南京航空航天大学

上 机 实 验 报 告

课　　程	数据结构
班　　级	1819001
学　　号	161940233
姓　　名	颜 宇 明
指导教师	秦 小 麟

目录

一、 Lab1

（一）数据结构

链表，顺序表

（二）算法设计思想

主要描述插入操作与删除操作的思路

（三）源程序

```
#include <malloc.h>
#include <stdio.h>
class ADT_list{
public:
    typedef struct node
    {
        int value;
        struct node *next;
    } node;

    node *head;
    int length = -1;

    void InitLIst(){
        node *tmp = (node *)malloc(sizeof(node));
        length = 0;
        head = tmp;
        tmp->next = NULL;
    }
    void DestoryList(){
        node *p, *temp;
        p = head;
        while (p != NULL){
            temp = p;
            p = p->next;
            free(temp);
        }
        length = -1;
    }
```

```
30      void ClearList(){
31          node *p, *temp;
32          p = head->next;
33          while (p != NULL){
34              temp = p;
35              p = p->next;
36              free(temp);
37          }
38          length = 0;
39      }
40      bool ListEmpty(){
41          if (length < 1) return true;
42          return false;
43      }
44      int ListLength(){
45          return length;
46      }
47      int GetElem(int index){
48          node *p;
49          int i = 0;
50          p = head->next;
51          while (p != NULL){
52              if (index == ++i)
53                  return p->value;
54              p = p->next;
55          }
56          return 0;
57      }
58      int LocateElem(int num){
59          node *p;
60          p = head->next;
61          int index = 0;
62          while (p != NULL){
63              if (p->value == num)
64                  return index;
65              p = p->next;
66              index++;
67          }
68          return -1;
69      }
70      int 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->value == cur_num)
77                  return temp->value;
78          }
79          return 0;
80      }
81      int NextElem(int cur_num){
82          node *p, *temp;
83          p = head->next;
84          while (p != NULL){
85              temp = p;
86              p = p->next;
87              if (p != NULL && temp->value == cur_num)
88                  return p->value;
89          }
90          return NULL;
91      }
92      void ListTraverse(){
93          node *p;
94          p = head->next;
95          while (p != NULL){
96              printf("%d ", p->value);
97              p = p->next;
98          }
99          printf("\n");
100     }
101     int SetElem(int index, int num){
102         node *p;
103         int i = 0;
104         p = head->next;
105         while (p != NULL){
106             if (index != i++) {
107                 p = p->next;
108                 continue;
109             }
110             int old = p->value;
111             p->value = num;
112             return old;
113         }
```

```
        return 0;
    }
    void InsertElem(int index, int num){
        node *p, *temp;
        int i = 0;
        p = head;
        length++;
        while (p != NULL){
            if (index != ++i) {
                temp = p;
                p = p->next;
                continue;
            }
            temp = p->next;
            node *tmp = (node *)malloc(sizeof(node));
            p->next = tmp;
            tmp->next = temp;
            tmp->value = num;
            return;
        }
        node *tmp = (node *)malloc(sizeof(node));
        temp->next = tmp;
        tmp->value = num;
        tmp->next = NULL;
    }
    void DeleteElem(int index){
        node *p, *temp;
        int i = 0;
        temp = head;
        p = head->next;
        length--;
        while (p != NULL){
            if (index != ++i) {
                temp = p;
                p = p->next;
                continue;
            }
            temp->next = p->next;
            free(p);
        }
    }
} ADTlist;
```

```
156
157 int main(){
158     ADTlist.InitLIst();
159     ADTlist.InsertElem(1, 1);
160     ADTlist.InsertElem(2, 1);
161     ADTlist.InsertElem(3, 3);
162     ADTlist.InsertElem(4, 3);
163     ADTlist.InsertElem(5, 3);
164     ADTlist.InsertElem(6, 4);
165     ADTlist.InsertElem(7, 5);
166     ADTlist.ListTraverse();
167     ADTlist.SetElem(2, 7);
168     printf("%d\n", ADTlist.NextElem(1));
169     ADTlist.ListTraverse();
170     ADTlist.DestoryList();
171 }
```

代码 1: 1Linked_list.cpp

```
1  #include <stdio.h>
2  class ADT_list{
3  public:
4      int list[999];
5      int rear = -1;
6      void InitLIst(){
7          rear++;
8      }
9      void DestoryList(){
10         rear = -1;
11     }
12     void ClearList(){
13         for(int i = 0; i <= rear; i++){
14             list[i] = 0;
15         }
16     }
17     bool ListEmpty(){
18         if (rear == 0 && list[rear] == 0){
19             return true;
20         }
21         return false;
22     }
23     int ListLength(){
```

```
24          return rear;
25      }
26      int GetElem(int num){
27          return *(list + num);
28      }
29      int LocateElem(int num){
30          for (int i = 0; i < rear; i++){
31              if (list[i] == num){
32                  return i;
33              }
34          }
35          return -1;
36      }
37      int PriorElem(int cur_num){
38          int pre;
39          for (int i = 0; i < rear; i++){
40              if (list[i] == cur_num){
41                  return i - 1;
42              }
43          }
44          return -1;
45      }
46      int NextElem(int cur_num){
47          for (int i = 0; i < rear; i++){
48              if (list[i] == cur_num && i != rear){
49                  return i + 1;
50              }
51          }
52          return -1;
53      }
54      void ListTraverse(){
55          for (int i = 0; i < rear; i++){
56              printf("%d ", list[i]);
57          }
58          printf("\n");
59      }
60      int SetElem(int index, int num){
61          list[index] = num;
62          return num = list[index];
63      }
64      void InsertElem(int index, int num){
65          rear++;
```

```
66          for (int i = rear; i > index - 1; i--){
67              list[i] = list[i - 1];
68          }
69          list[index - 1] = num;
70      }
71      void DeleteElem(int index){
72          for (int i = index - 1; i < rear; i++){
73              list[i] = list[i + 1];
74          }
75          rear--;
76      }
77  } ADTlist;
```

代码 2: 1Sequence_list.cpp

```
1   #include<stdio.h>
2   #include <malloc.h>
3
4   class ADT_list{
5   public:
6       typedef struct node
7       {
8           int value;
9           struct node *next;
10      } node;
11
12      node *head;
13      int length = -1;
14
15      void InitLIst(){
16          node *tmp = (node *)malloc(sizeof(node));
17          length = 0;
18          head = tmp;
19          tmp->next = NULL;
20      }
21      void DestoryList(){
22          node *p, *temp;
23          p = head;
24          while (p != NULL){
25              temp = p;
26              p = p->next;
27              free(temp);
```

```
28          }
29          length = -1;
30      }
31      void ClearList(){
32          node *p, *temp;
33          p = head->next;
34          while (p != NULL){
35              temp = p;
36              p = p->next;
37              free(temp);
38          }
39          length = 0;
40      }
41      bool ListEmpty(){
42          if (length < 1) return true;
43          return false;
44      }
45      int ListLength(){
46          return length;
47      }
48      int GetElem(int index){
49          node *p;
50          int i = 0;
51          p = head->next;
52          while (p != NULL){
53              if (index == ++i)
54                  return p->value;
55              p = p->next;
56          }
57          return 0;
58      }
59      int LocateElem(int num){
60          node *p;
61          p = head->next;
62          int index = 0;
63          while (p != NULL){
64              if (p->value == num)
65                  return index;
66              p = p->next;
67              index++;
68          }
69          return -1;
```

```
70      }
71      int PriorElem(int cur_num){
72          node *p, *temp;
73          p = head->next;
74          while (p != NULL){
75              temp = p;
76              p = p->next;
77              if (p != NULL && p->value == cur_num)
78                  return temp->value;
79          }
80          return 0;
81      }
82      int 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->value == cur_num)
89                  return p->value;
90          }
91          return 0;
92      }
93      void ListTraverse(){
94          node *p;
95          p = head->next;
96          while (p != NULL){
97              printf("%d ", p->value);
98              p = p->next;
99          }
100         printf("\n");
101     }
102     int SetElem(int index, int num){
103         node *p;
104         int i = 0;
105         p = head->next;
106         while (p != NULL){
107             if (index != i++) {
108                 p = p->next;
109                 continue;
110             }
111             int old = p->value;
```

```
112             p->value = num;
113             return old;
114         }
115         return 0;
116     }
117     void InsertElem(int index, int num){
118         node *p, *temp;
119         int i = 0;
120         p = head;
121         length++;
122         while (p != NULL){
123             if (index != ++i) {
124                 temp = p;
125                 p = p->next;
126                 continue;
127             }
128             temp = p->next;
129             node *tmp = (node *)malloc(sizeof(node));
130             p->next = tmp;
131             tmp->next = temp;
132             tmp->value = num;
133             return;
134         }
135         node *tmp = (node *)malloc(sizeof(node));
136         temp->next = tmp;
137         tmp->value = num;
138         tmp->next = NULL;
139     }
140     void DeleteElem(int index){
141         node *p, *temp;
142         int i = 0;
143         temp = head;
144         p = head->next;
145         length--;
146         while (p != NULL){
147             if (index != ++i) {
148                 temp = p;
149                 p = p->next;
150                 continue;
151             }
152             temp->next = p->next;
153             free(p);
```

```
154             }
155         }
156         void Reverse(){
157             if (length < 1) return;
158             node *p, *temp, *tmp = NULL;
159             temp = p = head->next;
160             while(temp != NULL){
161                 temp = p->next;
162                 p->next = tmp;
163                 tmp = p;
164                 p = temp;
165             }
166             head->next = tmp;
167         }
168     } ADTlist;
169
170     int main(){
171         ADTlist.InitLIst();
172         ADTlist.InsertElem(1, 1);
173         ADTlist.InsertElem(2, 2);
174         ADTlist.InsertElem(3, 3);
175         ADTlist.InsertElem(4, 4);
176         ADTlist.ListTraverse();
177         ADTlist.Reverse();
178         ADTlist.ListTraverse();
179         ADTlist.DestoryList();
180     }
```

代码 3: 2Linked_list_Reverse.cpp

```
1   #include<stdio.h>
2   int list[3] = {1, 2, 3};
3   void Reverse(int list[], int length){
4       length--;
5       for ( int i = 0; i < length; i++){
6           int temp = list[i];
7           list[i] = list[length - i];
8           list[length - i] = temp;
9       }
10  }
11
12  int main(){
```

```
    int length = 3;
    Reverse(list, length);
    for( int i = 0; i < length; i++){
        printf("%d ", list[i]);
    }
}
```

代码 4: 2Sequence_list_Reverse.cpp

```
#include <malloc.h>
#include <stdio.h>
class ADT_list{
public:
    typedef struct node
    {
        int value;
        struct node *next;
    } node;

    node *head;
    int length = -1;

    void InitLIst(){
        node *tmp = (node *)malloc(sizeof(node));
        length = 0;
        head = tmp;
        tmp->next = NULL;
    }
    void DestoryList(){
        node *p, *temp;
        p = head;
        while (p != NULL){
            temp = p;
            p = p->next;
            free(temp);
        }
        length = -1;
    }
    void ClearList(){
        node *p, *temp;
        p = head->next;
        while (p != NULL){
```

```
            temp = p;
            p = p->next;
            free(temp);
        }
        length = 0;
    }
    bool ListEmpty(){
        if (length < 1) return true;
        return false;
    }
    int ListLength(){
        return length;
    }
    int GetElem(int index){
        node *p;
        int i = 0;
        p = head->next;
        while (p != NULL){
            if (index == ++i)
                return p->value;
            p = p->next;
        }
        return 0;
    }
    int LocateElem(int num){
        node *p;
        p = head->next;
        int index = 0;
        while (p != NULL){
            if (p->value == num)
                return index;
            p = p->next;
            index++;
        }
        return -1;
    }
    int PriorElem(int cur_num){
        node *p, *temp;
        p = head->next;
        while (p != NULL){
            temp = p;
            p = p->next;
```

```
76              if (p != NULL && p->value == cur_num)
77                  return temp->value;
78          }
79          return 0;
80      }
81      int NextElem(int cur_num){
82          node *p, *temp;
83          p = head->next;
84          while (p != NULL){
85              temp = p;
86              p = p->next;
87              if (p != NULL && temp->value == cur_num)
88                  return p->value;
89          }
90          return 0;
91      }
92      void ListTraverse(){
93          node *p;
94          p = head->next;
95          while (p != NULL){
96              printf("%d ", p->value);
97              p = p->next;
98          }
99          printf("\n");
100     }
101     int SetElem(int index, int num){
102         node *p;
103         int i = 0;
104         p = head->next;
105         while (p != NULL){
106             if (index != i++) {
107                 p = p->next;
108                 continue;
109             }
110             int old = p->value;
111             p->value = num;
112             return old;
113         }
114         return 0;
115     }
116     void InsertElem(int index, int num){
117         node *p, *temp;
```

```
118         int i = 0;
119         p = head;
120         length++;
121         while (p != NULL){
122             if (index != ++i) {
123                 temp = p;
124                 p = p->next;
125                 continue;
126             }
127             temp = p->next;
128             node *tmp = (node *)malloc(sizeof(node));
129             p->next = tmp;
130             tmp->next = temp;
131             tmp->value = num;
132             return;
133         }
134         node *tmp = (node *)malloc(sizeof(node));
135         temp->next = tmp;
136         tmp->value = num;
137         tmp->next = NULL;
138     }
139     void DeleteElem(int index){
140         node *p, *temp;
141         int i = 0;
142         temp = head;
143         p = head->next;
144         length--;
145         while (p != NULL){
146             if (index != ++i) {
147                 temp = p;
148                 p = p->next;
149                 continue;
150             }
151             temp->next = p->next;
152             free(p);
153         }
154     }
155     void Remove(){
156         node *p = head->next, *temp = head;
157         int use[999] = {0};
158         while (p != NULL){
159             if (use[p->value]) {
```

```
                temp->next = p->next;
                free(p);
                length--;
                p = temp->next;
                continue;
            }
            use[p->value] = 1;
            temp = p;
            p = p->next;
        }
    }
} ADTlist;

int main(){
    ADTlist.InitLIst();
    ADTlist.InsertElem(1, 1);
    ADTlist.InsertElem(2, 1);
    ADTlist.InsertElem(3, 3);
    ADTlist.InsertElem(4, 3);
    ADTlist.InsertElem(5, 3);
    ADTlist.InsertElem(6, 4);
    ADTlist.InsertElem(7, 4);
    ADTlist.ListTraverse();
    ADTlist.Remove();
    ADTlist.ListTraverse();
    ADTlist.DestoryList();
}
```

代码 5: 3Linked_list_Remove.cpp

```
#include <stdio.h>
class ADT_list{
public:
    int list[999];
    int rear = -1;
    void InitLIst(){
        rear++;
    }
    void DestoryList(){
        rear = -1;
    }
    void ClearList(){
```

```
13          for(int i = 0; i <= rear; i++){
14              list[i] = 0;
15          }
16      }
17      bool ListEmpty(){
18          if (rear == 0 && list[rear] == 0){
19              return true;
20          }
21          return false;
22      }
23      int ListLength(){
24          return rear;
25      }
26      int GetElem(int num){
27          return *(list + num);
28      }
29      int LocateElem(int num){
30          for (int i = 0; i < rear; i++){
31              if (list[i] == num){
32                  return i;
33              }
34          }
35          return -1;
36      }
37      int PriorElem(int cur_num){
38          int pre;
39          for (int i = 0; i < rear; i++){
40              if (list[i] == cur_num){
41                  return i - 1;
42              }
43          }
44          return -1;
45      }
46      int NextElem(int cur_num){
47          for (int i = 0; i < rear; i++){
48              if (list[i] == cur_num && i != rear){
49                  return i + 1;
50              }
51          }
52          return -1;
53      }
54      void ListTraverse(){
```

```
55          for (int i = 0; i < rear; i++){
56              printf("%d ", list[i]);
57          }
58          printf("\n");
59      }
60      int SetElem(int index, int num){
61          list[index] = num;
62          return num = list[index];
63      }
64      void InsertElem(int index, int num){
65          rear++;
66          for (int i = rear; i > index - 1; i--){
67              list[i] = list[i - 1];
68          }
69          list[index - 1] = num;
70      }
71      void DeleteElem(int index){
72          for (int i = index - 1; i < rear; i++){
73              list[i] = list[i + 1];
74          }
75          rear--;
76      }
77      void Remove(){
78          int use[999];
79          for (int i = 0; i < rear; i++){
80              if (use[list[i]]){
81                  DeleteElem(i + 1);
82                  i--;
83                  continue;
84              }
85              use[list[i]] = 1;
86              i++;
87          }
88      }
89  } ADTlist;
90
91  int main(){
92      ADTlist.InitLIst();
93      ADTlist.InsertElem(1, 1);
94      ADTlist.InsertElem(1, 1);
95      ADTlist.InsertElem(2, 2);
96      ADTlist.InsertElem(2, 2);
```

```
97      ADTlist.InsertElem(3, 3);
98      ADTlist.InsertElem(3, 3);
99      ADTlist.InsertElem(3, 3);
100     ADTlist.InsertElem(4, 4);
101     ADTlist.InsertElem(4, 4);
102     ADTlist.InsertElem(4, 4);
103     ADTlist.ListTraverse();
104     ADTlist.Remove();
105     ADTlist.ListTraverse();
106     ADTlist.DestoryList();
107 }
```

代码 6: 3Sequence_list_Remove.cpp

```
1   #include <stdio.h>
2   int n, m, l, a[255555], num, h[333];
3   int main(){
4       freopen("4CSP1", "r", stdin);
5       scanf("%d%d%d", &n, &m, &l);
6       while (scanf("%d", &a[num]) != EOF) num++;
7       for (int i = 0; i < n; i++)
8       for (int j = 0; j < m; j++)
9           h[a[i * m + j]]++;
10      for (int i = 0; i < l; i++)
11          printf("%d ", h[i]);
12  }
```

代码 7: 4CSP.cpp

```
1   #include <stdio.h>
2   #include <algorithm>
3   int n, l, t, a[999], num, speed[999], use[999];
4   int main(){
5       freopen("5CSP1", "r", stdin);
6       scanf("%d%d%d", &n, &l, &t);
7       while (scanf("%d", &a[num]) != EOF) num++;
8       std::fill(speed, speed + 999, 1);
9       while(t--){
10          std::fill(use, use + 999, -1);
11          for (int j = 0; j < n; j++){
12              a[j] += speed[j];
13              if (a[j] == l || a[j] == 0) speed[j] *= -1;
14              if(use[a[j]] >= 0) speed[j] *= -1, speed[use[a[j]]] *= -1;
```

```
            else use[a[j]] = j;
        }
    }
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
}
```

代码 8: 5CSP.cpp

（四）测试数据及其结果

```
1 1 3 3 3 4 5
1
1 1 7 3 3 4 5
```

代码 9: 1Linked_list.cpp

```
1 2 3 4
4 3 2 1
```

代码 10: 2Linked_list_Reverse.cpp

```
3 2 1
```

代码 11: 2Sequence_list_Reverse.cpp

```
1 1 3 3 3 4 4
1 3 4
```

代码 12: 3Linked_list_Remove.cpp

```
1 2 3 4 4 4 3 3 2 1
1 2 3 4
```

代码 13: 3Sequence_list_Remove.cpp

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

代码 14: 4CSP.cpp

```
7 9 9
```

代码 15: 5CSP.cpp

（五）时间复杂度

数组实现

1) 插入元素:

$$O(n)$$

2) 删除元素：

$$O(n)$$

链表实现

1) 插入元素:

$$O(1)$$

2) 删除元素：

$$O(n)$$

（六）改进方法

链表的基本操作，不需要改进。

二、 Lab2

（一）数据结构

链表，顺序表

（二）算法设计思想

主要描述插入操作与删除操作的思路

（三）源程序

```
#include "Linked_list.h"
ADT_list ADTlist;
int main(){
    ADTlist.InitLIst();
    ADTlist.InsertElem(1, 3);
```

```
    ADTlist.InsertElem(2, 5);
    ADTlist.InsertElem(3, 1);
    ADTlist.InsertElem(4, 2);
    ADTlist.InsertElem(5, 6);
    ADTlist.InsertElem(6, 7);
    ADTlist.InsertElem(7, 0);
    ADTlist.ListTraverse();
    ADTlist.Select_sort();
    ADTlist.ListTraverse();
    ADTlist.DestoryList();
}
```

代码 16: 1Linked_list_sort.cpp

```
#include "Sequence_list.h"
ADT_list ADTlist;
int main(){
    ADTlist.InitLIst();
    ADTlist.InsertElem(1, 3);
    ADTlist.InsertElem(2, 5);
    ADTlist.InsertElem(3, 1);
    ADTlist.InsertElem(4, 2);
    ADTlist.InsertElem(5, 6);
    ADTlist.InsertElem(6, 7);
    ADTlist.InsertElem(7, 0);
    ADTlist.ListTraverse();
    ADTlist.Select_sort();
    ADTlist.ListTraverse();
    ADTlist.DestoryList();
}
```

代码 17: 1Sequence_list_sort.cpp

```
#include "Linked_list.h"
ADT_list A, B, C;
int main(){
    A.InitLIst();
    A.InsertElem(1, 7);
    A.InsertElem(2, 6);
    A.InsertElem(3, 5);
    A.InsertElem(4, 4);
    A.InsertElem(5, 3);
    A.InsertElem(6, 2);
```

```
    A.InsertElem(7, 1);
    A.ListTraverse();

    B.InitLIst();
    B.InsertElem(1, 9);
    B.InsertElem(2, 8);
    B.InsertElem(3, 7);
    B.InsertElem(4, 6);
    B.InsertElem(5, 5);
    B.InsertElem(6, 4);
    B.InsertElem(7, 3);
    B.ListTraverse();

    C = A.Union(B);
    C.ListTraverse();

    A.DestoryList();
    B.DestoryList();
    C.DestoryList();
}
```

代码 18: 2Linked_list_union.cpp

```
#include "Linked_list.h"
#include <stdio.h>
ADT_list A, B, C;
int main(){
    A.InitLIst();
    A.InsertElem(1, 1);
    A.InsertElem(2, 2);
    A.InsertElem(3, 3);
    A.InsertElem(4, 4);
    A.InsertElem(5, 5);
    A.InsertElem(6, 6);
    A.InsertElem(7, 7);
    A.Josephus(3);
}
```

代码 19: 3Linked_list_Josephus.cpp

```
#include "Sequence_list.h"
ADT_list A;
int main(){
```

```
    A.InitLIst();
    A.InsertElem(1, 1);
    A.InsertElem(2, 2);
    A.InsertElem(3, 3);
    A.InsertElem(4, 4);
    A.InsertElem(5, 5);
    A.InsertElem(6, 6);
    A.InsertElem(7, 7);
    A.Josephus(2);
}
```

代码 20: 3Sequence_list_Josephus.cpp

```
#include <stdio.h>
int n, a[1111], num, maxx = 0, ans = 0;
int count(int a[]){
    int flag = 0, num = a[0] == 0 ? -1 : 0;
    for(int i = 0; i < n; i++){
        if (a[i] == 0 && flag == 0) flag = 1;
        else if (flag == 1 && a[i]) flag = 0, num++;
    }
    return num + 1;
}
int main(){
    freopen("4CSP1", "r", stdin);
    scanf("%d", &n);
    while (scanf("%d", &a[num++]) != EOF);
    for (int i = 0; i < n; i++) maxx = a[i] > maxx ? a[i] : maxx;
    for (int i = 0; i < maxx + 2; i++){
        for (int j= 0; j < n; j++)
            a[j] = a[j] < i ? 0 : a[j];
        ans = count(a) > ans ? count(a) : ans;
    }
    printf("%d", ans);
}
```

代码 21: 4CSP.cpp

```
#include <stdio.h>
int n, a[2222], b[2222], c[2222], d[2222], num = -1, numm = -1, maxx =
    0, sum, time[1111111] = {0};
int main(){
    freopen("5CSP1", "r", stdin);
```

```
    scanf("%d", &n);
    while (++num != n) scanf("%d%d", &a[num], &b[num]);
    while (++numm != n) scanf("%d%d", &c[numm], &d[numm]);
    for (int i = 0; i < n; i++) {
        for (int j = a[i] + 1; j <= b[i]; j++) time[j]++, maxx = j >
            maxx ? j : maxx;
        for (int j = c[i] + 1; j <= d[i]; j++) time[j]++, maxx = j >
            maxx ? j : maxx;
    }
    for (int i = 0; i <= maxx; i++) sum = time[i] == 2 ? sum + 1 : sum;
    printf("%d", sum);
}
```

代码 22: 5CSP.cpp

```
#include <malloc.h>
#include <stdio.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;
```

```
28          p = p->next;
29          free(temp);
30      }
31      length = 0;
32  }
33
34  bool ADT_list::ListEmpty(){
35      if (length < 1) return true;
36      return false;
37  }
38
39  int ADT_list::ListLength(){
40      return length;
41  }
42
43  int ADT_list::GetElem(int index){
44      node *p;
45      int i = 0;
46      p = head->next;
47      while (p != NULL){
48          if (index == ++i)
49              return p->value;
50          p = p->next;
51      }
52      return 0;
53  }
54
55  int ADT_list::LocateElem(int num){
56      node *p;
57      p = head->next;
58      int index = 0;
59      while (p != NULL){
60          if (p->value == num)
61              return index;
62          p = p->next;
63          index++;
64      }
65      return -1;
66  }
67
68  int ADT_list::PriorElem(int cur_num){
69      node *p, *temp;
```

```
70      p = head->next;
71      while (p != NULL){
72          temp = p;
73          p = p->next;
74          if (p != NULL && p->value == cur_num)
75              return temp->value;
76      }
77      return 0;
78  }
79
80  int ADT_list::NextElem(int cur_num){
81      node *p, *temp;
82      p = head->next;
83      while (p != NULL){
84          temp = p;
85          p = p->next;
86          if (p != NULL && temp->value == cur_num)
87              return p->value;
88      }
89      return 0;
90  }
91
92  void ADT_list::ListTraverse(){
93      node *p;
94      p = head->next;
95      while (p != NULL){
96          printf("%d ", p->value);
97          p = p->next;
98      }
99      printf("\n");
100 }
101
102 int ADT_list::SetElem(int index, int num){
103     node *p;
104     int i = 0;
105     p = head->next;
106     while (p != NULL){
107         if (index != i++) {
108             p = p->next;
109             continue;
110         }
111         int old = p->value;
```

```
112         p->value = num;
113         return old;
114     }
115     return 0;
116 }
117 
118 void ADT_list::InsertElem(int index, int num){
119     node *p, *temp;
120     int i = 0;
121     p = head;
122     length++;
123     while (p != NULL){
124         if (index != ++i) {
125             temp = p;
126             p = p->next;
127             continue;
128         }
129         temp = p->next;
130         node *tmp = (node *)malloc(sizeof(node));
131         p->next = tmp;
132         tmp->next = temp;
133         tmp->value = num;
134         return;
135     }
136     node *tmp = (node *)malloc(sizeof(node));
137     temp->next = tmp;
138     tmp->value = num;
139     tmp->next = NULL;
140 }
141 
142 void ADT_list::DeleteElem(int index){
143     node *p, *temp;
144     int i = 0;
145     temp = head;
146     p = head->next;
147     length--;
148     while (p != NULL){
149         if (index != ++i) {
150             temp = p;
151             p = p->next;
152             continue;
153         }
```

```
154         temp->next = p->next;
155         free(p);
156     }
157 }
158 
159 void ADT_list::Remove(){
160     node *p = head->next, *temp = head;
161     int use[999] = {0};
162     while (p != NULL){
163         if (use[p->value]) {
164             temp->next = p->next;
165             free(p);
166             length--;
167             p = temp->next;
168             continue;
169         }
170         use[p->value] = 1;
171         temp = p;
172         p = p->next;
173     }
174 }
175 
176 void ADT_list::Reverse(){
177     if (length < 1) return;
178     node *p, *temp, *tmp = NULL;
179     temp = p = head->next;
180     while(temp != NULL){
181         temp = p->next;
182         p->next = tmp;
183         tmp = p;
184         p = temp;
185     }
186     head->next = tmp;
187 }
188 
189 void ADT_list::Bubble_Sort(){
190     for (int i = 1; i <= length ; i++){
191         node *p = head, *temp = p->next, *tmp = temp->next;
192         int times = length - i;
193         while (times--){
194             if (temp->value > tmp->value){
195                 p->next = temp->next;
```

```
                temp->next = tmp->next;
                tmp->next = temp;
            }
            p = p->next;
            temp = p->next;
            tmp = temp->next;
        }
    }
}

void ADT_list::Select_sort(){
    node *p = head->next, *temp = p->next;
    for (int i = 0; i < length - 1; i++){
        int minn = 0x3f3f3f3f;
        while(temp){
            minn = temp->value < minn ? temp->value : minn;
            temp = temp->next;
        }
        SetElem(LocateElem(minn), p->value);
        SetElem(i, minn);
        p = p->next;
        temp = p->next;
    }
}

ADT_list ADT_list::Union(ADT_list B){
    ADT_list C;
    C.InitLIst();
    int index = 0;
    node *p = head->next;
    while(p){
        if (B.LocateElem(p->value) >= 0){
            C.InsertElem(++index, p->value);
        }
        p = p->next;
    }
    // ListTraverse();
    // B.ListTraverse();
    return C;
}

void ADT_list::Josephus(int m){
```

```
238     node *p = head->next, *last = p, *tmp = head;
239     while(last->next)
240         last = last->next;
241     last->next = head->next;
242     while(length--){
243         int temp = m;
244         while(--temp){
245             p = p->next;
246             tmp = tmp->next;
247         }
248         printf("%d ", p->value);
249         tmp->next = p->next;
250         last = p;
251         p = p->next;
252         free(last);
253     }
254 }
```

代码 23: Linked_list.cpp

```
1  #ifndef LINKED_LIST_H
2  #define LINKED_LIST_H
3  class ADT_list{
4  public:
5      typedef struct node
6      {
7          int value;
8          struct node *next;
9      } node;
10     node *head;
11     int length = -1;
12
13     void InitLIst();
14     void DestoryList();
15     void ClearList();
16     bool ListEmpty();
17     int ListLength();
18     int GetElem(int);
19     int LocateElem(int);
20     int PriorElem(int);
21     int NextElem(int);
22     void ListTraverse();
```

```
    int SetElem(int index, int num);
    void InsertElem(int index, int num);
    void DeleteElem(int index);
    void Remove();
    void Reverse();
    void Bubble_Sort();
    void Select_sort();
    ADT_list Union(ADT_list B);
    void Josephus(int);
};
#endif
```

代码 24: Linked_list.h

```
#include <stdio.h>
#include "Sequence_list.h"

void ADT_list::InitLIst(){
    rear++;
}
void ADT_list::DestoryList(){
    rear = -1;
}
void ADT_list::ClearList(){
    for(int i = 0; i < rear; i++){
        list[i] = 0;
    }
}
bool ADT_list::ListEmpty(){
    if (rear == 0 && list[rear] == 0){
        return true;
    }
    return false;
}
int ADT_list::ListLength(){
    return rear;
}
int ADT_list::GetElem(int num){
    return *(list + num);
}
int ADT_list::LocateElem(int num){
    for (int i = 0; i < rear; i++){
```

```
29          if (list[i] == num){
30              return i;
31          }
32      }
33      return -1;
34  }
35  int ADT_list::PriorElem(int cur_num){
36      int pre;
37      for (int i = 0; i < rear; i++){
38          if (list[i] == cur_num){
39              return i - 1;
40          }
41      }
42      return -1;
43  }
44  int ADT_list::NextElem(int cur_num){
45      for (int i = 0; i < rear; i++){
46          if (list[i] == cur_num && i != rear){
47              return i + 1;
48          }
49      }
50      return -1;
51  }
52  void ADT_list::ListTraverse(){
53      for (int i = 0; i < rear; i++){
54          printf("%d ", list[i]);
55      }
56      printf("\n");
57  }
58  int ADT_list::SetElem(int index, int num){
59      list[index] = num;
60      return num = list[index];
61  }
62  void ADT_list::InsertElem(int index, int num){
63      rear++;
64      for (int i = rear; i > index - 1; i--){
65          list[i] = list[i - 1];
66      }
67      list[index - 1] = num;
68  }
69  void ADT_list::DeleteElem(int index){
70      for (int i = index - 1; i < rear; i++){
```

```
        list[i] = list[i + 1];
    }
    rear--;
}
void ADT_list::Remove(){
    int use[999];
    for (int i = 0; i < rear; i++){
        if (use[list[i]]){
            DeleteElem(i + 1);
            i--;
            continue;
        }
        use[list[i]] = 1;
        i++;
    }
}
void ADT_list::Bubble_Sort(){
    for(int i = 0; i < rear; i++)
    for(int j = 0; j < rear - i - 1; j++)
        if (list[j] > list[j + 1]){
            int temp = list[j];
            list[j] = list[j + 1];
            list[j + 1] = temp;
        }
}
void ADT_list::Select_sort(){
    for(int i = 0; i < rear; i++){
        int minn = 0x3f3f3f3f;
        for (int j = i; j < rear; j++){
            minn = list[j] < minn ? list[j] : minn;
        }
        SetElem(LocateElem(minn), list[i]);
        SetElem(i, minn);
    }
}
ADT_list ADT_list::Union(ADT_list B){
    ADT_list C;
    C.InitLIst();
    int index = 0;
    for (int i = 0; i < rear; i++){
        if (B.LocateElem(list[i]) >= 0){
            C.InsertElem(++index, list[i]);
```

```
        }
    }
    return C;
}

void ADT_list::Josephus(int m){
    int index = 0;
    while(rear){
        index = (index + m - 1) % rear;
        printf("%d ", list[index]);
        DeleteElem(index + 1);
    }
}
```

代码 25: Sequence_list.cpp

```
#ifndef SEQUENCE_LIST_H
#define SEQUENCE_LIST_H
class ADT_list{
public:
    int list[999];
    int rear = -1;
    void InitLIst();
    void DestoryList();
    void ClearList();
    bool ListEmpty();
    int ListLength();
    int GetElem(int num);
    int LocateElem(int num);
    int PriorElem(int cur_num);
    int NextElem(int cur_num);
    void ListTraverse();
    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_list Union(ADT_list);
    void Josephus(int);
};
#endif
```

代码 26: Sequence_list.h

（四）测试数据及其结果

```
3 5 1 2 6 7 0
0 1 2 3 5 6 7
```

代码 27: 1Linked_list_sort.cpp

```
3 5 1 2 6 7 0
0 1 2 3 5 6 7
```

代码 28: 1Sequence_list_sort.cpp

```
7 6 5 4 3 2 0
9 8 7 6 3 2 1
7 6 3 2
```

代码 29: 2Sequence_list_union.cpp

```
3 6 2 7 5 1 4
```

代码 30: 3Linked_list_Josephus.cpp

```
2 4 6 1 5 3 7
```

代码 31: 3Sequence_list_Josephus.cpp

```
5
```

代码 32: 4CSP.cpp

```
3
```

代码 33: 5CSP.cpp

（五）时间复杂度

数组实现

1) 插入元素:

$$O(n)$$

2) 删除元素：

$$O(n)$$

链表实现

1) 插入元素:

$$O(1)$$

2) 删除元素：

$$O(n)$$

（六）改进方法

可以尝试用一行代码解决约瑟夫问题。

三、 Lab3

（一）数据结构

链表，顺序表

（二）算法设计思想

主要描述插入操作与删除操作的思路。

按行摆放，在确定一个皇后应该摆的列时，需要检查当前列是否合法，如果合法，则将皇后放置在当前位置，并进行递归，回溯。每行都摆满皇后时，则产生了一种解法，将所有解法收集并返回。

合法性判断方法：当前将要摆放皇后的位置和其他已摆放皇后的位置不能在同一列，且不能在同一条斜线上。这里判断是否在同一条斜线上可以通过两个皇后的位置横坐标之差和纵坐标之差的绝对值是否相等来判断。

（三）源程序

```
1 #include <stdio.h>
2 #include "Sequence_Stack.h"
```

```
ADT_Stack solution;
int N = 8;
bool conflict(Pos point) {
    for (int i = 1; i <= solution.StackLength(); i++)
      if (point.y == solution[i].y || (point.x + point.y) == (solution[i
          ].x + solution[i].y) || (point.x - point.y) == (solution[i].x
          - solution[i].y) || point.x >= N || point.y >= N)
         return true;
    return false;
}
void find(Pos pos) {
   if (solution.StackLength() >= N || (pos.y >= N && pos.x == N)){
      for (int i = 1; i <= solution.StackLength(); i++)
         printf("%d ", solution[i].y);
      printf("\n");
      return;
   }
   if(!conflict(pos)){
      solution.Push(pos);
      if (pos.x < N){
         pos.x++;
         pos.y = 0;
      }
   }
   else {
      if (pos.y >= N)
         pos = solution.Pop();
      pos.y++;
   }
   find(pos);
}
int main(){
   Pos pos;
   solution.InitStack();
   pos.x = 0, pos.y = 0;
   find(pos);
}
```

代码 34: 3Eight_Queens.cpp

```
#include <stdio.h>
#include <algorithm>
```

```cpp
using namespace std;
int m, num = 1, maxx = 0, ans, onenum, indexx, error, temp;
typedef struct Test
{
       int y;
       int result;
       int state;
}person;
person list[111111];
bool Compare(const person &a, const person &b)
{
    return a.y < b.y ? true : (a.y == b.y ? a.result < b.result : false)
        ;
}
int main(){
    freopen("4CSP2", "r", stdin);
    scanf("%d", &m);
    while (scanf("%d%d", &list[num].y, &list[num].result) != EOF) num++;
    sort(list + 1, list + m + 1, Compare);
    while (++indexx != num)  list[indexx].state = list[indexx - 1].state
         + list[indexx].result, onenum += list[indexx].result;
    for(int i = 1; i <= m; i++){
        error = list[i].state - list[i].result + m - i + 1 - (onenum - (
            list[i].state - list[i].result)) + (temp = list[i].y == list
            [i - 1].y && i != 1 ? 0x3f3f : 0);
        maxx <= m - error ? maxx = m - error, ans = list[i].y : maxx =
            maxx;
    }
    printf("%d", ans);
}
```

代码 35: 4CSP.cpp

```cpp
#include <stdio.h>
#include "5Linked_list.h"
int n, m, num = 0, maxx = 0, ans = 0, onenum, indexx, error, temp, x1,
    x2, y1, y2, win[3333][2222];
ADT_list list;
ADT_list::node *point;
typedef struct Click
{
    int x, y;
```

```
}ClickList;
ClickList Clicks[11];
int main(){
    list.InitLIst();
    freopen("5CSP1", "r", stdin);
    scanf("%d%d", &n, &m);
    while (num++ != n){
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2), list.InsertElem(1, x1, y1
            , x2, y2);
        for (int i = x1; i <= x2; i++)
            for (int j = y1; j <= y2; j++)
                win[i][j] = num;
    }
    num = 1;
    while (scanf("%d%d", &Clicks[num].x, &Clicks[num].y) != EOF) {
        int index = win[Clicks[num].x][Clicks[num].y];
        if (index) {
            printf("%d\n", index);
            point = list.GetElem(index);
            for (int i = point->x1; i <= point->x2; i++)
                for (int j = point->y1; j <= point->y2; j++)
                    win[i][j] = index;
        }
        else printf("IGNORED\n");
        num++;
    }
}
```

代码 36: 5CSP.cpp

```
#include <malloc.h>
#include <stdio.h>
#include "5Linked_list.h"

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

void ADT_list::DestoryList(){
```

```
13      node *p, *temp;
14      p = head;
15      while (p != NULL){
16          temp = p;
17          p = p->next;
18          free(temp);
19      }
20      length = -1;
21  }
22
23  void ADT_list::ClearList(){
24      node *p, *temp;
25      p = head->next;
26      while (p != NULL){
27          temp = p;
28          p = p->next;
29          free(temp);
30      }
31      length = 0;
32  }
33
34  bool ADT_list::ListEmpty(){
35      if (length < 1) return true;
36      return false;
37  }
38
39  int ADT_list::ListLength(){
40      return length;
41  }
42
43  ADT_list::node* ADT_list::GetElem(int index){
44      node *p;
45      int i = 0;
46      p = head->next;
47      while (p != NULL){
48          if (index == ++i)
49              return p;
50          p = p->next;
51      }
52      return NULL;
53  }
54
```

```
55 // int ADT_list::LocateElem(int num){
56 //     node *p;
57 //     p = head->next;
58 //     int index = 0;
59 //     while (p != NULL){
60 //         if (p->value == num)
61 //             return index;
62 //         p = p->next;
63 //         index++;
64 //     }
65 //     return -1;
66 // }
67
68 // int ADT_list::PriorElem(int cur_num){
69 //     node *p, *temp;
70 //     p = head->next;
71 //     while (p != NULL){
72 //         temp = p;
73 //         p = p->next;
74 //         if (p != NULL && p->value == cur_num)
75 //             return temp->value;
76 //     }
77 //     return 0;
78 // }
79
80 // int ADT_list::NextElem(int cur_num){
81 //     node *p, *temp;
82 //     p = head->next;
83 //     while (p != NULL){
84 //         temp = p;
85 //         p = p->next;
86 //         if (p != NULL && temp->value == cur_num)
87 //             return p->value;
88 //     }
89 //     return 0;
90 // }
91
92 void ADT_list::ListTraverse(){
93     node *p;
94     p = head->next;
95     while (p != NULL){
96         printf("%d %d %d %d\n", p->x1, p->y1, p->x2, p->y2);
```

```
97          p = p->next;
98      }
99      printf("\n");
100 }
101 
102 // int ADT_list::SetElem(int index, int num){
103 //     node *p;
104 //     int i = 0;
105 //     p = head->next;
106 //     while (p != NULL){
107 //         if (index != i++) {
108 //             p = p->next;
109 //             continue;
110 //         }
111 //         int old = p->value;
112 //         p->value = num;
113 //         return old;
114 //     }
115 //     return 0;
116 // }
117 
118 void ADT_list::InsertElem(int index, int x1, int y1, int x2, int y2){
119     node *p, *temp;
120     int i = 0;
121     p = head;
122     length++;
123     while (p != NULL){
124         if (index != ++i) {
125             temp = p;
126             p = p->next;
127             continue;
128         }
129         temp = p->next;
130         node *tmp = (node *)malloc(sizeof(node));
131         p->next = tmp;
132         tmp->next = temp;
133         tmp->x1 = x1;
134         tmp->y1 = y1;
135         tmp->x2 = x2;
136         tmp->y2 = y2;
137         return;
138     }
```

```
139     node *tmp = (node *)malloc(sizeof(node));
140     temp->next = tmp;
141     tmp->x1 = x1;
142     tmp->y1 = y1;
143     tmp->x2 = x2;
144     tmp->y2 = y2;
145     tmp->next = NULL;
146 }
147 
148 void ADT_list::DeleteElem(int index){
149     node *p, *temp;
150     int i = 0;
151     temp = head;
152     p = head->next;
153     length--;
154     while (p != NULL){
155         if (index != ++i) {
156             temp = p;
157             p = p->next;
158             continue;
159         }
160         temp->next = p->next;
161         free(p);
162     }
163 }
164 
165 // void ADT_list::Remove(){
166 //     node *p = head->next, *temp = head;
167 //     int use[999] = {0};
168 //     while (p != NULL){
169 //         if (use[p->value]) {
170 //             temp->next = p->next;
171 //             free(p);
172 //             length--;
173 //             p = temp->next;
174 //             continue;
175 //         }
176 //         use[p->value] = 1;
177 //         temp = p;
178 //         p = p->next;
179 //     }
180 // }
```

```
181 
182 void ADT_list::Reverse(){
183     if (length < 1) return;
184     node *p, *temp, *tmp = NULL;
185     temp = p = head->next;
186     while(temp != NULL){
187         temp = p->next;
188         p->next = tmp;
189         tmp = p;
190         p = temp;
191     }
192     head->next = tmp;
193 }
194 
195 // void ADT_list::Bubble_Sort(){
196 //     for (int i = 1; i <= length ; i++){
197 //         node *p = head, *temp = p->next, *tmp = temp->next;
198 //         int times = length - i;
199 //         while (times--){
200 //             if (temp->value > tmp->value){
201 //                 p->next = temp->next;
202 //                 temp->next = tmp->next;
203 //                 tmp->next = temp;
204 //             }
205 //             p = p->next;
206 //             temp = p->next;
207 //             tmp = temp->next;
208 //         }
209 //     }
210 // }
211 
212 // void ADT_list::Select_sort(){
213 //     node *p = head->next, *temp = p->next;
214 //     for (int i = 0; i < length - 1; i++){
215 //         int minn = 0x3f3f3f3f;
216 //         while(temp){
217 //             minn = temp->value < minn ? temp->value : minn;
218 //             temp = temp->next;
219 //         }
220 //         SetElem(LocateElem(minn), p->value);
221 //         SetElem(i, minn);
222 //         p = p->next;
```

```
223 //         temp = p->next;
224 //     }
225 // }
226 
227 // ADT_list ADT_list::Union(ADT_list B){
228 //     ADT_list C;
229 //     C.InitLIst();
230 //     int index = 0;
231 //     node *p = head->next;
232 //     while(p){
233 //         if (B.LocateElem(p->value) >= 0){
234 //             C.InsertElem(++index, p->value);
235 //         }
236 //         p = p->next;
237 //     }
238 //     // ListTraverse();
239 //     // B.ListTraverse();
240 //     return C;
241 // }
242 
243 // void ADT_list::Josephus(int m){
244 //     node *p = head->next, *last = p, *tmp = head;
245 //     while(last->next)
246 //         last = last->next;
247 //     last->next = head->next;
248 //     while(length--){
249 //         int temp = m;
250 //         while(--temp){
251 //             p = p->next;
252 //             tmp = tmp->next;
253 //         }
254 //         printf("%d ", p->value);
255 //         tmp->next = p->next;
256 //         last = p;
257 //         p = p->next;
258 //         free(last);
259 //     }
260 // }
```

代码 37: 5Linked_list.cpp

```
1 #ifndef LINKED_LIST_H
```

```
#define LINKED_LIST_H
class ADT_list{
public:
    typedef struct node
    {
        int x1, y1, x2, y2;
        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();
    int SetElem(int index, int num);
    void InsertElem(int , int ,int ,int ,int);
    void DeleteElem(int index);
    void Remove();
    void Reverse();
    void Bubble_Sort();
    void Select_sort();
    ADT_list Union(ADT_list B);
    void Josephus(int);
};
#endif
```

代码 38: 5Linked_list.h

（四）测试数据及其结果

```
    0 4 7 5 2 6 1 3
```

代码 39: 3Eight_Queens.cpp

```
    100000000
```

代码 40: 4CSP.cpp

```
    2
    1
    1
    IGNORED
```

代码 41: 5CSP.cpp

（五）时间复杂度

$$O(n^2)$$

（六）改进方法

可以尝试把八皇后扩展到 n 皇后。

四、 Lab4

（一）数据结构

三元组

（二）算法设计思想

遍历两次把行和列换一下，最后完成转置。

（三）源程序

```
#include <stdio.h>
#include <algorithm>
typedef struct
{
    int i, j, v;
} Triple;
typedef struct
{
    Triple arr[256];
    int Rows, Cols, Nums;
} SqSMatrix;
```

```
bool cmp(Triple a, Triple b){
    return a.i < b.i;
}
void TransposeSMatrix(SqSMatrix &A, SqSMatrix &B){
    B.Rows = A.Cols;
    B.Cols = A.Rows;
    B.Nums = A.Nums;
    if (A.Nums > 0){
        int q = 0;
        for (int k = 0; k < A.Cols; k++){
            for (int p = 0; p < A.Nums; p++){
                if (A.arr[p].j == k){
                    B.arr[q].i = A.arr[p].j;
                    B.arr[q].j = A.arr[p].i;
                    B.arr[q].v = A.arr[p].v;
                    q++;
                }
            }
        }
    }
}
void FastTransposeSMatrix(SqSMatrix &A, SqSMatrix &B){
    int rowNum[256], rowStart[256];
    B.Rows = A.Cols;
    B.Cols = A.Rows;
    B.Nums = A.Nums;
    if (A.Nums > 0){
        int q = 0;
        for (int k = 0; k < A.Cols; k++) rowNum[k] = 0;
        for (int p = 0; p < A.Nums; p++) rowNum[A.arr[p].j]++;
        rowStart[0] = 0;
        for (int k = 0; k < A.Cols; k++) rowStart[k] = rowStart[k - 1] +
             rowNum[k - 1];
        for (int p = 0; p < A.Nums; p++){
            q = rowStart[A.arr[p].j];
            B.arr[q].i = A.arr[p].j;
            B.arr[q].j = A.arr[p].i;
            B.arr[q].v = A.arr[p].v;
            rowStart[A.arr[p].j]++;
        }
    }
}
```

```
53 int main(){
54     SqSMatrix A, B;
55     A.arr[0].i = 0;
56     A.arr[0].j = 2;
57     A.arr[0].v = 4;
58 
59     A.arr[1].i = 1;
60     A.arr[1].j = 1;
61     A.arr[1].v = 6;
62 
63     A.arr[2].i = 3;
64     A.arr[2].j = 0;
65     A.arr[2].v = 5;
66 
67     A.arr[3].i = 3;
68     A.arr[3].j = 4;
69     A.arr[3].v = 3;
70 
71     A.arr[4].i = 4;
72     A.arr[4].j = 2;
73     A.arr[4].v = 7;
74 
75     A.Cols = 5;
76     A.Rows = 5;
77     A.Nums = 5;
78 
79 // 简单方法
80     // for (int i = 0; i < 5; i++){
81     //     printf("%d %d %d\n", A.arr[i].i, A.arr[i].j, A.arr[i].v);
82     //     int temp = A.arr[i].i;
83     //     A.arr[i].i = A.arr[i].j;
84     //     A.arr[i].j = temp;
85     // }
86     // puts("");
87     // std::sort(A.arr, A.arr + 5, cmp);
88     // for (int i = 0; i < 5; i++){
89     //     printf("%d %d %d\n", A.arr[i].i, A.arr[i].j, A.arr[i].v);
90     // }
91 
92 
93     for (int i = 0; i < 5; i++)
94         printf("%d %d %d\n", A.arr[i].i, A.arr[i].j, A.arr[i].v);
```

```
95      puts("");
96      FastTransposeSMatrix(A, B); // 快速转置法
97      // TransposeSMatrix(A, B); // 列序遍历法
98      for (int i = 0; i < 5; i++)
99          printf("%d %d %d\n", B.arr[i].i, B.arr[i].j, B.arr[i].v);
100
101 }
```

代码 42: 1triple.cpp

```
1  #include <stdio.h>
2  #define M 4
3  #define N 4
4  int A[M][N]={{9,7,6,8},{20,26,22,25},{28,36,25,30},{12,4,2,6}}, max
       [256], min[256];
5  void minMax(){
6      for (int i = 0; i < M; i++){
7          min[i] = A[i][0];
8          for (int j = 0; j < N; j++)
9              if (min[i] > A[i][j])
10                 min[i] = A[i][j];
11     }
12     for (int i = 0; i < M; i++){
13         max[i] = A[0][i];
14         for (int j = 0; j < N; j++)
15             if (max[i] < A[j][i])
16                 max[i] = A[j][i];
17     }
18     for (int i = 0; i < M; i++)
19         for (int j = 0; j < N; j++)
20             if (min[i] == max[j])
21                 printf("%d\n", A[i][j]);
22 }
23 int main(){
24     minMax();
25 }
```

代码 43: 2Saddle_point.cpp

```
1  #include <stdio.h>
2  int n, k[111111], t[111111], num, state[4], tran[4] = {0, 3, 1, 2}, summ
       = 0;
3  void change(int time, int &sta, int &curtime){
```

```
    if (!sta) return;
    int nexttime = time - curtime;
    if (time >= curtime){
        sta = tran[sta];
        curtime = state[sta] - nexttime;
    }
    else
        curtime -= time;
}
int main(){
    freopen("3CSP", "r", stdin);
    scanf("%d%d%d", &state[1], &state[2], &state[3]);
    scanf("%d", &n);
    while (scanf("%d%d", &k[num], &t[num]) != EOF) num++;
    for (int i = 0; i < n; i++)
        if (k[i] != 3 && k[i] != 2){
            summ += t[i];
            for (int j = i + 1; j < n; j++)
                change(t[i], k[j], t[j]);
        }
    printf("%d\n", summ);
}
```

代码 44: 3CSP.cpp

```
#include <stdio.h>
int n, k, viste[22], num, summ = 0;
int main(){
    freopen("4CSP", "r", stdin);
    scanf("%d", &n);
    while (scanf("%d", &k) != EOF)
    for (int j = 0; j < 20; j++)
        if (5 - viste[j] >= k){
            for (int kk = 1; kk <= k; kk++)
                printf("%d ", 5 * j + kk + viste[j]);
            puts("");
            viste[j] = k;
            break;
        }
}
```

代码 45: 4CSP.cpp

（四）测试数据及其结果

```
0 2 4
1 1 6
3 0 5
3 4 3
4 2 7
```

代码 46: 1triple.cpp

```
25
```

代码 47: 2Saddle_point.cpp

```
46
```

代码 48: 3CSP.cpp

```
1 2
6 7 8 9 10
11 12 13 14
3 4
```

代码 49: 4CSP.cpp

（五）时间复杂度

$$O(n)$$

五、 Lab5

（一）数据结构

二叉树

（二）算法设计思想

二叉树的基本操作，插入，删除。

（三）源程序

```cpp
#include <malloc.h>
#include <stdio.h>
#include "lBinaryTree.h"

void BinaryTree::InitBiTree(){
    BiTNode *tmp = (BiTNode *)malloc(sizeof(BiTNode));
    root = tmp;
    root->data = NULL;
    root->left = NULL;
    root->right = NULL;
}
void BinaryTree::DestoryBiTree(BiTNode *p){
    if (p){
        if (p->left) DestoryBiTree(p->left), p->left = NULL;
        if (p->right) DestoryBiTree(p->right), p->right = NULL;
        free(p);
    }
}
BinaryTree::BiTNode* BinaryTree::CreateBiTree(){
    BiTNode *tmp = NULL;
    char ch;
    scanf("%c", &ch);
    if (ch != '#') {
        tmp = (BiTNode *)malloc(sizeof(BiTNode));
        tmp->data = ch;
        Nodenum++;
        tmp->left = CreateBiTree();
        tmp->right = CreateBiTree();
    }
    return tmp;
}
void BinaryTree::ClearBiTree(){
    if (root){
        DestoryBiTree(root->left);
        DestoryBiTree(root->right);
    }
}
bool BinaryTree::BiTreeEmpty(){
    if(!root->left && !root->right) return true;
    return false;

```

```
42 }
43 int BinaryTree::BiTreeDepth(BiTNode *p){
44     if (!p) return 0;
45     int left = BiTreeDepth(p->left);
46     int right = BiTreeDepth(p->right);
47     return left >= right ? left + 1 : right + 1;
48 }
49 char BinaryTree::Root(){
50     return root->data;
51 }
52 char BinaryTree::Value(BiTNode *p){
53     return p->data;
54 }
55 BinaryTree::BiTNode* BinaryTree::Parent(BiTNode *p, BiTNode *target){
56     if (!p) return NULL;
57     if (p == target)
58         return p;
59     if (Parent(p->left, target))
60         return Parent(p->left, target);
61     if (Parent(p->right, target))
62         return Parent(p->right, target);
63     return p;
64 }
65 BinaryTree::BiTNode* BinaryTree::LeftChild(BiTNode *p){
66     return p->left;
67 }
68 BinaryTree::BiTNode* BinaryTree::RightChild(BiTNode *p){
69     return p->right;
70 }
71 BinaryTree::BiTNode* BinaryTree::LeftBrother(BiTNode *p){
72     return Parent(root, p)->left == p ? NULL : Parent(root, p)->left;
73 }
74 BinaryTree::BiTNode* BinaryTree::RightBrother(BiTNode *p){
75     return Parent(root, p)->right == p ? NULL : Parent(root, p)->right;
76 }
77 BinaryTree::BiTNode* BinaryTree::FindtargetandDelete(BiTNode *p, char ch
    ){
78     if (p->left && p->left->data == ch) DestoryBiTree(p->left), p->left
        = NULL;
79     if (p->right && p->right->data == ch) DestoryBiTree(p->right), p->
        right = NULL;
80     if (p->left) return FindtargetandDelete(p->left, ch);
```

```
81      if (p->right) return FindtargetandDelete(p->right, ch);
82      return NULL;
83  }
84  void BinaryTree::PreOrderTreaverse(BiTNode *p){
85      if (!p) return;
86      printf("%c", p->data);
87      PreOrderTreaverse(p->left);
88      PreOrderTreaverse(p->right);
89  }
90  void BinaryTree::InOrderTreaverse(BiTNode *p){
91      if (!p) return;
92      InOrderTreaverse(p->left);
93      printf("%c", p->data);
94      InOrderTreaverse(p->right);
95  }
96  void BinaryTree::PostOrderTreaverse(BiTNode *p){
97      if (!p) return;
98      PostOrderTreaverse(p->left);
99      PostOrderTreaverse(p->right);
100     printf("%c", p->data);
101 }
102 void BinaryTree::LevelOrderTreaverse(){
103     BiTNode *queue[999];
104     int head = 0, rear = 0;
105     queue[rear++] = root;
106     while(head < rear){
107         printf("%c", queue[head]->data);
108         if (queue[head]->left) queue[rear++] = queue[head]->left;
109         if (queue[head]->right) queue[rear++] = queue[head]->right;
110         head++;
111     }
112     return;
113 }
114
115 void BinaryTree::Nonrecursive_PreOrderTreaverse(BiTNode *p){
116     BiTNode *stack[999];
117     int top = 0;
118     stack[++top] = root;
119     while (top){
120         printf("%c", stack[top]->data);
121         BiTNode* tmp = stack[top--];
122         if (tmp->right) stack[++top] = tmp->right;
```

```
123         if (tmp->left) stack[++top] = tmp->left;
124     }
125 }
126 void BinaryTree::Nonrecursive_InOrderTreaverse(BiTNode *p){
127     BiTNode *stack[999], *node = p;
128     int top = 0;
129     while (top || node){
130         if (node){
131             stack[++top] = node;
132             node = node->left;
133         }
134         else {
135             node = stack[top--];
136             printf("%c", node->data);
137             node = node->right;
138         }
139     }
140 }
141 void BinaryTree::Nonrecursive_PostOrderTreaverse(BiTNode *p){
142     BiTNode *stack1[999], *stack2[999], *node;
143     int top1 = 0, top2 = 0;
144     stack1[++top1] = p;
145     while (top1){
146         node = stack1[top1--];
147         stack2[++top2] = node;
148         if(node->left) stack1[++top1] = node->left;
149         if(node->right) stack1[++top1] = node->right;
150     }
151     while (top2)
152         printf("%c", stack2[top2--]->data);
153 }
154 
155 void BinaryTree::Assign(BiTNode *p, char value){
156     p->data = value;
157 }
158 void BinaryTree::InsertChild(BiTNode *p, BiTNode *c, int LR){
159     BiTNode *tmp = LR ? p->right : p->left;
160     if (LR)
161         p->right = c;
162     else
163         p->left = c;
164     c->right = tmp;
```

```
}
void BinaryTree::DeleteChild(BiTNode *p, int LR){
    if (LR)
        DestoryBiTree(p->right);
    else
        DestoryBiTree(p->left);
}
int BinaryTree::Complete_binary_tree(BiTNode *p){
    BiTNode *queue[999];
    int head = 0, rear = 0, num = 1;
    queue[rear++] = root;
    while(head < rear){
        if (queue[head]->left) queue[rear++] = queue[head]->left, num++;
        else if (num != Nodenum) return 0;
        if (queue[head]->right) queue[rear++] = queue[head]->right, num
            ++;
        else if (num != Nodenum) return 0;
        head++;
    }
    return 1;
}
```

```
#ifndef BinaryTree_H
#define BinaryTree_H
class BinaryTree{
public:
    typedef struct BiTNode
    {
        char data;
        struct BiTNode *left, *right;
    } BiTNode;
    BiTNode *root;
    int Nodenum = 0;
    void InitBiTree();
    void DestoryBiTree(BiTNode *p);
    BiTNode* CreateBiTree();
    void ClearBiTree();
    bool BiTreeEmpty();
    int BiTreeDepth(BiTNode *p);
    char Root();
    char Value(BiTNode *p);
```

```
20      BiTNode* Parent(BiTNode *p, BiTNode *target);
21      BiTNode* LeftChild(BiTNode *p);
22      BiTNode* RightChild(BiTNode *p);
23      BiTNode* LeftBrother(BiTNode *p);
24      BiTNode* RightBrother(BiTNode *p);
25      BiTNode* FindtargetandDelete(BiTNode *p, char ch);
26      void PreOrderTreaverse(BiTNode *p);
27      void InOrderTreaverse(BiTNode *p);
28      void PostOrderTreaverse(BiTNode *p);
29      void LevelOrderTreaverse();
30      void Nonrecursive_PreOrderTreaverse(BiTNode *p);
31      void Nonrecursive_InOrderTreaverse(BiTNode *p);
32      void Nonrecursive_PostOrderTreaverse(BiTNode *p);
33      void Nonrecursive_LevelOrderTreaverse(BiTNode *p);
34      void Assign(BiTNode *p, char value);
35      void InsertChild(BiTNode *p, BiTNode *c, int LR);
36      void DeleteChild(BiTNode *p, int LR);
37      int Complete_binary_tree(BiTNode *p);
38  };
39  #endif
```

```
1   #include "lBinaryTree.h"
2   #include <stdio.h>
3   int main(){
4       BinaryTree Tree;
5       Tree.InitBiTree();
6       freopen("tree", "r", stdin);
7       Tree.root = Tree.CreateBiTree();
8       Tree.PreOrderTreaverse(Tree.root);
9       puts("");
10      Tree.Nonrecursive_PreOrderTreaverse(Tree.root);
11      puts("");
12      Tree.InOrderTreaverse(Tree.root);
13      puts("");
14      Tree.Nonrecursive_InOrderTreaverse(Tree.root);
15      puts("");
16      Tree.PostOrderTreaverse(Tree.root);
17      puts("");
18      Tree.Nonrecursive_PostOrderTreaverse(Tree.root);
19      puts("");
20      Tree.LevelOrderTreaverse();
```

```
}
```

```
#include "lBinaryTree.h"
#include <stdio.h>
int main(){
    BinaryTree Tree;
    Tree.InitBiTree();
    freopen("tree", "r", stdin);
    Tree.root = Tree.CreateBiTree();
    Tree.PreOrderTreaverse(Tree.root);
    puts("");
    Tree.FindtargetandDelete(Tree.root, 'D');
    Tree.PreOrderTreaverse(Tree.root);
}
```

```
#include "lBinaryTree.h"
#include <stdio.h>
int main(){
    BinaryTree Tree;
    Tree.InitBiTree();
    freopen("tree2", "r", stdin);
    Tree.root = Tree.CreateBiTree();
    printf("%d", Tree.Complete_binary_tree(Tree.root));
}
```

```
#include <stdio.h>
int n, L, r, t, A[666][666], num;
float sum;
int main(){
    freopen("5CSP2", "r", stdin);
    scanf("%d%d%d%d", &n, &L, &r, &t);
    for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        scanf("%d", &A[i][j]);
    for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++){
        int x1 = i - r > 0 ? i - r : 0;
        int x2 = i + r < n ? i + r : n - 1;
        int y1 = j - r > 0 ? j - r : 0;
        int y2 = j + r < n ? j + r : n - 1;
        sum = 0;
```

```
        for (int k = x1; k <= x2; k++)
            for (int kk = y1; kk <= y2; kk++)
                sum += A[k][kk];
        sum /= (x2 - x1 + 1) * (y2 - y1 + 1) * 1.0;
        if (sum <= t) num++;
    }
    printf("%d", num);
}
```

（四）测试数据及其结果

```
ABDHIECFJG
ABDHIECFJG
HDIBEAFJCG
HDIBEAFJCG
HIDEBJFGCA
HIDEBJFGCA
ABCDEFGHIJ
```

代码 50: 2Nonrecursive.cpp

```
ABDHIECFJG
ABECFJG
```

代码 51: 3delete.cpp

```
1
```

代码 52: 4Complete_binary_tree.cpp

（五）时间复杂度

$$O(log_2 n)$$

六、 Lab6

（一）数据结构

图，孩子兄弟表示法，二叉树

（二）算法设计思想

构建哈夫曼树, 孩子兄弟表示法就是把二叉树稍微改一改。

（三）源程序

```
#include <malloc.h>
#include <string.h>
#include <stdio.h>
#include "1Graph.h"
int Graph::CreateGraph(){
    scanf("%d%d", &G.n, &G.e);
    memset(G.edges, 0, sizeof(G.edges));
    for (int i = 0; i < G.n; i++)
        scanf("%c", &G.v[i]);
    int x, y;
    for (int i = 0; i < G.e; i++){
        scanf("%d%d", &x, &y);
        G.edges[x][y] = 1;
        G.edges[y][x] = 1;
    }
}
void Graph::DestoryGraph(){
    memset(G.edges, 0, sizeof(G.edges));
    G.n = 0;
    G.e = 0;
}
char Graph::GetVex(int index){
    return G.v[index];
}
int Graph::FirstAdjVex(int start){
    for (int i = 0; i < G.n; i++){
        if (G.edges[start][i]) return i;
    }
    return NULL;
}
int Graph::NextAdjVex(int start, int now){
    int flag = 0;
    for (int i = 0; i < G.n; i++){
        if (flag && G.edges[start][i]) return i;
        if (i == now && G.edges[start][i] && !flag) flag = 1;
```

```
    }
    return NULL;
}
void Graph::DFSTraverse(){
    memset(visited, 0, sizeof(visited));
    for (int i = 0; i < G.n; i++)
        if (!visited[i])
            DFS(i);
}
void Graph::DFS(int index){
    printf("%c", G.v[index]);
    visited[index] = 1;
    for (int i = 0; i < G.n; i++)
        if (!visited[i] && G.edges[index][i])
            DFS(i);
}
void Graph::BFSTraverse(){
    memset(visited, 0, sizeof(visited));
    for (int i = 0; i < G.n; i++){
        if (!visited[i])
            BFS(i);
    }
}
void Graph::BFS(int index){
    int queue[999], rear = 0, front = 0;
    queue[++rear] = index;
    visited[index] = 1;
    while (front != rear){
        int first = queue[front++];
        printf("%d ", first);
        for (int i = 0; i < G.n; i++){
            if (!visited[i] && G.edges[index][i]){
                visited[i] = 1;
                queue[++rear] = i;
            }
        }

    }
}
void Graph::InsertVex(char ch){
    G.v[G.n++] = ch;
}
```

```
78 void Graph::InsertArc(int start, int end){
79     G.edges[start][end] = 1;
80     G.edges[end][start] = 1;
81 }
82 void Graph::DeleteVex(int index){
83     for (int i = index; i < G.n - 1; i++){
84         G.v[i] = G.v[i + 1];
85     }
86     G.n--;
87 }
88 void Graph::DeleteArc(int start, int end){
89     G.edges[start][end] = 0;
90     G.edges[end][start] = 0;
91 }
```

```
1  #ifndef Graph_H
2  #define Graph_H
3  class Graph{
4  public:
5      typedef struct MGrape
6      {
7          int edges[999][999];
8          int n, e;
9          char v[999];
10     } MGrape;
11     MGrape G;
12     int visited[999];
13     int CreateGraph();
14     void DestoryGraph();
15     char GetVex(int);
16     int FirstAdjVex(int);
17     int NextAdjVex(int, int);
18     void DFSTraverse();
19     void DFS(int);
20     void BFSTraverse();
21     void BFS(int);
22     void InsertVex(char);
23     void InsertArc(int, int);
24     void DeleteVex(int);
25     void DeleteArc(int, int);
26 };
```

```
#endif
```

```
#include <stdio.h>
class HuffmanTree{
public:
    typedef struct HNode{
        int weight;
        int parent;
        int lchild, rchild;
        char *code;
        HNode() {
            weight = 0;
            parent = lchild = rchild = -1;
        }
    } HNode;
    HNode *Tree;
    int TreeSize, flags[99];
    void CreateTree(int* a, int n){
        TreeSize = 2 * n - 1;
        Tree = new HNode[TreeSize];
        for (int i = 0; i < n ;i++){
            Tree[i].weight = a[i];
        }
        int s1, s2, nextPos = n;
        for (int i = 0; i < n - 1; i++){
            SelectTwoMin(nextPos, s1, s2);
            Tree[nextPos].lchild = s1;
            Tree[nextPos].rchild = s2;
            Tree[nextPos].weight = Tree[s1].weight + Tree[s2].weight;
            Tree[s1].parent = Tree[s2].parent = nextPos++;
        }
    };
    void SelectTwoMin(int nextPos, int &s1, int &s2){
        int index = 0;
        while(Tree[index].parent != -1) index++;
        s1 = index++;
        while(Tree[index].parent != -1) index++;
        s2 = index;
        KeepOrder(s1, s2);
        for (int i = index + 1; i < nextPos; i++){
            if (Tree[i].parent == -1 && Tree[i].weight < Tree[s2].weight
```

```
                ){
40              s2 = i;
41              KeepOrder(s1, s2);
42          }
43      }
44  };
45  void KeepOrder(int& n1,int& n2){
46      if (Tree[n1].weight > Tree[n2].weight)
47      {
48          int tmp = n1;
49          n1 = n2;
50          n2 = tmp;
51      }
52  };
53  void printtree(int index, int tab, int flag){
54      int nextTab = tab;
55      flags[tab] = 1;
56      printTabs(tab);
57      if (Tree[index].weight){
58          if (flag == 2) {
59              printf ("\\--> %d", Tree[index].weight);
60              flags[tab] = 0;
61          }
62          else{
63              printf ("|--> %d", Tree[index].weight);
64          }
65      }
66      printf("\n");
67      if (Tree[index].lchild != -1) {
68          printtree(Tree[index].lchild, nextTab + 1, 1);
69      }
70      if (Tree[index].rchild != -1) {
71          flags[tab + 1] = 0;
72          printtree(Tree[index].rchild, tab + 1, 2);
73      }
74  }
75  void printTabs(int numOfTabs) {
76      int i;
77      for (i = 0; i < numOfTabs; i++) {
78          if (flags[i] == 0)
79              printf("   ");
80          else
```

```
                printf("|  ");
            }
        }
        int findroot(){
            int index = 0;
            while (Tree[index].parent != -1) index = Tree[index].parent;
            return index;
        }
};
int main(){
    HuffmanTree hTree;
    int x[] = {5,29,7,8,14,23,3,11};
    hTree.CreateTree(x, sizeof(x) / sizeof(x[0]));
    hTree.printtree(hTree.findroot(), 0, 2);
}
```

```
#include <malloc.h>
#include <stdio.h>
#include "3BinaryTree.h"

void BinaryTree::InitBiTree(){
    BiTNode *tmp = (BiTNode *)malloc(sizeof(BiTNode));
    root = tmp;
    root->data = NULL;
    root->left = NULL;
    root->right = NULL;
}
void BinaryTree::DestoryBiTree(BiTNode *p){
    if (p){
        if (p->left) DestoryBiTree(p->left), p->left = NULL;
        if (p->right) DestoryBiTree(p->right), p->right = NULL;
        free(p);
    }
}
BinaryTree::BiTNode* BinaryTree::CreateBiTree(){
    BiTNode *tmp = NULL;
    char ch;
    scanf("%c", &ch);
    if (ch != '#') {
        tmp = (BiTNode *)malloc(sizeof(BiTNode));
        tmp->data = ch;
```

```
        Nodenum++;
        tmp->left = CreateBiTree();
        tmp->right = CreateBiTree();
    }
    return tmp;
}
void BinaryTree::ClearBiTree(){
    if (root){
        DestoryBiTree(root->left);
        DestoryBiTree(root->right);
    }
}
bool BinaryTree::BiTreeEmpty(){
    if(!root->left && !root->right) return true;
    return false;

}
int BinaryTree::BiTreeDepth(BiTNode *p){
    if (!p) return 0;
    int left = BiTreeDepth(p->left);
    int right = BiTreeDepth(p->right);
    return left >= right ? left + 1 : right + 1;
}
char BinaryTree::Root(){
    return root->data;
}
char BinaryTree::Value(BiTNode *p){
    return p->data;
}
BinaryTree::BiTNode* BinaryTree::Parent(BiTNode *p, BiTNode *target){
    if (!p) return NULL;
    if (p == target)
        return p;
    if (Parent(p->left, target))
        return Parent(p->left, target);
    if (Parent(p->right, target))
        return Parent(p->right, target);
    return p;
}
BinaryTree::BiTNode* BinaryTree::LeftChild(BiTNode *p){
    return p->left;
}
```

```
68  BinaryTree::BiTNode* BinaryTree::RightChild(BiTNode *p){
69      return p->right;
70  }
71  BinaryTree::BiTNode* BinaryTree::LeftBrother(BiTNode *p){
72      return Parent(root, p)->left == p ? NULL : Parent(root, p)->left;
73  }
74  BinaryTree::BiTNode* BinaryTree::RightBrother(BiTNode *p){
75      return Parent(root, p)->right == p ? NULL : Parent(root, p)->right;
76  }
77  BinaryTree::BiTNode* BinaryTree::FindtargetandDelete(BiTNode *p, char ch
        ){
78      if (p->left && p->left->data == ch) DestoryBiTree(p->left), p->left
            = NULL;
79      if (p->right && p->right->data == ch) DestoryBiTree(p->right), p->
            right = NULL;
80      if (p->left) return FindtargetandDelete(p->left, ch);
81      if (p->right) return FindtargetandDelete(p->right, ch);
82      return NULL;
83  }
84  void BinaryTree::PreOrderTreaverse(BiTNode *p){
85      if (!p) return;
86      printf("%c", p->data);
87      PreOrderTreaverse(p->left);
88      PreOrderTreaverse(p->right);
89  }
90  void BinaryTree::InOrderTreaverse(BiTNode *p){
91      if (!p) return;
92      InOrderTreaverse(p->left);
93      printf("%c", p->data);
94      InOrderTreaverse(p->right);
95  }
96  void BinaryTree::PostOrderTreaverse(BiTNode *p){
97      if (!p) return;
98      PostOrderTreaverse(p->left);
99      PostOrderTreaverse(p->right);
100     printf("%c", p->data);
101 }
102 void BinaryTree::LevelOrderTreaverse(){
103     BiTNode *queue[999];
104     int head = 0, rear = 0;
105     queue[rear++] = root;
106     while(head < rear){
```

```
107         printf("%c", queue[head]->data);
108         if (queue[head]->left) queue[rear++] = queue[head]->left;
109         if (queue[head]->right) queue[rear++] = queue[head]->right;
110         head++;
111     }
112     return;
113 }
114 
115 void BinaryTree::Nonrecursive_PreOrderTreaverse(BiTNode *p){
116     BiTNode *stack[999];
117     int top = 0;
118     stack[++top] = root;
119     while (top){
120         printf("%c", stack[top]->data);
121         BiTNode* tmp = stack[top--];
122         if (tmp->right) stack[++top] = tmp->right;
123         if (tmp->left) stack[++top] = tmp->left;
124     }
125 }
126 void BinaryTree::Nonrecursive_InOrderTreaverse(BiTNode *p){
127     BiTNode *stack[999], *node = p;
128     int top = 0;
129     while (top || node){
130         if (node){
131             stack[++top] = node;
132             node = node->left;
133         }
134         else {
135             node = stack[top--];
136             printf("%c", node->data);
137             node = node->right;
138         }
139     }
140 }
141 void BinaryTree::Nonrecursive_PostOrderTreaverse(BiTNode *p){
142     BiTNode *stack1[999], *stack2[999], *node;
143     int top1 = 0, top2 = 0;
144     stack1[++top1] = p;
145     while (top1){
146         node = stack1[top1--];
147         stack2[++top2] = node;
148         if(node->left) stack1[++top1] = node->left;
```

```
149         if(node->right) stack1[++top1] = node->right;
150     }
151     while (top2)
152         printf("%c", stack2[top2--]->data);
153 }
154 
155 void BinaryTree::Assign(BiTNode *p, char value){
156     p->data = value;
157 }
158 void BinaryTree::InsertChild(BiTNode *p, BiTNode *c, int LR){
159     BiTNode *tmp = LR ? p->right : p->left;
160     if (LR)
161         p->right = c;
162     else
163         p->left = c;
164     c->right = tmp;
165 }
166 void BinaryTree::DeleteChild(BiTNode *p, int LR){
167     if (LR)
168         DestoryBiTree(p->right);
169     else
170         DestoryBiTree(p->left);
171 }
172 int BinaryTree::Complete_binary_tree(BiTNode *p){
173     BiTNode *queue[999];
174     int head = 0, rear = 0, num = 1;
175     queue[rear++] = root;
176     while(head < rear){
177         if (queue[head]->left) queue[rear++] = queue[head]->left, num++;
178         else if (num != Nodenum) return 0;
179         if (queue[head]->right) queue[rear++] = queue[head]->right, num
              ++;
180         else if (num != Nodenum) return 0;
181         head++;
182     }
183     return 1;
184 }
185 int BinaryTree::getwidth(BiTNode *p){
186     BiTNode *stack[999];
187     int top = 0, maxx = -1, nownum = 1, start = 1, end = 1;
188     stack[++top] = p;
189     while (nownum){
```

```
190         nownum = 0;
191         for (int i = start;i <= end; i++){
192             if (stack[i]->left) stack[++top] = stack[i]->left, nownum++;
193             if (stack[i]->right) stack[++top] = stack[i]->right, nownum
                ++;
194         }
195         start = end + 1;
196         end = end + nownum;
197         maxx = maxx < nownum ? nownum : maxx;
198     }
199     return maxx;
200 }
201
202 void BinaryTree::printtree(BiTNode *node, int tab, int flag){
203     int nextTab = tab;
204     flags[tab] = 1;
205     printTabs(tab);
206     if (node){
207         if (flag == 2) {
208             printf ("\\--> %c", node->data);
209             flags[tab] = 0;
210         }
211         else{
212             printf ("|--> %c", node->data);
213         }
214     }
215     printf("\n");
216     if (node->left) {
217         printtree(node->left, nextTab + 1, 1);
218     }
219     if (node->right) {
220         flags[tab + 1] = 0;
221         printtree(node->right, tab + 1, 2);
222     }
223 }
224 void BinaryTree::printTabs(int numOfTabs) {
225     int i;
226     for (i = 0; i < numOfTabs; i++) {
227         if (flags[i] == 0)
228             printf("   ");
229         else
230             printf("|  ");
```

```
    }
}
```

```
#ifndef BinaryTree_H
#define BinaryTree_H
class BinaryTree{
public:
    typedef struct BiTNode
    {
        char data;
        struct BiTNode *left, *right;
    } BiTNode;
    BiTNode *root;
    int Nodenum = 0, flags[999];
    void InitBiTree();
    void DestoryBiTree(BiTNode *p);
    BiTNode* CreateBiTree();
    void ClearBiTree();
    bool BiTreeEmpty();
    int BiTreeDepth(BiTNode *p);
    char Root();
    char Value(BiTNode *p);
    BiTNode* Parent(BiTNode *p, BiTNode *target);
    BiTNode* LeftChild(BiTNode *p);
    BiTNode* RightChild(BiTNode *p);
    BiTNode* LeftBrother(BiTNode *p);
    BiTNode* RightBrother(BiTNode *p);
    BiTNode* FindtargetandDelete(BiTNode *p, char ch);
    void PreOrderTreaverse(BiTNode *p);
    void InOrderTreaverse(BiTNode *p);
    void PostOrderTreaverse(BiTNode *p);
    void LevelOrderTreaverse();
    void Nonrecursive_PreOrderTreaverse(BiTNode *p);
    void Nonrecursive_InOrderTreaverse(BiTNode *p);
    void Nonrecursive_PostOrderTreaverse(BiTNode *p);
    void Nonrecursive_LevelOrderTreaverse(BiTNode *p);
    void Assign(BiTNode *p, char value);
    void InsertChild(BiTNode *p, BiTNode *c, int LR);
    void DeleteChild(BiTNode *p, int LR);
    int Complete_binary_tree(BiTNode *p);
    int getwidth(BiTNode *p);
```

```
39      void printtree(BiTNode *node, int tab, int flag);
40      void printTabs(int);
41  };
42  #endif
```

```
1   #include "3BinaryTree.h"
2   #include <stdio.h>
3   int main(){
4       BinaryTree Tree;
5       Tree.InitBiTree();
6       freopen("tree", "r", stdin);
7       Tree.root = Tree.CreateBiTree();
8       // Tree.PreOrderTreaverse(Tree.root);
9       // puts("");
10      Tree.printtree(Tree.root, 0, 2);
11      printf("%d", Tree.getwidth(Tree.root));
12  }
```

```
1   #include <malloc.h>
2   #include <stdio.h>
3   #include "4Kidbrother.h"
4
5   void BinaryTree::InitBiTree(){
6       BiTNode *tmp = (BiTNode *)malloc(sizeof(BiTNode));
7       root = tmp;
8       root->data = NULL;
9       root->kid = NULL;
10      root->borther = NULL;
11  }
12  void BinaryTree::DestoryBiTree(BiTNode *p){
13      if (p){
14          if (p->kid) DestoryBiTree(p->kid), p->kid = NULL;
15          if (p->borther) DestoryBiTree(p->borther), p->borther = NULL;
16          free(p);
17      }
18  }
19  BinaryTree::BiTNode* BinaryTree::CreateBiTree(){
20      BiTNode *tmp = NULL;
21      char ch;
22      scanf("%c", &ch);
23      if (ch != '#') {
```

```
24          tmp = (BiTNode *)malloc(sizeof(BiTNode));
25          tmp->data = ch;
26          Nodenum++;
27          tmp->kid = CreateBiTree();
28          tmp->borther = CreateBiTree();
29      }
30      return tmp;
31  }
32  void BinaryTree::ClearBiTree(){
33      if (root){
34          DestoryBiTree(root->kid);
35          DestoryBiTree(root->borther);
36      }
37  }
38  bool BinaryTree::BiTreeEmpty(){
39      if(!root->kid && !root->borther) return true;
40      return false;
41
42  }
43  int BinaryTree::BiTreeDepth(BiTNode *p){
44      if (!p) return 0;
45      int kid = BiTreeDepth(p->kid);
46      int borther = BiTreeDepth(p->borther);
47      return kid >= borther ? kid + 1 : borther + 1;
48  }
49  char BinaryTree::Root(){
50      return root->data;
51  }
52  char BinaryTree::Value(BiTNode *p){
53      return p->data;
54  }
55  BinaryTree::BiTNode* BinaryTree::Parent(BiTNode *p, BiTNode *target){
56      if (!p) return NULL;
57      if (p == target)
58          return p;
59      if (Parent(p->kid, target))
60          return Parent(p->kid, target);
61      if (Parent(p->borther, target))
62          return Parent(p->borther, target);
63      return p;
64  }
65  BinaryTree::BiTNode* BinaryTree::kidChild(BiTNode *p){
```

```
66      return p->kid;
67  }
68  BinaryTree::BiTNode* BinaryTree::bortherChild(BiTNode *p){
69      return p->borther;
70  }
71  BinaryTree::BiTNode* BinaryTree::kidBrother(BiTNode *p){
72      return Parent(root, p)->kid == p ? NULL : Parent(root, p)->kid;
73  }
74  BinaryTree::BiTNode* BinaryTree::bortherBrother(BiTNode *p){
75      return Parent(root, p)->borther == p ? NULL : Parent(root, p)->
            borther;
76  }
77  BinaryTree::BiTNode* BinaryTree::FindtargetandDelete(BiTNode *p, char ch
        ){
78      if (p->kid && p->kid->data == ch) DestoryBiTree(p->kid), p->kid =
            NULL;
79      if (p->borther && p->borther->data == ch) DestoryBiTree(p->borther),
             p->borther = NULL;
80      if (p->kid) return FindtargetandDelete(p->kid, ch);
81      if (p->borther) return FindtargetandDelete(p->borther, ch);
82      return NULL;
83  }
84  void BinaryTree::PreOrderTreaverse(BiTNode *p){
85      if (!p) return;
86      printf("%c", p->data);
87      PreOrderTreaverse(p->kid);
88      PreOrderTreaverse(p->borther);
89  }
90  void BinaryTree::InOrderTreaverse(BiTNode *p){
91      if (!p) return;
92      InOrderTreaverse(p->kid);
93      printf("%c", p->data);
94      InOrderTreaverse(p->borther);
95  }
96  void BinaryTree::PostOrderTreaverse(BiTNode *p){
97      if (!p) return;
98      PostOrderTreaverse(p->kid);
99      PostOrderTreaverse(p->borther);
100     printf("%c", p->data);
101 }
102 void BinaryTree::LevelOrderTreaverse(){
103     BiTNode *queue[999];
```

```
104     int head = 0, rear = 0;
105     queue[rear++] = root;
106     while(head < rear){
107         printf("%c", queue[head]->data);
108         if (queue[head]->kid) queue[rear++] = queue[head]->kid;
109         if (queue[head]->borther) queue[rear++] = queue[head]->borther;
110         head++;
111     }
112     return;
113 }
114 
115 void BinaryTree::Nonrecursive_PreOrderTreaverse(BiTNode *p){
116     BiTNode *stack[999];
117     int top = 0;
118     stack[++top] = root;
119     while (top){
120         printf("%c", stack[top]->data);
121         BiTNode* tmp = stack[top--];
122         if (tmp->borther) stack[++top] = tmp->borther;
123         if (tmp->kid) stack[++top] = tmp->kid;
124     }
125 }
126 void BinaryTree::Nonrecursive_InOrderTreaverse(BiTNode *p){
127     BiTNode *stack[999], *node = p;
128     int top = 0;
129     while (top || node){
130         if (node){
131             stack[++top] = node;
132             node = node->kid;
133         }
134         else {
135             node = stack[top--];
136             printf("%c", node->data);
137             node = node->borther;
138         }
139     }
140 }
141 void BinaryTree::Nonrecursive_PostOrderTreaverse(BiTNode *p){
142     BiTNode *stack1[999], *stack2[999], *node;
143     int top1 = 0, top2 = 0;
144     stack1[++top1] = p;
145     while (top1){
```

```
146         node = stack1[top1--];
147         stack2[++top2] = node;
148         if(node->kid) stack1[++top1] = node->kid;
149         if(node->borther) stack1[++top1] = node->borther;
150     }
151     while (top2)
152         printf("%c", stack2[top2--]->data);
153 }
154 
155 void BinaryTree::Assign(BiTNode *p, char value){
156     p->data = value;
157 }
158 void BinaryTree::InsertChild(BiTNode *p, BiTNode *c, int LR){
159     BiTNode *tmp = LR ? p->borther : p->kid;
160     if (LR)
161         p->borther = c;
162     else
163         p->kid = c;
164     c->borther = tmp;
165 }
166 void BinaryTree::DeleteChild(BiTNode *p, int LR){
167     if (LR)
168         DestoryBiTree(p->borther);
169     else
170         DestoryBiTree(p->kid);
171 }
172 int BinaryTree::Complete_binary_tree(BiTNode *p){
173     BiTNode *queue[999];
174     int head = 0, rear = 0, num = 1;
175     queue[rear++] = root;
176     while(head < rear){
177         if (queue[head]->kid) queue[rear++] = queue[head]->kid, num++;
178         else if (num != Nodenum) return 0;
179         if (queue[head]->borther) queue[rear++] = queue[head]->borther,
                num++;
180         else if (num != Nodenum) return 0;
181         head++;
182     }
183     return 1;
184 }
185 int BinaryTree::getwidth(BiTNode *p){
186     BiTNode *stack[999];
```

```
187     int top = 0, maxx = -1, nownum = 1, start = 1, end = 1;
188     stack[++top] = p;
189     while (nownum){
190         nownum = 0;
191         for (int i = start;i <= end; i++){
192             if (stack[i]->kid) stack[++top] = stack[i]->kid, nownum++;
193             if (stack[i]->borther) stack[++top] = stack[i]->borther,
                    nownum++;
194         }
195         start = end + 1;
196         end = end + nownum;
197         maxx = maxx < nownum ? nownum : maxx;
198     }
199     return maxx;
200 }
201 
202 void BinaryTree::printtree(BiTNode *node, int tab, int flag){
203     int nextTab = tab;
204     flags[tab] = 1;
205     printTabs(tab);
206     if (node){
207         if (flag == 2) {
208             printf ("\\--> %c", node->data);
209             flags[tab] = 0;
210         }
211         else{
212             printf ("|--> %c", node->data);
213         }
214     }
215     printf("\n");
216     if (node->kid) {
217         printtree(node->kid, nextTab + 1, 1);
218     }
219     if (node->borther) {
220         flags[tab + 1] = 0;
221         printtree(node->borther, tab + 1, 2);
222     }
223 }
224 void BinaryTree::printTabs(int numOfTabs) {
225     int i;
226     for (i = 0; i < numOfTabs; i++) {
227         if (flags[i] == 0)
```

```
            printf("  ");
        else
            printf("| ");
    }
}
void BinaryTree::Output_layer_elements(int index){
    BiTNode *node = root, *stack[999];
    int top = 0;
    while (node){
        stack[++top] = node;
        node = node->borther;
    }
    int k = 0;
    while (k++ != top){
        int times = index;
        while (--times){
            if (stack[k]->kid) stack[k] = stack[k]->kid;
            else {
                for (int j = k; j < top - 1; j++){
                    stack[j] = stack[j + 1];
                }
                k--,top--;
                break;
            };
        }
    }
    int set[333];
    for (int i = 1; i <= top; i++){
        while (stack[i]){
            if (!set[(int)stack[i]->data])
                printf("%c ", stack[i]->data), set[(int)stack[i]->data
                    ]++;
            if (stack[i]->borther) stack[i] = stack[i]->borther;
            else break;
        }
    }
}
```

```
#ifndef Kidbrother_H
#define Kidbrother_H
class BinaryTree{
```

```
4  public:
5      typedef struct BiTNode
6      {
7          char data;
8          struct BiTNode *kid, *borther;
9      } BiTNode;
10     BiTNode *root;
11     int Nodenum = 0, flags[999];
12     void InitBiTree();
13     void DestoryBiTree(BiTNode *p);
14     BiTNode* CreateBiTree();
15     void ClearBiTree();
16     bool BiTreeEmpty();
17     int BiTreeDepth(BiTNode *p);
18     char Root();
19     char Value(BiTNode *p);
20     BiTNode* Parent(BiTNode *p, BiTNode *target);
21     BiTNode* kidChild(BiTNode *p);
22     BiTNode* bortherChild(BiTNode *p);
23     BiTNode* kidBrother(BiTNode *p);
24     BiTNode* bortherBrother(BiTNode *p);
25     BiTNode* FindtargetandDelete(BiTNode *p, char ch);
26     void PreOrderTreaverse(BiTNode *p);
27     void InOrderTreaverse(BiTNode *p);
28     void PostOrderTreaverse(BiTNode *p);
29     void LevelOrderTreaverse();
30     void Nonrecursive_PreOrderTreaverse(BiTNode *p);
31     void Nonrecursive_InOrderTreaverse(BiTNode *p);
32     void Nonrecursive_PostOrderTreaverse(BiTNode *p);
33     void Nonrecursive_LevelOrderTreaverse(BiTNode *p);
34     void Assign(BiTNode *p, char value);
35     void InsertChild(BiTNode *p, BiTNode *c, int LR);
36     void DeleteChild(BiTNode *p, int LR);
37     int Complete_binary_tree(BiTNode *p);
38     int getwidth(BiTNode *p);
39     void printtree(BiTNode *node, int tab, int flag);
40     void printTabs(int);
41     void Output_layer_elements(int index);
42 };
43 #endif
```

```cpp
#include "4Kidbrother.h"
#include <stdio.h>
int main(){
    BinaryTree Tree;
    Tree.InitBiTree();
    freopen("tree", "r", stdin);
    Tree.root = Tree.CreateBiTree();
    Tree.PreOrderTreaverse(Tree.root);
    puts("");
    Tree.printtree(Tree.root, 0, 2);
    Tree.Output_layer_elements(1);
    puts("");
    Tree.Output_layer_elements(2);
    puts("");
    Tree.Output_layer_elements(3);
    puts("");
    Tree.Output_layer_elements(4);
}
```

```cpp
#include <stdio.h>
int A[22][11], B[5][5], n, max = 0, map[44], xmin = 0x3f3f3f3f, ymin = 0
    x3f3f3f3f;
float sum;
int main(){
    freopen("5CSP", "r", stdin);
    for (int i = 0; i < 15; i++)
    for (int j = 0; j < 10; j++)
        scanf("%d", &A[i][j]);
    for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
        scanf("%d", &B[i][j]);
    scanf("%d", &n);

    int index = 0;
    for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
        if (B[i][j]) {
            map[index * 2] = i;
            map[index++ * 2 + 1] = j;
            xmin = xmin > i ? i : xmin;
            ymin = ymin > j ? j : ymin;
```

```
        }

    for (int i = 2; i < 15; i++){
        int flag = 0;
        for (int j = 0; j < index; j++){
            if (A[map[j * 2] + i - xmin - 1][map[j * 2 + 1] + n - ymin]
                || map[j * 2] + i - xmin - 1> 14) {
                printf("%d %d %d\n", A[map[j * 2] + i][map[j * 2 + 1] +
                    n], map[j * 2] + i, map[j * 2 + 1] + n);
                flag = 1;
                break;
            }
        }
        if (flag) break;
        else max = i;
    }
    for (int i = 0; i < index; i++)
        A[map[i * 2] + max - xmin - 1][map[i * 2 + 1] + n - ymin] = 1;

    for (int i = 0; i < 15; i++){
        for (int j = 0; j < 10; j++)
            printf("%d ", A[i][j]);
        puts("");
    }

}
```

（四）测试数据及其结果

```
    ABDHIECFJG
    \--> A
      |--> B
      | |--> D
      | | |--> H
      | | \--> I
      | \--> E
      \--> C
        |--> F
        | \--> J
        \--> G
    A C G
```

```
   B E F J
   D I
   H
```

代码 53: 4Outputelement.cpp

```
   \--> A
   |--> B
   | |--> D
   | | |--> H
   | | \--> I
   | \--> E
   \--> C
     |--> F
     | \--> J
     \--> G
4
```

代码 54: 3BinaryTreeweigh.cpp

```
   \--> 100
   |--> 42
   | |--> 19
   | | |--> 8
   | | | |--> 3
   | | | \--> 5
   | | \--> 11
   | \--> 23
   \--> 58
     |--> 29
     \--> 29
       |--> 14
       \--> 15
         |--> 7
         \--> 8
```

代码 55: 2HuffmanTree.cpp

```
   0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0
```

```
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 1 0 0
    0 0 0 0 0 0 1 0 0 0
    0 0 0 0 0 0 1 0 0 0
    1 1 1 1 1 1 1 1 1 1
    0 0 0 0 1 1 0 0 0 0
```

代码 56: 5CSP.cpp

七、 Lab7

（一）数据结构

广度优先搜索，深度优先搜索，Dijkstra 算法

（二）算法设计思想

广度优先搜索，深度优先搜索，Dijkstra 算法的基本思想

（三）源程序

```cpp
#include <iostream>
using namespace std;
#define INF 2147483647
#define maxn 20000

struct Edge{
    int v;
    int len;
    int next;
} edge[1111111];
int head[maxn], cnt = 0, m, from, to, len, queue[1111111], front, rear;
bool visit[maxn];
long long dis[maxn];

void addedge(int from, int to, int len){
    edge[++cnt].v = to;
    edge[cnt].len = len;
```

```
    edge[cnt].next = head[from];
    head[from] = cnt;
}

void BFS(int temp){
    queue[++rear] = temp;
    front = rear;
    while (front <= rear){
        int tmp = queue[front++];
        printf("%d ", tmp);
        for (int j = head[tmp]; j; j = edge[j].next)
            queue[++rear] = edge[j].v;
    }
}


int main(){
    freopen("1BFS", "r", stdin);
    cin >> m;
    for (int i = 0; i < m; i++){
        cin >> from >> to >> len;
        addedge(from, to, len);
    }
    BFS(1);
    return 0;
}
```

```
#include <iostream>
using namespace std;
#define INF 2147483647
#define maxn 20000

struct Edge{
    int v;
    int len;
    int next;
} edge[1111111];
int head[maxn], cnt = 0, m, from, to, len;
bool visit[maxn];
long long dis[maxn];

```

```cpp
void addedge(int from, int to, int len){
    edge[++cnt].v = to;
    edge[cnt].len = len;
    edge[cnt].next = head[from];
    head[from] = cnt;
}

void DFS(int temp){
    printf("%d ", temp);
    for (int j = head[temp]; j; j = edge[j].next)
        DFS(edge[j].v);
}


int main(){
    freopen("1DFS", "r", stdin);
    cin >> m;
    for (int i = 0; i < m; i++){
        cin >> from >> to >> len;
        addedge(from, to, len);
    }
    DFS(1);
    return 0;
}
```

```cpp
#include <malloc.h>
#include <string.h>
#include <stdio.h>
#include "1Graph.h"
int Graph::CreateGraph(){
    scanf("%d%d", &G.n, &G.e);
    memset(G.edges, 0, sizeof(G.edges));
    for (int i = 0; i < G.n; i++)
        scanf("%c", &G.v[i]);
    int x, y;
    for (int i = 0; i < G.e; i++){
        scanf("%d%d", &x, &y);
        G.edges[x][y] = 1;
        G.edges[y][x] = 1;
    }
}
```

```
17 void Graph::DestoryGraph(){
18     memset(G.edges, 0, sizeof(G.edges));
19     G.n = 0;
20     G.e = 0;
21 }
22 char Graph::GetVex(int index){
23     return G.v[index];
24 }
25 int Graph::FirstAdjVex(int start){
26     for (int i = 0; i < G.n; i++){
27         if (G.edges[start][i]) return i;
28     }
29     return NULL;
30 }
31 int Graph::NextAdjVex(int start, int now){
32     int flag = 0;
33     for (int i = 0; i < G.n; i++){
34         if (flag && G.edges[start][i]) return i;
35         if (i == now && G.edges[start][i] && !flag) flag = 1;
36     }
37     return NULL;
38 }
39 void Graph::DFSTraverse(){
40     memset(visited, 0, sizeof(visited));
41     for (int i = 0; i < G.n; i++)
42         if (!visited[i])
43             DFS(i);
44 }
45 void Graph::DFS(int index){
46     printf("%c", G.v[index]);
47     visited[index] = 1;
48     for (int i = 0; i < G.n; i++)
49         if (!visited[i] && G.edges[index][i])
50             DFS(i);
51 }
52 void Graph::BFSTraverse(){
53     memset(visited, 0, sizeof(visited));
54     for (int i = 0; i < G.n; i++){
55         if (!visited[i])
56             BFS(i);
57     }
58 }
```

```
59 void Graph::BFS(int index){
60     int queue[999], rear = 0, front = 0;
61     queue[++rear] = index;
62     visited[index] = 1;
63     while (front != rear){
64         int first = queue[front++];
65         printf("%d ", first);
66         for (int i = 0; i < G.n; i++){
67             if (!visited[i] && G.edges[index][i]){
68                 visited[i] = 1;
69                 queue[++rear] = i;
70             }
71         }
72 
73     }
74 }
75 void Graph::InsertVex(char ch){
76     G.v[G.n++] = ch;
77 }
78 void Graph::InsertArc(int start, int end){
79     G.edges[start][end] = 1;
80     G.edges[end][start] = 1;
81 }
82 void Graph::DeleteVex(int index){
83     for (int i = index; i < G.n - 1; i++){
84         G.v[i] = G.v[i + 1];
85     }
86     G.n--;
87 }
88 void Graph::DeleteArc(int start, int end){
89     G.edges[start][end] = 0;
90     G.edges[end][start] = 0;
91 }
```

```
1 #ifndef Graph_H
2 #define Graph_H
3 class Graph{
4 public:
5     typedef struct MGrape
6     {
7         int edges[999][999];
```

```
        int n, e;
        char v[999];
    } MGrape;
    MGrape G;
    int visited[999];
    int CreateGraph();
    void DestoryGraph();
    char GetVex(int);
    int FirstAdjVex(int);
    int NextAdjVex(int, int);
    void DFSTraverse();
    void DFS(int);
    void BFSTraverse();
    void BFS(int);
    void InsertVex(char);
    void InsertArc(int, int);
    void DeleteVex(int);
    void DeleteArc(int, int);
};
#endif
```

```
#include <iostream>
using namespace std;
#define INF 2147483647
#define maxn 20000

struct Edge{
    int v;
    int len;
    int next;
} edge[1111111];
int head[maxn], cnt = 0;
int n, m, s, from, to, len;
bool visit[maxn];
long long dis[maxn];

void addedge(int from, int to, int len){
    edge[++cnt].v = to;
    edge[cnt].len = len;
    edge[cnt].next = head[from];
    head[from] = cnt;
```

```
}

void dijkstra(){
    for (int i = 1; i <= n; i++)
        dis[i] = INF;
    int temp = s;
    dis[temp] = 0;
    long long minn;
    while (!visit[temp])
    {
        visit[temp] = true;
        for (int j = head[temp]; j; j = edge[j].next)
            if (!visit[edge[j].v] && dis[edge[j].v] > dis[temp] + edge[j
                ].len)
                dis[edge[j].v] = dis[temp] + edge[j].len;
        minn = INF;
        for (int j = 1; j <= n; j++)
            if (!visit[j] && minn > dis[j])
                minn = dis[j], temp = j;
    }
}

int main(){
    freopen("2", "r", stdin);
    cin >> n >> m >> s;
    for (int i = 0; i < m; i++){
        cin >> from >> to >> len;
        addedge(from, to, len);
    }
    dijkstra();
    for (int i = 1; i <= n; i++){
        cout << dis[i] << " ";
    }
    return 0;
}
```

```
#include <stdio.h>
int m, n, A, T, D, E, sum, state[999];
int main(){
    freopen("3CSP1", "r", stdin);
    scanf("%d", &n);
```

```
    for (int i = 0; i < n; i++){
        scanf("%d", &m);
        sum = 0;
        for (int j = 0; j < m; j++){
            scanf("%d", &A);
            if (A > 0){
                if (j != 0 && sum != A) D++, state[i]++;
                sum = A;
            }
            else sum += A;
        }
        T += sum;
    }
    for (int i = 0; i < n; i++)
        if (state[i] && state[(i + 1) % n] && state[(i + 2) % n]) E++;
    printf("%d %d %d", T, D, E);
}
```

```
#include <algorithm>
#include <cstdio>
using namespace std;
int n, m, f[5555], sum, line;
int root(int x){
    return f[x] == x ? x : f[x] = root(f[x]);
}
struct tree{
    int x, y, z;
} t[222222];

bool cmp(tree x, tree y){
    return x.z < y.z;
}
int main(){
    freopen("4CSP", "r", stdin);
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++)
        scanf("%d%d%d", &t[i].x, &t[i].y, &t[i].z);
    for (int i = 1; i <= n; i++)
        f[i] = i;
    sort(t + 1, t + m + 1, cmp);
    for (int i = 1; i <= m; i++){
```

```
        int x = t[i].x;
        int y = t[i].y;
        if (root(x) == root(y))
            continue;
        f[root(x)] = root(y);
        sum += t[i].z;
        line++;
    }
    if (line == n - 1)
        printf("%d", sum);
    else
        printf("orz");
}
```

（四）测试数据及其结果

```
1 3 7 6 2 4 5
```

代码 57: 1DFS.cpp

```
1 3 7 6 2 4 5
```

代码 58: 1BFS.cpp

```
0 1 1 2 3 2
```

代码 59: 2Dijkstra.cpp

```
222 1 0
```

代码 60: 3CSP.cpp

```
6
```

代码 61: 4CSP.cpp

八、 Lab8

（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

```
   \--> 62
   |--> 58
   | |--> 47
   |   |--> 35
   |   | |--> 29
   |   | \--> 37
   |   |   |--> 36
   |   \--> 51
   |     |--> 49
   |     | |--> 48
   |     | \--> 50
   |     \--> 56
   \--> 88
     |--> 73
     \--> 99
       |--> 93
\--> 62
   |--> 58
   | |--> 37
   |   |--> 35
   |   | |--> 29
   |   | \--> 36
   |   \--> 51
   |     |--> 49
   |     | |--> 48
   |     | \--> 50
   |     \--> 56
   \--> 88
     |--> 73
     \--> 99
       |--> 93
```

代码 62: 1BinarySortTree.cpp

所有的排序结果都是：

```
   62 58 88 47 73 99 35 51 93 29 37 49 56 36 48 50
   29 35 36 37 47 48 49 50 51 56 58 62 73 88 93 99
```

```
   0 0
   1 2
   2 1
```

```
4       3 0
5       4 0
```

代码 63: 5CSP.cpp

（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 1: 实验截图