# 数据结构作业第二章、第三章

庄震丰 22920182204393

Sept. $30^{th}$, 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

```cpp
#include<bits/stdc++.h>
#define MAXN 1000
using namespace std;
int n;
int a[MAXN];
int main()
{
        cin>>n;
    for (int i=1;i<=n;i++)
        cin>>a[i];
    for (int i=1;i<=n/2;i++)
        swap(a[i],a[n-i+1]);
    for (int i=1;i<=n;i++)
    cout<<a[i]<<" ";
    return 0;
}
```

**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

```cpp
#include<bits/stdc++.h>
#define MAXN 1000
using namespace std;
int a[MAXN]={0},b[MAXN]={0},c[MAXN]={0},a_new[MAXN]={0};
bool dela[MAXN];
void init(int Na,int Nb,int Nc)
{
    for (int i=1;i<=Na;i++)
        cin>>a[i];
    for (int i=1;i<=Nb;i++)
        cin>>b[i];
    for (int i=1;i<=Nc;i++)
        cin>>c[i];
    memset(dela,sizeof(dela),false);
}
int main()
{
    int na,nb,nc;
```

```
    cin>>na>>nb>>nc;
    init(na,nb,nc);
    int pb=1,pc=1;
    int cnt=0;
    cout<<endl;
    for (int i=1;i<=na;i++)
    {
        while(b[pb]<a[i]) ++pb;
        while(c[pc]<a[i]) ++pc;
        cout<<i<<" "<<pb<<" "<<pc<<endl;
        if (a[i]==b[pb]&&a[i]==c[pc])
            {
                dela[i]=true; //标记删除的点
                cnt++;
            }
    }
    int *p=a+1;
    for (int i=1;i<=na-cnt;i++)
    {
        while (dela[p-a]) p++;
        a_new[i]=*p;p++;
    }
    for (int i=1;i<=na-cnt;i++)
        cout<<a_new[i]<<" ";
    return 0;
}
```

**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

```
#include<bits/stdc++.h>
#define MAXN 1000
using namespace std;
struct node
{
    int data;
    node* next;
    node* pre;
};
node *heada,*headb,*headc;
node* build(int N)
{
    int tmp;
    node* head;
    node* p1;
    node* p2;
    head=(node *)malloc(sizeof(node));
    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  }
```

```
void getlst(node * Ha)
{
    while(Ha!=NULL)
    {
        printf("%d ",Ha->data);
        Ha=Ha->next;
    }
}
int main()
{
    int na,nb,nc;
    cin>>na>>nb>>nc;
    heada=build(na);
    headb=build(nb);
    headc=build(nc);
    lstdel();
    getlst(heada);
    return 0;
}
```

**2-39**

题目描述：计算稀疏多项式。

算法分析：先计算好 $x..x^{e(m)}$，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

```
#include<bits/stdc++.h>
#define MAXN 100
using namespace std;
long long x[100]={1};
long long c[100]={0};
long long e[100]={0};
int main()
{
    long long p=0,x0,m;
    cin>>x0>>m;
    for (int i=1;i<=m;i++)
        cin>>c[i];
    for (int i=1;i<=m;i++)
        cin>>e[i];
    for (int i=1;i<=e[m];i++)
        x[i]=x[i-1]*x0;
    for (int i=1;i<=m;i++)
        p+=c[i]*x[e[i]];
    cout<<p;
    return 0;
}
```

**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

```cpp
#include<bits/stdc++.h>
#define MAXN 100
using namespace std;
int stacklength;
int a[MAXN]={0};//栈数组
int *p0,*p1;//栈顶指针
void inistack()//初始化栈
{
    memset(a,sizeof(a),0);
    p0=a,p1=a+stacklength;
}
int pushtwi(int i,int x)//在栈顶插入元素
{
    if (i)
    {
        if (p1-p0>1)
            {
                p1--;
                *p1=x;
                return 1;
            }
        else return -1;
    }
    else
    {
        if (p1-p0>1)
        {
            p0++;
            *p0=x;
            return 1;
        }
        else return -1;
    }

}
int poptwi(int i)//弹出栈顶元素
{
    if (i)
    {
        if (a+stacklength>p1)
        {
            int num=*p1;
            *p1=0;
            p1++;
            return num;
        }
        else return -1;
    }
    else
    {
        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,I,X,op;
65      cout<<"input the operator times:";
66      cin>>opnum;
67      for (int i=1;i<=4;++i)
68      {
69          cin>>I>>X>>op;
70          if (op)
71              {
72                  statupush=pushtwi(I,X);cout<<statupush<<" ";
73              }
74          else
75              {
76                  statupop=poptwi(I);
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()
```

```
{
    string c;
    cin>>c;
    while(!a.empty())
        a.pop();
    for (int i=0;i<c.length();i++)
    {
        if (a.size()==0)
            {
                a.push(c[i]);//若栈为空则直接push
                continue;
            }
        if (match_braket(a.top(),c[i]))//如果当前元素栈顶元素匹配则弹出
            a.pop();
            else a.push(c[i]);
    }
    if (!a.empty())
        cout<<"The brakets don't match";
        else cout<<"The brakets match";
return 0;
}
```

**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

```
#include <stdio.h>
#include <stdlib.h>
#define ERROR 0
#define OK 1
#define OVERFLOW 0
typedef int qelemType;
typedef struct queue
{
    qelemType data;
    struct queue *next;
}queue,*linkqueue;
typedef struct
{
    linkqueue rear;
    int length;
}sqqueue;
void initQueue(sqqueue &queue)//置空队列
{
    queue.rear=(linkqueue)malloc(sizeof(queue));
    queue.rear->next=queue.rear;
}

int emptyQueue(sqqueue &queue)//判队列是否为空
```

```
{
    if(queue.rear->next==queue.rear)
        return OK;
    else
        return 0;
}
int enqueue(sqqueue &queue,qelemType e)
{
    linkqueue p;
    p=(linkqueue)malloc(sizeof(queue));
    if(!p)
        return OVERFLOW;
    p->data=e;
    p->next=queue.rear->next;
    queue.rear->next=p;
    queue.rear=p;
    return OK;
}
int delqueue(sqqueue &queue,qelemType &e)
{
    linkqueue p;
    if(queue.rear->next==queue.rear)
        return ERROR;//若队列为空返回0
    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()
    e=p->data;
    queue.rear->next->next=p->next;
    free(p);
    return OK;
}
int main()
{
    sqqueue queue2;
    qelemType num;
    initQueue(queue2);
    if(emptyQueue(queue2))
        printf("该队列目前为空!\n");
    else
        printf("该队列不为空!\n");
    for(int i=1;i<=10;i++)
        if(enqueue(queue2,i))
            printf("元素%成功入列d!\n",i);
    printf("\n\n");
    for(int j=1;j<=9;j++)
        if(delqueue(queue2,num))
            printf("元素%成功出列d!\n",num);
    if(emptyQueue(queue2))
        printf("该队列目前为空!\n");
    else
        printf("该队列不为空!\n");
    return 0;
}
```

**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。

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

$$\begin{aligned} f_i &= \sum_{j=i-k-1}^{i-1} f_j \\ f_{i-1} &= \sum_{j=i-k-2}^{i-2} f_j \end{aligned} \tag{1}$$

所以，可以递推得到，

$$f_i = 2 \cdot f_{i-1} - f_{i-1-k} \tag{2}$$

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

```cpp
#include <bits/stdc++.h>
using namespace std;
queue <int> a;
int main()
{
    int n,k;
    \\阶斐波那契kf0=0,f1=0,...fk-2=0,fk-1=0;
    for (int i=0;i<k-1;i++)
        a.push(0);
        a.push(1);
    for (int i=k;i<=n;i++)
        {
            a.push(2*a.front()-a.back());//fi=2*f(i-1)-f(i-k-1) 分别为队头和队尾
            if (a.size()>k) a.pop();
        }
    cout<<a.front();
}
```

# 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*times*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，$\cdots$,64 依次填入一个 8*times*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

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

int dx[9]={0,2,2,-2,-2,1,1,-1,-1};
int dy[9]={0,-1,1,1,-1,-2,2,-2,2};
bool MAP[N+1][N+1];
int sign[N+1][N+1]={0};
void print()
{
    for (int i=1;i<=N;i++)
        {
            for (int j=1;j<=N;j++)
                cout<<sign[i][j]<<" ";
        cout<<endl;
        }
}
```

```
bool judge(int i,int j)
{
    if (!MAP[i][j]&&i>0&&i<N+1&&j>0&&j<N+1)
    return true;
    else return false;
}
int judgepath(int x_,int y_)
{
   int c;
   int tag=0;
   for (c = 0; c < 8; c++)
   {
      if (judge(x_ + dx[c], y_ + dy[c]))
      {
         tag++;
      }
   }
   return tag;
}
void walk(int x,int y,int cnt)
{
    cout<<x<<" "<<y<<endl;
    if (cnt==N*N)
        {
            print();
            exit(0);
        }
    int MIN=INF,rec=-1,rec2=0;
    for (int i=1;i<=8;i++)
    {
        if (!judge(x+dx[i],y+dy[i]))
              continue;
        int path=judgepath(x+dx[i],y+dy[i]);
            if (MIN>path)
            {
                MIN=path;
                if (rec) rec2=rec;
                rec=i;
            }
    }
    if (MIN!=INF)
    {
                sign[x+dx[rec]][y+dy[rec]]=cnt+1;
                MAP[x+dx[rec]][y+dy[rec]]=true;
                walk(x+dx[rec],y+dy[rec],cnt+1);
                MAP[x+dx[rec]][y+dy[rec]]=false;
                sign[x+dx[rec]][y+dy[rec]]=0;
    }
        return;
}
void init()
{
    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。