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

（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

```
#ifndef LINKED_LIST_H
#define LINKED_LIST_H
class ADT_list{
public:
    typedef struct node
    {
        int number;
        char information[100];
        int Checkcode;
        node *next;
    } node;
    node *head;
    int length = -1;

    void InitLIst();
    void DestoryList();
    void ClearList();
    bool ListEmpty();
    int ListLength();
    node* GetElem(int);
    int LocateElem(int);
    int PriorElem(int);
    int NextElem(int);
    void ListTraverse();
    void CreateList(char*, int);
    int CheckList();
    void SetStr(int, const char*);
```

```
    void InsertElem(char*);
    int GetCheckcode();
    void DeleteElem(int index);
    void Reverse();
};
#endif
```

代码 1: Linked_list.h

```
#include <malloc.h>
#include <stdio.h>
#include <string.h>
#include "Linked_list.h"

void ADT_list::InitLIst(){
    node *tmp = (node *)malloc(sizeof(node));
    length = 0;
    head = tmp;
    tmp->next = NULL;
}

void ADT_list::DestoryList(){
    node *p, *temp;
    p = head;
    while (p != NULL){
        temp = p;
        p = p->next;
        free(temp);
    }
    length = -1;
}

void ADT_list::ClearList(){
    node *p, *temp;
    p = head->next;
    while (p != NULL){
        temp = p;
        p = p->next;
        free(temp);
    }
    length = 0;
    head->next = NULL;
```

```
}

bool ADT_list::ListEmpty(){
    if (length < 1) return true;
    return false;
}

int ADT_list::ListLength(){
    return length;
}

ADT_list::node* ADT_list::GetElem(int index){
    node *p;
    int i = 0;
    p = head->next;
    while (p != NULL){
        if (index == ++i)
            return p;
        p = p->next;
    }
    return NULL;
}

int ADT_list::LocateElem(int num){
    node *p;
    p = head->next;
    int index = 0;
    while (p != NULL){
        if (p->number == num)
            return index;
        p = p->next;
        index++;
    }
    return -1;
}

int ADT_list::PriorElem(int cur_num){
    node *p, *temp;
    p = head->next;
    while (p != NULL){
        temp = p;
        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：

```
3
2
1
```

文件 CheckBlockChain 创建一个错误的区块链：

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

main.cpp 输出为：

```
Block Chain Output:
0 3 51
1 2 102
2 1 40

Block Chain Output:
0 3 51
1 2 102
2 1 4

The 2th node Error!

Block Chain Output:
0 3 51
1 22 39
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

```
#include <stdio.h>
#include <stdlib.h>
#include "Sequence_Stack.h"
#include <fstream>
using namespace std;
int maze[100][100], n, visited[100][100];
int xx[] = {-1, 1, 0, 0};
int yy[] = {0, 0, -1, 1};
int endx, endy, flag;
ADT_Stack Stack;
string temp;
int main(){
    ifstream ReadFile("maze");
    while(getline(ReadFile, temp)) n++;

    Stack.InitStack();
    freopen("maze", "r", stdin);
    for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        scanf("%d", &maze[i][j]);
    for (int i = n; i >= 1; i--)
        if (!maze[i][n]) endx = i, endy = n;
```

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

```
    //创造迷宫，（2，2）为起点
    CreateMaze(Maze, startx, starty + 1);

    //画迷宫的入口和出口
    Maze[2][1] = ROUTE;

    //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
    for (int i = L - 3; i >= 0; i--) {
        if (Maze[i][L - 3] == ROUTE) {
            Maze[i][L - 2] = ROUTE;
            break;
        }
    }

    //画迷宫
    for (int i = 0; i < L; i++) {
        for (int j = 0; j < L; j++) {
            if (Maze[i][j] == ROUTE) {
                printf("  ");
            }
            else {
                printf("■■");
            }
        }
        printf("\n");
    }
    ofstream out("maze");
    if (out.is_open()){
        for (int i = 1; i < L - 1; i++) {
            for (int j = 1; j < L - 1; j++) {
                if (Maze[i][j] == ROUTE)
                    out << "0 ";
                else
                    out << "1 ";
            }
            out << "\n";
        }
    }

    for (int i = 0; i < L; i++) free(Maze[i]);
    free(Maze);

```

```
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
                        四个特定位置
119                     if (abs(j - dx) + abs(k - dy) == 1 && maze[j][k] ==
```

```
                        ROUTE) {
120                         count++;
121                     }
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();
```

```
    void StackTraverse();
    void Push(Pos);
    Pos Pop();
    Pos operator [] (int);
    // int LocateElem(int num);
    // int PriorElem(int cur_num);
    // int NextElem(int cur_num);
    // int SetElem(int index, int num);
    // void InsertElem(int index, int num);
    // void DeleteElem(int index);
    // void Remove();
    // void Bubble_Sort();
    // void Select_sort();
    // ADT_Stack Union(ADT_Stack);
    // void Josephus(int);
};
#endif
```

代码 6: Sequence_Stack.h

```
CC = g++

OUT1 = MazeGeneration.exe
OBJ1 = MazeGeneration.o

OUT2 = main.exe
OBJ2 = main.o Sequence_Stack.o

IN = main.cpp MazeGeneration.cpp Sequence_Stack.cpp

build: $(OUT1) $(OUT2)

run: $(OUT1) $(OUT2)
    @./$(OUT1);./$(OUT2)

clean:
    @rm -f *.o $(OUT1) $(OUT2)

$(OUT1): $(OBJ1)
    @$(CC) $(OBJ1) -o $(OUT1)

$(OUT2): $(OBJ2)
```

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

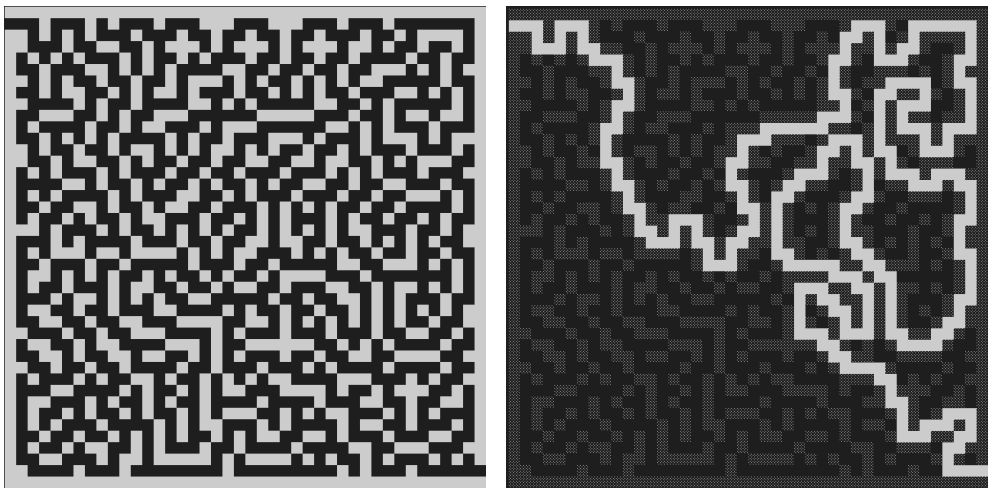
代码 7: Makefile

（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

各种排序算法，都是数组。

（二）算法设计思想

1) 直接插入排序：将一条记录插入到已排好的有序表中，从而得到一个新的、记录数量增 1 的有序表。

2) 希尔排序：希尔排序是把记录按下标的一定增量分组，对每组使用直接插入排序算法排序；随着增量逐渐减少，每组包含的关键词越来越多，当增量减至 1 时，整个文件恰被分成一组，算法便终止。

3) 冒泡排序：依次比较两个相邻的元素，如果顺序（如从大到小、首字母从 Z 到 A）错误就把他们交换过来。走访元素的工作是重复地进行直到没有相邻元素需要交换，也就是说该元素列已经排序完成。

4) 快速排序：挑选基准值：从数列中挑出一个元素，称为“基准”（pivot），分割：重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（与基准值相等的数可以到任何一边）。在这个分割结束之后，对基准值的排序就已经完成，递归排序子序列：递归地将小于基准值元素的子序列和大于基准值元素的子序列排序。

5) 选择排序：第一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后再从剩余的未排序元素中寻找到最小（大）元素，然后放到已排序的序列的末尾。以此类推，直到全部待排序的数据元素的个数为零。

6) 堆排序：利用堆这种数据结构所设计的一种排序算法。堆积是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

7) 归并排序：将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。

8) 基数排序：将整数按位数切割成不同的数字，然后按每个位数分别比较。具体做法是：将所有待比较数值统一为同样的数位长度，数位较短的数前面补零。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后, 数列就变成一个有序序列。

（三）源程序

代码 8: Sort.cpp

```
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <time.h>
using namespace std;
int num = 50000, n, a = num;
int main(void) {
    srand((unsigned)time(NULL));
    ofstream out("1Sample");
    while (num)
        out << a - num-- << endl;
    out.close();

    out.open("2Sample");
    num = a;
    while (num--) out << num << endl;
    out.close();

    num = 10;
    for (int i = 3; i <= num; i++){
        out.open(to_string(i) + "Sample");
        n = a;
        while (n--)
            out << rand() % a << endl;
        out.close();
    }
    return 0;
}
```

代码 9: NumGeneration.cpp

```
CC = g++
SOURCE := NumGeneration.cpp Sort.cpp

build:
    @$(foreach var,$(SOURCE),\
        $(CC) -c $(var); \
        $(CC) $(subst .cpp,.o,$(var)) -o $(subst .cpp,.exe,$(var)); \
        ./$(subst .cpp,.exe,$(var));\
    )
```

```
10
11 clean:
12     @rm -f *.o *.exe *Sample
```

代码 10: Makefile

（四）测试数据及其结果

测试数据用 NumGeneration.cpp 生成 10 个每个含有 50000 个元素的样本。

```
1           Insert   Shell  Bubble  Select    Heap   Merge   Radix  Insert
2  1Sample:  0.000s  0.003s  3.064s  2.685s  0.010s  0.000s  0.015s
3  2Sample:  4.707s  0.003s  5.329s  2.825s  0.009s  0.004s  0.004s
4  3Sample:  2.413s  0.010s  7.257s  2.761s  0.011s  0.007s  0.004s
5  4Sample:  2.793s  0.010s  7.300s  2.763s  0.011s  0.007s  0.004s
6  5Sample:  2.566s  0.011s  7.392s  2.733s  0.011s  0.007s  0.004s
7  6Sample:  2.472s  0.010s  7.217s  2.662s  0.012s  0.007s  0.004s
8  7Sample:  2.367s  0.010s  6.910s  2.611s  0.011s  0.006s  0.004s
9  8Sample:  2.302s  0.010s  6.812s  2.656s  0.011s  0.006s  0.003s
10 9Sample:  2.412s  0.011s  7.042s  2.717s  0.011s  0.007s  0.004s
11 10Sample: 2.337s  0.010s  7.519s  2.815s  0.016s  0.008s  0.005s
```

代码 11: 结果

（五）时间复杂度

排序方法	时间复杂度	最坏情况	空间复杂度	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
希尔排序	$O(n^{1.5})$	$O(n^{1.5})$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
快速排序	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	不稳定
简单选择排序	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序	$O(dn)$	$O(dn)$	$O(rd)$	稳定

图 2: 排序算法比较

（六）改进方法

每种排序算法都是固定的，不需要改进。

七、 总结

（一）代码行数

题目	行数
区块链	283
迷宫问题	388
JSON 查找	×

总代码行数为：1000.

（二）心得体会