



电子科技大学

University of Electronic Science and Technology of China

第四讲

基本数据结构

数学科学学院 汪小平

一、堆栈 (LIFO)

堆栈是一种先进后出的结构, 有栈底和栈顶, 操作主要有入栈 (push) 和出栈 (pop), 如下图的示:

1. 原堆栈



2. pop



3. push



逆序问题

从键盘读入一个长度不超过1000的以‘#’作为输入结束符的一系列字符，然后按相反顺序输出。

```
#include <stdio.h>
int main()
{
    char ch, stack[1002];
    int top = -1;
    while((ch = getchar()) != '#')
        stack[++top] = ch;
    while(top >= 0)
        putchar(stack[top--]);
    return 0;
}
```

```
E:\Projects\C\2013\temp\bin\Debug\temp
abcdef 123456#sdfsd
654321 fedcba
Process returned 0 (0x0)
s
Press any key to continue.
```



C++实现

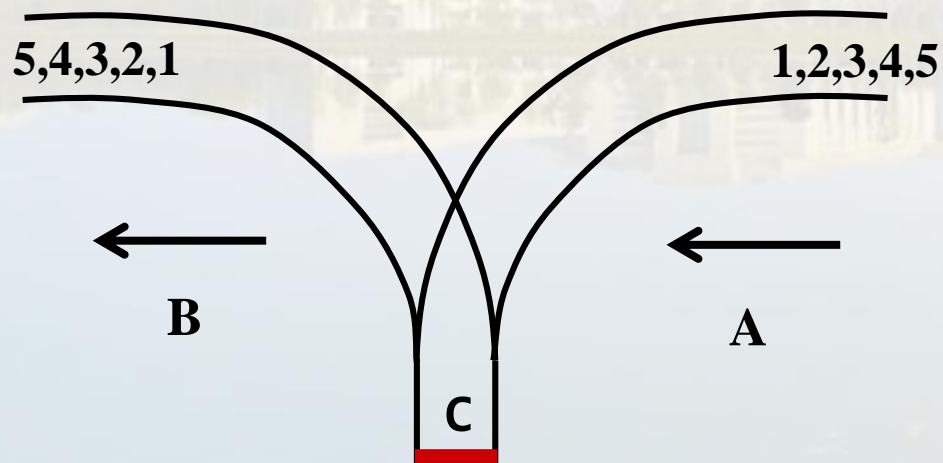
```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    char ch;
    stack<char> s;
    while((ch = getchar()) != '#')
        s.push(ch);
    while(!s.empty())
    {
        putchar(s.top());
        s.pop();
    }
    return 0;
}
```



size(): 返回堆栈中元素的个数;

铁轨问题

某城市有一个火车站, 铁轨铺设如下图. 有 n 节车厢从A方向驶入车站, 按进站顺序编号为 $1 \sim n$. 你的任务是让它们按照某种特定的顺序进入B方向的铁轨并驶出车站. 为了重组车厢, 你可以借助中转站C. 这是一个可以停放任意多节车厢的车站, 但由于末端封顶, 驶入C的车厢必须按照相反的顺序驶出C. 对于每个车厢, 一旦从A移入C, 就不能再回到A了; 一旦从C移入B, 就不能回到C了. 换句话说, 在任意时刻, 只有两种选择: $A \rightarrow C$ 和 $C \rightarrow B$.

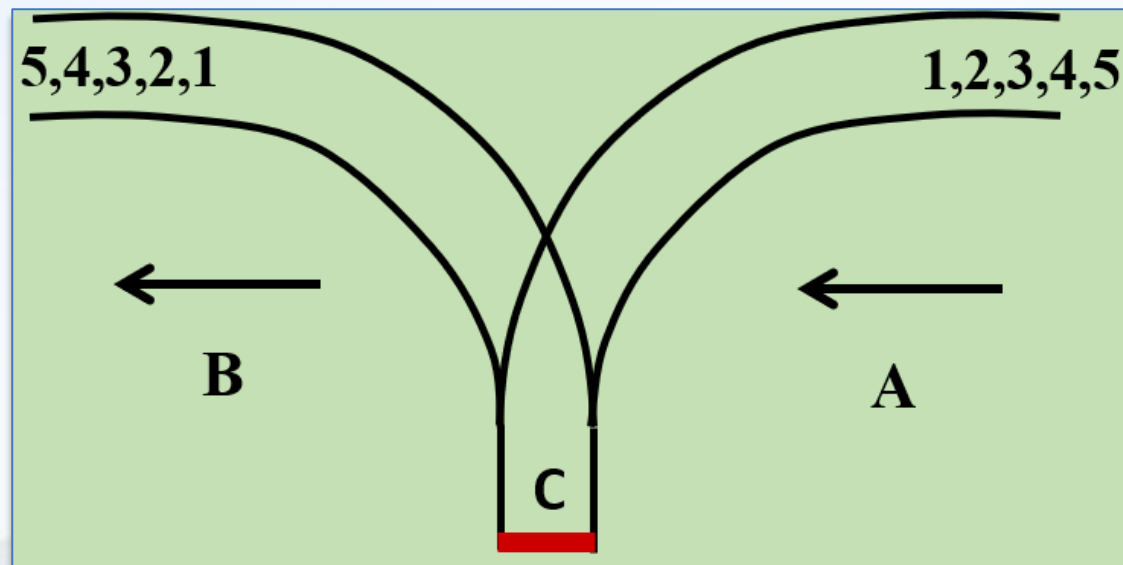


样例输入	样例输出
5	Yes
1 2 3 4 5	No
5	Yes
5 4 1 2 3	
6	
6 5 4 3 2 1	

题目分析

将驶入的车厢编号及站内最后进入的车厢编号与目标车厢编号对比再决定操作:

- 将进入的车厢编号与目标车厢编号相同, 则车厢进入车站马上驶出;
- 站内最后进入的车厢编号与目标车厢编号相同, 则该车厢马上驶出;
- 不属于上两种情况, 但还有驶入车厢, 则让它入站;
- 没有车厢驶入, 也不存在最前面两种情况, 则无法达到目标.



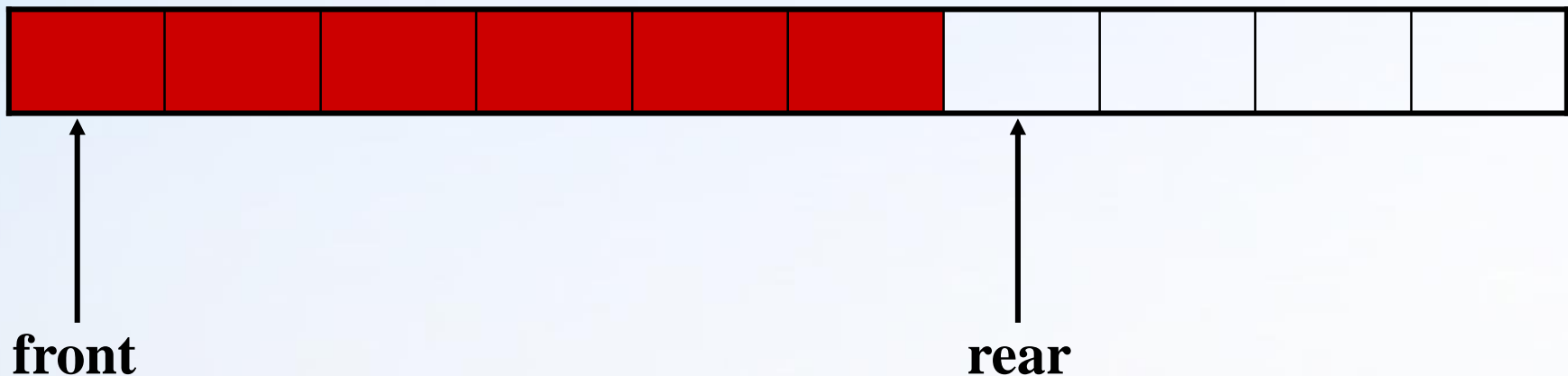
数组实现



```
#include <stdio.h>
#define N 1000
int main()
{
    int i, n, target[N];
    while(scanf("%d", &n) == 1)
    {
        int stack[N], top = -1;
        for(i = 0; i < n; i++) scanf("%d", &target[i]);
        int A = 1, B = 0;
        int ok = 1;
        while(B < n)
        {
            if(A == target[B]) { A++; B++; }
            else if(top >= 0 && stack[top] == target[B]) {top--; B++; }
            else if(A <= n) { stack[++top] = A++; }
            else { ok = 0; break; }
        }
        printf("%s\n", ok ? "Yes" : "No");
    }
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
#define N 1000
int main()
{
    int i, n, target[N];
    while(scanf("%d", &n) == 1)
    {
        stack<int> s;
        for(i = 0; i < n; i++)    scanf("%d", &target[i]);
        int A = 1, B = 0, ok = 1;
        while(B < n)
        {
            if(A == target[B]) { A++; B++; }
            else if(!s.empty() && s.top() == target[B]) { s.pop(); B++; }
            else if(A <= n) { s.push(A++); }
            else { ok = 0; break; }
        }
        printf("%s\n", ok ? "Yes" : "No");
    }
    return 0;
}
```


二、队列 (FIFO)



翻牌问题 桌上有一又叠牌，从第一张牌（即位于顶面的牌）开始从上往下依次编号为1~n。当至少还剩两张牌时进行以下操作：把第一张牌扔掉，然后把新的第一张放到整叠牌的最后。输入n，输出每次扔掉的牌，以及最后剩下的牌。

样例输入：7

样例输出：1 3 5 7 4 2 6

数组实现

```
#include <stdio.h>
#define N 64
int main()
{
    int i, n, front, rear, q[2*N];
    scanf("%d", &n);
    for(i = 0; i < n; i++)
        q[i] = i + 1;
    front = 0;
    rear = n;
    while(rear > front)
    {
        printf("%d ", q[front]);
        front++;
        q[rear++] = q[front++];
    }
    return 0;
}
```

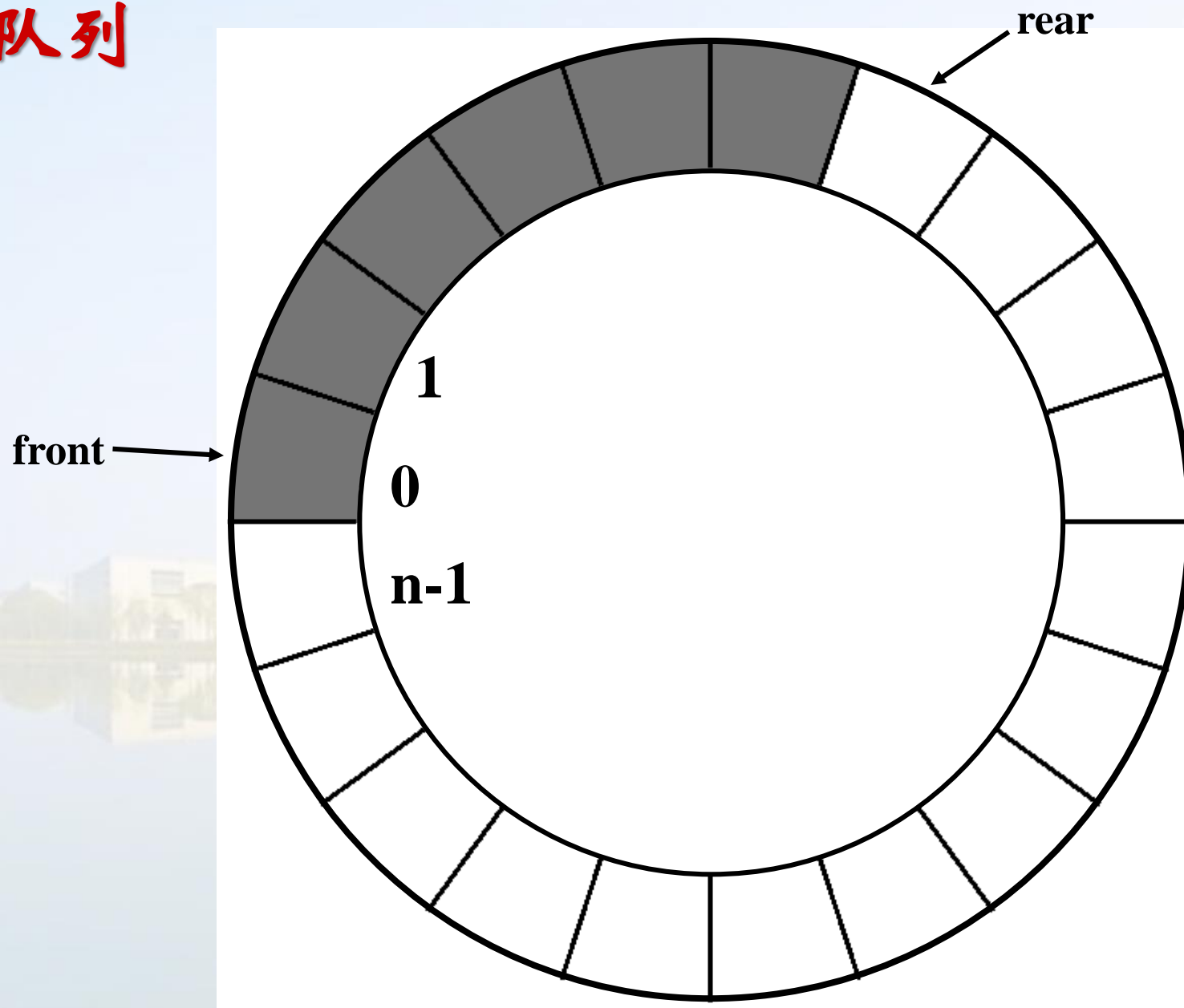
C++ STL实现

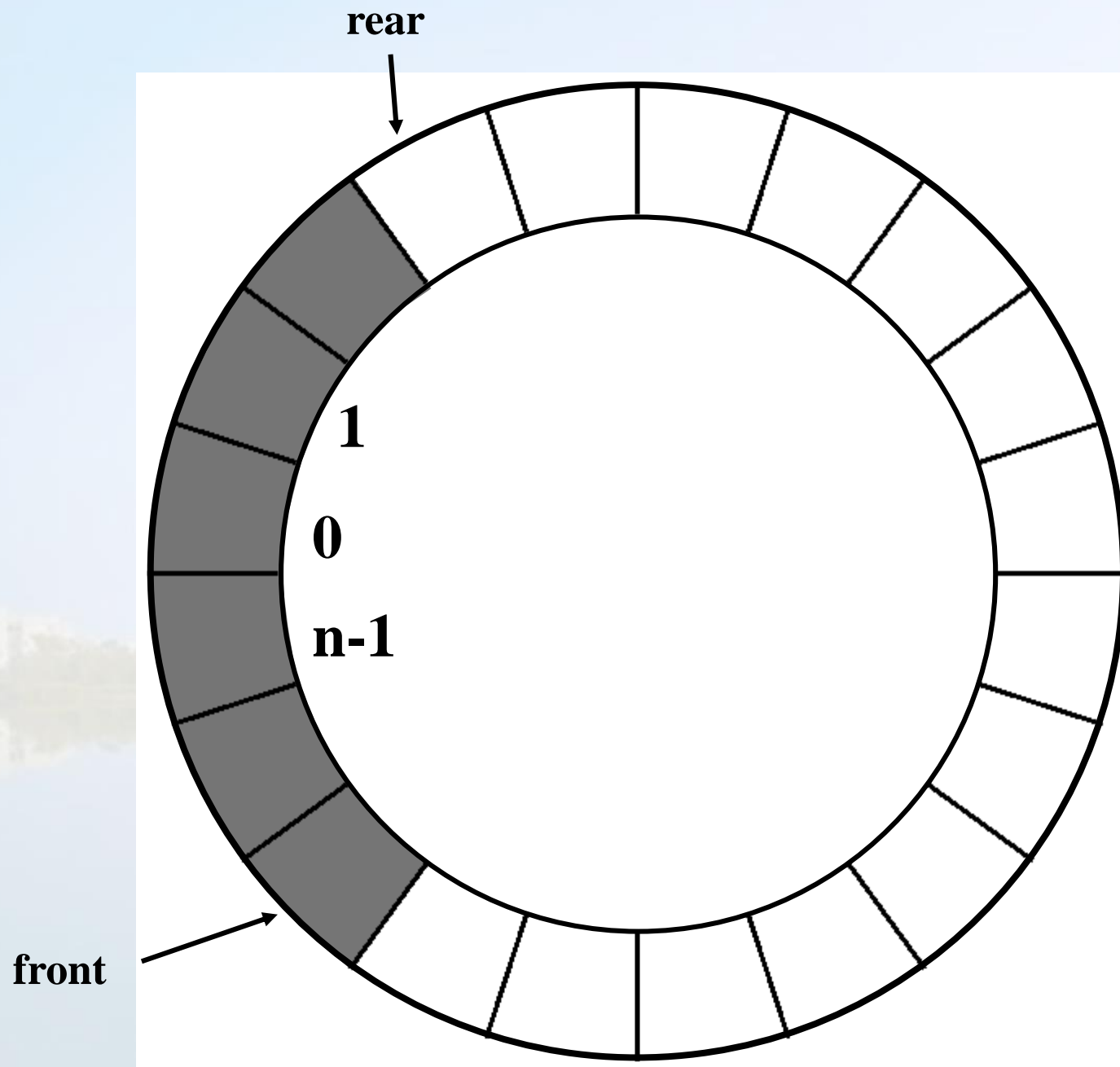
```
#include <cstdio>
#include <queue>
using namespace std;
int main()
{
    queue<int> q;
    int i, n;
    scanf("%d", &n);
    for(i = 0; i < n; i++)
        q.push(i + 1);
    while(!q.empty())
    {
        printf("%d ", q.front());
        q.pop();
        q.push(q.front());
        q.pop();
    }
    return 0;
}
```

size(): 返回队列元素数;

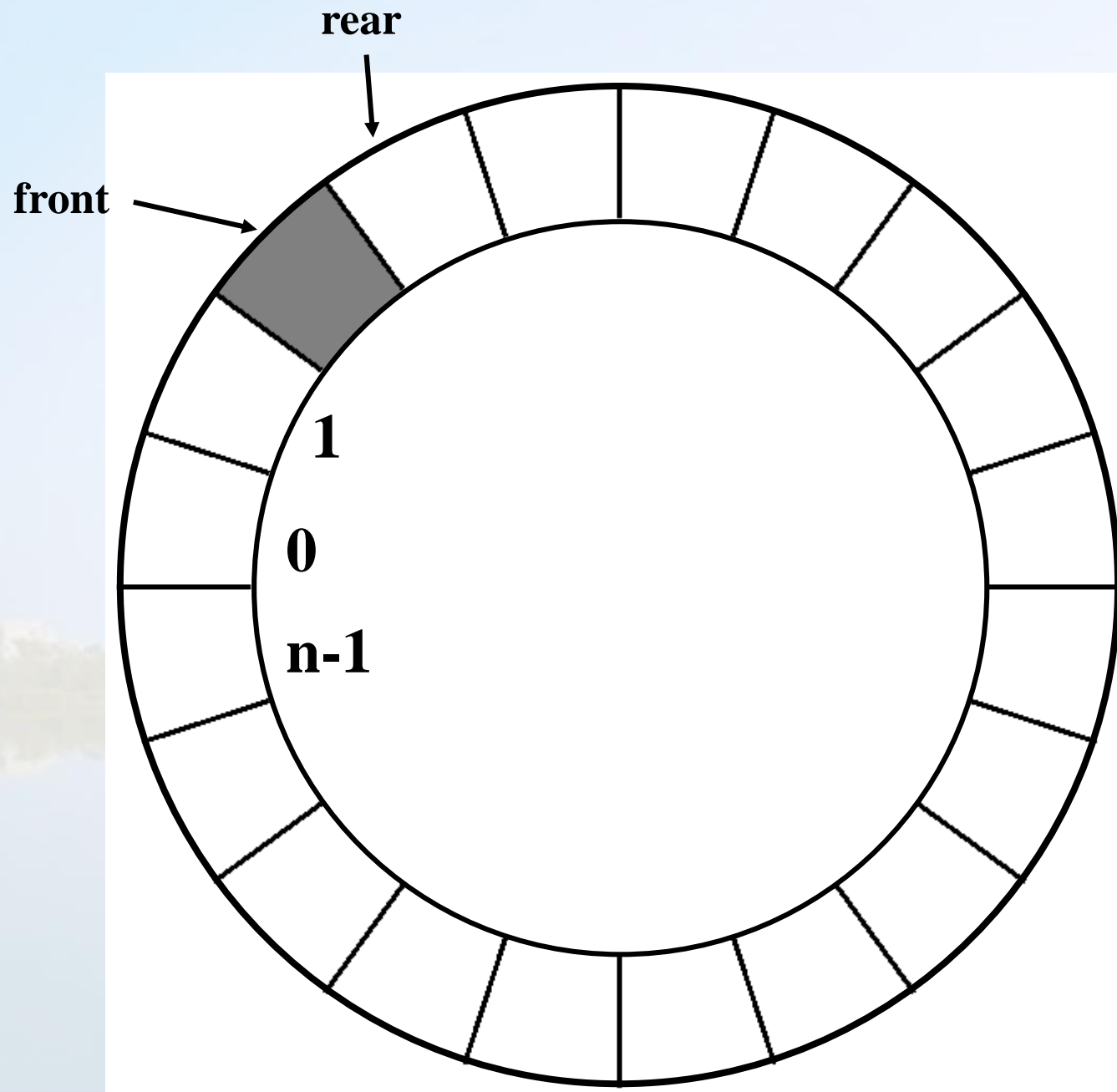
back(): 返回队尾元素.

循环队列

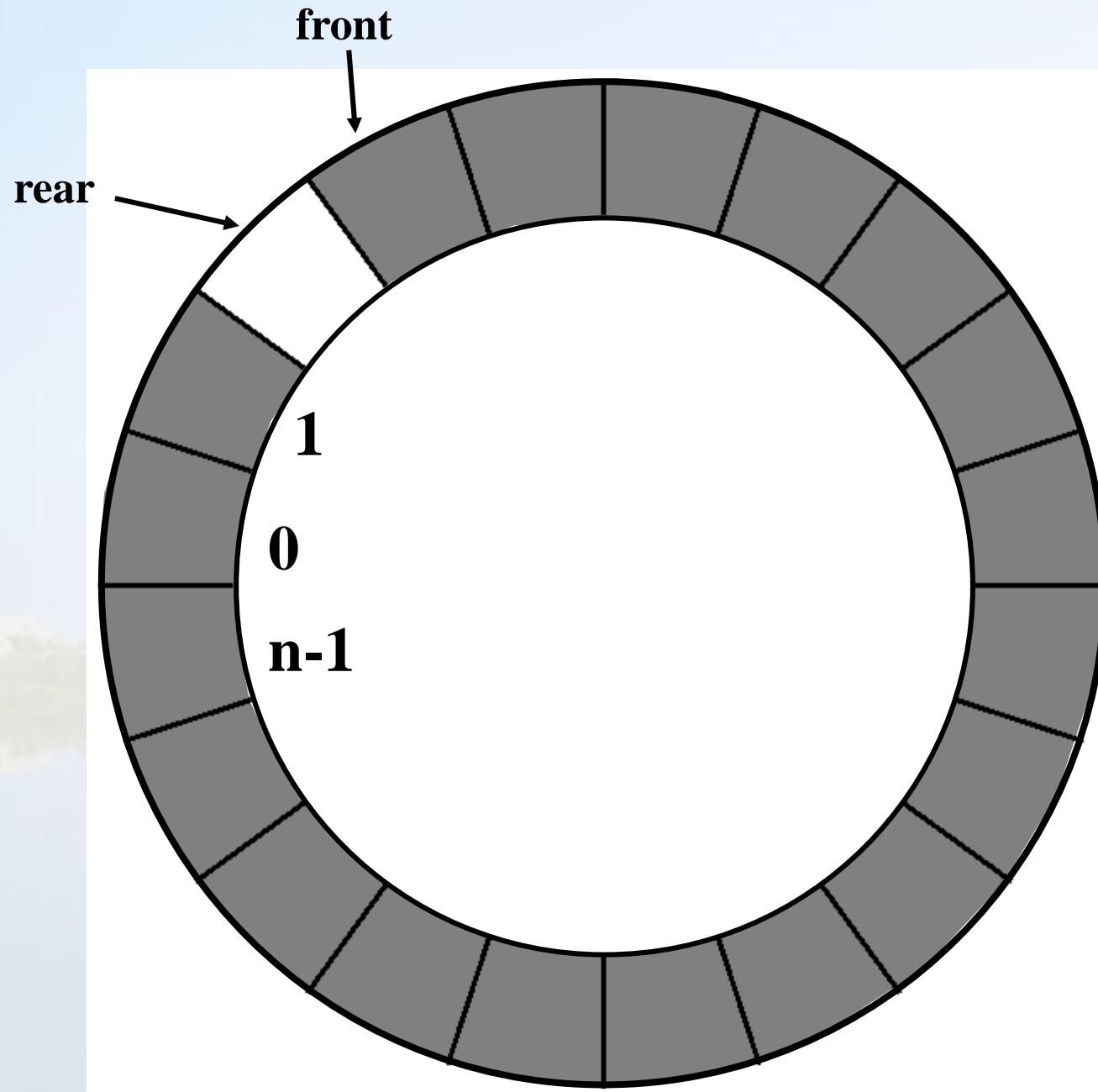




- $\text{front} > \text{rear}$



- $|\text{rear} - \text{front}| = 1$



- $|\text{rear} - \text{front}| = 1$

三、优先队列(priority_queue)

优先队列类似队列,但**先出队的是队列中优先级最高的元素**.
即入队时会进行按优先级排序. 由于出队元素并不是最先入队的元素,所以**使用方法是top(),而不是front()**.

```
std::priority_queue
```

```
C++ Containers library std::priority_queue
```

```
Defined in header <queue>
```

```
template<
```

```
    class T,
```

```
    class Container = std::vector<T>,
```

```
    class Compare = std::less<typename Container::value_type>
```

```
> class priority_queue;
```

测试priority_queue

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int k;
    priority_queue<int> pq;
    for(int i = 0; i < 10; i++) {
        k = rand() % 100;
        pq.push(k);
        printf("%d ", k);
    }
    printf("\n");
    while(!pq.empty()) {
        printf("%d ", pq.top());
        pq.pop();
    }
    return 0;
}
```

size(): 返回队列中元素的个数;

E:\Projects\C_C++\temp\bin\Debug\temp.exe

41 67 34 0 69 24 78 58 62 64

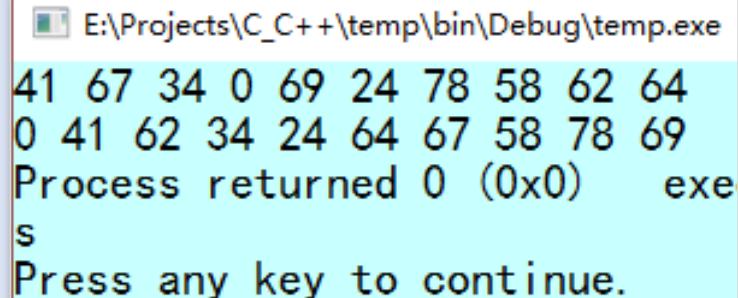
78 69 67 64 62 58 41 34 24 0

Process returned 0 (0x0) execution
s

Press any key to continue.

测试priority_queue(个位数大优先级小)

```
#include <bits/stdc++.h>
using namespace std;
struct cmp {
    bool operator()(const int a, const int b)const {
        return b%10 < a%10;
    }
};
int main() {
    int k;
    priority_queue<int, vector<int>, cmp> pq;
    for(int i = 0; i < 10; i++) {
        k = rand() % 100;    pq.push(k);    printf("%d ", k);
    }
    printf("\n");
    while(!pq.empty()) {
        printf("%d ", pq.top());    pq.pop();
    }
    return 0;
}
```



```
E:\Projects\C_C++\temp\bin\Debug\temp.exe
41 67 34 0 69 24 78 58 62 64
0 41 62 34 24 64 67 58 78 69
Process returned 0 (0x0)   exe
s
Press any key to continue.
```

std::vector: Defined in header
<vector>

operator[]: 访问特定元素
empty(), size(), clear(), insert(),
erase(), push_back()

丑数

丑数是指不能被2,3,5之外的素数整除的正整数. 把丑数从小到大排列起来, 结果如下:

1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ...

求第1500个丑数.

分析: 本题实现方法有多种. 这儿介绍一种.

- 1是最小丑数, 而对于丑数 x , 则 $2x, 3x, 5x$ 也是丑数;
- 用priority_queue存放丑数, 让最小丑数出队不断生成后续丑数, 同时进行计数, 以找到第1500个丑数;
- 需要注意的是, 同一个丑数, 可能多次生成, 所以需要判断生成的丑数是否已入队, 这里用set容器实现.

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int coeff[3] = {2, 3, 5};
int main(){
    priority_queue<LL, vector<LL>, greater<LL> > pq;
    set<LL> s;
    pq.push(1); s.insert(1);
    for(int i = 1; ; i++) {
        LL x = pq.top(); pq.pop();
        if(i == 1500) {
            printf("%d", x); break;
        }
        for(int j = 0; j < 3; j++) {
            LL x2 = x * coeff[j];
            if(s.count(x2) == 0) {
                s.insert(x2); pq.push(x2);
            }
        }
    }
    return 0;
}
```

```
E:\Projects\C_C++\temp\bin\Debug\ter
859963392
Process returned 0 (0x0)
s
Press any key to continue.
```

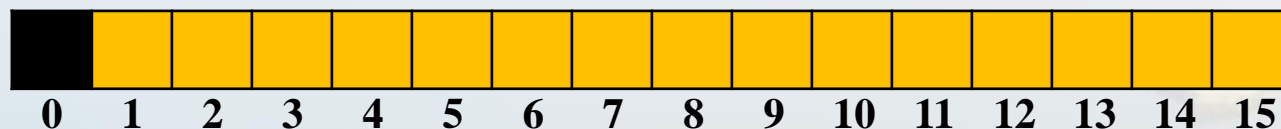
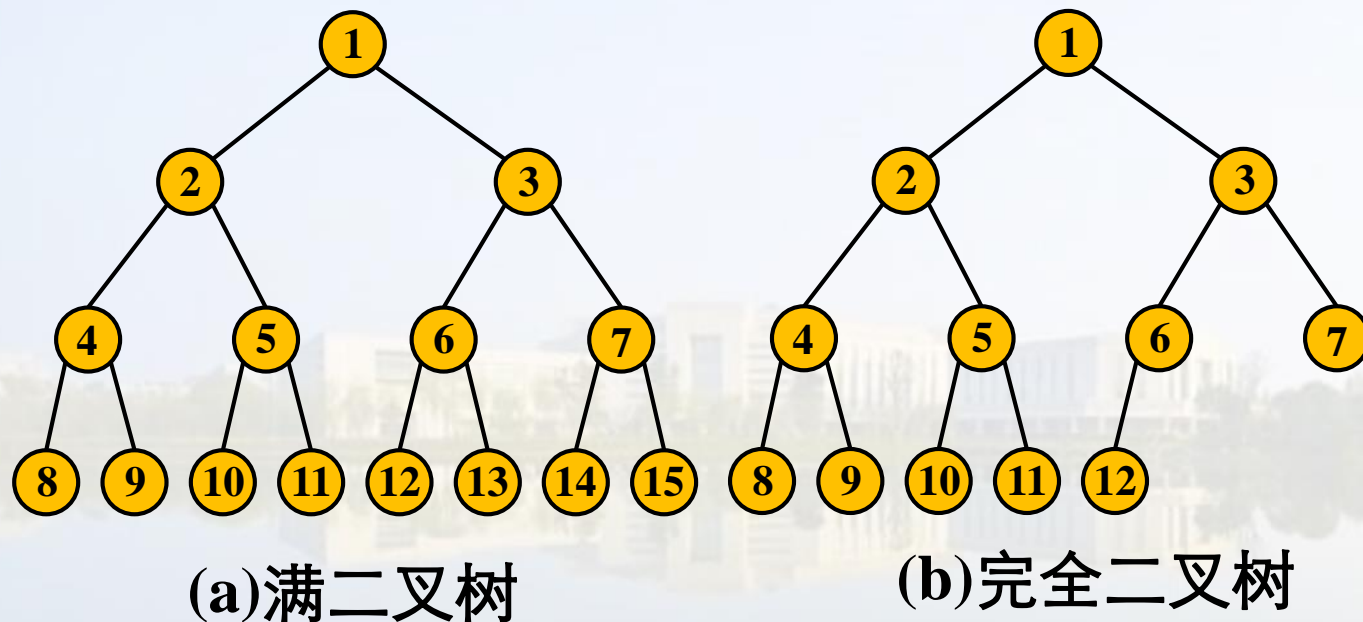
std::set: Defined in header <set>

- **bool empty() const;**
- **size_type size() const;**
- **void clear();**
- **insert(const value_type& value);**
- **size_type erase(const key_type& key);**
- **size_type count(const Key& key) const;**

四、二叉树

1. 二叉树及其存储

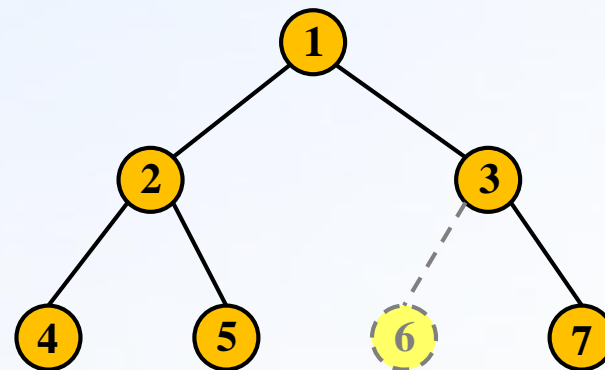
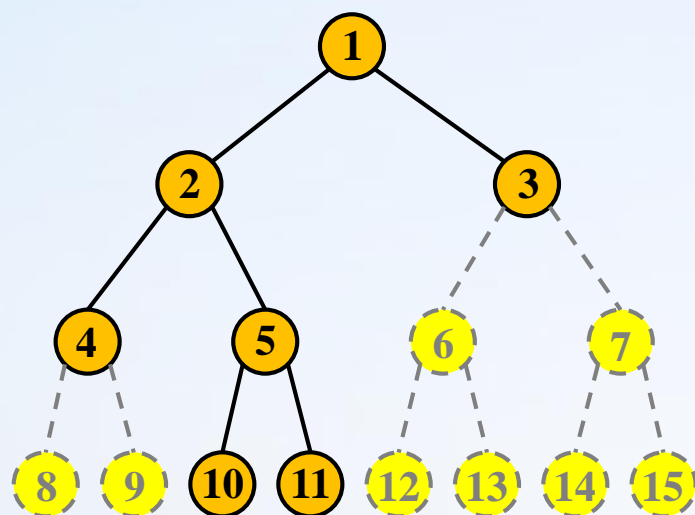
树是一种特殊的**非线性结构**, 更一般的情形是图. 最常用的树形结构是二叉树, 下面是几种二叉树的情形:



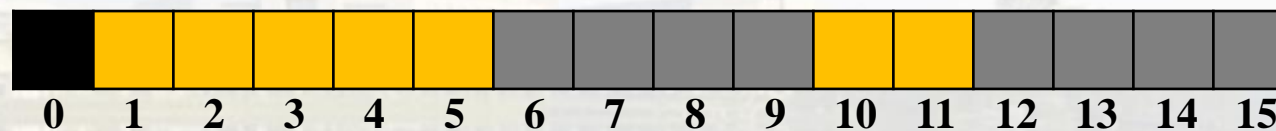
结点的编号为 k , 如果存在, 则其左儿子编号为 $2k$, 右儿子为 $2k + 1$, 其父结点编号为 $\lfloor k/2 \rfloor$.

几个概念:

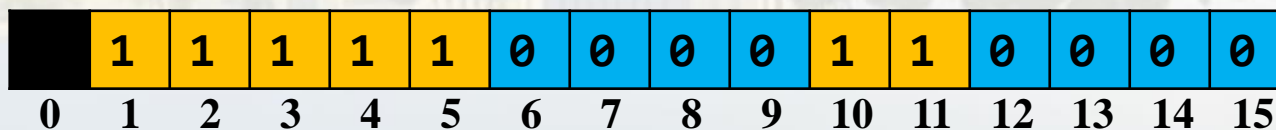
- 根结点
- 叶子结点
- 父结点
- 左右儿子
- 左右子树
- 兄弟
- 堂兄弟
- 祖先
- 树深



值数组:



标记数组:



2. 二叉树的遍历

(1) **前序遍历**: 首先访问根结点, 然后按前序遍历左子树, 最后按前序遍历右子树;

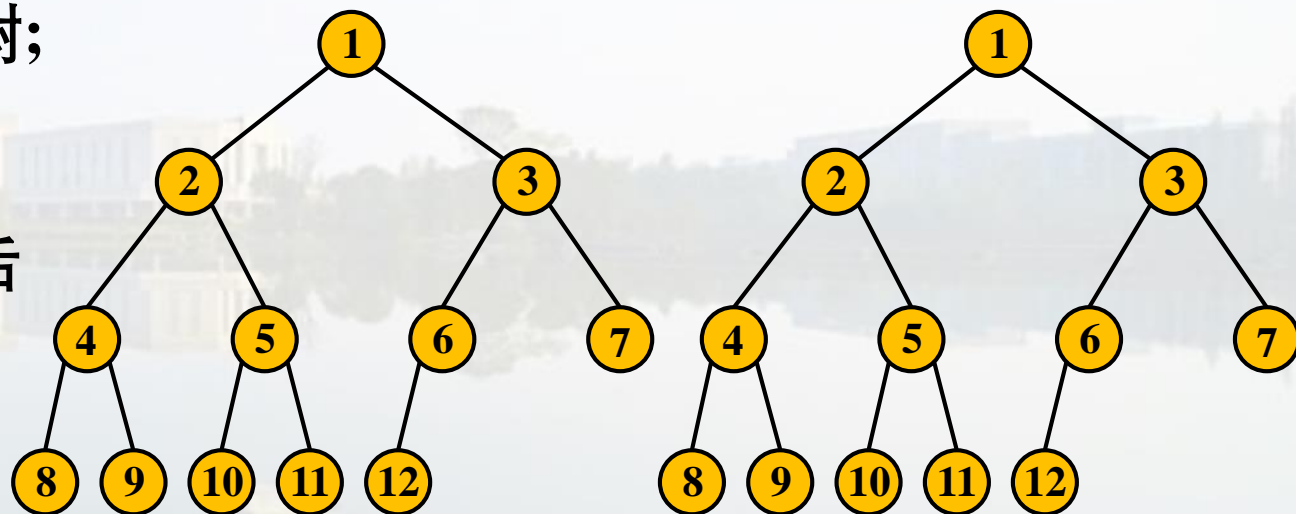
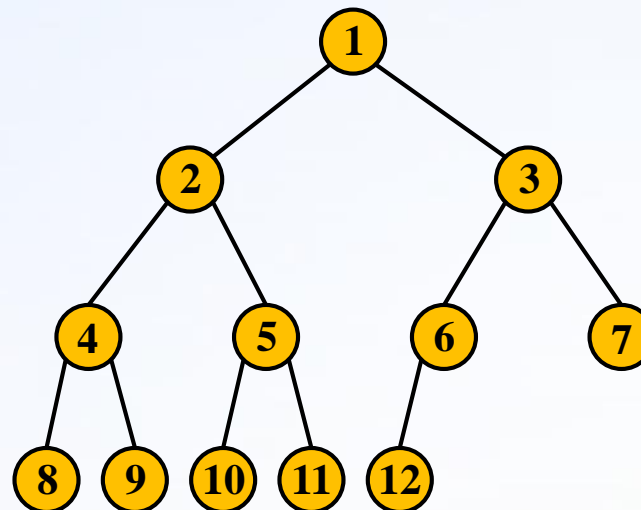
1, 2, 4, 8, 9, 5, 10, 11, 3, 6, 12, 7

(2) **中序遍历**: 先按中序遍历左子树, 然后访问根结点, 最后按中序遍历右子树;

8, 4, 8, 2, 10, 5, 11, 1, 12, 6, 3, 7

(3) **后序遍历**: 先后序遍历左子树, 然后按后序遍历右子树, 最后访问根结点.

8, 9, 4, 10, 11, 5, 2, 12, 6, 7, 3, 1

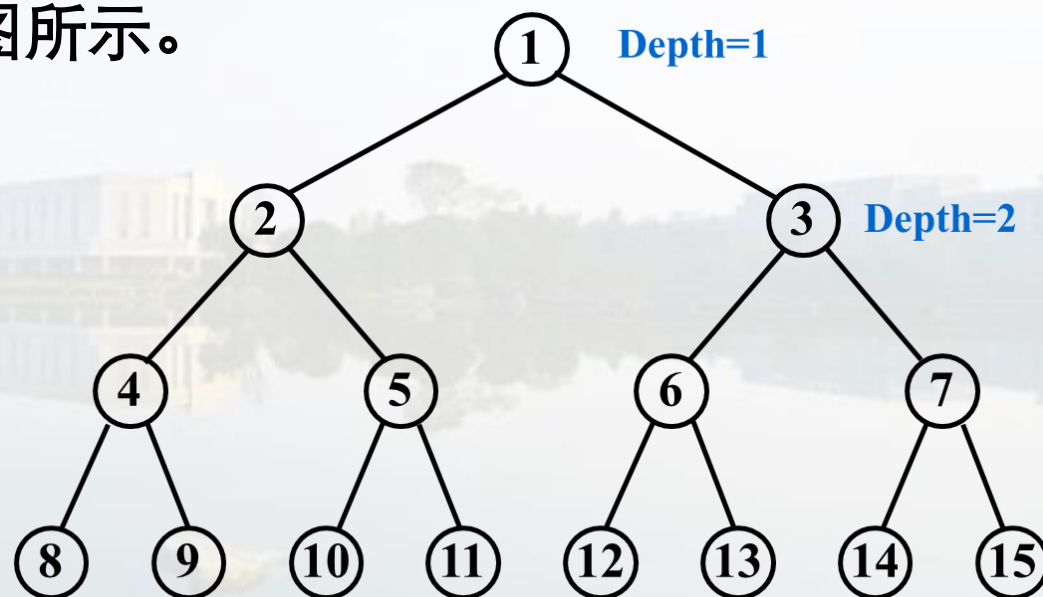


问题: 中序遍历结果和前(后)序遍历结果能否确定二叉树?

小球下落问题

有一棵二叉树，最大深度为 D ，且所有叶子的深度都相同。所有结点从上到下从左到右编号为 $1, 2, 3, \dots, 2^D - 1$ 。在结点 1 处放一个小球，它会往下落。每个内点上都有一个开关，初始全部关闭，当每次有小球落到一个开关上时，它的状态都会改变。当小球到达一个内结点时，如果该结点上的开关关闭，则往左走，否则往右走。如下图所示。

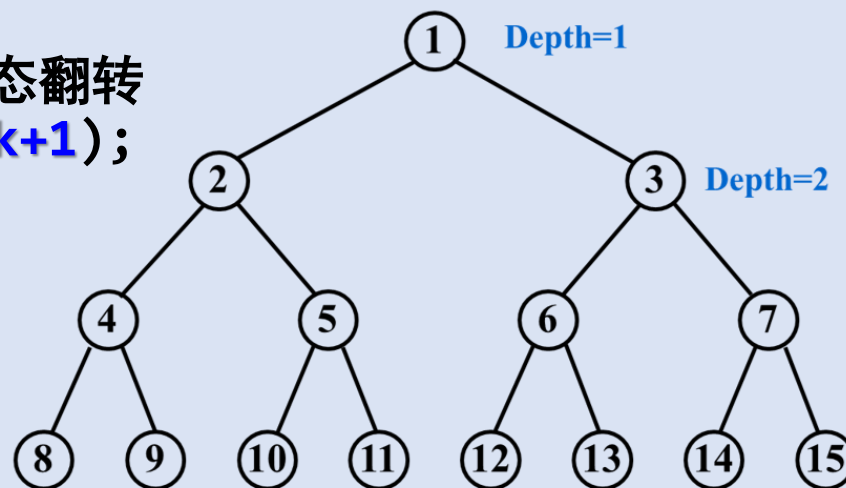
一些小球从结点 1 处依次开始下落，最后一个小球将会落到哪儿呢？
输入叶子深度 D 和小球个数 I ，输出第 I 个小球最后所在的叶子编号。
假设 I 不超过整棵树的叶子个数。
 $D \leq 20$ 。输入最多包含 1000 组数据。



样例输入	样例输出
4 2	12
3 4	7
10 1	512
2 2	3
8 128	255
16 12345	36358

方法一：直接模拟

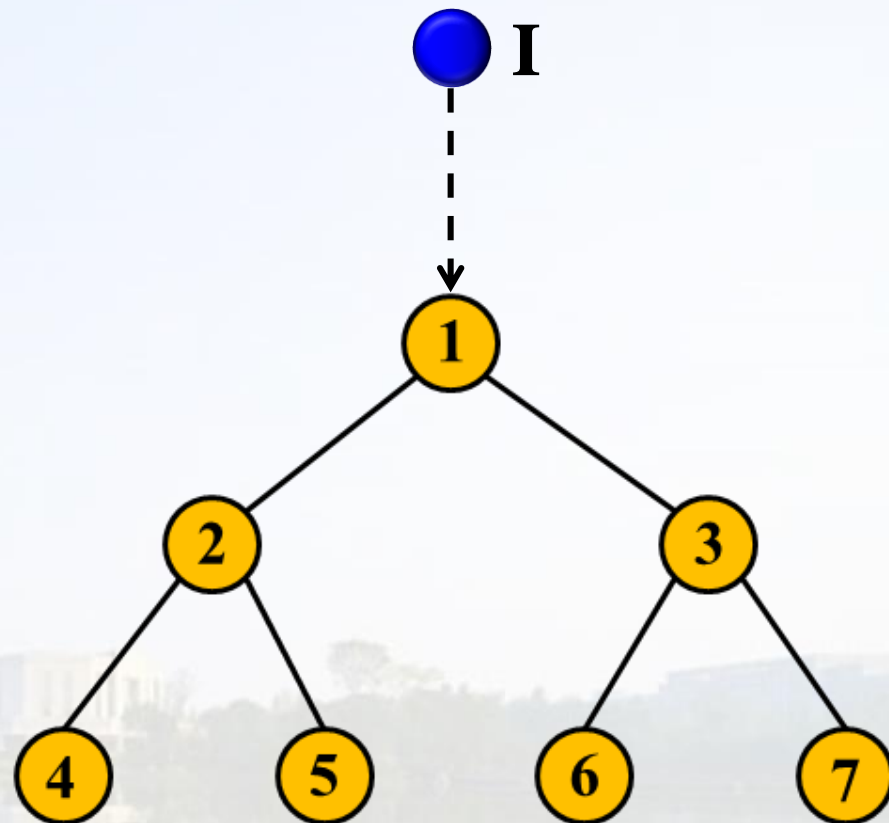
```
#include <bits/stdc++.h>
using namespace std;
#define MAXD 20
int state[1<<MAXD];
int main() {
    int D, I;
    while(scanf("%d%d", &D, &I) == 2) {
        int i, k, n = (1<<D)-1;
        memset(state, 0, sizeof(state)); //清零
        for(i = 0; i < I; i++) {
            for(k = 1;;) {
                state[k] = !state[k]; //状态翻转
                k = state[k] ? (2*k) : (2*k+1);
                if(k > n) break;
            }
            printf("%d\n", k/2);
        }
        return 0;
    }
}
```



方法二

按落在开关 k 上的小球计数，假设现在是第 I 个。由于开关 k 刚开始是关闭的，则容易发现，则若 I 为奇数，则小球会向左落，否则会向右落。

开关 k 的左儿子为 $2k$ ，右儿子为 $2k+1$ 。若第 I 个小球落向左儿子，则到目前落在开关 $2k$ 的小球个数为 $(I+1)/2$ ，若第 I 个小于落向右儿子，则到目前落在开关 $2k+1$ 的小球个数为 $I/2$ 。



因此，要看第 I 个小球到底落在哪个叶子上，只须看 I 为奇数还是偶数，就知道它落在左儿子还是左儿子，同时也就知道落在左儿子或右儿子上已经是第多少个小球，因此就能知道该小球是落在左儿子的左儿子还是左儿子的右儿子，……，这样下去，就知道它落在哪个叶子。

参考程序二

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int D, I;
    while(scanf("%d%d", &D, &I) == 2) {
        int i, k = 1;
        for(i = 0; i < D - 1; i++) {//逐层判断
            if(I % 2 == 1) {//落在左儿子2k上
                k = k * 2;
                I = (I + 1)/2; //落在左儿子上的小球数
            }
            else {//落在右儿子2k+1上
                k = k * 2 + 1;
                I = I / 2; //落在右儿子上的小球数
            }
        }
        printf("%d\n", k);
    }
    return 0;
}
```

