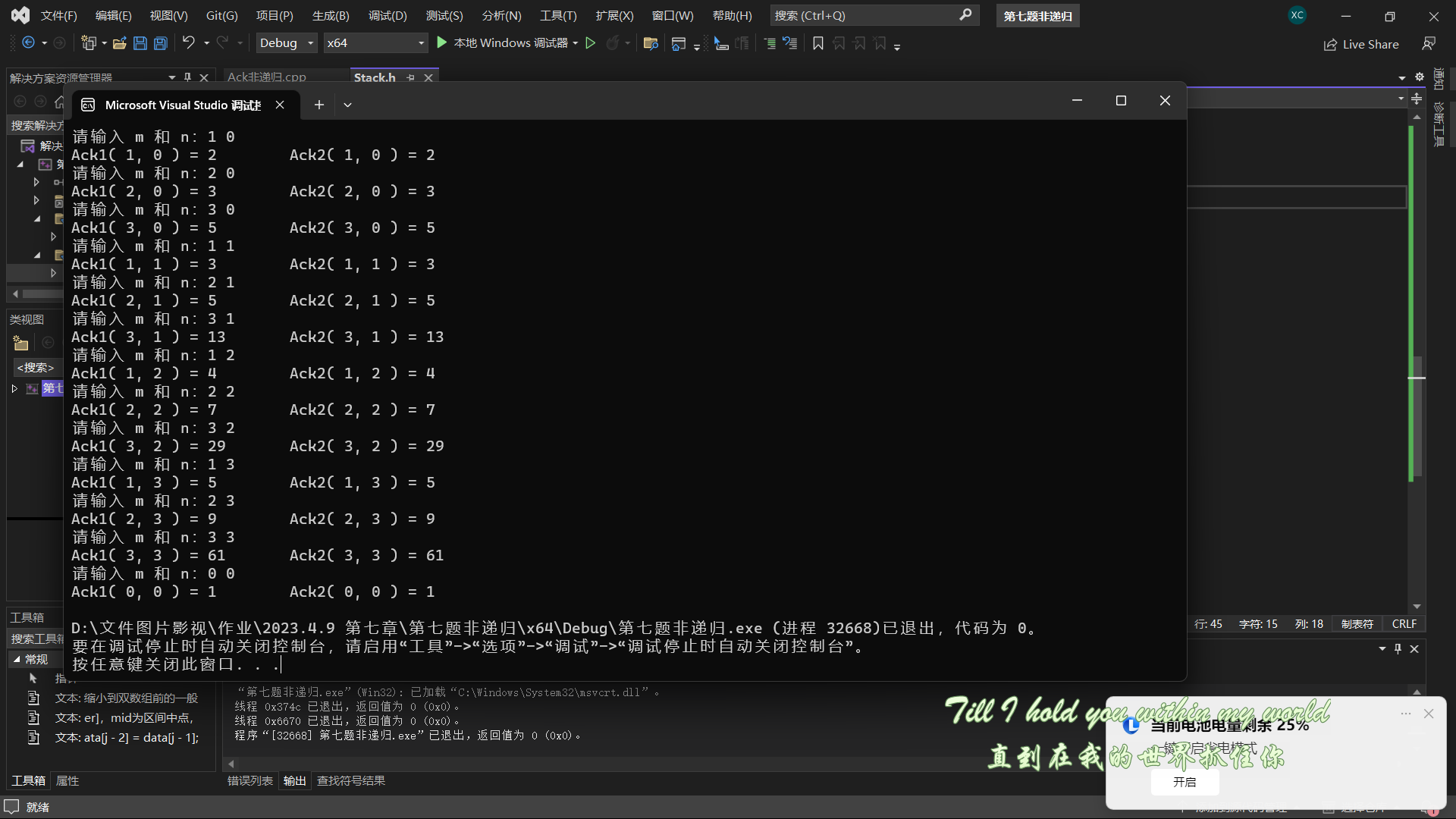
temp->\_next = NULL;

delete temp;

\_count--;

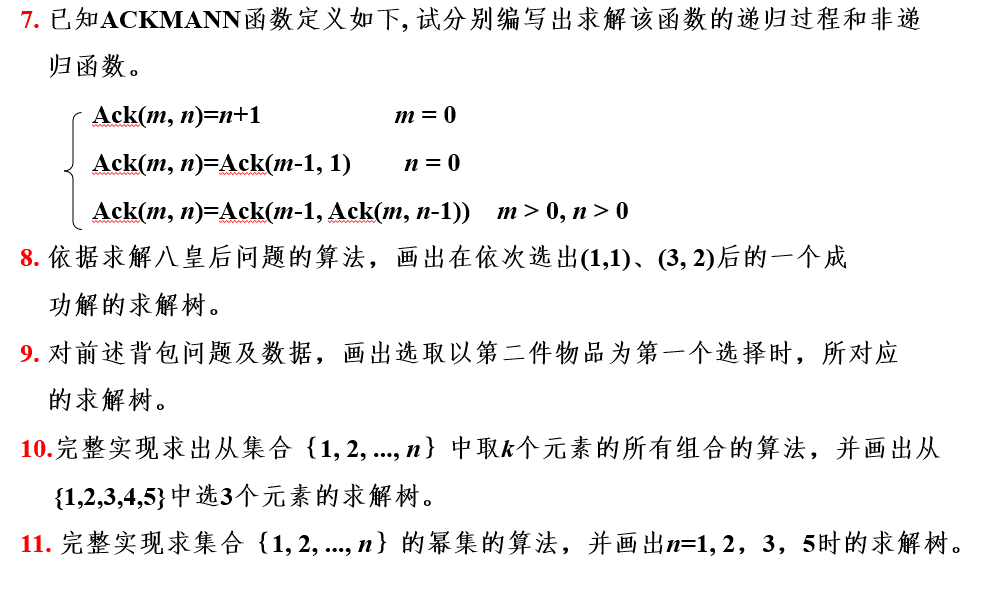
}

测试运行：



**第九题**

**题目要求：**



**思路：**试探法

先将物品从重到轻排列好（假设已经排列好，从大到小排可以减少选取物品的次数），依次选取物品，判断条件如下：

①放置后正好满足条件，输出放置方法

②放置后中总重量 < S，需要继续放置：

如果后面还有物品，依次选取物品，即回到开头进行递归；

如果后面没有物品，则结束当前选取；

③放置后中总重量 > S，说明当前情况该物品不能放置。原因有：

（1）这个物品总量大，后面的可以放下；

（2）或者前面的有的东西大了，需要取出来，再试后面的；

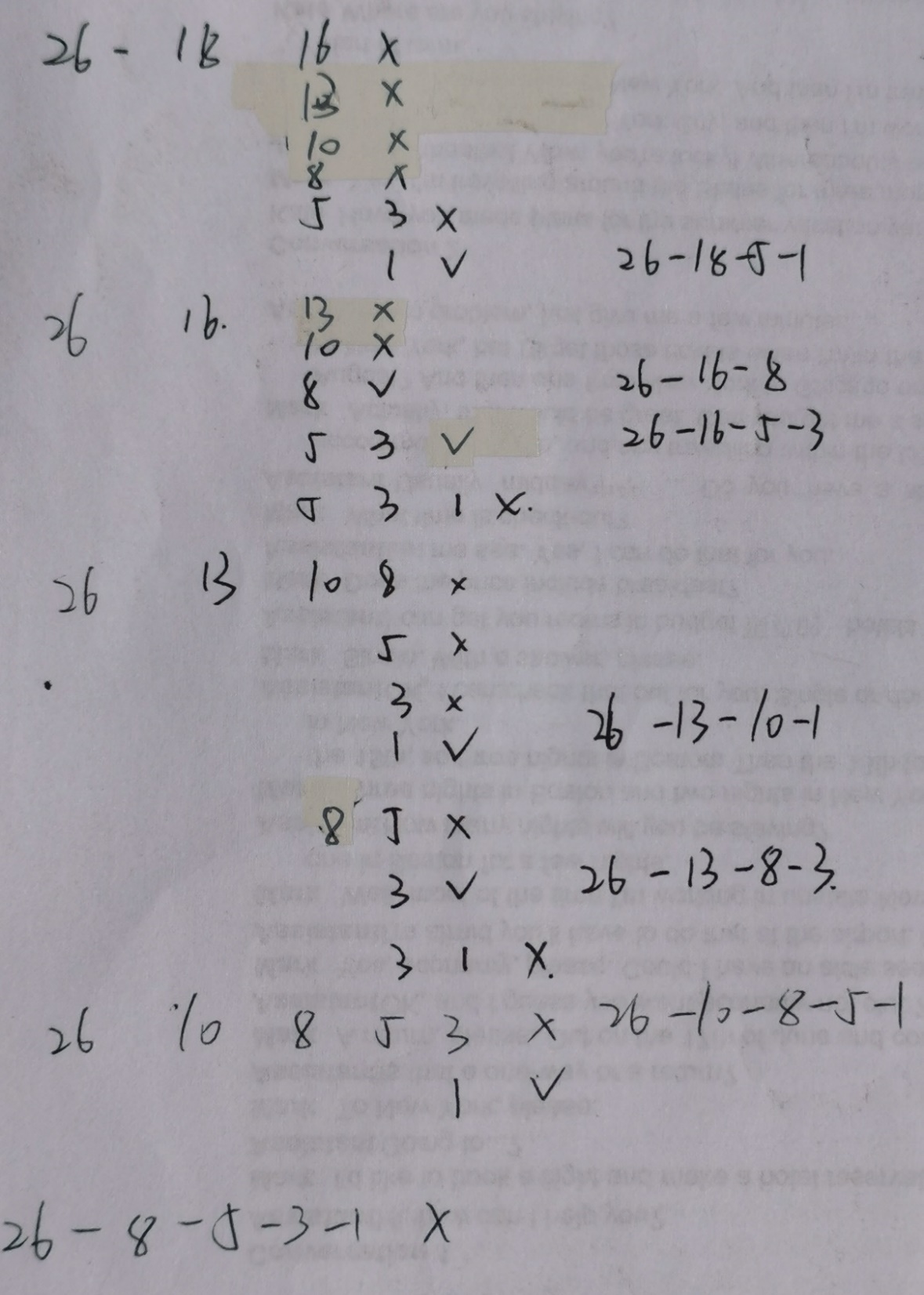
（3）再或者就是再没有比这个更小的了，相当于情况②中的第二个条件。

由（2）知，书包物品的存储结构相较于数组，栈更好用，情况（3）相当于②中第二条情况，就可以代替；情况（1）则需要取出该物品，然后进入情况②中的第一种条件，直到情况（3），此时说明前面有不合理的，相当于情况（2）；

直到从某一个开始到后面所有的取到但都小于题目要求的，结束试探。

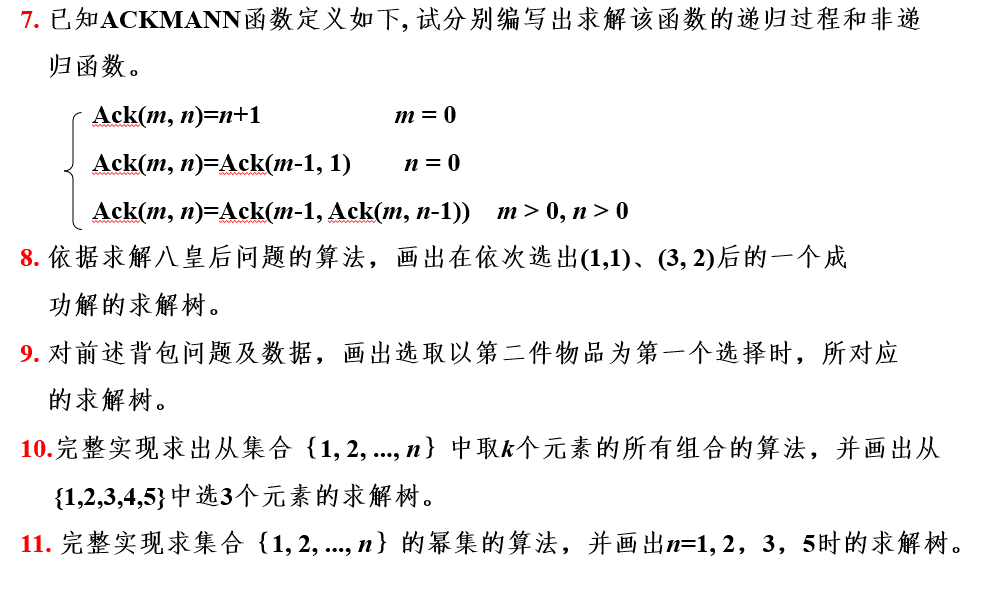
总过程：将背包看成栈，每取出一个，就用其后面的所有物品进行试探。若其后面的所有物品放置都不符合条件，就继续取出背包中的物品再试探。在这样的程序中，每个物品在当前可以放置时，都要试探放与不放两种情况。直到找到所有符合条件的解。

**结果：**



**第十题**

**题目要求：**



**思路：**

getson用于进入每一层

for的结束条件作为递归的结束条件

输出为count-1为0时，即到达最底层输出

**代码：**

#include<iostream>

using namespace std;

const int n = 5;

const int k = 3;

void getson(int\* arr, int start, int\* result, int count)//start 为变量起始位置, count 为 result 的索引值，且为还需要找多少个数

{

for (int i = start; i < n + 1 - count; i++)

{

result[count - 1] = i; //存储值

if (count - 1 == 0) //到达三个时，输出

{

for (int j = k - 1; j >= 0; j--)

cout << arr[result[j]] << "\t";

cout << endl;

}

else

getson(arr, i + 1, result, count - 1);

}

}

int main()

{

cout << "n = " << n << endl;

cout << "k = " << k << endl;

int arr[n]; //原数组

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

{

arr[i] = i + 1;

}

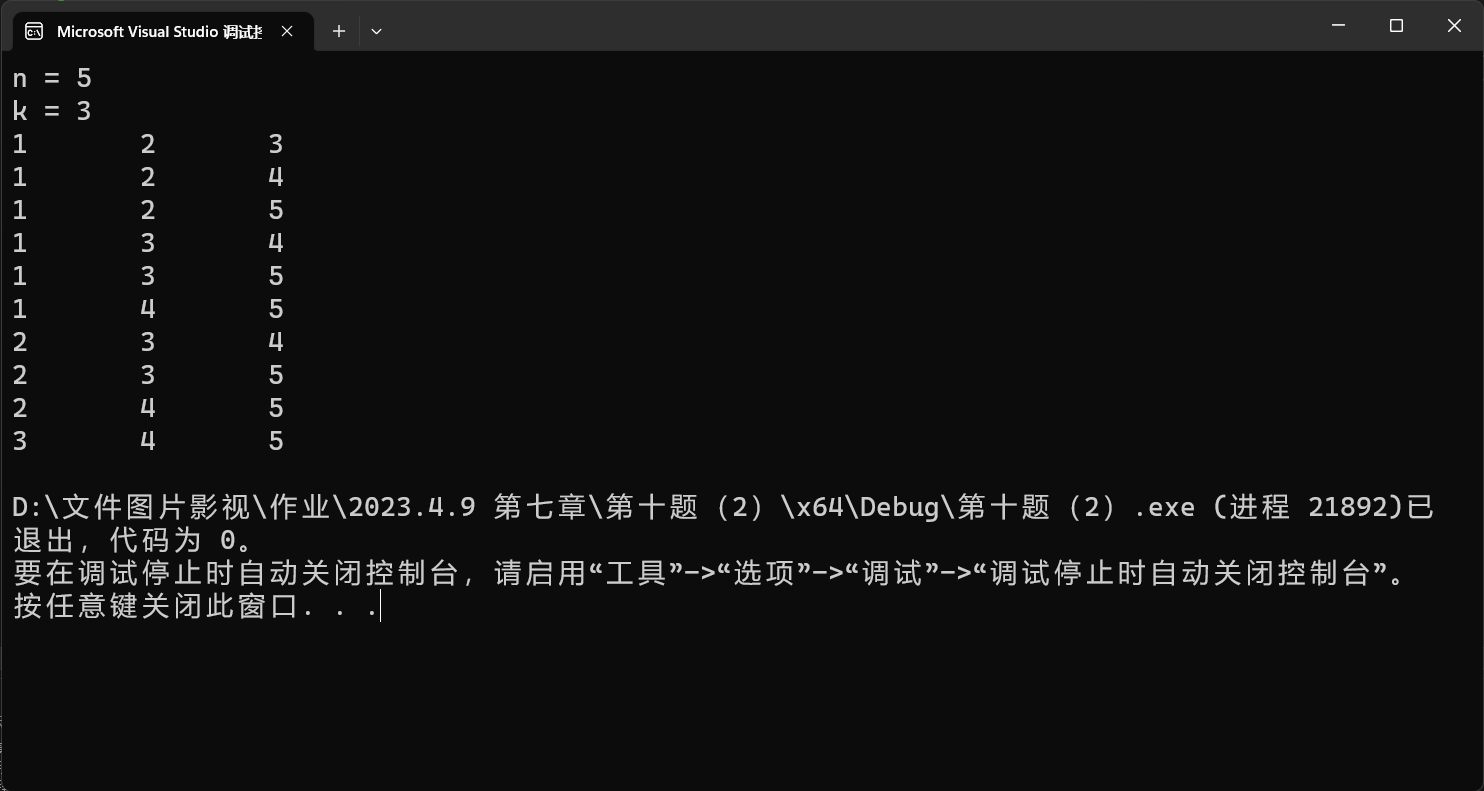
int result[n]; //result保存结果，保存所需要的输出的数在原数组中的位置

getson(arr, 0, result, k);

return 0;

}

**测试结果：**



**求解树：**

