# 题目

给定两个以升序排列的整形数组 nums1 和 nums2, 以及一个整数 k。

定义一对值 (u,v)，其中第一个元素来自 nums1，第二个元素来自 nums2。

找到和最小的 k 对数字 (u1,v1), (u2,v2) ... (uk,vk)。

**示例 1:**

**输入:** nums1 = [1,7,11], nums2 = [2,4,6], k = 3

**输出:** [1,2],[1,4],[1,6]

**解释:** 返回序列中的前 3 对数：

[1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]

**示例 2:**

**输入:** nums1 = [1,1,2], nums2 = [1,2,3], k = 2

**输出:** [1,1],[1,1]

**解释:** 返回序列中的前 2 对数：

  [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]

**示例 3:**

**输入:** nums1 = [1,2], nums2 = [3], k = 3

**输出:** [1,3],[2,3]

**解释:** 也可能序列中所有的数对都被返回:[1,3],[2,3]

# 分析

## 方法一：堆/优先队列

**思路：**

**代码：**

class Solution {

public:

struct cmp{

bool operator ()(pair<int, int> &a, pair<int, int> &b) {

return a.first+a.second<b.first+b.second;

//选择较小的那一组

}

};

vector<vector<int>> kSmallestPairs(vector<int>& nums1, vector<int>& nums2, int k) {

priority\_queue<pair<int,int>,vector<pair<int,int>>,cmp>q;

/\*

这里定义堆的基本元素是pair，不是根据返回值定义vector<int>，

根据我们具体比较运算的需要设计合理的入参

也可以这样定义：priority\_queue<vector<int>,vactor<vector<int>>>，仿函数使用数组下标访问，参考LeetCode 973

\*/

vector<vector<int>>res;

for(int i=0;i<nums1.size();i++){

for(int j=0;j<nums2.size();j++){

if(q.size()<k)

q.push({nums1[i],nums2[j]});

else if(nums1[i]+nums2[j]<q.top().first+q.top().second){

q.pop();

q.push({nums1[i],nums2[j]});

}

/\*

上述算法不能这样写：

if(q.size()<k)

q.push({nums1[i],nums2[j]});

else if(i+j>k)

//这个判断逻辑错误

q.pop();

q.push({nums1[i],nums2[j]});

\*/

}

}

while(!q.empty()){

pair<int,int> top=q.top();

res.push\_back({top.first,top.second});

q.pop();

}

return res;

}

};

另一种写法（推荐）：

class Solution {

public:

struct compare{

bool operator()(pair<int,int> &a, pair<int,int> &b)

{

return a.first+a.second < b.first+b.second;

}

};

public:

vector<vector<int>> kSmallestPairs(vector<int>& nums1, vector<int>& nums2, int k) {

std::priority\_queue<pair<int,int>,vector<pair<int,int>>,compare> pri\_queue;

vector<vector<int>> tmpArr;

for(int i=0;i<nums1.size();i++)

{

for(int j=0;j<nums2.size();j++)

{

pri\_queue.push({nums1.at(i),nums2.at(j)});

if(pri\_queue.size() > k)

{

pri\_queue.pop();

}

}

}

while(!pri\_queue.empty())

{

pair<int,int> top = pri\_queue.top();

tmpArr.push\_back({top.first,top.second});

pri\_queue.pop();

}

return tmpArr;

}

};

## 方法二：二分搜索