

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 26

(五) 时间复杂度	28
(六) 改进方法	28
五、 Hash 表应用	28
(一) 数据结构	28
(二) 算法设计思想	28
(三) 源程序	28
(四) 测试数据及其结果	28
(五) 时间复杂度	28
(六) 改进方法	28
六、 排序算法比较	28
(一) 数据结构	28
(二) 算法设计思想	28
(三) 源程序	29
(四) 测试数据及其结果	30
(五) 时间复杂度	31
(六) 改进方法	31
七、 总结	31
(一) 代码行数	31
(二) 心得体会	32

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     }
55     for (int i = 0; i < 4; i++)
56         if (!maze[p->x + xx[i]][p->y + yy[i]] && !visited[p->x + xx[
57             i][p->y + yy[i]]) {
58             point = new Pos;
59             point->x = p->x + xx[i];
60             point->y = p->y + yy[i];
61             point->last = p;
62             Stack.Push(*point);
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)
```

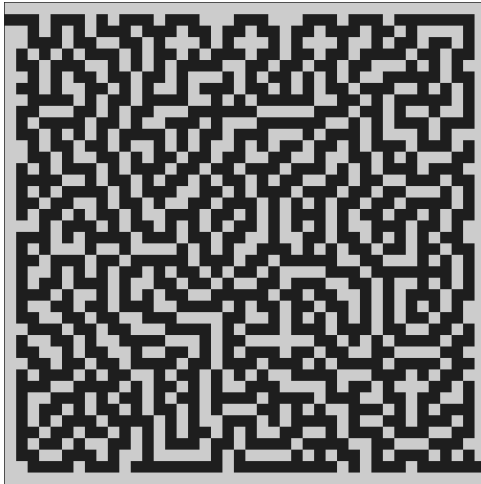
代码 7: Makefile

(四) 测试数据及其结果

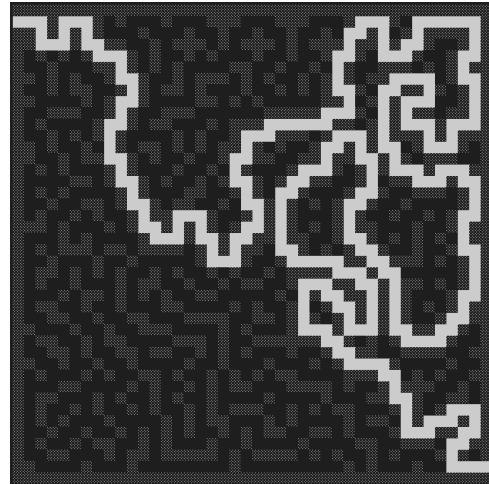
在项目目录下运行命令：

```
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

栈，队列，邻接矩阵，Dijkstra 算法，广度优先搜索

- (二) 算法设计思想

1) 最短路径：利用邻接矩阵 Dijkstra 算法，计算两点之间的最短路径。

2) 最少转乘：采用广度优先遍历算法，判断起点所涉及的每辆公交车经过的所有站点，如果有终点，则无需转车。若无终点，则让起点所涉及的每辆公交车经过的所有站点进入队列，通过 Visited 数组判断该站点是否已经访问过。队列元素出队，继续判断该点所涉及的每辆公交车经过的所有站点，重复以上步骤，直至找到所需终点。

- (三) 源程序

```
1 #include <map>
2 #include <list>
3 #include <queue>
4 #include <string>
5 #include <vector>
6 #include <fstream>
7 #include <cstring>
8 #include <iostream>
9 #include <algorithm>
```

```

10 using namespace std;
11 #define INF 0x3f3f3f3f
12 #define maxn 40000
13
14 vector<string> split(const string& str, const string& delim) {
15     vector<string> res;
16     if( "" == str) return res;
17     //先将要切割的字符串从string类型转换为char*类型
18     char * strs = new char[str.length() + 1];
19     strcpy(strs, str.c_str());
20
21     char * d = new char[delim.length() + 1];
22     strcpy(d, delim.c_str());
23
24     char *p = strtok(strs, d);
25     while(p) {
26         string s = p; //分割得到的字符串转换为string类型
27         res.push_back(s); //存入结果数组
28         p = strtok(NULL, d);
29     }
30     return res;
31 }
32
33 class FindKey
34 {
35 private:
36     int num;
37 public:
38     FindKey(int n) :num(n){}
39     bool operator()(map<string, int>::value_type item){
40         return item.second == num;
41     }
42 };
43
44 class BusStationDatabase
45 {
46 private:
47     typedef struct Pos
48     {
49         int StationNum;
50         int BusNum;
51         Pos *last;

```

```

52     } Pos;
53
54     list<string> stations;
55     int StationNumber;
56     int RouteNumber;
57     int StartStaionNum;
58     map<string, int> StationMap;           //站点及其编号的映射
59     map<int, vector<string> > BusInfo;    //第几路公交车及其经过站
        点的映射
60     map<string, vector<int> > StationInfo; //每个站点都有哪些公交车
        经过
61     queue<Pos> Queue;
62
63     struct Edge{
64         int v;
65         int len;
66         int next;
67     } edge[111111];
68     int head[maxn], cnt = 0;
69     int from, to;
70     string start, end;
71     int visit[maxn];
72     int dis[maxn], path[maxn];
73
74
75 public:
76     BusStationDatabase(){
77         StationNumber = 1;
78         RouteNumber = 0;
79     }
80     void LoadData(){
81         ifstream inputfile("NanjingBusRoute");
82         if(inputfile.fail()){
83             cout<<"Open failed."<<endl;
84             inputfile.close();
85             return;
86         }
87         string input_line;
88         while(!inputfile.eof()){
89             getline(inputfile, input_line);
90             vector<string> splitarr = split(input_line, " ");
91             splitarr[0].pop_back();

```

```

92         vector<string> curBusStation = split(splitarr[1], ",");
93         BusInfo[stoi(splitarr[0])] = curBusStation;
94         for (auto i : curBusStation){
95             auto key = StationMap.find(i);
96             if (key == StationMap.end())
97                 StationMap[i] = StationNumber++;
98             StationInfo[i].push_back(stoi(splitarr[0]));
99         }
100     }
101     inputfile.close();
102     StationNumber--;
103     RouteNumber = BusInfo.size();
104     return;
105 }
106 void BuildGraph(){
107     for (auto i : BusInfo) {
108         for(int j = 0; j < i.second.size() - 1; j++){
109             from = StationMap[i.second[j]];
110             to = StationMap[i.second[j + 1]];
111             addedge(from, to, 1);
112             addedge(to, from, 1);
113         }
114     }
115 }
116 void PrintLeastStationRoute(int s){
117     if (!s) return;
118     PrintLeastStationRoute(path[s]);
119     auto it = find_if(StationMap.begin(), StationMap.end(),
120                     FindKey(s));
121     if (it != StationMap.end())
122         if (path[s])
123             cout << ">" << (*it).first;
124         else
125             cout << (*it).first;
126 }
127 void LeastStationRoute(){
128     freopen("StartEndStation", "r", stdin);
129     cin >> start >> end;
130     memset(path, 0, sizeof(path));
131     memset(visit, 0, sizeof(visit));
132     int flag = 0;
133     auto key = StationMap.find(start);

```

```

133         if (key == StationMap.end()) flag = 1;
134         key = StationMap.find(end);
135         if (key == StationMap.end()) flag = 1;
136         if (flag) {
137             cout << "error!" << endl;
138             return;
139         }
140         dijkstra();
141         for (int i = 1; i <= StationNumber; i++)
142             if (i == StationMap[end]) {
143                 cout << "经过站点最少: " << dis[i] << "站" << endl;
144                 int p = path[i];
145                 PrintLeastStationRoute(i);
146                 cout << endl << endl;
147             }
148     }
149     void LeastChangeRoute(){
150         memset(path, 0, sizeof(path));
151         memset(visit, 0, sizeof(visit));
152         Pos *point = new Pos;
153         point->BusNum = 0;
154         point->last = NULL;
155         point->StationNum = StationMap[start];
156         Queue.push(*point);
157         visit[point->StationNum] = 1;
158         while (!Queue.empty()) {
159             Pos front = Queue.front();
160             Queue.pop();
161             Pos *p = new Pos;
162             p->StationNum = front.StationNum;
163             p->BusNum = front.BusNum;
164             p->last = front.last;
165             visit[p->StationNum] = 1;
166             if (p->StationNum == StationMap[end]) {
167                 point = ReverseList(p);
168                 point->BusNum = point->last->BusNum;
169                 cout << "转车次数最少: " << endl;
170                 while (point->last){
171                     int tmp = point->StationNum;
172                     auto ss = find_if(StationMap.begin(), StationMap
                                   .end(), FindKey(tmp));
173                     auto ee = find_if(StationMap.begin(), StationMap

```



```

        .end(), FindKey(point->last->StationNum));
174     auto StartStation = find(BusInfo[point->last->
        BusNum].begin(), BusInfo[point->last->BusNum
        ].end(), (*ss).first);
175     auto EndStation = find(BusInfo[point->last->
        BusNum].begin(), BusInfo[point->last->BusNum
        ].end(), (*ee).first);
176     if (StartStation > EndStation) reverse(BusInfo[
        point->last->BusNum].begin(), BusInfo[point
        ->last->BusNum].end());
177     int flag = 0;
178     for (auto i : BusInfo[point->last->BusNum]) {
179         if (i == (*ss).first || flag) {
180             if (i == (*ee).first){
181                 cout << i << "(" << point->last->
                    BusNum << "路)";
182                 break;
183             }
184             cout << i << "(" << point->last->BusNum
                    << "路)->";
185             flag = 1;
186         }
187     }
188     if (point->last->last) cout << endl << "转" <<
        endl;
189     point = point->last;
190 }
191 return;
192 }
193 auto StationName = find_if(StationMap.begin(),
        StationMap.end(), FindKey(p->StationNum));
194 for (auto i : StationInfo[(*StationName).first]) { //能
        经过该站点所有公交车的列表
195     for (auto j : BusInfo[i]) { // 该公交车经过的所有站
        点列表
196         if (!visit[StationMap[j]]) { //如果这个站点没有
            被访问过，就加到队列里
197             point = new Pos;
198             point->StationNum = StationMap[j];
199             point->BusNum = i;
200             point->last = p;
201             Queue.push(*point);

```

```

202         }
203     }
204 }
205 }
206 }
207 Pos* ReverseList(Pos *root){
208     Pos *p, *temp, *tmp = NULL;
209     temp = p = root;
210     while(temp){
211         temp = p->last;
212         p->last = tmp;
213         tmp = p;
214         p = temp;
215     }
216     return tmp;
217 }
218 void addedge(int from, int to, int len){
219     edge[++cnt].v = to;
220     edge[cnt].len = len;
221     edge[cnt].next = head[from];
222     head[from] = cnt;
223 }
224 void dijkstra(){
225     for (int i = 1; i <= StationNumber; i++)
226         dis[i] = INF;
227     int temp = StationMap[start];
228     dis[temp] = 0;
229     int minn;
230     while (!visit[temp]){
231         visit[temp] = 1;
232         for (int j = head[temp]; j; j = edge[j].next)
233             if (!visit[edge[j].v] && dis[edge[j].v] > dis[temp]
234                 + edge[j].len)
235                 dis[edge[j].v] = dis[temp] + edge[j].len, path[
236                     edge[j].v] = temp;
237         minn = INF;
238         for (int j = 1; j <= StationNumber; j++)
239             if (!visit[j] && minn > dis[j])
240                 minn = dis[j], temp = j;
241     }
242     int get_StationNumber(){return StationNumber;}

```

```

242         int get_route_number(){return RouteNumber;}
243         list<string>& get_station_name_list(){return stations;}
244     };
245     BusStationDatabase lib;
246     int main()
247     {
248         lib.LoadData();
249         lib.BuildGraph();
250         lib.LeastStationRoute();
251         lib.LeastChangeRoute();
252         return 0;
253     }

```

代码 8: BusRoute.cpp

(四) 测试数据及其结果

起点：干休所站

终点：墨香路站

```

1  经过站点最少：10站
2  干休所站->河路道站->中央门北站->中央门东站->龙蟠路南京站西站->龙蟠路南京
   站东站->新庄广场南站->长途东站站->樱铁村站->经五立交站->墨香路站
3
4  转车次数最少：
5  干休所站(1路)->河路道站(1路)->中央门北站(1路)->中央门南站(1路)
6  转
7  中央门南站(22路)->中央门东站(22路)->龙蟠路南京站西站(22路)->龙蟠路南京站
   东站(22路)->新庄广场西站(22路)->新庄广场北站(22路)->曹后村站(22路)->
   江南公交一公司站(22路)->红山森林动物园站(22路)->十字街站(22路)->省中
   西医结合医院站(22路)->红山路
8  迈皋桥站(22路)->华电路站(22路)->长营村站(22路)->北苑新村站(22路)->月苑小
   区站(22路)->月苑南路站(22路)->墨香路站(22路)

```

代码 9: 测试用例 1

起点：干休所站

终点：国际青年文化中心站

```

1  经过站点最少：24站

```

2	干休所站->河路道站->中央门北站->中央门南站->许府巷站->玄武湖公园站->中央路鼓楼
3	站->中山路珠江路北站->新街口南站->三元巷站->中山南路升州路站->中华路瞻园路站->
4	钓鱼台站->窑湾街站->雨花西路站->能仁里站->龙福山庄站->秋叶村站->梦都大街庐山路
5	站->庐山路牡丹江街站->庐山路奥体大街站->庐山路中央公园站->庐山路嘉陵江东街站->
6	江东中路江山大街站->国际青年文化中心站
7	
8	转车次数最少：
9	干休所站(25路)->河路道站(25路)->中央门北站站(25路)->中央门南站站(25路)->许府巷
10	站(25路)->玄武湖公园站(25路)->中山路鼓楼站(25路)->鼓楼医院站(25路)->中山路珠江
11	路南站(25路)->新街口东站(25路)->大行宫西站(25路)->大行宫东站(25路)->中山东路逸
12	仙桥(毗卢寺)站(25路)->西安门站(25路)->瑞金北村站(25路)->瑞金路站(25路)
13	转
14	瑞金路站(7路)->解放南路站(7路)->大光路站(7路)->通济门站(7路)->建康路大中桥站(7
15	路)->建康路夫子庙站(7路)->升州路三山街站(7路)->评事街站(7路)->水西门站(7路)->莫
16	愁湖公园南门站(7路)->水西门大街大士茶亭站(7路)->茶亭东街站(7路)->江东门纪念馆站
17	(7路)->水西门大街江东门站(7路)->江东万达广场站(7路)->江东中路集庆门大街站(7路)->江东中路应天大街站(7路)->江东中路月安街站(7路)->江东中路兴隆大街站(7路)->兴隆
18	大街燕山路站(7路)->清竹园南门站(7路)->兴隆大街华山路站(7路)->乐山路梦都大街站(7路)->金陵图书馆站(7路)->奥体中心西门站(7路)->乐山路富春江西街站(7路)->乐山路楠
19	溪江西街站(7路)->乐山路河西大街站(7路)->乐山路白龙江西街站(7路)->乐山路仁恒江湾
20	城站(7路)->乐山路金沙江西街站(7路)->青年文化中心东门站(7路)->国际青年文化中心站
21	(7路)

代码 10: 测试用例 2

(五) 时间复杂度

1) Dijkstra 算法时间复杂度:

$$O(V^2)$$

2) 广度优先搜索时间复杂度:

$$O(V)$$

(六) 改进方法

由于节点太多，转乘 3 次以上会导致广度优先搜索是速度变慢，可以使用减少不必要的步骤，来减少时间开销。

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后，数列就变成一个有序序列。

（三）源程序

代码 11: Sort.cpp

```
1 #include <stdio.h>
2 #include <iostream>
3 #include <fstream>
4 #include <time.h>
5 using namespace std;
```

```

6  int num = 50000, n, a = num;
7  int main(void) {
8      srand((unsigned)time(NULL));
9      ofstream out("1Sample");
10     while (num)
11         out << a - num-- << endl;
12     out.close();
13
14     out.open("2Sample");
15     num = a;
16     while (num-- > 0) out << num << endl;
17     out.close();
18
19     num = 10;
20     for (int i = 3; i <= num; i++){
21         out.open(to_string(i) + "Sample");
22         n = a;
23         while (n-- > 0)
24             out << rand() % a << endl;
25         out.close();
26     }
27     return 0;
28 }

```

代码 12: NumGeneration.cpp

```

1  CC = g++
2  SOURCE := NumGeneration.cpp Sort.cpp
3
4  build:
5      @$(foreach var,$(SOURCE),\
6          $(CC) -c $(var); \
7          $(CC) $(subst .cpp,.o,$(var)) -o $(subst .cpp,.exe,$(var)); \
8          ./$(subst .cpp,.exe,$(var));\
9      )
10
11 clean:
12     @rm -f *.o *.exe *Sample

```

代码 13: 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 14: 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

总代码行数为：1000.

题目	行数
区块链	283
迷宫问题	388
JSON 查找	×

(二) 心得体会