# 题目

编写一个程序，找出第n个丑数。

丑数就是只包含质因数2, 3, 5的正整数。

**示例:**

**输入:** n = 10

**输出:** 12

**解释:** 1, 2, 3, 4, 5, 6, 8, 9, 10, 12是前10个丑数。

**说明:**

1是丑数。

n不超过1690。

# 分析

## 方法一：暴力破解

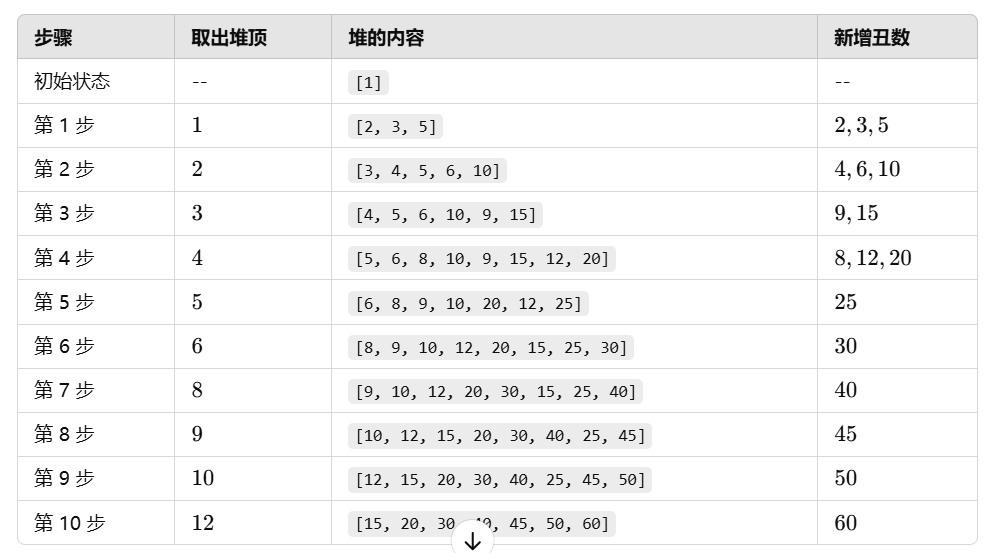
## 方法二：堆/优先队列（推荐）

**思路：**

这里是求第N个丑数，其实就是第K值的问题，只是这里是满足某种特性的第K值。因此，可以利用优先队列有自动排序的功能。

每次取出队头元素，存入队头元素\*2、队头元素\*3、队头元素\*5

但注意，像12这个元素，可由4乘3得到，也可由6乘2得到，所以要注意去重



**代码：**

class Solution {

public:

int nthUglyNumber(int n) {

priority\_queue <long,vector<long>,greater<long> > pri\_queue;

long ret = 1; // int计算后可能溢出，使用long接收

for (int i = 1; i < n; i++)

{

pri\_queue.push(ret \* 2);

pri\_queue.push(ret \* 3);

pri\_queue.push(ret \* 5); //这里不是i\*5

ret = pri\_queue.top();

pri\_queue.pop();

while (!pri\_queue.empty() && ret==pri\_queue.top()) //这里上一个top与后面的top可能相等，循环判断

//不是if，因为可能多个重复

pri\_queue.pop();

}

return (int)ret;

}

};

或：

class Solution {

public:

int nthUglyNumber(int n) {

priority\_queue <double,vector<double>,greater<double> > q;

double answer=1;

for (int i=1;i<n;++i)

{

q.push(answer\*2);

q.push(answer\*3);

q.push(answer\*5); //这里不是i\*5

answer=q.top();

q.pop();

while (!q.empty() && answer==q.top()) //这里上一个top与后面的top可能相等，循环判断

//不是if，因为可能多个重复

q.pop();

}

return answer;

}

};

或：

采用set来识别有无重复

class Solution {

public:

int nthUglyNumber(int n) {

priority\_queue <double,vector<double>,greater<double> > q;

set<int> s;

s.insert(1);

vector<int> mask({2,3,5});

double answer=1;

for (int i=1;i<n;++i)

{

for (int &j:mask)

if (s.count(answer\*j)==0)

{

q.push(answer\*j);

s.insert(answer\*j);

}

answer=q.top();

q.pop();

}

return answer;

}

};

## 方法三：动态规划（推荐）

**解题思路**

动态规划（双指针法）

1、动态规划定义：

- 用一个数组dp存储第n个丑数，其中dp[i]表示第i+1个丑数。

- 初始值：第一个丑数dp[0] = 1。

2、生成丑数：

- 用三个指针 p2, p3, p5 分别表示当前需要乘以2、3、5的位置。

- 下一个丑数为 min(dp[p2] \* 2, dp[p3] \* 3, dp[p5] \* 5)。

- 将选中的丑数放入dp，并移动对应指针。

3、重复操作：

- 每次选择最小值作为新的丑数，直到找到第n个丑数。

4、避免重复：

- 如果某个丑数由多个因子生成，所有对应的指针都要移动。

**实现代码：**

class Solution {

public:

int nthUglyNumber(int n) {

vector<int> dp(n); // 动态规划数组

dp[0] = 1; // 第一个丑数是 1

int p2 = 0, p3 = 0, p5 = 0; // 指针初始化

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

// 找出当前最小丑数

int nextUgly = min({dp[p2] \* 2, dp[p3] \* 3, dp[p5] \* 5});

dp[i] = nextUgly;

// 移动指针

if (nextUgly == dp[p2] \* 2) ++p2;

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

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

}

return dp[n - 1];

}

};

**代码解析**

1、初始化 dp 数组：

vector<int> dp(n);

dp[0] = 1;

- 丑数序列从1开始。

2、使用三个指针p2, p3, p5：

- p2表示当前乘2的位置。

- p3表示当前乘3的位置。

- p5表示当前乘5的位置。

3、在每一步找到最小丑数：

int nextUgly = min({dp[p2] \* 2, dp[p3] \* 3, dp[p5] \* 5});

4、更新对应指针，避免重复选取：

if (nextUgly == dp[p2] \* 2) ++p2;

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

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

5、最终返回第n个丑数：

return dp[n - 1];

**复杂度分析**

1、时间复杂度：

- 遍历n次，每次更新指针和计算最小值。

- 总复杂度为O(n)。

2、空间复杂度：

- 需要一个大小为n的数组dp，空间复杂度为O(n)。

扩展优化

如果需要优化空间，可以使用最小堆的方法或滑动窗口，将空间复杂度降低到O(1)。动态规划方法已经可以高效解决大多数实际问题。

**思路：**

我们先模拟手写丑数的过程

1打头，1乘2 1乘3 1乘5，现在是{1,2,3,5}

轮到2，2乘2 2乘3 2乘5，现在是{1,2,3,4,5,6,10}

手写的过程和采用小顶堆的方法很像，但是怎么做到提前排序呢

小顶堆的方法是先存再排，dp的方法则是先排再存

我们设3个指针p\_2，p\_3，p\_5

代表的是第几个数的2倍、第几个数3倍、第几个数5倍

动态方程dp[i]=min(dp[p\_2]\*2,dp[p\_3]\*3,dp[p\_5]\*5)

小顶堆是一个元素出来然后存3个元素

动态规划则是标识3个元素，通过比较他们的2倍、3倍、5倍的大小，来一个一个存

**代码：**

class Solution {

public:

int nthUglyNumber(int n) {

vector<int> dp(n);

dp.at(0)=1;

int p\_2,p\_3,p\_5;

p\_2=p\_3=p\_5=0;

for (int i=1;i<n;++i)

{

dp.at(i)=min(min(2\*dp.at(p\_2),3\*dp.at(p\_3)),5\*dp.at(p\_5));

if (dp.at(i)==2\*dp.at(p\_2))

++p\_2;

if (dp.at(i)==3\*dp.at(p\_3))

++p\_3;

if (dp.at(i)==5\*dp.at(p\_5))

++p\_5;

}

return dp.at(n-1);

}

};

或：

// 1, 2, 3, 4, 5, 6, 8, 9, 10, 12

class Solution {

public:

int nthUglyNumber(int n) {

vector<int> dp(n, 0);

dp[0] = 1;

int p2 = 0, p3 = 0, p5 = 0;

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

dp[i] = min(min(dp[p2] \* 2, dp[p3] \* 3), dp[p5] \* 5);

if (dp[i] == dp[p2] \* 2)

p2++;

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

p3++;

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

p5++;

}

return dp[n - 1];

}

};