stack

栈(std::stack) 是一种后进先出 (Last In, First Out) 的容器适配器，仅支持查询或删除最后一个加入的元素（栈顶元素），不支持随机访问，且为了保证数据的严格有序性，不支持迭代器。

成员函数

函数名	功能	时间复杂度
top()	返回栈顶元素	O(1)
empty()	判断是否为空	O(1)
size()	返回元素个数	O(1)
push()	在栈顶插入元素	O(1)
pop()	删除栈顶元素	O(1)

示例

```
#include <iostream>
#include <stack>  //头文件

int main(){
    std::stack<int> s;//定义一个空栈
    s.push(1);
    s.push(2);
    s.push(3);
    //s={1,2,3(top)}
    s.pop();//移除栈顶元素
    //s={1,2(top)}
    cout<<"此时的栈顶元素为"<<s.top()<<endl;
    //此时的栈顶元素为2
    return 0;
}
```

queue

队列(std::queue) 是一种先进先出 (First In, First Out) 的容器适配器，仅支持查询或删除第一个加入的元素（队首元素），不支持随机访问，且为了保证数据的严格有序性，不支持迭代器。

成员函数

函数名	功能	时间复杂度
front()	返回队首元素	O(1)
back()	返回队尾元素	O(1)

函数名	功能	时间复杂度
empty()	判断是否为空	O(1)
size()	返回元素个数	O(1)
push()	在队尾插入元素	O(1)
pop()	删除队首元素	O(1)

示例

```
#include <iostream>
#include <queue>  //头文件

int main(){
    std::queue<int> q;
    q.push(1);
    q.push(2);
    q.push(3);
    //q={1(front),2,3(back)}
    q.pop();//移除队首元素
    //q={2(front),3(back)}
    cout<<"此时的队首元素为"<<q.front()<<endl;
    //此时的队首元素为2
    cout<<"此时的队尾元素为"<<q.back()<<endl;
    //此时的队尾元素为3
    return 0;
}
```

priority_queue

优先队列 std::priority_queue 是一种堆，一般为二叉堆。

成员函数

函数名	功能	时间复杂度
top()	返回最大元素	O(1)
empty()	判断是否为空	O(1)
size()	返回元素个数	O(1)
push()	插入元素	O(log n)
pop()	删除最大元素	O(log n)

示例

```cpp
#include <iostream>
#include <queue>  //头文件
using namespace std;
int main(){
    priority_queue<int> q;
    /*默认为大根堆
    小根堆写法 priority_queue<int, vector<int>,greater<int> > p;
    自定义比较方式
    1、重载operator <
    bool operator<(Node a, Node b){//返回true时，说明a的优先级低于b
    //x值较大的Node优先级低（x小的Node排在队前）
    //x相等时，y大的优先级低（y小的Node排在队前）
    if(a.x==b.x) return a.y> b.y;
        return a.x> b.x;
    }
    2、重写仿函数 priority_queue<Node, vector<Node>, cmp> q;
    cmp为自定义的比较类
    struct compare {
        bool operator()(int a, int b) {
            return a > b; // 定义最小堆
        }
    };
    */
    q.push(10);
    q.push(1);
    q.push(20);
    //q={1,10,20(top)}
    q.pop();//移除队首元素
    //q={1,10(top)}
    cout<<"此时的队首元素为"<<q.top()<<endl;
    //此时的队首元素为10
    return 0;
}
```

ST表

ST表是一种基于倍增思想，用于解决可重复贡献问题的数据结构。

可重复贡献问题:对于运算opt满足$xoptx = x$，则对应的区间询问就是一个可重复贡献问题。例如，最大值有$max(x, x) = x$，gcd有$gcd(x, x) = x$，所以 RMQ(区间最大最小值) 和区间GCD就是一个可重复贡献问题。像区间和就不具有这个性质，如果求区间和的时候采用的预处理区间重叠了，则会导致重叠部分被计算两次，这是我们所不愿意看到的。另外，opt还必须满足结合律才能使用 ST 表求解

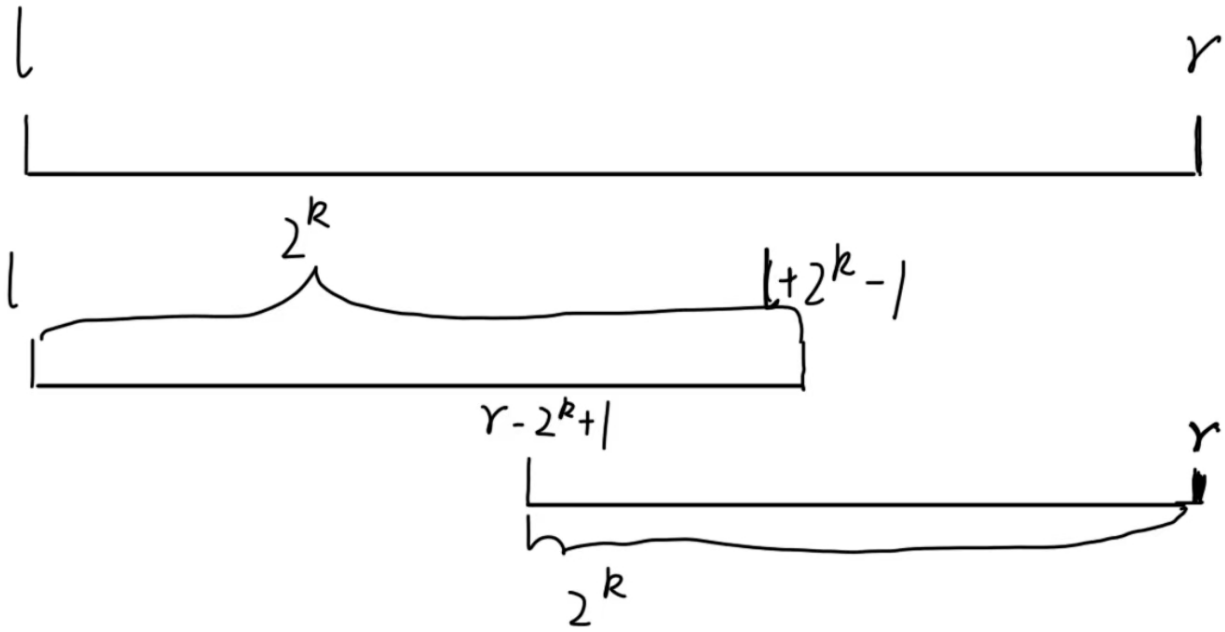
引入

问题：给定n个数，m个询问，对于每个询问了l,r，需要回答区间[l, r]中的最大值或最小值。

暴力求解对每次都对区间[l,r]扫描一遍显然会超时。 ST表基于倍增的思想,可以实现O(nlogn)下进行预处理，并在0(1)时间内回答每个询问。

实现

$f[i][j]$表示区间 $[i, i+2^j-1]$的最大值。 那么我们可以给出状态转移方程
$f[i][j]=max(f[i][j-1], f[i+2^{j-1}][j-1])$。 对于每个查询$[l,r]$,我们只需将区间分成两个子区间 $[l, l+2^k-1], [r-2^k+1, r]$,两端区间必须包含了所有所查区间的数



因此我们需要得出最大的k,即第一个字区间的右端点尽可能的接近r，第二个区间的左端点尽可能地接近l，得到$s=[log_2(r-l+1)]$ ($[x]$ 表示不超过x的最大整数，如[1.2]=1,[-2.3]=-3),为了保证查询复杂度为O(1)，我们需要提前预处理出每个$lo_2(r-l+1)$向下取整后的值。

$$log_2 i=\begin{cases}0 & i=1\\ log_2\frac{i}{2}+1 & i>1\end{cases}$$

总结 预处理有两部分1、$log_2[1\ldots n]$ 2、$f[i][j]=max(f[i][j-1], f[i+2^{j-1}][j-1])$

```
#include <bits/stdc++.h>
using namespace std;
const int N=5e6;
int smax[N][31],log_2[N];

int main(){
    int n,m;
    cin>>n>>m;
    log_2[1]=0;
    for(int i=2;i<=n;i++){
```

```
        log_2[i]=log_2[i/2]+1;//预处理log2
    }
    //也可以直接用以下函数计算log2,头文件为#include <cmath>
    //_lg(m):计算以2为底的对数,接受一个整数,返回一个整数
    //log2(m):计算以2为底的对数,接受一个参数(不一定要是整型)，返回一个double、float 或
    for(int i=1;i<=n;i++){
        cin>>smax[i][0];//smax[i][0]即区间(i,i)的最大值，即i
    }
    for(int j=1;j<=log_2[n];j++){
        for(int i=1;i+(1<<j)-1<=n;i++){
            smax[i][j]=max(smax[i][j-1],smax[i+(1<<(j-1))][j-1]);
        }
    }
    int l,r;
    for(int i=1;i<=m;i++){
        cin>>l>>r;
        int s=log_2[r-l+1];
        cout<<max(smax[l][s],smax[r-(1<<s)+1][s])<<endl;
    }
    return 0;
}
```