

南京航空航天大学

数据结构课程设计

班 级	1819001
学 号	161940233
姓 名	颜 宇 明
指导教师	秦 小 麟

目录

一、 区块链	3
(一) 数据结构	3
(二) 算法设计思想	3
(三) 源程序	3
(四) 测试数据及其结果	9
(五) 时间复杂度	10
(六) 改进方法	11
二、 迷宫问题	11
(一) 数据结构	11
(二) 算法设计思想	11
(三) 源程序	11
(四) 测试数据及其结果	18
(五) 时间复杂度	18
(六) 改进方法	18
三、 JSON 查找	19
(一) 数据结构	19
(二) 算法设计思想	19
(三) 源程序	19
(四) 测试数据及其结果	19
(五) 时间复杂度	19
(六) 改进方法	19
四、 公交线路提示	19
(一) 数据结构	19
(二) 算法设计思想	19
(三) 源程序	19
(四) 测试数据及其结果	19

(五) 时间复杂度	19
(六) 改进方法	19
五、 Hash 表应用	19
(一) 数据结构	19
(二) 算法设计思想	19
(三) 源程序	19
(四) 测试数据及其结果	19
(五) 时间复杂度	19
(六) 改进方法	19
六、 排序算法比较	19
(一) 数据结构	19
(二) 算法设计思想	19
(三) 源程序	19
(四) 测试数据及其结果	19
(五) 时间复杂度	19
(六) 改进方法	19
七、 总结	19
(一) 代码行数	19
(二) 心得体会	20

一、区块链

(一) 数据结构

链表

(二) 算法设计思想

将区块设计成链表节点，每增加一个节点，计算校验码都要结合之前所有的节点的信息。

(三) 源程序

```
1  #ifndef LINKED_LIST_H
2  #define LINKED_LIST_H
3  class ADT_list{
4  public:
5      typedef struct node
6      {
7          int number;
8          char information[100];
9          int Checkcode;
10         node *next;
11     } node;
12     node *head;
13     int length = -1;
14
15     void InitList();
16     void DestoryList();
17     void ClearList();
18     bool ListEmpty();
19     int ListLength();
20     node* GetElem(int);
21     int LocateElem(int);
22     int PriorElem(int);
23     int NextElem(int);
24     void ListTraverse();
25     void CreateList(char*, int);
26     int CheckList();
27     void SetStr(int, const char*);
```

```

28     void InsertElem(char*);
29     int GetCheckcode();
30     void DeleteElem(int index);
31     void Reverse();
32 };
33 #endif

```

代码 1: Linked_list.h

```

1  #include <malloc.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include "Linked_list.h"
5
6  void ADT_list::InitList(){
7      node *tmp = (node *)malloc(sizeof(node));
8      length = 0;
9      head = tmp;
10     tmp->next = NULL;
11 }
12
13 void ADT_list::DestoryList(){
14     node *p, *temp;
15     p = head;
16     while (p != NULL){
17         temp = p;
18         p = p->next;
19         free(temp);
20     }
21     length = -1;
22 }
23
24 void ADT_list::ClearList(){
25     node *p, *temp;
26     p = head->next;
27     while (p != NULL){
28         temp = p;
29         p = p->next;
30         free(temp);
31     }
32     length = 0;
33     head->next = NULL;

```

```

34     }
35
36     bool ADT_list::ListEmpty(){
37         if (length < 1) return true;
38         return false;
39     }
40
41     int ADT_list::ListLength(){
42         return length;
43     }
44
45     ADT_list::node* ADT_list::GetElem(int index){
46         node *p;
47         int i = 0;
48         p = head->next;
49         while (p != NULL){
50             if (index == ++i)
51                 return p;
52             p = p->next;
53         }
54         return NULL;
55     }
56
57     int ADT_list::LocateElem(int num){
58         node *p;
59         p = head->next;
60         int index = 0;
61         while (p != NULL){
62             if (p->number == num)
63                 return index;
64             p = p->next;
65             index++;
66         }
67         return -1;
68     }
69
70     int ADT_list::PriorElem(int cur_num){
71         node *p, *temp;
72         p = head->next;
73         while (p != NULL){
74             temp = p;
75             p = p->next;

```

```

76         if (p != NULL && p->number == cur_num)
77             return temp->number;
78     }
79     return 0;
80 }
81
82 int ADT_list::NextElem(int cur_num){
83     node *p, *temp;
84     p = head->next;
85     while (p != NULL){
86         temp = p;
87         p = p->next;
88         if (p != NULL && temp->number == cur_num)
89             return p->number;
90     }
91     return 0;
92 }
93
94 void ADT_list::ListTraverse(){
95     node *p;
96     p = head->next;
97     puts("Block Chain Output:");
98     while (p != NULL){
99         printf("%d %s %d\n", p->number, p->information, p->Checkcode);
100         p = p->next;
101     }
102     printf("\n");
103 }
104
105 void ADT_list::CreateList(char* str, int check){
106     node *p = head;
107     while (p->next != NULL) p = p->next;
108     node *tmp = (node *)malloc(sizeof(node));
109     p->next = tmp;
110     tmp->next = NULL;
111     tmp->number = length;
112     strcpy(tmp->information, str);
113     length++;
114     tmp->Checkcode = check;
115 }
116
117 int ADT_list::CheckList(){

```

```

118     node* p = head->next, *tmp = p;
119     if (!p) return 1;
120     while(p) {
121         int sum = 0;
122         for (int i = 0; i < strlen(p->information); i++)
123             sum += p->information[i];
124         if (p->number) {
125             node *temp = head;
126             sum += p->number;
127             while (temp->next != p) temp = temp->next;
128             sum += temp->Checkcode;
129         }
130         // printf("%d %d %d\n", sum % 113, p->Checkcode, p->number);
131         if (sum % 113 != p->Checkcode) return p->number + 100;
132         p = p->next;
133     }
134     return 1;
135 }
136
137 void ADT_list::SetStr(int index, const char* str){
138     node *p;
139     int i = 0;
140     p = head->next;
141     while (p && index != i++) p = p->next;
142     strcpy(p->information, str);
143     while(p) {
144         int sum = 0;
145         for (int i = 0; i < strlen(p->information); i++)
146             sum += p->information[i];
147         if (p->number) {
148             node *temp = head;
149             sum += p->number;
150             while (temp->next != p) temp = temp->next;
151             sum += temp->Checkcode;
152         }
153         // printf("%d %d %d\n", sum % 113, p->Checkcode, p->number);
154         // if (sum % 113 != p->Checkcode) return p->number + 100;
155         p->Checkcode = sum % 113;
156         p = p->next;
157     }
158 }
159

```



```

160 void ADT_list::InsertElem(char* str){
161     node *p = head;
162     while (p->next != NULL) p = p->next;
163     node *tmp = (node *)malloc(sizeof(node));
164     p->next = tmp;
165     tmp->next = NULL;
166     tmp->number = length;
167     strcpy(tmp->information, str);
168     length++;
169     tmp->Checkcode = GetCheckcode();
170 }
171 int ADT_list::GetCheckcode(){
172     node* p = head;
173     while (p->next != NULL) p = p->next;
174     int sum = 0;
175     for (int i = 0; i < strlen(p->information); i++)
176         sum += p->information[i];
177     if (length > 1) {
178         sum += p->number;
179         p = head;
180         while (p->next->next) p = p->next;
181         sum += p->Checkcode;
182     }
183     return sum % 113;
184 }
185 void ADT_list::DeleteElem(int index){
186     node *p, *temp;
187     int i = 0;
188     temp = head;
189     p = head->next;
190     length--;
191     while (p != NULL){
192         if (index != ++i) {
193             temp = p;
194             p = p->next;
195             continue;
196         }
197         temp->next = p->next;
198         free(p);
199     }
200 }
201

```

```

202 void ADT_list::Reverse(){
203     if (length < 1) return;
204     node *p, *temp, *tmp = NULL;
205     temp = p = head->next;
206     while(temp != NULL){
207         temp = p->next;
208         p->next = tmp;
209         tmp = p;
210         p = temp;
211     }
212     head->next = tmp;
213 }

```

代码 2: Linked_list.cpp

```

1  #include <stdio.h>
2  #include "Linked_list.h"
3  char str[100];
4  int num, check;
5  int main(){
6      ADT_list list;
7      list.InitList();
8      freopen("insertnode", "r", stdin);
9      while (scanf("%s", &str) != EOF) list.InsertElem(str);
10     list.ListTraverse();
11     list.ClearList();
12     freopen("CheckBlockChain", "r", stdin);
13     while (scanf("%s%d", &str, &check) != EOF) list.CreateList(str,
        check);
14     list.ListTraverse();
15     (num = list.CheckList()) < 100 ? puts("Accept!") : printf("The %
        dth node Error!\n\n", num - 100);
16     list.SetStr(1, "22");
17     list.ListTraverse();
18
19 }

```

代码 3: main.cpp

(四) 测试数据及其结果

文件 insertnode 创建区块链输入:

```
1      3
2      2
3      1
```

文件 CheckBlockChain 创建一个错误的区块链：

```
1      3 51
2      2 102
3      1 4
```

main.cpp 输出为：

```
1      Block Chain Output:
2      0 3 51
3      1 2 102
4      2 1 40
5
6      Block Chain Output:
7      0 3 51
8      1 2 102
9      2 1 4
10
11     The 2th node Error!
12
13     Block Chain Output:
14     0 3 51
15     1 22 39
16     2 1 90
```

由以上输出结果可知，题目要求所有完全达到了。

(五) 时间复杂度

$O(n)$

(六) 改进方法

算校验码不需要每次都把前面所有的节点都遍历一遍，可以直接用最后一个节点的校验码。

二、 迷宫问题

(一) 数据结构

用栈来实现深度优先搜索

(二) 算法设计思想

先生成迷宫，设定起点，然后用栈实现的深度优先搜索来寻找迷宫的出口，迷宫保证路径唯一。

(三) 源程序

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "Sequence_Stack.h"
4  #include <fstream>
5  using namespace std;
6  int maze[100][100], n, visited[100][100];
7  int xx[] = {-1, 1, 0, 0};
8  int yy[] = {0, 0, -1, 1};
9  int endx, endy, flag;
10 ADT_Stack Stack;
11 string temp;
12 int main(){
13     ifstream ReadFile("maze");
14     while(getline(ReadFile, temp)) n++;
15
16     Stack.InitStack();
17     freopen("maze", "r", stdin);
18     for (int i = 1; i <= n; i++)
19         for (int j = 1; j <= n; j++)
20             scanf("%d", &maze[i][j]);
21     for (int i = n; i >= 1; i--)
22         if (!maze[i][n]) endx = i, endy = n;
```

```

23 Pos *point = new Pos;
24 point->x = 2;
25 point->y = 1;
26 point->last = NULL;
27 Stack.Push(*point);
28 visited[point->x][point->y] = 1;
29 while (Stack.rear) {
30     Pos pp = Stack.Pop();
31     Pos *p = new Pos;
32     p->x = pp.x;
33     p->y = pp.y;
34     p->last = pp.last;
35     visited[p->x][p->y] = 1;
36     if (p->x == endx && p->y == endy) {
37         point = p;
38         while (point)
39             maze[point->x][point->y] = 2, point = point->last;
40
41         for (int i = 0; i <= n; i++) {
42             for (int j = 0; j <= n; j++) {
43                 if (maze[i][j] == 0)
44                     printf(" ");
45                 else if (maze[i][j] == 2)
46                     printf("■");
47                 else
48                     printf("░");
49             }
50             printf("\n");
51         }
52         return 0;
53     }
54     for (int i = 0; i < 4; i++)
55         if (!maze[p->x + xx[i]][p->y + yy[i]] && !visited[p->x + xx[
56             i][p->y + yy[i]]) {
57             point = new Pos;
58             point->x = p->x + xx[i];
59             point->y = p->y + yy[i];
60             point->last = p;
61             Stack.Push(*point);
62         }
63 }

```

64 }

代码 4: main.cpp

```
1  #include<stdio.h>
2  #include<fstream>
3  #include<Windows.h>
4  #include<time.h>
5  #include<math.h>
6
7  //地图长度L，包括迷宫主体40，外侧的包围的墙体2，最外侧包围路径2（之后会
   解释）
8  #define L 44
9
10 //墙和路径的标识
11 #define WALL 0
12 #define ROUTE 1
13 using namespace std;
14 //控制迷宫的复杂度，数值越大复杂度越低，最小值为0
15 static int Rank = 0;
16 int startx = 2, starty = 1;
17
18 //生成迷宫
19 void CreateMaze(int **maze, int x, int y);
20
21 int main(void) {
22     srand((unsigned)time(NULL));
23
24     int **Maze = (int**)malloc(L * sizeof(int *));
25     for (int i = 0; i < L; i++) {
26         Maze[i] = (int*)calloc(L, sizeof(int));
27     }
28
29     //最外围层设为路径的原因，为了防止挖路时挖出边界，同时为了保护迷宫主
   体外的一圈墙体被挖穿
30     for (int i = 0; i < L; i++){
31         Maze[i][0] = ROUTE;
32         Maze[0][i] = ROUTE;
33         Maze[i][L - 1] = ROUTE;
34         Maze[L - 1][i] = ROUTE;
35     }
36 }
```

```

37 //创造迷宫, (2, 2) 为起点
38 CreateMaze(Maze, startx, starty + 1);
39
40 //画迷宫的入口和出口
41 Maze[2][1] = ROUTE;
42
43 //由于算法随机性, 出口有一定概率不在 (L-3,L-2) 处, 此时需要寻找出口
44 for (int i = L - 3; i >= 0; i--) {
45     if (Maze[i][L - 3] == ROUTE) {
46         Maze[i][L - 2] = ROUTE;
47         break;
48     }
49 }
50
51 //画迷宫
52 for (int i = 0; i < L; i++) {
53     for (int j = 0; j < L; j++) {
54         if (Maze[i][j] == ROUTE) {
55             printf(" ");
56         }
57         else {
58             printf("■");
59         }
60     }
61     printf("\n");
62 }
63 ofstream out("maze");
64 if (out.is_open()){
65     for (int i = 1; i < L - 1; i++) {
66         for (int j = 1; j < L - 1; j++) {
67             if (Maze[i][j] == ROUTE)
68                 out << "0 ";
69             else
70                 out << "1 ";
71         }
72         out << "\n";
73     }
74 }
75
76 for (int i = 0; i < L; i++) free(Maze[i]);
77 free(Maze);
78

```

```

79     return 0;
80 }
81
82 void CreateMaze(int **maze, int x, int y) {
83     maze[x][y] = ROUTE;
84
85     //确保四个方向随机
86     int direction[4][2] = { { 1,0 }, { -1,0 }, { 0,1 }, { 0,-1 } };
87     for (int i = 0; i < 4; i++) {
88         int r = rand() % 4;
89         int temp = direction[0][0];
90         direction[0][0] = direction[r][0];
91         direction[r][0] = temp;
92
93         temp = direction[0][1];
94         direction[0][1] = direction[r][1];
95         direction[r][1] = temp;
96     }
97
98     //向四个方向开挖
99     for (int i = 0; i < 4; i++) {
100         int dx = x;
101         int dy = y;
102
103         //控制挖的距离，由Rank来调整大小
104         int range = 1 + (Rank == 0 ? 0 : rand() % Rank);
105         while (range > 0) {
106             dx += direction[i][0];
107             dy += direction[i][1];
108
109             //排除掉回头路
110             if (maze[dx][dy] == ROUTE) {
111                 break;
112             }
113
114             //判断是否挖穿路径
115             int count = 0;
116             for (int j = dx - 1; j < dx + 2; j++) {
117                 for (int k = dy - 1; k < dy + 2; k++) {
118                     //abs(j - dx) + abs(k - dy) == 1 确保只判断九宫格的
119                     //四个特定位置
120                     if (abs(j - dx) + abs(k - dy) == 1 && maze[j][k] ==

```



```

120             ROUTE) {
121                 count++;
122             }
123         }
124     }
125     if (count > 1) {
126         break;
127     }
128
129     //确保不会挖穿时，前进
130     --range;
131     maze[dx][dy] = ROUTE;
132 }
133
134 //没有挖穿危险，以此为节点递归
135 if (range <= 0) {
136     CreateMaze(maze, dx, dy);
137 }
138 }
139 }

```

代码 5: MazeGeneration.cpp

```

1  #ifndef SEQUENCE_Stack_H
2  #define SEQUENCE_Stack_H
3  typedef struct Pos
4  {
5      int x;
6      int y;
7      Pos *last;
8  } Pos;
9  class ADT_Stack{
10 public:
11     Pos Stack[9999];
12     int rear = -1;
13     void InitStack();
14     void DestoryStack();
15     void ClearStack();
16     bool StackEmpty();
17     int StackLength();
18     Pos GetTop();

```

```

19     void StackTraverse();
20     void Push(Pos);
21     Pos Pop();
22     Pos operator [] (int);
23     // int LocateElem(int num);
24     // int PriorElem(int cur_num);
25     // int NextElem(int cur_num);
26     // int SetElem(int index, int num);
27     // void InsertElem(int index, int num);
28     // void DeleteElem(int index);
29     // void Remove();
30     // void Bubble_Sort();
31     // void Select_sort();
32     // ADT_Stack Union(ADT_Stack);
33     // void Josephus(int);
34 };
35 #endif

```

代码 6: Sequence_Stack.h

```

1  CC = g++
2
3  OUT1 = MazeGeneration.exe
4  OBJ1 = MazeGeneration.o
5
6  OUT2 = main.exe
7  OBJ2 = main.o Sequence_Stack.o
8
9  IN = main.cpp MazeGeneration.cpp Sequence_Stack.cpp
10
11 build: $(OUT1) $(OUT2)
12
13 run: $(OUT1) $(OUT2)
14     @./$(OUT1);./$(OUT2)
15
16 clean:
17     @rm -f *.o $(OUT1) $(OUT2)
18
19 $(OUT1): $(OBJ1)
20     @$(CC) $(OBJ1) -o $(OUT1)
21
22 $(OUT2): $(OBJ2)

```

```
23     @$(CC) $(OBJ2) -o $(OUT2)
24
25 $(OBJ): $(IN)
26     @$(CC) -c $(IN)
```

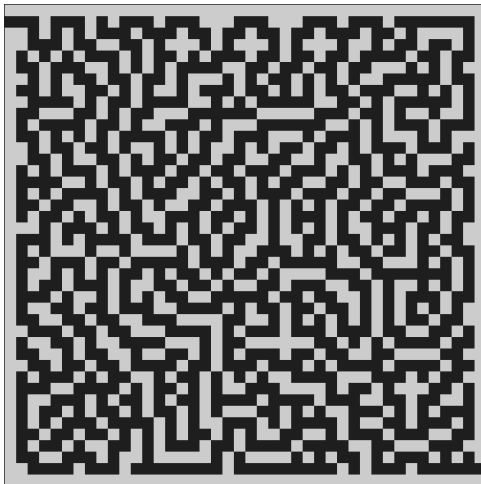
代码 7: Makefile

(四) 测试数据及其结果

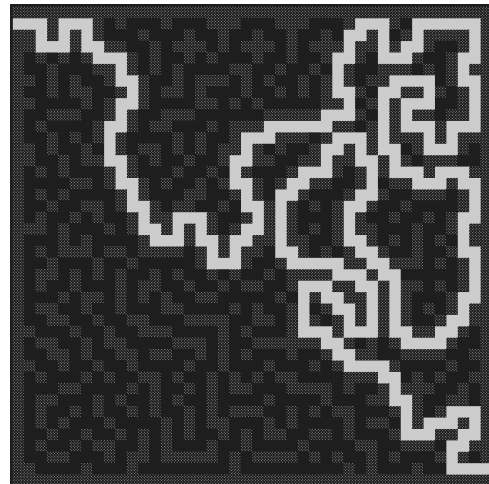
在项目目录下运行命令：

```
1 clear;make clean;make run
```

运行结果为：



(a) 自动生成的迷宫



(b) 基于 DFS 的迷宫路径可视化

图 1: 实验结果

由以上结果可知，实现题目的全部要求。

(五) 时间复杂度

$$O(n^2)$$

(六) 改进方法

可以尝试用 Dijkstra 算法来解决这个问题。

三、JSON 查找

- (一) 数据结构
- (二) 算法设计思想
- (三) 源程序
- (四) 测试数据及其结果
- (五) 时间复杂度
- (六) 改进方法

四、公交线路提示

- (一) 数据结构
- (二) 算法设计思想
- (三) 源程序
- (四) 测试数据及其结果
- (五) 时间复杂度
- (六) 改进方法

五、Hash 表应用

- (一) 数据结构
- (二) 算法设计思想
- (三) 源程序
- (四) 测试数据及其结果
- (五) 时间复杂度
- (六) 改进方法

六、排序算法比较

- (一) 数据结构
- (二) 算法设计思想
- (三) 源程序
- (四) 测试数据及其结果

题目	行数
区块链	283
迷宫问题	388
JSON 查找	×

(二) 心得体会