# 题目

请定义一个队列并实现函数 max\_value 得到队列里的最大值，要求函数max\_value、push\_back 和 pop\_front 的均摊时间复杂度都是O(1)。

若队列为空，pop\_front 和 max\_value 需要返回 -1

**示例 1：**

输入:

["MaxQueue","push\_back","push\_back","max\_value","pop\_front","max\_value"]

[[],[1],[2],[],[],[]]

输出: [null,null,null,2,1,2]

**示例 2：**

输入:

["MaxQueue","pop\_front","max\_value"]

[[],[],[]]

输出: [null,-1,-1]

**限制：**

1 <= push\_back,pop\_front,max\_value的总操作数 <= 10000

1 <= value <= 10^5

# 分析

## 方法一：暴力

**代码：**

class MaxQueue {

int q[20000];

int begin = 0, end = 0;

public:

MaxQueue() {

}

int max\_value() {

int ans = -1;

for (int i = begin; i != end; ++i)

ans = max(ans, q[i]);

return ans;

}

void push\_back(int value) {

q[end++] = value;

}

int pop\_front() {

if (begin == end)

return -1;

return q[begin++];

}

};

复杂度分析：

时间复杂度：O(1)（插入，删除），O(n)（求最大值）。

插入与删除只需要普通的队列操作，为O(1)，求最大值需要遍历当前的整个队列，最坏情况下为O(n)。

空间复杂度：O(n)，需要用队列存储所有插入的元素。

## 方法二：滑动窗口

class MaxQueue {

private:

queue<int> que;

deque<int> deq;

public:

MaxQueue() {

}

int max\_value() {

if(deq.empty())return -1;

return deq.front();

}

void push\_back(int value) {

while(!deq.empty() && deq.back()<value)

deq.pop\_back();

deq.push\_back(value);

que.push(value);

}

int pop\_front() {

if(que.empty())return -1;

int ret = que.front();

que.pop();

if(ret == deq.front()) deq.pop\_front();

return ret;

}

};

/\*\*

\* Your MaxQueue object will be instantiated and called as such:

\* MaxQueue\* obj = new MaxQueue();

\* int param\_1 = obj->max\_value();

\* obj->push\_back(value);

\* int param\_3 = obj->pop\_front();

\*/