# 题目

有些数的素因子只有 3，5，7，请设计一个算法找出第 k 个数。注意，不是必须有这些素因子，而是必须不包含其他的素因子。例如，前几个数按顺序应该是 1，3，5，7，9，15，21。

**示例 1:**

输入: k = 5

输出: 9

# 分析

## 方法一：堆

**思路：**

跟丑数类似，小顶堆（priority\_queue<> 默认最大堆,改成最小堆）

**代码：**

class Solution {

public:

int getKthMagicNumber(int k) {

typedef long long int64;

set<int64> values;

priority\_queue<int64,vector<int64>,greater<int64>> queue;

queue.push(1);

while (true) {

int64 value = queue.top();

queue.pop();

//处理重复数据（使用数据结构set），如不考虑去重不用set

if (!values.count(value)) {

values.insert(value);

if (value<INT\_MAX/3){

queue.push(value \* 3);

}

if (value<INT\_MAX/5){

queue.push(value \* 5);

}

if (value<INT\_MAX/7){

queue.push(value \* 7);

}

}

if (values.size() == k) {

return value;

}

}

return -1;

}

};

或者：

class Solution {

public:

int getKthMagicNumber(int k) {

priority\_queue<int,deque<int>,greater<int>> smallqueue;

unordered\_set<int> numset;

vector<int> factor = {3,5,7};

smallqueue.push(1);

int count = 0;

long long value = 0;

while(!smallqueue.empty() && count<k){

value = smallqueue.top();

smallqueue.pop();

count++;

for(auto &num:factor){

long long nextnum =value\*num ;

if(!numset.count(nextnum) &&nextnum<= INT32\_MAX){

smallqueue.push(nextnum);

numset.insert(nextnum);

}

}

}

return value;

}

};

或：

class Solution {

public:

int getKthMagicNumber(int k) {

set<long> q;//可以看做小顶堆

long ans;

q.insert(1);

while(k--)

{

ans = \*q.begin();

q.erase(q.begin());

q.insert(ans\*3);

q.insert(ans\*5);

q.insert(ans\*7);

// cout << ans << endl;

}

return ans;

}

};

## 方法二：动态规划

**思路：**

1、定义三个指针p3,p5,p7，p3指向的数字永远乘3，p5指向的数字永远乘5，p7指向的数字永远乘7

2、初始化所有指针都指向第一个丑数，即1

3、我们从dp[p3]\*3,dp[p5]\*5,dp[p7]\*7选取最小的一个数字，作为新的丑数。这边新的丑数就是3\*dp[p3]=3\*1=3，然后p3++

4、此时p5和p7指向第1个丑数，p3指向第2个丑数。然后重复上一步

5、这里基于的一个事实是，丑数数列是递增的，当p5指针在当前位置时，后面的数乘以5必然比前面的数乘以5大，所以下一个丑数必然是先考虑前面的数乘以5。p3,p7同理，所以才可以使用指针

**代码：**

public int getKthMagicNumber(int k) {

int p3=0,p5=0,p7=0;

int[] dp=new int[k];

dp[0]=1;

for(int i=1;i<k;i++){

dp[i]=Math.min(dp[p3]\*3,Math.min(dp[p5]\*5,dp[p7]\*7));

if(dp[i]==dp[p3]\*3) p3++;

if(dp[i]==dp[p5]\*5) p5++;

if(dp[i]==dp[p7]\*7) p7++;

}

return dp[k-1];

}