

数据结构实验报告

计算机 46 班

2140504137

何宜晖

实验一 背包问题的求解

（一）问题描述

假设有一个能装入总体积为 T 的背包和 n 件体积分别为 w_1, w_2, \dots, w_n 的物品能否从 n 件物品中挑选若干件恰好装满背包，即使 $w_1 + w_2 + \dots + w_m = T$ ，要求找

出所有满足上述条件的解。例如：当 $T=10$ ，各件物品的体积 $\{1, 8, 4, 3, 5, 2\}$ 时，可找到下列 4 组

解：

$(1, 4, 3, 2)$

$(1, 4, 5)$

$(8, 2)$

$(3, 5, 2)$ 。

（二）实验提示

可利用回溯法的设计思想来解决背包问题。首先，将物品排成一列，然后，顺序选取物品装入背包，若已选取第 i 件物品后未满足，则继续选取

第 $i+1$ 件，若该件物品“太大”不能装入，则弃之，继续选取下一件，直至

背包装满为止。

如果在剩余的物品中找不到合适的物品以填满背包，则说明“刚刚”装入的物品“不合适”，应将它取出“弃之一边”，继续再从“它之后”的物品中

选取，如此重复，直到求得满足条件的解，或者无解。由于回溯求解的规则是“后进先出”，自然要用到“栈”。

（三）代码实现

```
#include <iostream>
using namespace std;
int T;
int n;
int *w, *flag;

void knapsack(int sum, int k) {
```

```

    if (sum == T) {
        int first = true;
        cout << "(";
        for (int i = 0; i < n; i++) {
            if (flag[i]) {
                if (first) {
                    first = false;
                }
                else
                    cout << ",";
                cout << w[i];
            }

        }
        cout << ")"<<endl;
        return;
    }
    if (sum > T || k >= n)
        return;
    flag[k] = 0;
    knapsack(sum, k + 1);
    flag[k] = 1;
    knapsack(sum+w[k], k + 1);
}

int main() {
    cout << "T=";
    cin >> T;
    cout << "n=";
    cin >> n;
    cout << "w?" << endl;
    w = new int[n];
    flag = new int[n];
    for (int i = 0; i < n; i++) {
        cin >> w[i];
        flag[i] = 0;
    }
    knapsack(0, 0);
    return 0;
}

```

(四) 运行结果

```
C:\Users\dell01\Desktop\1.exe
T=10
n=6
w?
1
8
4
3
5
2
(3,5,2)
(8,2)
(1,4,5,2)
(1,4,3,2)

-----
Process exited with return value 0
Press any key to continue . . .

搜狗拼音输入法 全 :
```

(五) 进一步考虑

如果每件物品都有体积和价值，背包又有大小限制，求解背包中存放物品总价值最大的问题解——最优解或近似最优解。

代码实现

```
#include<stdio.h>
int c;
int n;
int weight[100];
int price[100];
```

```

int currentweight=0;
int currentprice=0;
int bestprice=0;
int bestanswer[100];
int bp=0;
int ba[100];
int times=0;
void print();
void backtracking(int i)
{
    times+=1;
    if(i>n)
    {
        print ();
        if(bestprice>bp)
        {
            bp=bestprice;
            for(int j=1;j<=n;j++)
                ba[j]=bestanswer[j];
        }
        return;
    }
    if(currentweight+weight[i]<=c)
    {
        bestanswer[i]=1;
        currentweight+=weight[i];
        bestprice+=price[i];
        backtracking(i+1);
        currentweight-=weight[i];
        bestprice-=price[i];

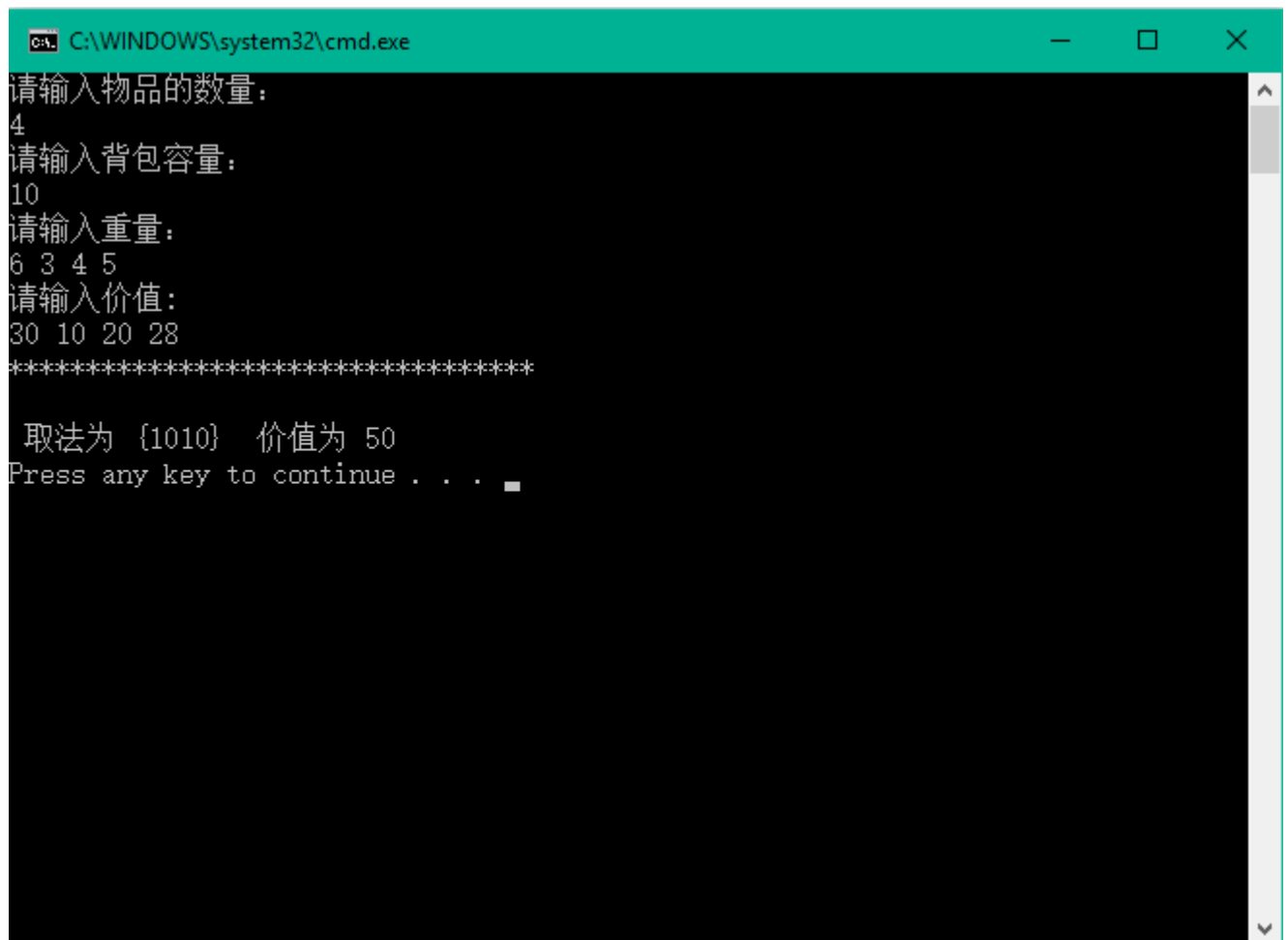
        }bestanswer[i]=0;
        backtracking(i+1);
    }

void main()
{
    int i;
    printf("请输入物品的数量:\n");
    scanf("%d",&n);
    printf("请输入背包容量:\n");
    scanf("%d",&c);
    printf("请输入重量:\n",n);
    for(i=1;i<=n;i++)

```

```
scanf("%d",&weight[i]);
printf("请输入价值:\n",n);
for(i=1;i<=n;i++)
scanf("%d",&price[i]);
printf("个符合条件的取法将为:\n");
backtracking(1);
printf("*****\n");
printf("\n 取法为 {");
for(i=1;i<n;i++)
    printf("%d",ba[i]);
printf("%d}\t 价值为 %d\n",ba[i],bp);
}
```

运行结果



```
C:\WINDOWS\system32\cmd.exe
请输入物品的数量:
4
请输入背包容量:
10
请输入重量:
6 3 4 5
请输入价值:
30 10 20 28
*****

 取法为 {1010}  价值为 50
Press any key to continue . . .
```

实验二 农夫过河问题的求解

1.问题描述

一个农夫带着一只狼、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北岸。他面前只有一条小船，船只能容下他和一件物品，另外只有

农夫才能撑船。如果农夫在场，则狼不能吃羊，羊不能吃白菜，否则狼会吃羊，羊会吃白菜，所以农夫不能留下羊和白菜自己离开，也不能留下狼和羊自

己离开，而狼不吃白菜。请求出农夫将所有的东西运过河的方案。

2.实现提示

求解这个问题的简单方法是一步一步进行试探，每一步搜索所有可能的选择，对前一步合适的选择后再考虑下一步的各种方案。要模拟农夫过河问题，首

先需要对问题中的每个角色的位置进行描述。可用 4 位二进制数顺序分别表示农夫、狼、白菜和羊的位置。用 0 表在南岸，1 表示在北岸。例如，整数 5

(0101) 表示农夫和白菜在南岸，而狼和羊在北岸。现在问题变成：从初始的状态二进制 0000 (全部在河的南岸) 出发，寻找一种全部由安全状态构成的状态

序列，它以二进制 1111 (全部到达河的北岸) 为最终目标。总状态共 16 种 (0000 到 1111)，(或者看成 16 个顶点的有向图) 可采用广度优先或深度优先的搜

索策略---得到从 0000 到 1111 的安全路径。

以广度优先为例：整数队列---逐层存放下一步可能的安全状态；Visited[16] 数组标记该状态是否已访问过，若访问过，则记录前驱状态值---

安全路径。

最终的过河方案应用汉字显示出每一步的两岸状态。

(三) 代码实现

```
#include <iostream>
#include <stack>
#include <queue>
#include <list>
```

```

#include <string>
#include <algorithm>
#include <iomanip>
#include <stdio.h>

using namespace std;

bool vis[16];
int farmer = 1 << 3;
int wolf = 1 << 2;
int sheep = 1 << 1;
int cabbage = 1;
int route[16];

string chinese[4] = { "白菜", "羊", "狼", "农夫" };
int role[4] = { (1 << 3), (1 << 2), (1 << 1), 1 };

int main() {
    // freopen("input.txt", "r", stdin);
    for (int i = 0; i < 16; i++) {
        if (farmer&i) {
            if ((!(cabbage&i) && !(sheep&i)) || (!(wolf&i)
&& !(sheep&i))) {
                vis[i] = true;
                continue;
            }
        }
        if (!(farmer&i)) {
            if (((cabbage&i) && (sheep&i)) || ((wolf&i) && (sheep&i)))
{
                vis[i] = true;
                continue;
            }
        }
        vis[i] = false;
    }
}

```



```

}
queue<int> q;
q.push(0);
while (!q.empty()) {
    int now = q.front();
    int next;
    q.pop();
    vis[now] = true;
    if (now == 15)
        break;

    if (now & farmer) {
        next = now - farmer;
        if (!vis[next]) {
            q.push(next);
            route[next] = now;
        }
        for (int i = 1; i != (1 << 3); i = (i << 1)) {
            if (i & now) {
                int carry = next - i;
                if (!vis[carry]) {
                    q.push(carry);
                    route[carry] = now;
                }
            }
        }
    }
    else {
        next = now + farmer;
        if (!vis[next]) {
            q.push(next);
            route[next] = now;
        }
        for (int i = 1; i != (1 << 3); i = (i << 1)) {
            if (!(i & now)) {

```

```

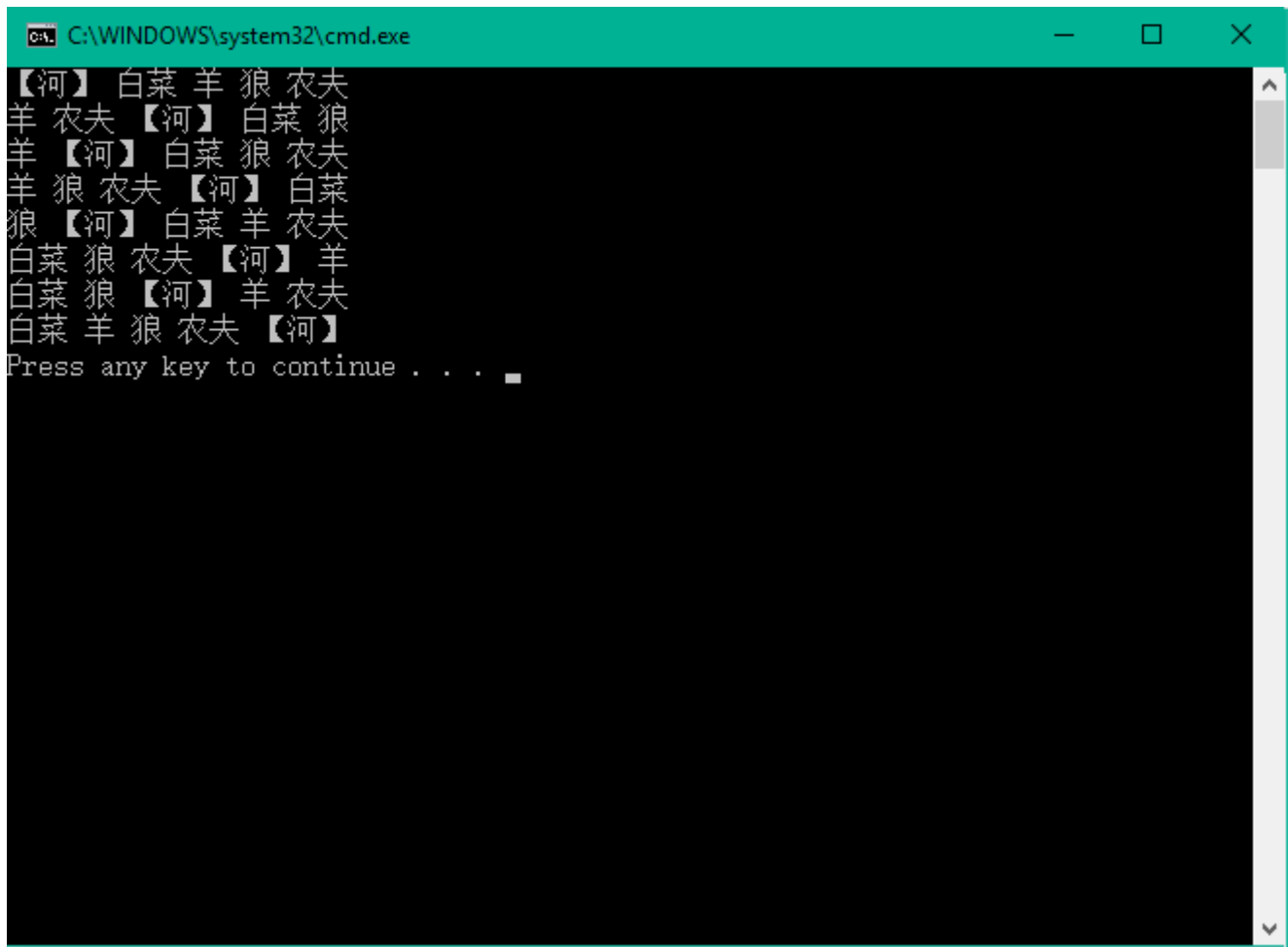
        int carry = next + i;
        if (!vis[carry]) {
            q.push(carry);
            route[carry] = now;
        }
    }
}

int now = 15;
stack<int> s;
while (now) {
    s.push(now);
    now = route[now];
}
s.push(now);
while (!s.empty()) {
    now = s.top();
    s.pop();
    for (int i = 0; i < 4; i++) {
        if (now & (1 << i))
            cout << chinese[i] << ' ';
    }
    cout << "【河】 ";
    for (int i = 0; i < 4; i++) {
        if (!(now & (1 << i)))
            cout << chinese[i] << ' ';
    }
    cout << endl;
}

return 0;
}

```

（四）实验结果



```
C:\WINDOWS\system32\cmd.exe
【河】 白菜 羊 狼 农夫
羊 农夫 【河】 白菜 狼
羊 【河】 白菜 狼 农夫
羊 狼 农夫 【河】 白菜
狼 【河】 白菜 羊 农夫
白菜 狼 农夫 【河】 羊
白菜 狼 【河】 羊 农夫
白菜 羊 狼 农夫 【河】
Press any key to continue . . . .
```

实验十三 迷宫问题

（一）问题描述

迷宫实验是取自心理学的一个古典实验。在该实验中，把一只老鼠从一个无顶大盒子的门放入，在盒中设置了许多墙，对行进方向形成了多处阻挡。盒子仅有一个出口，在出口处放置一块奶酪，吸引老鼠在迷宫中寻找道路以到达出口。对同一只老鼠重复进行上述实验，一直到老鼠从入口到出口，而不走错一步。老鼠经多次试验终于得到它学习走迷宫的路线。

(二) 设计要求

迷宫由 m 行 n 列的二维数组设置, 0 表示无障碍, 1 表示有障碍。设入口为 $(1, 1)$, 出口为 (m, n) , 每次只能从一个无障碍单元移到周围四个方向上任

一无障碍单元。编程实现对任意设定的迷宫, 求出一条从入口到出口的通路, 或得出没有通路的结论。

算法输入: 代表迷宫入口的坐标

算法输出: 穿过迷宫的结果。 算法要点: 创建迷宫, 试探法查找路。

(三) 代码实现

```
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<windows.h>
#include<cstring>
using namespace std;

const int MAX = 100;
int MAZE[MAX][MAX];
int n, m;
int dx[4] = { 1, -1, 0, 0 },
dy[4] = { 0, 0, 1, -1 };

typedef struct Pointer {
    int x;
    int y;
    int tag;
}P;
P st, en;
typedef struct {
    P p[500];
    int top;
}stack;

void creat(stack &s) {
    s.top = 0;
}

void push(stack &s, P q) {
    s.p[s.top].x = q.x;
    s.p[s.top].y = q.y;
```

```

s.p[s.top].tag = q.tag;
s.top++;
}
P pop(stack &s) {
    s.top--;
    return s.p[s.top];
}
P get_first(stack &s) {
    s.p[s.top - 1].tag++;
    return s.p[s.top - 1];
}
int isempty(stack s) {
    if (s.top == 0) {
        return 1;
    }
    else {
        return 0;
    }
}
}
void init() {
// freopen("map.txt", "r", stdin);
cin >> n >> m;
cin >> en.x >> en.y;
en.tag = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        cin >> MAZE[i][j];
    }
}
for (int j = 0; j <= m + 1; j++) {
    MAZE[0][j] = MAZE[n + 1][j] = 1;
}
for (int i = 1; i <= n; i++) {
    MAZE[i][0] = MAZE[i][m + 1] = 1;
}
}
void disp() {
    for (int i = 0; i <= n + 1; i++) {
        for (int j = 0; j <= m + 1; j++) {
            if ((i == st.x && j == st.y) || (i == 9 && j == 9)) {
                cout << "* ";
            }
            else {
                switch (MAZE[i][j]) {
                    case 0: cout << " "; break;

```

```

        case 1: cout << "@ "; break;
        case 2: cout << "$ "; break;
        case 3: cout << " "; break;
        default:;
    }
}

}
cout << endl;
}
Sleep(300);
}

void path(stack &s) {
    cout << "走出迷宫的路径为：" << endl;
    int j = 0;
    for (int i = 0; i < s.top - 1; i++) {
        cout << "(" << s.p[i].x << "," << s.p[i].y << ">>";
        j++;
        if (j >= 5) {
            cout << endl;
            j = 0;
        }
    }
    cout << "(" << s.p[s.top - 1].x << "," << s.p[s.top - 1].y << ")";
}

int maze(P start) {
    stack T;
    creat(T);
    push(T, start);
    P q;
    while (!isempty(T)) {
        q = get_first(T);
        if (q.x == en.x && q.y == en.y) {
            break;
        }
        if (!MAZE[q.x][q.y]) {
            MAZE[q.x][q.y] = 2;
            system("cls");
            disp();
        }
        if (q.tag < 4)
        {
            if (!MAZE[q.x + dx[q.tag]][q.y + dy[q.tag]]) {
                P Q;
                Q.x = q.x + dx[q.tag];
                Q.y = q.y + dy[q.tag];
            }
        }
    }
}

```

```

        Q.tag = -1;
        push(T, Q);
    }
}
else {
    q = pop(T);
    MAZE[q.x][q.y] = 3;
    system("cls");
    disp();
}
}
if (isempty(T)) {
    return 1;
}
else {
    system("cls");
    disp();
    path(T);
    return 0;
}
}

P input_start() {
// freopen("CON", "r", stdin);

cout << "迷宫的大小为" << n << "*" << m << endl;

cout << "请在此范围内输入迷宫入口坐标：" << endl;
char a[100], b[100];
int n1, m1, x, y;
int flag = 1;
while (flag) {
    flag = 0;
    cin >> a >> b;
    n1 = strlen(a);
    m1 = strlen(b);
    for (int i = 0; i < n1; i++) {
        if (a[i] < '0' || a[i] > '9') {
            flag = 1;
            cout << "error" << endl;
            break;
        }
    }
    if (flag) continue;
    for (int i = 0; i < m1; i++) {
        if (b[i] < '0' || b[i] > '9') {

```

```

        flag = 1;
        cout << "error" << endl;
        break;
    }
    if (flag) continue;
    x = 0;
    y = 0;
    for (int i = 0; i < n1; i++) {
        x = x * 10 + a[i] - '0';
    }
    for (int i = 0; i < m1; i++) {
        y = y * 10 + b[i] - '0';
    }
    if (x > m || x < 1 || y > n || y < 1) {
        flag = 1;
        cout << "error" << endl;
        continue;
    }
    if (MAZE[x][y]) {
        cout << "error" << endl;
        flag = 1;
        continue;
    }
}

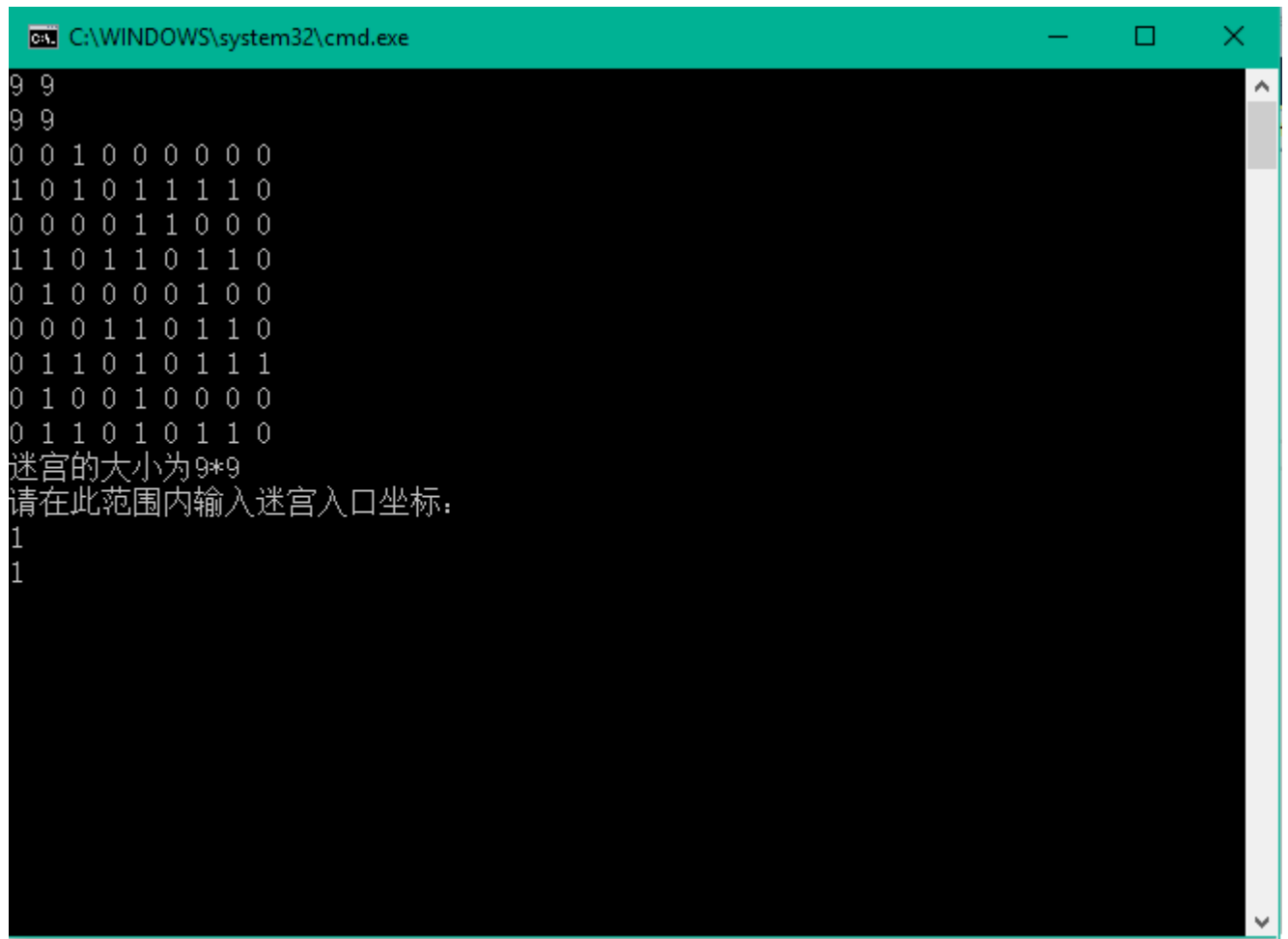
}
P px;
px.x = x;
px.y = y;
px.tag = -1;
return px;
}

int main() {
    init();
    st = input_start();
    if (!maze(st)) {
    }
    else {
        cout << "wrong" << endl;
    }
    return 0;
}

```

(四) 运行结果

输入



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window has a black background with white text. The text displays a 9x9 maze grid, followed by the text "迷宫的大小为 9*9" and "请在此范围内输入迷宫入口坐标:". Below this, the coordinates "1" and "1" are entered on separate lines. The maze grid is as follows:

```
9 9
9 9
0 0 1 0 0 0 0 0 0
1 0 1 0 1 1 1 1 0
0 0 0 0 1 1 0 0 0
1 1 0 1 1 0 1 1 0
0 1 0 0 0 0 1 0 0
0 0 0 1 1 0 1 1 0
0 1 1 0 1 0 1 1 1
0 1 0 0 1 0 0 0 0
0 1 1 0 1 0 1 1 0
```

迷宫的大小为 9*9
请在此范围内输入迷宫入口坐标:
1
1

输出

```
C:\WINDOWS\system32\cmd.exe

@ @ @ @ @ @ @ @ @ @ @ @
@ * $ @ @ @ @ @ @ @
@ @ $ @ @ @ @ @ @
@ $ $ @ @ @ @
@ @ @ $ @ @ @ @
@ @ $ $ $ $ @ @
@ @ @ $ @ @ @
@ @ @ @ $ @ @ @
@ @ @ @ $ @ @ @
@ @ @ @ $ $ $ $ @
@ @ @ @ @ @ * @
@ @ @ @ @ @ @ @ @ @

走出迷宫的路径为：
(1, 1)>>(1, 2)>>(2, 2)>>(3, 2)>>(3, 3)>>
(4, 3)>>(5, 3)>>(5, 4)>>(5, 5)>>(5, 6)>>
(6, 6)>>(7, 6)>>(8, 6)>>(8, 7)>>(8, 8)>>
(8, 9)>>(9, 9)Press any key to continue . . . █
```