线性数据结构

栈，队列，链表

线性结构的特点：所有元素排成一排

在数据元素的非空有限集合中：

- **存在唯一的一个被称作“第一个”的元素；**
- **存在唯一的一个被称作“最后一个”的元素；**
- **除第一个之外，集合中的每个元素都只有一个前驱；**
- **除最后一个之外，集合中的每个元素都只有一个后继。**

以数组为例：int a[100];

第一个元素：a[0]；最后一个元素：a[99]。

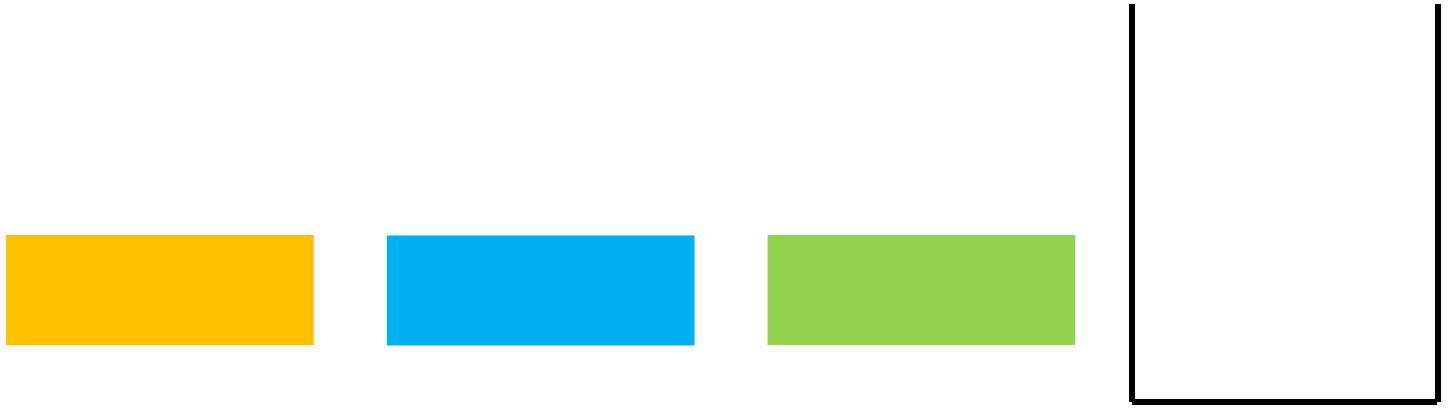
0 < i < 100 时，a[i]的前驱是a[i – 1]；

0 <= i < 99 时，a[i]的后继是a[i + 1]。

栈

栈的定义

- 栈（stack）是一种特殊的线性数据结构，栈中的元素是按照入栈顺序线性的排列。
- 栈的结构如下图所示，仅允许在表的一端进行插入和删除运算，这一端被称为栈顶，相对地，把另一端称为栈底。



- 栈的特点是后进先出（LIFO，Last In First Out），即最后入栈的元素最先出栈。

栈的实现

- 栈本质上就是一个线性表。我们可以用一个数组简单的实现栈的所有操作。

```
int Stack[MAX_SIZE];     //int可以换成其他类型
int top;     //栈顶元素的下标，top==0表示栈为空

bool empty()     //判断栈是否为空
{
    return top==0;
}

void push(int x)     //元素x入栈
{
    top++;
    Stack[top]=x;
}

void pop()   //栈顶元素出栈
{
    top--;   //相当于原来栈顶元素下标-1的元素成为新的栈顶元素
}
```

栈的实现

- 事实上，C++语言的STL库自带了一个栈的实现。我们也可以这样来简单的实现栈。

```
#include <stack>
stack<int> st;  //定义一个数据元素为int的栈
st.empty(); //判断栈是否为空
st.push(x); //元素x入栈
st.top();   //返回栈顶元素
st.pop();   //栈顶元素出栈
```

栈的应用

- 大家可能会问，这种东西的功能限制那么多，比数组还弱，为什么还要学这个？有什么用？

- 实际上，栈在计算机科学中应用十分广泛。
- 比如递归函数的过程。我们在调用函数的时候相当于把函数压到了一个栈里。每次调用栈顶的函数。一个函数结束相当于从栈顶被弹出，然后继续执行栈顶的函数。

例1. 初识栈

```
#include <iostream>
#include <stack>
using namespace std;
int main()
{
    stack<int> s;//定义栈
    s.push(5);s.push(6);s.push(7);//入栈
    s.pop();//出栈
    cout<<"栈顶元素: "<<s.top();
    return 0;
}
```

例2. 判断回文noi.openjudge.cn/ch0107/33/

- “xyzyx”是一个回文字符串，所谓回文字符串就是指正读反读均相同的字符序列，如“aha”和“ahaha”均是回文，但“ahah”不是回文。通过栈这个数据结构我们将很容易判断一个字符串是否为回文。

我们先将字符串中间位置mid之前的部分的字符全部入栈。

```
for(i=0;i<mid;i++)
        s.push(a[i]);
```

接下来进入判断回文的关键步骤。将当前栈中的字符依次出栈，看看是否能与mid之后的字符一一匹配，如果都能匹配则说明这个字符串是回文字符串，否则这个字符串就不是回文字符串。注意分析字符串长度的奇偶性。

STL栈

```
stack<char> s;
char a[100];
int i,mid,len;
gets(a);
len=strlen(a);
mid=len/2;
for(i=0;i<mid;i++) s.push(a[i]);//mid前的字符入栈
if(len%2!=0) mid++;//考虑字符串长度的奇偶性
for(i=mid;i<len;i++)//开始匹配
{
    if(a[i]!=s.top()) break;
    s.pop();
}
if(s.empty()) cout<<"yes";
else cout<<"NO";
```

手工栈

```
char st[109];
char a[100];
int i,mid,len,top;
gets(a);
len=strlen(a);
mid=len/2;
top=0;//栈的初始化
for(i=0;i<mid;i++) st[++top]=a[i];//mid前的字符入栈
if(len%2!=0) mid++;//考虑字符串长度的奇偶性
for(i=mid;i<len;i++)//开始匹配
{
    if(a[i]==st[top]) top--;
    else break;
}
if(top==0) cout<<"yes";
else cout<<"NO";
```

例3. 1113 括号匹配

- 时间限制: 1000 ms 空间限制: 262144 KB 具体限制
- 【题目描述】
- 给定一个只包含左右括号的合法括号序列，按右括号从左到右的顺序输出每一对配对的括号出现的位置(括号序列以0开始编号)。
- 【输入】
- 仅一行，表示一个合法的括号序列。
- 【输出】
- 设括号序列有n个右括号。则输出包括n行，每行两个整数l和r，表示配对的括号左括号出现在第l位，右括号出现在第r位。
- 【样例输入】
- (())()
- 【样例输出】
- 1 2
- 0 3
- 4 5

位置	0	1	2	3	4	5
括号	(	(	)	)	(	)
待匹配的左括号	0	01	0		4	
匹配结果			12	03		45

待匹配的左括号序列：在同一端进行元素的添加和删除。
栈：添加(进栈)、删除(出栈)元素的一端——栈顶，另一端——栈底。后进先出。

手工栈

```
#include <iostream>
using namespace std;
int main()
{
    const int maxl = 110;
    char s[maxl];
    int st[maxl],top=0;
    cin>>s;
    for(int i=0;s[i];i++)
    {
        if(s[i]=='(') st[top++]=i;
        else cout<<st[--top]<<' '<<i<<endl;
    }
    return 0;
}
```

STL栈

```
const int maxl = 110;
char s[maxl];
stack<int> st;
cin>>s;
for(int i=0;i<strlen(s);i++)
{
    if(s[i]=='(')
        st.push(i);
    else
    {
        cout<<st.top()<<' '<<i<<endl;
        st.pop();
    }
}
```

例4. P1739 表达式括号匹配

【题目描述】
假设一个表达式有英文字母（小写）、运算符（+，-，*，/）和左右小（圆）括号构成，以"@"作为表达式的结束符。请编写一个程序检查表达式中的左右圆括号是否匹配，若匹配，则返回"YES"；否则返回"NO"。表达式长度小于255，左圆括号少于20个。
【输入输出格式 】
输入格式：
一行：表达式
输出格式：
一行："YES" 或 "NO"

输入样例1：2*(x+y)/(1-x)@
输出样例1：YES
输入样例2：(25+x)*(a*(a+b+b)@
输出样例2：NO

【说明】
表达式长度小于255，左圆括号少于20个。

算法分析：

读到左括号进栈，读到右括号，栈顶元素出栈。

不能匹配的情况：

- 匹配完所有的右括号之后，栈不空(存在多余的左括号)；
- 读到右括号，需要栈顶元素出栈时，栈是空的(右括号数量超过左括号)。

```cpp
#include<iostream>
#include<cstdio>
using namespace std;
int main(){
    char c;
    int top=0;
    while ((c=getchar())!='@'){
        if(c=='(') top++;
        if(c==')')
            if(top>0) top--;
            else{
                top=-1;
                break;
            }
    }
    if(top==0) cout<<"YES\n";
    elsec out<<"NO\n";
    return 0;
}
```

```cpp
#include<iostream>
using namespace std;
int main()
{
    char str[300];
    int count=0;
    cin>>str;
    for(int i=0;str[i]!='@';i++)
    {
        if(str[i]=='(') count++;
        else if(str[i]==')') count--;
        if(count<0) break;   //此处是关键
    }
    cout<<(count==0 ? "YES\n" : "NO\n");
    return 0;
}
```

队列

队列的定义

- 队列（queue）是一种特殊的线性数据结构，队列中的元素也是按照入队顺序线性的排列。
- 队列的结构如下图所示，队列只允许在队列的前端（队头）进行删除操作，后端（队尾）进行插入操作。

- 队列的特点是先进先出（FIFO，First In First Out），即最先入队列的元素最先出队列，就和我们平时排队一样。

先进先出FIFO

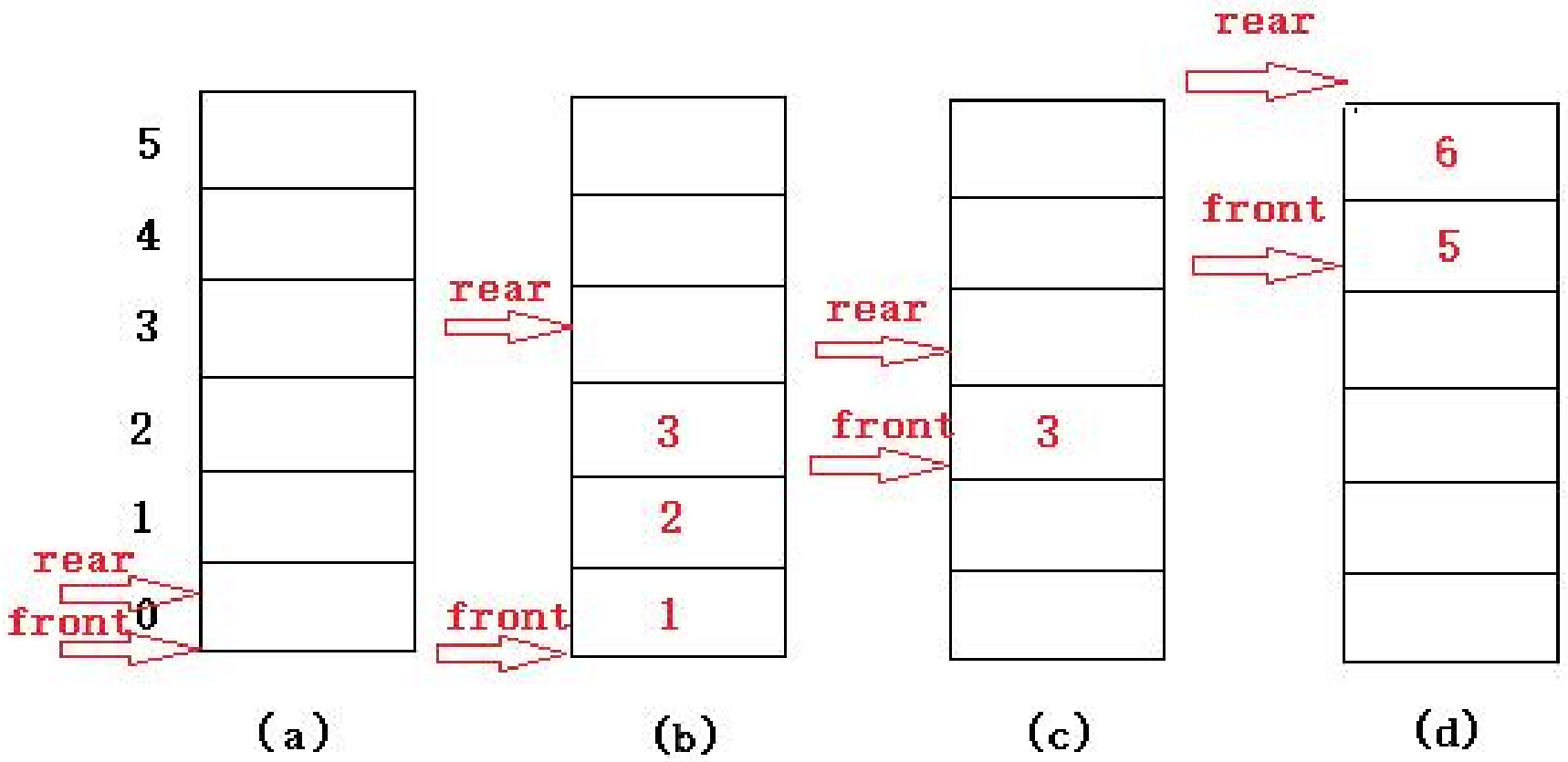
队头删除

队尾插入

队列的实现

- 队列的基本操作包括入队、出队、判断队列是否为空、求队列中元素的个数（队列长度）等。我们可以用一个数组简单的实现队列的所有操作。

```
#define MAX_SIZE 100000 //定义队列的最大容量
int Queue[MAX_SIZE];    //int可以换成其他类型
```

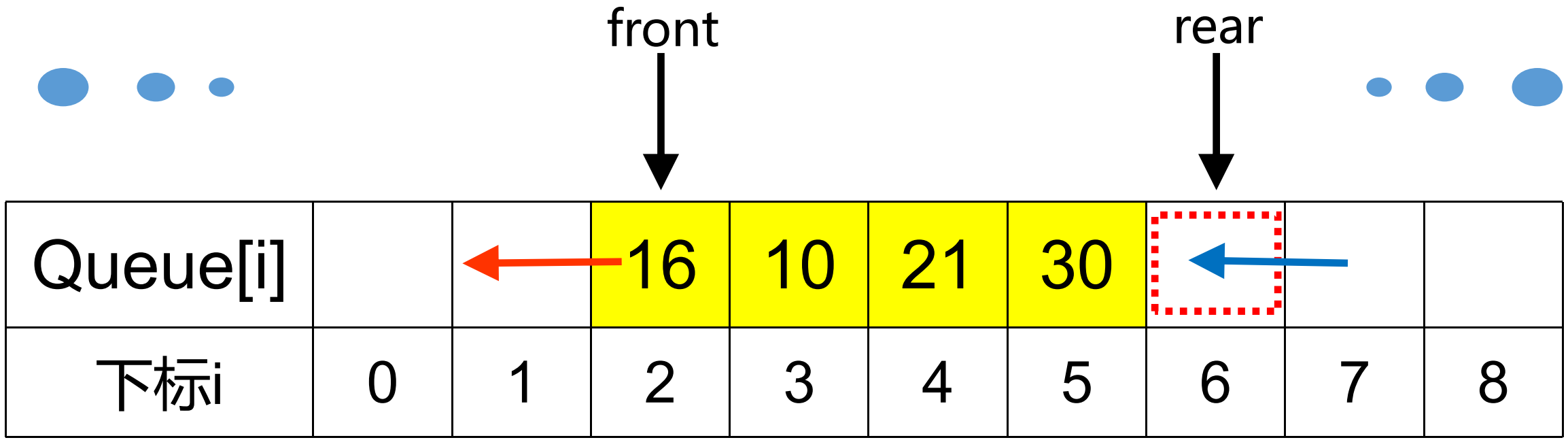


a. 初始建空队列时，令 front = rear = 0

b. 每当插入新的队列尾元素时，“尾指针增1”

c. 每当删除队列头元素时，“头指针增1”。

缺陷：假设当前队列分配的最大空间为6，队列处于图（d）状态时不可在继续插入新的队尾元素---这种现象为“假溢出”



出队(注意顺序)：

t=Queue[front];

front++;

或：t=Queue[front++];

入队(注意顺序)：

Queue[rear]=x;

rear++;

或：Queue[rear++]=x;

队列为空：

front==rear

队列长度：

rear-front

```
#define MAX_SIZE 100000 //定义队列的最大容量
int Queue[MAX_SIZE];    //int可以换成其他类型
int front=0, rear=0;    //front为队头元素的下标，rear为队尾元素后一个位置的下标

void push(int x)     //将元素x入队
{
    Queue[rear]=x;
    rear++;
}

int pop()    //队头元素出队，并返回队头元素的值
{
    int t;
    t=Queue[front];
    front++;
    return t;
}

bool empty()     //判断队列是否为空
{
    return (front==rear);    //当front==rear时，队列为空，即front<rear时，队列非空
}

int size()   //求队列中元素的个数
{
    return (rear-front);
}
```

- 事实上，C++语言的STL库自带了一个队列的实现。我们也可以这样来简单的实现队列。

```
#include <queue>
queue<int> q;//定义一个数据元素为int的队列
q.push(x);   //元素x入队
q.pop();     //队头元素出队，但不会返回元素的值
q.front();   //返回队头元素
q.back();    //返回队尾元素
q.empty();   //判断队列是否为空
q.size();    //返回队列中元素的个数
```

队列的应用

- 队列在计算机科学中应用也十分广泛。
- 队列的主要应用是在BFS（广度优先搜索）中。
- 此外，我们要讲的「单调队列」也是队列的重要用途之一。

例1. 初识队列

```
#include <iostream>
using namespace std;
int main()
{
    int q[100];
    int i,n,front,rear;
    cin>>n;
    for(i=0;i<n;i++) q[i]=i+1;
    front=0;rear=n;
    while(front<rear) cout<<q[front++]<<' ';
    return 0;
}
```

例2. 1332 周末舞会

【题目描述】
假设在周末舞会上，男士们和女士们进入舞厅时，各自排成一队。跳舞开始时，依次从男队和女队的队头上各出一人配成舞伴。规定每个舞曲能有一对跳舞者。若两队初始人数不相同，则较长的那一队中未配对者等待下一轮舞曲。现要求写一个程序，模拟上述舞伴配对问题。

【输入】
第一行两队的人数;
第二行舞曲的数目。

【输出】配对情况。

【输入样例】

```
4 6
7
```

【输出样例】

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

```cpp
#include<iostream>
#include<queue>
using namespace std;
int main(){
    int n1,n2;
    cin>>n1>>n2;
    int m;
    cin>>m;
    queue<int> q1;
    queue<int> q2;
    for(int i=1;i<=n1;i++) q1.push(i);
    for(int i=1;i<=n2;i++) q2.push(i);
    for(int i=0;i<m;i++){
        cout<<q1.front()<<' '<<q2.front()<<endl;
        q1.push(q1.front());
        q2.push(q2.front());
        q1.pop();
        q2.pop();
    }
    return 0;
}
```

```cpp
#include<iostream>
#include<queue>
using namespace std;
int main()
{
    int n1,n2;
    cin>>n1>>n2;
    int m;
    cin>>m;
    int q1[100],q2[100];
    for(int i=1;i<=n1;i++) q1[i]=i;
    for(int i=1;i<=n2;i++) q2[i]=i;
    int f1=1,r1=n1,f2=1,r2=n2;
    for(int i=0;i<m;i++){
        cout<<q1[f1]<<" "<<q2[f2]<<endl;
        q1[++r1]=q1[f1++];
        q2[++r2]=q2[f2++];
    }
    return 0;
}
```

例3. 1334 围圈报数

【题目描述】
有n个人依次围成一圈，从第1个人开始报数，数到第m个人出列，然后从出列的下一个人开始报数，数到第m个人又出列，…，如此反复到所有的人全部出列为止。设n个人的编号分别为1，2，…，n，打印出列的顺序。
【输入】
n和m。
【输出】
出列的顺序。
【输入样例】
4 17
【输出样例】
1 3 4 2

```cpp
#include<iostream>
#include<queue>
using namespace std;
int main(){
    int q[100];
    int f=0,r=0;
    int n,m,now=1;
    cin>>n>>m;
    for(int i=1;i<=n;i++) q[r++]=i;
    while (f<r){
        if(now==m){
            cout<<q[f]<<' ';
            f++;
            now=1;
        }
        else{
            q[r++]=q[f++];
            now++;
        }
    }
    return 0;
}
```

```cpp
#include<queue>
using namespace std;
int main()
{
    queue<int> q;
    int n,m;
    int now=1;
    cin>>n>>m;
    for(int i=1;i<=n;i++) q.push(i);
    while(!q.empty()){
        if(now==m){
            cout<<q.front()<<' ';
            q.pop();
            now=1;
        }
        else{
            q.push(q.front());
            q.pop();
            now++;
        }
    }
    return 0;
}
```

例4. 1111 Blash数集

时间限制: 1000 ms 空间限制: 262144 KB 具体限制

【题目描述】大数学家高斯小时候偶然间发现一种有趣的自然数集合Blash，对应以a为基的集合Ba定义如下：

（1）a是集合Ba的基，且a是Ba的第一个元素；

（2）如果x在集合Ba中，则2x+1和3x+1也都在集合Ba中；

（3）没有其他元素在集合Ba中了。

现在小高斯想知道如果将集合Ba中元素按照升序排列，第n个元素会是多少？

【输入】

输入包含很多行，每行输入包括两个数字，集合的基a(1<=a<=50)以及所求元素序号n(1<=n<=1000000)。

【输出】

对应每个输入，输出集合Ba的第n个元素值。

【样例输入】

1 100

28 5437

【样例输出】

418

900585

【数据范围限制】

1<=n<=1000000，1<=a<=50，数据组数不超过10。

	1	2	3	4	5	6	7	8	9
Blash	1	3	4	7	9	10	13	15	19
2x+1	3	7	9	15	19	21	27	31	39
3x+1	4	10	13	22	28	31	40	46	58

作用：存储暂时还没有处理，且需要按照一定顺序处理的元素。

特点：从一端添加元素，从另一端处理(删除)元素。

概念：队列、先进先出、队尾、队头。

算法分析：

维护三个数组q1,q2,q3;

取q2、q3队头元素的较小者k出队，并加入q1，2*k+1、3*k+1分别加入q2、q3;

直到q1中的元素个数达到n个。

实际上，q2、q3中的元素都来自于q1，实际上，q2、q3中的元素都来自于q1，只要维护head2、head3两个位置，表示q2中的下一个数由q1[head2] *2+1得到，q3中的下一个数由q1[head3]*3+1得到，这样就不需要q2、q3这两个数组了。

特殊情况的处理：q2、q3的队头元素相同。

```cpp
#include <iostream>
using namespace std;
const int M=1000009;
typedef long long ll;
ll q[M];
void work(int a,int n)
{
    q[1]=a;//初始元素入队
    int head2=1,head3=1,tail=1;//初始队列指针，tail是队尾元素下标
    while(tail<=n)
    {
        ll t1=2*q[head2]+1,t2=3*q[head3]+1;
        ll t=min(t1,t2);
        if(t1<t2) head2++;//出队
        else head3++;
        if(t!=q[tail]) q[++tail]=t;//入队，且重复元素不入队
    }
    cout<<q[n]<<endl;
}
int main()
{
    int a,n;
    while(cin>>a>>n) work(a,n);
    return 0;
}
```

例5. P1540 机器翻译

【题目背景 】
小晨的电脑上安装了一个机器翻译软件，他经常用这个软件来翻译英语文章。
【题目描述 】
这个翻译软件的原理很简单，它只是从头到尾，依次将每个英文单词用对应的中文含义来替换。对于每个英文单词，软件会先在内存中查找这个单词的中文含义，如果内存中有，软件就会用它进行翻译；如果内存中没有，软件就会在外存中的词典内查找，查出单词的中文含义然后翻译，并将这个单词和译义放入内存，以备后续的查找和翻译。
假设内存中有M个单元，每单元能存放一个单词和译义。每当软件将一个新单词存入内存前，如果当前内存中已存入的单词数不超过M-1，软件会将新单词存入一个未使用的内存单元；若内存中已存入M个单词，软件会清空最早进入内存的那个单词，腾出单元来，存放新单词。
假设一篇英语文章的长度为N个单词。给定这篇待译文章，翻译软件需要去外存查找多少次词典？假设在翻译开始前，内存中没有任何单词。
【输入输出格式 】
输入格式：
输入文件共2行。每行中两个数之间用一个空格隔开。
第一行为两个正整数M和N，代表内存容量和文章的长度。
第二行为N个非负整数，按照文章的顺序，每个数（大小不超过1000）代表一个英文单词。文章中两个单词是同一个单词，当且仅当它们对应的非负整数相同。
输出格式：
包含一个整数，为软件需要查词典的次数。

【输入输出样例】
输入样例#1：
3 7
1 2 1 5 4 4 1
输出样例#1 ：
5
【说明】
每个测试点1s
对于10%的数据有M=1，N≤5。
对于100%的数据有0<=M<=100，0<=N<=1000。
整个查字典过程如下：每行表示一个单词的翻译，冒号前为本次翻译后的内存状况：
空：内存初始状态为空。
1．1：查找单词1并调入内存。
2．1 2：查找单词2并调入内存。
3．1 2：在内存中找到单词1。
4．1 2 5：查找单词5并调入内存。
5．2 5 4：查找单词4并调入内存替代单词1。
6．2 5 4：在内存中找到单词4。
7．5 4 1：查找单词1并调入内存替代单词2。
共计查了5次词典。

算法分析：

维护内存单元中的单词：状态数组 + 队列。

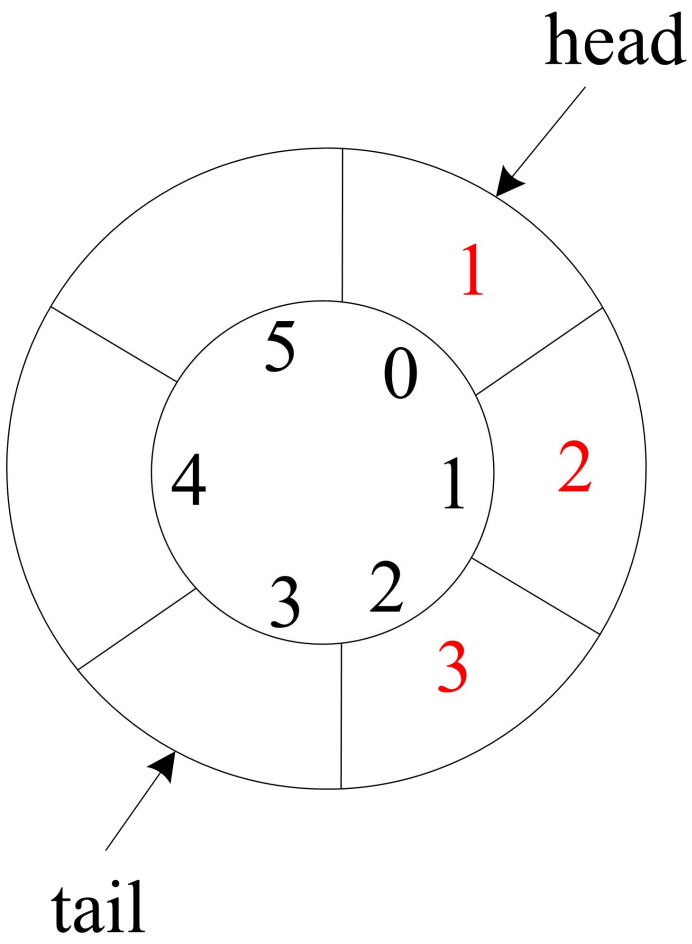
需要查询一个单词时，如果已在队列中，跳过；

否则，存入新单词（如果队列满，先清空最早进入的单词，即队头元素出队）。

M <= 100，N <=1000。

队列中最多有M个元素，但大小须定义到N，存在空间的浪费。

改进：使用循环队列（为了克服“假溢出”，把存储空间想象为一个首尾相接的圆环）



	普通队列	循环队列
出队：	head++;	head=(head+1)%M;
入队：	tail++;	tail=(tail+1)%M;
判空：	head==tail	head==tail
判满：	tail–head==M	(tail+1)%M==head

由于head==tail 可能代表队列空，也可能代表队列满。

改进：声明队列时多申请一个元素的空间

例：M=3; int q[4]; head=2; 时，

队列空： tail=2。

队列满： 队列元素存储在q[2]、q[3]、q[0]，tail=1。

```cpp
#include<iostream>
using namespace std;
int main()
{
    int n,m,a[10001],num,k=0,t=1,i,j;
    cin>>m>>n;
    for(i=1;i<=n;i++)
    {
        cin>>num;
        for(j=1;j<=m;j++)
            if(num==a[j]) break;
        if(j>m)
        {
            k++;
            a[t++]=num;
            if(t>m) t=1;
        }
    }
    cout<<k<<endl;
}
```

```cpp
#include <iostream>
#include <stdio.h>
using namespace std;
int n,m,x,ans,f,r,a[1005],b[1005];
int main() {
    cin>>m>>n;
    f=0;
    r=0;//初始化两个指针
    for (int i=1; i<=n; i++) {
        scanf("%d",&x);//边读入边做
        if (a[x]==0) {
            ans++;
            r++;
            b[r]=x;
            a[x]=1;
            if (r>m) {
                f++;
                a[b[f]]=0;
            }
        }
    }
    cout<<ans;
    return 0;
}
```

例6. 瑞士轮

总时间限制: 1000ms 内存限制: 65535kB

【背景】

在双人对决的竞技性比赛，如乒乓球、羽毛球、国际象棋中，最常见的赛制是淘汰赛和循环赛。前者的特点是比赛场数少，每场都紧张刺激，但偶然性较高。后者的特点是较为公平，偶然性较低，但比赛过程往往十分冗长。 本题中介绍的瑞士轮赛制，因最早使用于 1895 年在瑞士举办的国际象棋比赛而得名。它可以看作是淘汰赛与循环赛的折衷，既保证了比赛的稳定性，又能使赛程不至于过长。

【问题描述】

2*N 名编号为 1~2N的选手共进行 R轮比赛。每轮比赛开始前，以及所有比赛结束后，都会按照总分从高到低对选手进行一次排名。选手的总分为第一轮开始前的初始分数加上已参加过的所有比赛的得分和。总分相同的，约定编号较小的选手排名靠前。 每轮比赛的对阵安排与该轮比赛开始前的排名有关：第 1 名和第 2 名、第 3 名和第 4 名、……、第 2K-1 名和第 2K名、…… 、第 2N-1 名和第2N名，各进行一场比赛。每场比赛胜者得 1 分，负者得 0 分。也就是说除了首轮以外，其它轮比赛的安排均不能事先确定，而是要取决于选手在之前比赛中的表现。

现给定每个选手的初始分数及其实力值，试计算在 R 轮比赛过后，排名第 Q 的选手编号是多少。我们假设选手的实力值两两不同，且每场比赛中实力值较高的总能获胜。

【输入】

输入的第一行是三个正整数 N、R、Q，每两个数之间用一个空格隔开，表示有 2*N 名选手、R 轮比赛，以及我们关心的名次 Q。

第二行是 2*N个非负整数 $s_1, s_2, \ldots, s_{2N}$，每两个数之间用一个空格隔开，其中 s_i 表示编号为 i 的选手的初始分数。

第三行是 2*N个正整数 $w_1, w_2, \ldots, w_{2N}$，每两个数之间用一个空格隔开，其中 w_i 表示编号为 i 的选手的实力值。

【输出】

输出只有一行，包含一个整数，即 R 轮比赛结束后，排名第 Q 的选手的编号。

【样例输入】
2 4 2
7 6 6 7
10 5 20 15
【样例输出】
1
【提示】
对于 30%的数据，$1 \le N \le 100$；
对于 50%的数据，$1 \le N \le 10,000$；
对于 100%的数据，$1 \le N \le 100,000$，$1 \le R \le 50$，$1 \le Q \le 2N$，$0 \le s_1, s_2, \ldots, s_{2N} \le 10^8$，$1 \le w_1, w_2, \ldots, w_{2N} \le 10^8$。

	本轮对阵	本轮结束后的得分			
选手编号	/	①	②	③	④
初始	/	7	6	6	7
第 1 轮	①—④ ②—③	7	6	7	8
第 2 轮	④—① ③—②	7	6	8	9
第 3 轮	④—③ ①—②	8	6	9	9
第 4 轮	③—④ ①—②	9	6	10	9

算法分析：

按比赛规则，每轮比赛前先按总分排序，然后对阵，直到R轮比赛结束。

复杂度：R * (2 * n) * log(2 * n)

= 50 * (2 * 10^5) * log(2 * 10^5)　约10^8。 TLE

优化：1—2、3—4、…、2*i-1—2*i

每组比赛的胜者：赛前，总分是按降序排的；获胜后都得1分，仍是降序；

每组比赛的负者：赛前，总分是按降序排的；不得分，仍是降序。

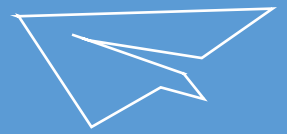
先按初始分数排序，然后按分数高低两人一组比赛；

胜者入队A，负者入队B。这样A、B自身仍是有序的；

合并A、B是O(2*n)的，总复杂度O(R*2*n)=O(10^7) 。

```cpp
#include<iostream>
#include<algorithm>
using namespace std;
const int N=200000+5;
int n,m,q;
int id[N],win[N],lose[N],s[N],w[N];
bool cmp(int a,int b){
    if(s[a]!=s[b]) return s[a]>s[b];
    return a<b;
}
void merge(){
    int i=1,j=1;
    id[0] = 0;
    while(i<=win[0] && j<=lose[0])
        if(cmp(win[i],lose[j])) id[++id[0]]=win[i++];
        else id[++id[0]]=lose[j++];
    while(i<=win[0]) id[++id[0]]=win[i++];
    while(j<=lose[0]) id[++id[0]]=lose[j++];
}
```

```
int main(){
    cin>>n>>m>>q;
    n*=2;
    for(int i=1;i<=n;i++) cin>>s[i];
    for(int i=1;i<=n;i++){
        cin>>w[i];
        id[i]=i;
    }
    sort(id+1,id+n+1,cmp);//初始排序
    for(int i=1;i<=m;i++){
        win[0]=lose[0]=0;
        for(int j=1;j<=n;j+=2){
            if(w[id[j]]>w[id[j+1]]){
                s[id[j]]++;
                win[++win[0]]=id[j];
                lose[++lose[0]]=id[j+1];
            }
            else{
                s[id[j+1]]++;
                win[++win[0]]=id[j+1];
                lose[++lose[0]]=id[j];
            }
        }
        merge();
    }
    cout << id[q] << endl;
    return 0;
```



链表

单链表

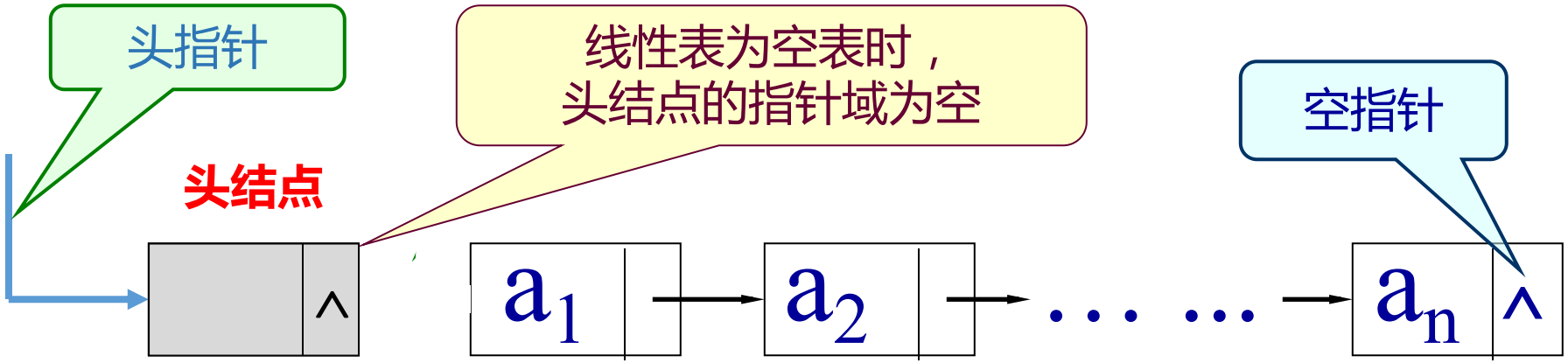
用一组地址任意的存储单元存放线性表中的数据元素。

数据元素（数据域）+ 指针（指针域，指示后继元素存储位置）= 结点

以“结点的序列”表示线性表 —— 称作链表。

以线性表中第一个数据元素a1的存储地址作为线性表的地址，称作线性表的头指针。

有时为了操作方便，会在第一个结点之前虚加一个“头结点”，以指向头结点的指针为链表的头指针。



例1. 1109 维护序列

时间限制: 1000 ms 空间限制: 262144 KB

【题目描述】

给定一个长度为n的整数序列。现在有m个操作，操作分为三类，格式如下：

① 1 i：询问序列中第i个元素的值，保证i小于等于当前序列长度；

② 2 i v：在序列中第i个元素前加入新的元素v，保证i小于等于当前序列长度；

③ 3 i：删除序列中的第i个元素，保证i小于等于当前序列长度。

【输入】

第一行输入n(1<=n<=1000)，表示序列最初的长度。

第二行输入n个空格隔开的数表示原始的整数序列。

第三行输入m(1<=m<=1000)，表示操作数。

第四到m+3行依次输入一个操作。

【输出】

对于操作①输出对应的答案，一行输出一个数。

【样例输入】

```
5
6 31 23 14 5
5
1 2
2 2 7
1 2
3 3
1 3
```

【样例输出】

```
31
7
23
```

【数据范围限制】

1<=n<=1000, 1<=m<=1000,每个元素都是不超过1000000的正整数。

算法分析：(数组实现)

1 i：询问序列中第i个元素的值，保证i小于等于当前序列长度；

输出数组中第i个元素的值；

2 i v：在序列中第i个元素前加入新的元素v，保证i小于等于当前序列长度；

第i个元素以及它后面的元素依次后移，第i个元素赋为v；

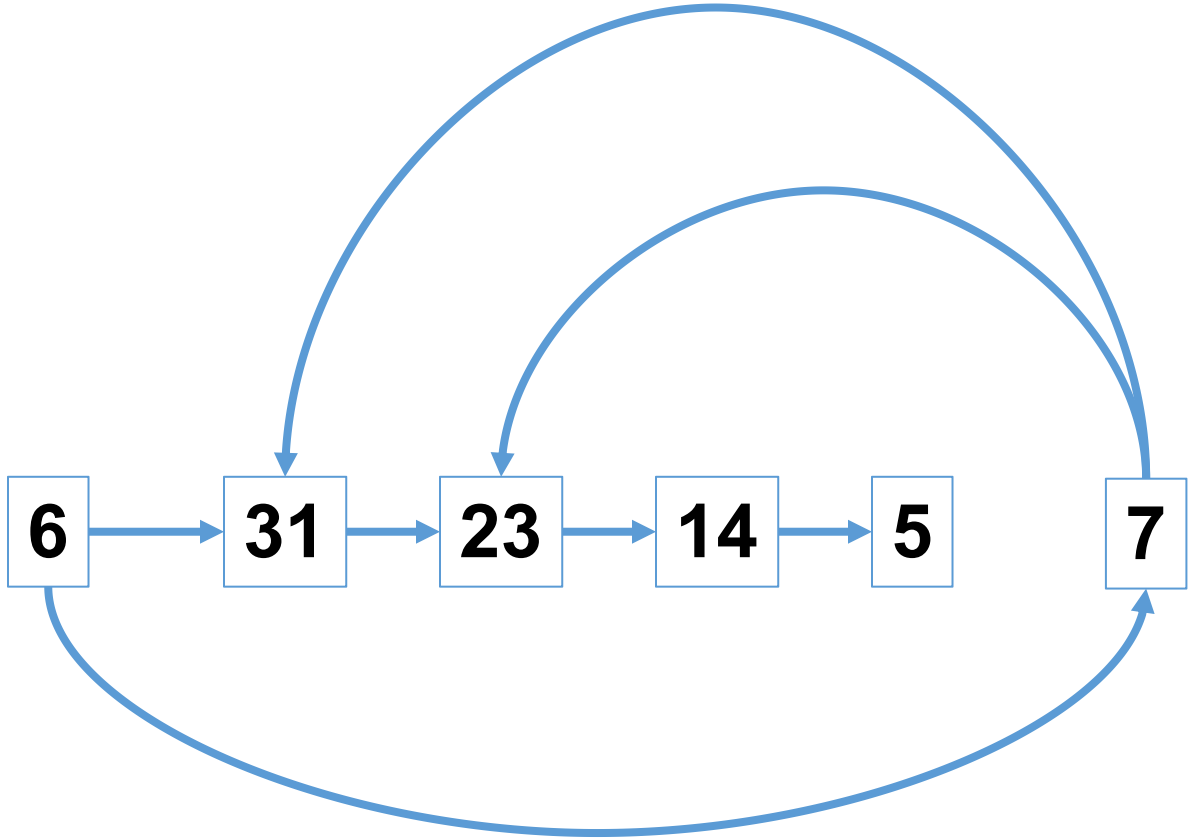
3 i：删除序列中的第i个元素，保证i小于等于当前序列长度；

第i+1个元素以及它后面的元素依次前移。

其中，2、3操作中都可能需要移动大量元素。

总复杂度：O(n * m)。

```
5

1  2

2  2  7

1  2

3  3

1  3
```



链表：

- 元素数组：data[maxn];
- 指针数组：next[maxn];　　//元素data[i]的后继元素所在位置是next[i]
- 头结点指针：head，head的后继是第一个元素

不要求逻辑上相邻的元素存储位置也相邻，不能随机存储。

```
5
1 2
2 2 7
1 2
3 3
1 3
```

位置	1	2	3	4	5	6
data	6	31	23	14	5	7
next	2	3	4	5	-1	
head = 0						

代码实现：

访问第i个元素：

next[head]是第一个元素，……

cur=next[head]; for(int j=1;j<i;j++) cur=next[cur];

在第i个元素之前加入一个新元素data[n]：

先把当前位置cur从head移动到第i-1个元素，

for(j=0,cur=head;j<i-1; j++) cur=next[cur];

然后插入新元素，

next[n] = next[cur]; next[cur] = n;

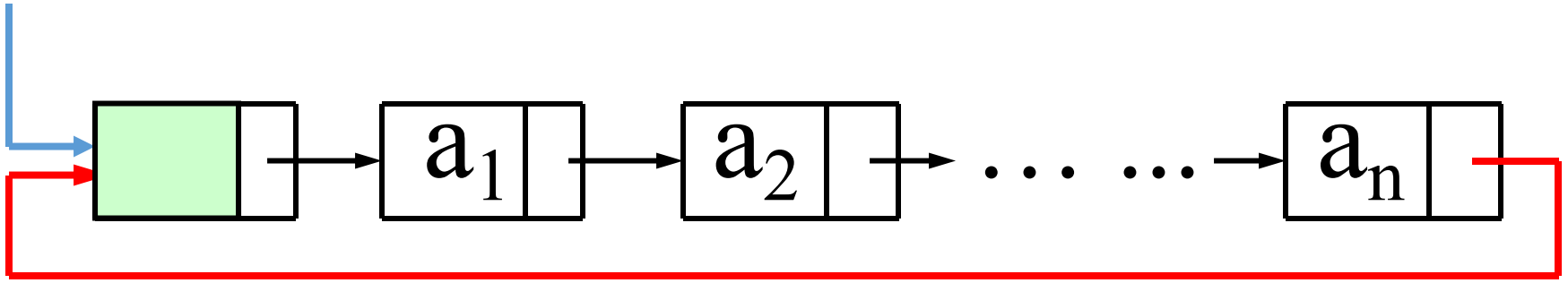
删除第i个元素：

先把当前位置cur从head移动到第i-1个元素，再删除第i个元素

next[cur] = next[next[cur]];

循环单链表

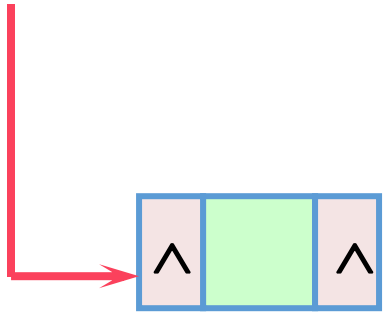
最后一个结点的指针域的指针又指回第一个结点的链表。
和单链表的差别仅在于，判别链表中最后一个结点的条件不再是“后继是否为空”，而是“后继是否为头结点”。



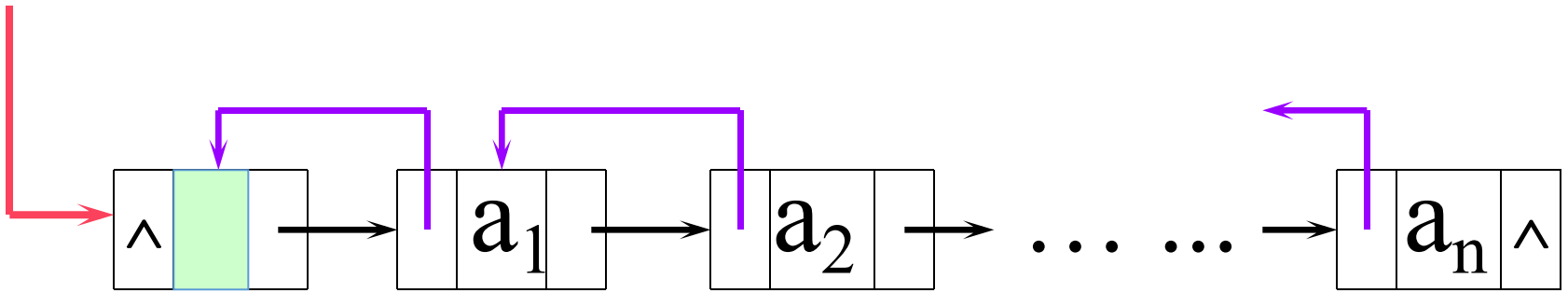
双链表

每个结点有两个指针域，一个指向前驱，一个指向后继。

空表



非空表



例2. P1160 队列安排

【题目描述 】
一个学校里老师要将班上N个同学排成一列，同学被编号为1～N，他采取如下的方法：
1.先将1号同学安排进队列，这时队列中只有他一个人；
2.2～N号同学依次入列，编号为i的同学入列方式为：老师指定编号为i的同学站在编号为1～i -1中某位同学（即之前已经入列的同学）的左边或右边；
3.从队列中去掉M（M<N）个同学，其他同学位置顺序不变。
在所有同学按照上述方法队列排列完毕后，老师想知道从左到右所有同学的编号。
【输入输出格式 】
输入格式：
输入文件arrange.in的第1行为一个正整数N，表示了有N个同学。
第2～第N行，第i行包含两个整数k，p，其中k为小于i的正整数，p为0或者1。若p为0，则表示将i号同学插入到k号同学的左边，p为1则表示插入到右边。
第N+1行为一个正整数M，表示去掉的同学数目。
接下来M行，每行一个正整数x，表示将x号同学从队列中移去，如果x号同学已经不在队列中则忽略这一条指令。
输出格式：
输入文件arrange.out仅包括1行，包含最多N个空格隔开的正整数，表示了队列从左到右所有同学的编号，行末换行且无空格。

4 2 3 4

1 0

2 1

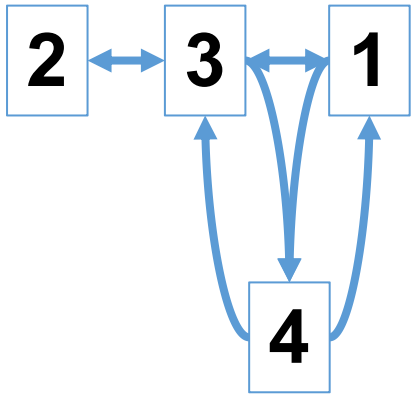
1 0

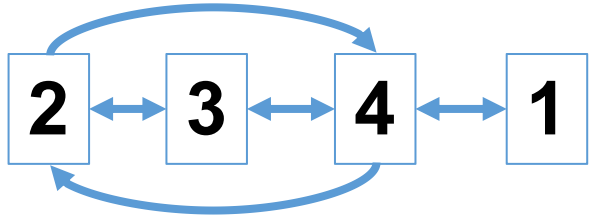
2

3

3

如果以数组的形式来维护队伍，与插入排序类似，插入或删除元素时，存在元素的频繁移动，时间复杂度O(n^2)。





代码实现:

```
struct node{  int l,r; } a[100009];
```

i插在x的左侧：

```
void lf(int i,int x){// p-x 变成 p-i-x
    int p=a[x].l;
    a[x].l=i; a[i].r=x;
    a[p].r=i; a[i].l=p;
}
```

删除x：

```
void de(int x){ //p-x-q 变成 p-q;
    int p=a[x].l,q=a[x].r;
    a[x].l=a[x].r=-1;
    if(p==-1&&q==-1)return;
    a[p].r=q;
    if(q!=-1) a[q].l=p;
}
```

Thanks for listening !