

数据结构作业第二章、第三章

庄震丰 22920182204393

Sept. 30th, 2019

2-21

题目要求：将顺序表的元素倒置。

算法分析：直接将头尾元素交换即可。

2-21.cpp

```
1 #include<bits/stdc++.h>
2 #define MAXN 1000
3 using namespace std;
4 int n;
5 int a[MAXN];
6 int main()
7 {
8     cin>>n;
9     for (int i=1;i<=n;i++)
10         cin>>a[i];
11     for (int i=1;i<=n/2;i++)
12         swap(a[i],a[n-i+1]);
13     for (int i=1;i<=n;i++)
14         cout<<a[i]<<" ";
15     return 0;
16 }
```

2-29

题目要求：将顺序表 A 中元素在 B 和 C 中的元素删除。

算法分析：对 A 和 B 和 C 顺序表进行遍历，每次指向比 A 中元素小或者相等的元素，符合判断要求的则标记当前元素，最后再删除。

时间复杂度 $O(n)$, 空间复杂度 $O(n)$

2-29.cpp

```
1 #include<bits/stdc++.h>
2 #define MAXN 1000
3 using namespace std;
4 int a[MAXN]={0},b[MAXN]={0},c[MAXN]={0},a_new[MAXN]={0};
5 bool dela[MAXN];
6 void init(int Na,int Nb,int Nc)
```

```
7 {
8     for (int i=1;i<=Na;i++)
9         cin>>a[i];
10    for (int i=1;i<=Nb;i++)
11        cin>>b[i];
12    for (int i=1;i<=Nc;i++)
13        cin>>c[i];
14    memset(dela,sizeof(dela),false);
15 }
16 int main()
17 {
18     int na,nb,nc;
19     cin>>na>>nb>>nc;
20     init(na,nb,nc);
21     int pb=1,pc=1;
22     int cnt=0;
23     cout<<endl;
24     for (int i=1;i<=na;i++)
25     {
26         while(b[pb]<a[i]) ++pb;
27         while(c[pc]<a[i]) ++pc;
28         cout<<i<<" "<<pb<<" "<<pc<<endl;
29         if (a[i]==b[pb]&&a[i]==c[pc])
30         {
31             dela[i]=true; 标记删除的点//
32             cnt++;
33         }
34     }
35     int *p=a+1;
36     for (int i=1;i<=na-cnt;i++)
37     {
38         while (dela[p-a]) p++;
39         a_new[i]=*p;p++;
40     }
41     for (int i=1;i<=na-cnt;i++)
42         cout<<a_new[i]<<" ";
43     return 0;
44 }
```

2-30

题目要求：在链表数据结构中解决问题。

题目分析：建立双向链表，当前元素符合要求时利用 pre 指针和 next 指针进行删除当前元素，不用再进行链表头尾特判。

时间复杂度 $O(n)$, 空间复杂度 $O(n)$

2-30.cpp

```
1 #include<bits/stdc++.h>
2 #define MAXN 1000
3 using namespace std;
4 struct node
5 {
6     int data;
7     node* next;
8     node* pre;
9 };
10 node *heada,*headb,*headc;
11 node* build(int N)
12 {
13     int tmp;
14     node* head;
15     node* p1;
16     node* p2;
17     head=(node *) malloc(sizeof(node));
18     cin>>tmp;
19     head->data=tmp;
20     head->pre=NULL;
21     p1=head;
22     for (int i=1;i<=N-1;i++)
23     {
24         p2=(node *) malloc(sizeof(node));
25         cin>>tmp;
26         p2->data=tmp;
27         p1->next=p2;
28         p2->pre=p1;
29         p1=p2;
30     }
31     p1->next=NULL;
32     return head;
33 }
34 void lstdel()
35 {
36     node *pa=heada,*pb=headb,*pc=headc;
37     while (pa!=NULL)
```

```
38     {
39         while (pb->data<pa->data&&pb->next!=NULL) pb=pb->
            next;
40         while (pc->data<pa->data&&pc->next!=NULL) pc=pc->
            next;
41         if (pa->data==pb->data&&pa->data==pc->data)
42         {
43             if (pa==heada)
44             {
45                 heada->next->pre=NULL;
46                 node *ha=heada;
47                 heada=pa->next;
48                 pa=pa->next;
49                 free(ha);
50             }
51             else if (pa->next==NULL)
52             {
53                 pa->pre->next=NULL;
54                 free(pa);
55                 break;
56             }
57             else
58             {
59                 node *p=pa;
60                 pa->pre->next=pa->next;
61                 pa->next->pre=pa->pre;
62                 pa=pa->next;
63                 free(p);
64             }
65         }
66         else
67         {
68             pa=pa->next;
69         }
70     }
71 }
72 void getlst(node * Ha)
73 {
74     while (Ha!=NULL)
75     {
76         printf("%d ",Ha->data);
77         Ha=Ha->next;
```

```
78     }
79 }
80 int main()
81 {
82     int na,nb,nc;
83     cin>>na>>nb>>nc;
84     heada=build(na);
85     headb=build(nb);
86     headc=build(nc);
87     lstdel();
88     getlst(heada);
89     return 0;
90 }
```

2-39

题目描述：计算稀疏多项式。

算法分析：先计算好 $x \dots x^{e(m)}$ ，再针对每项进行相加操作。

时间复杂度 $O(n)$, 空间复杂度 $O(n)$

2-39.cpp

```
1 #include<bits/stdc++.h>
2 #define MAXN 100
3 using namespace std;
4 long long x[100]={1};
5 long long c[100]={0};
6 long long e[100]={0};
7 int main()
8 {
9     long long p=0,x0,m;
10    cin>>x0>>m;
11    for (int i=1;i<=m;i++)
12        cin>>c[i];
13    for (int i=1;i<=m;i++)
14        cin>>e[i];
15    for (int i=1;i<=e[m];i++)
16        x[i]=x[i-1]*x0;
17    for (int i=1;i<=m;i++)
18        p+=c[i]*x[e[i]];
19    cout<<p;
20    return 0;
```

21 }

3-15

题目要求：双向栈，实现初始化，push 和 pop 操作。

算法分析：模拟单向栈，当 push 是 0 栈和 1 栈重合，或者栈为空时，返回 ERROR。

时间复杂度 $O(n)$ ，空间复杂度 $O(n)$. 3-15.cpp

```
1  #include<bits/stdc++.h>
2  #define MAXN 100
3  using namespace std;
4  int stacklength;
5  int a[MAXN栈数组]={0};//
6  int *p0,*p1栈顶指针;//
7  void inistack()初始化栈//
8  {
9      memset(a,sizeof(a),0);
10     p0=a,p1=a+stacklength;
11 }
12 int pushtwi(int i,int x)在栈顶插入元素//
13 {
14     if (i)
15     {
16         if (p1-p0>1)
17         {
18             p1--;
19             *p1=x;
20             return 1;
21         }
22         else return -1;
23     }
24     else
25     {
26         if (p1-p0>1)
27         {
28             p0++;
29             *p0=x;
30             return 1;
31         }
32         else return -1;
33     }
```

```
34
35 }
36 int poptwi(int i)弹出栈顶元素//
37 {
38     if (i)
39     {
40         if (a+stacklength>p1)
41         {
42             int num=*p1;
43             *p1=0;
44             p1++;
45             return num;
46         }
47         else return -1;
48     }
49     else
50     {
51         if (p0>a)
52         {
53             int num=*p0;
54             *p0=0;
55             p0--;
56             return num;
57         }
58         else return -1;
59     }
60 }
61 void opstack()//或者操作poppush
62 {
63     int opnum;
64     int statupush,statupop,l,X,op;
65     cout<<"input the operator times:";
66     cin>>opnum;
67     for (int i=1;i<=4;++i)
68     {
69         cin>>l>>X>>op;
70         if (op)
71         {
72             statupush=pushtwi(l,X);cout<<statupush<<" ";
73         }
74         else
75         {
```

```

76             statupop=poptwi(l);
77             cout<<statupop<<" ";
78         }
79     }
80 }
81 int main()
82 {
83     cin>>stacklength;
84     inistack();
85     return 0;
86 }

```

3-19

题目要求：括号序列匹配

算法分析：建立栈，当栈为空，或者当前元素与栈顶元素不匹配就 push，否则元素不插入，并且 pop 栈顶元素

时间复杂度 $O(n)$, 空间复杂度 $O(n)$.

3-19.cpp

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  stack <int> a定义栈;//
4  bool match_braket(char s1,char s2)判断栈顶元素与当前元素是否匹配//
5  {
6      if (s1=='{'&& s2=='}') return true;
7      if (s1=='['&& s2==']') return true;
8      if (s1=='('&& s2==')') return true;
9      return false;
10 }
11 int main()
12 {
13     string c;
14     cin>>c;
15     while(!a.empty())
16         a.pop();
17     for (int i=0;i<c.length();i++)
18     {
19         if (a.size()==0)
20         {
21             a.push(c[i]) 若栈为空则直接;// push
22             continue;

```



```

23         }
24         if (match_braket(a.top(), c[i])) 如果当前元素栈顶元素匹配则弹出//
25             a.pop();
26         else a.push(c[i]);
27     }
28     if (!a.empty())
29         cout<<"The brackets don't match";
30     else cout<<"The brackets match";
31     return 0;
32 }

```

3-28

题目要求：实现循环链表队列的 init, push, pop 操作。

算法分析：首先建立循环链表，给定 head 和 rear 指针，当插入元素数目超过给定范围时，从 rear 指针下删除队尾元素。

时间复杂度 $O(n)$, 空间复杂度 $O(n)$.

3-28.cpp

```

1     #include <stdio.h>
2     #include <stdlib.h>
3     #define ERROR 0
4     #define OK 1
5     #define OVERFLOW 0
6     typedef int qelemType;
7     typedef struct queue
8     {
9         qelemType data;
10        struct queue *next;
11    }queue,*linkqueue;
12    typedef struct
13    {
14        linkqueue rear;
15        int length;
16    }sqqueue;
17    void initQueue(sqqueue &queue)置空队列//
18    {
19        queue.rear=(linkqueue)malloc(sizeof(queue));
20        queue.rear->next=queue.rear;
21    }
22

```

```
23     int emptyQueue(sqqueue &queue)判队列是否为空//
24     {
25         if(queue.rear->next==queue.rear)
26             return OK;
27         else
28             return 0;
29     }
30     int enqueue(sqqueue &queue, qelemType e)
31     {
32         linkqueue p;
33         p=(linkqueue) malloc( sizeof(queue));
34         if(!p)
35             return OVERFLOW;
36         p->data=e;
37         p->next=queue.rear->next;
38         queue.rear->next=p;
39         queue.rear=p;
40         return OK;
41     }
42     int delqueue(sqqueue &queue, qelemType &e)
43     {
44         linkqueue p;
45         if(queue.rear->next==queue.rear)
46             return ERROR若队列为空返回;//0
47         p=queue.rear->next->next循环链表队列队尾指针下一结点也即头结
            点;//()的下一结点即队头指针()
48         e=p->data;
49         queue.rear->next->next=p->next;
50         free(p);
51         return OK;
52     }
53     int main()
54     {
55         sqqueue queue2;
56         qelemType num;
57         initQueue(queue2);
58         if(emptyQueue(queue2))
59             printf该队列目前为空("\n");
60         else
61             printf该队列不为空("\n");
62         for(int i=1;i<=10;i++)
63             if(enqueue(queue2,i))
```

```

64         printf元素("%成功入列d!\n",i);
65     printf("\n\n");
66     for(int j=1;j<=9;j++)
67         if(delqueue(queue2,num))
68             printf元素("%成功出列d!\n",num);
69     if(emptyQueue(queue2))
70         printf该队列目前为空("!\n");
71     else
72         printf该队列不为空("!\n");
73     return 0;
74 }

```

3-32

题目要求：k 阶斐波那契数列。

算法分析：首先明确 k 阶斐波那契数列的定义。

$$f_0 = 0, f_1 = 0, f_2 = 0, \dots, f_{k-2} = 0, f_{k-1} = 0, f_k = 1.$$

$$f_i = \sum_{j=i-k-1}^{i-1} f_j \quad (1)$$

$$f_{i-1} = \sum_{j=i-k-2}^{i-2} f_j$$

所以，可以递推得到，

$$f_i = 2 \cdot f_{i-1} - f_{i-1-k} \quad (2)$$

而 f_{i-1} 与 f_{i-1-k} 分别是队头和队尾元素。用封装好的队列 push 和 pop 函数即可。

时间复杂度 $O(n)$, 空间复杂度即队列复杂度 $O(n)$.

3-32.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 queue<int> a;
4 int main()
5 {
6     int n,k;
7     \\阶斐波那契kf0=0,f1=0,...fk-2=0,fk-1=0;
8     for (int i=0;i<k-1;i++)
9         a.push(0);
10        a.push(1);
11    for (int i=k;i<=n;i++)

```

```
12         {
13             a.push(2*a.front()-a.back()); // fi=2*f(i-1)-f(i-k-1)
                分别为队头和队
                尾
14             if (a.size()>k) a.pop();
15         }
16     cout<<a.front();
17 }
```

马踏棋盘解题报告

2019.09.30

1 题目描述

将马随机放置再国际象棋 8×8 棋盘 `Board[8][8]` 的某个方格中，马按走棋的规则进行移动。要求每个方格只进入一次，走遍棋盘上全部 64 个方格，求出马的行走路线，并按要求输出行走路线，将 1, 2, ..., 64 依次填入一个 8×8 的方阵，输出之。

1.1 测试数据

自行指定一个马的初始位置 (i,j) ，得到路径结果。

1.2 实现提示

一般来说，马位于 (i,j) 时，可以走到以下八个位置之一。

$(i-2, j-1), (i-2, j+1), (i+2, j-1), (i+2, j+1), (i-1, j+2), (i-1, j-2), (i+1, j+2), (i+1, j-2)$

但是，当以上位置超出边界或者已经走过，就只能选择回溯。

2 算法分析

栈操作（深搜）+ 贪心算法。

每到达一个点试探八个方向是否能走，对于能进行的方向判断该方向能够继续走的方向，取其中子方案数最小的方向进行，这样最容易到达搜索树的根节点。

直到能够走完棋盘为止。

3 程序及运行结果

3.1 程序代码

horse.cpp

```
1 #include <bits/stdc++.h>
2 #define N 8
3 #define INF 999999999
4 using namespace std;
```

```
5
6 int dx[9]={0,2,2,-2,-2,1,1,-1,-1};
7 int dy[9]={0,-1,1,1,-1,-2,2,-2,2};
8 bool MAP[N+1][N+1];
9 int sign[N+1][N+1]={0};
10 void print()
11 {
12     for (int i=1;i<=N;i++)
13     {
14         for (int j=1;j<=N;j++)
15             cout<<sign[i][j]<<" ";
16         cout<<endl;
17     }
18 }
19 bool judge(int i,int j)
20 {
21     if (!MAP[i][j]&& i>0&&i<N+1&&j>0&&j<N+1)
22         return true;
23     else return false;
24 }
25 int judgepath(int x_,int y_)
26 {
27     int c;
28     int tag=0;
29     for (c = 0; c < 8; c++)
30     {
31         if (judge(x_ + dx[c], y_ + dy[c]))
32         {
33             tag++;
34         }
35     }
36     return tag;
37 }
38 void walk(int x,int y,int cnt)
39 {
40     cout<<x<<" "<<y<<endl;
41     if (cnt==N*N)
42     {
43         print();
44         exit(0);
45     }
46     int MIN=INF,rec=-1,rec2=0;
```

```
47     for (int i=1;i<=8;i++)
48     {
49         if (!judge(x+dx[i],y+dy[i]))
50             continue;
51         int path=judgepath(x+dx[i],y+dy[i]);
52         if (MIN>path)
53         {
54             MIN=path;
55             if (rec) rec2=rec;
56             rec=i;
57         }
58     }
59     if (MIN!=INF)
60     {
61         sign[x+dx[rec]][y+dy[rec]]=cnt+1;
62         MAP[x+dx[rec]][y+dy[rec]]=true;
63         walk(x+dx[rec],y+dy[rec],cnt+1);
64         MAP[x+dx[rec]][y+dy[rec]]=false;
65         sign[x+dx[rec]][y+dy[rec]]=0;
66     }
67     return;
68 }
69 void init()
70 {
71     int x0,y0;
72     memset(MAP,sizeof(MAP),false);
73     cin>>x0>>y0;
74     sign[x0][y0]=1;
75     MAP[x0][y0]=true;
76     walk(x0,y0,1);
77 }
78 int main()
79 {
80     init();
81     return 0;
82 }
```

3.2 样例输入输出

样例输入 1:

2 3

样例输出 1:

2 57 30 37 16 47 14 39
29 36 1 56 61 38 17 46
58 3 60 31 48 15 40 13
35 28 53 62 55 64 45 18
4 59 34 49 32 19 12 41
27 52 25 54 63 44 9 20
24 5 50 33 22 7 42 11
51 26 23 6 43 10 21 8

样例输入 2:

3 4

样例输出 2:

36 45 16 13 22 59 18 11
15 2 37 44 17 12 21 60
46 35 14 1 58 23 10 19
3 38 51 54 43 20 61 24
34 47 42 57 52 55 28 9
39 4 53 50 31 62 25 64
48 33 6 41 56 27 8 29
5 40 49 32 7 30 63 26

4 总结与思考

对于没有优化的搜索，不难求得时间复杂度达到 8^{64} ，显然难以在短时间得到结果。因此贪心优化利用后继数最少压缩搜索量，较快得到一组可行解，但是得到的解数目有限，由于没有完全搜索，这是本算法的弊端所在。