Given a nested list of integers, implement an iterator to flatten it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

Input: [[1,1],2,[1,1]]

Output: [1,1,2,1,1]

Explanation: By calling *next* repeatedly until *hasNext* returns false,

the order of elements returned by *next* should be: [1,1,2,1,1].

Example 2:

Input: [1,[4,[6]]]

Output: [1,4,6]

Explanation: By calling *next* repeatedly until *hasNext* returns false,

the order of elements returned by *next* should be: [1,4,6].

\*class NestedInteger {

\* public:

\* // Return true if this NestedInteger holds a single integer, rather than a nested list.

\* bool isInteger() const;

\*

\* // Return the single integer that this NestedInteger holds, if it holds a single integer

\* // The result is undefined if this NestedInteger holds a nested list

\* int getInteger() const;

\*

\* // Return the nested list that this NestedInteger holds, if it holds a nested list

\* // The result is undefined if this NestedInteger holds a single integer

\* const vector<NestedInteger> &getList() const;

\* };

[[],[a[,b,c]]]

class NestedIterator:

def \_\_init\_\_(self, nestedList: [NestedInteger]):

self.next\_element = None

self.underlying\_list = nestedList

self.stack = []

if self.underlying\_list:

self.stack.append((0, self.underlying\_list))

def next(self) -> int:

return self.next\_element.getInteger()

[a,[b,c]]

def hasNext(self) -> bool

# Retore the status

idx, active\_list = None, None

while self.stack:

idx, active\_list = self.stack.pop()

if not active\_list or idx >= len(active\_list):

continue

else:

break

if not active\_list:

return False

# [a, [b,c]]

for i in range(idx, len(active\_list):

obj = active\_list[i]

if obj.isInteger():

self.next\_element = obj

# Store current status and exit function

self.stack.append((i + 1, active\_list))

return True

elif obj.isList():

self.stack.append((i+1, active\_list))

self.stack.append((0, obj.getList())

return self.hasNext()

else:

assert False

973. K Closest Points to Origin

We have a list of points on the plane. Find the K closest points to the origin (0, 0).

#include <cmath>

//headers for pq??

#include <queue>

struct comp {   
 bool operator()(const pair<vector<int>, int>& p1, const pair<vector<int>, int>& p2) {  
 if (p1.second < p2.second) {   
 return true;

}

return false;

}

};

class Solution {

public:

vector<vector<int>> kClosest(vector<vector<int>>& points, int K) {

priority\_queue<vector<int>, vector<vector<int>>, comp> pq;

for (vector<int>& point : points) {

int dist = sqrt(point[0] \* point[0] + point[1] \* point[1]);  
pq.push({point, dist});

if (pq.size() > K) {   
 pq.pop();

}

}

vector<vector<int>> ans;

while (!pq.empty()) {

ans.push\_back(pq.top().first);

pq.pop();

}

return ans;

}

};

K = 2;

[[1,2], [2, 3], [-1, -5]]

nlogk

31. Next Permutation

1,2,3 -> 1,3,2

1, 3, 2, -> 2, 1, 3

nums -> List

vector<int> nums,

vector<int> next\_nums;

3, 2, 1 -> 1, 2, 3

5, 7, 5, 3, 2 -> 5, 7, -> 7 5 3 2 - 7 2 3 5 - 5 7 5 2

1 1 2 -> 1 2 1 -> 2 1 1

5 5 3 2, 7 2 3 5 5

2, 3, 5, 7, 5

vector<int> NextPermutation(vector<int>& nums) {

vector<int> ans = nums;  
 if (nums.size() <= 1) {   
 return nums;

}

int i = nums.size() - 2;

for (; i >= 0; i--) {

// <=

if (nums[i] < nums[i + 1]) {

break;

}

}

if (i == -1) {

reverse(begin(ans), end(ans));

return ans;

}

// 5, 7, 5, 3, 2

//7, 5, 5, 3, 2

//7, 2, 3, 5, 5

//start from i + 1, find the minimum number nums[k] > nums[i], switch nums[i], nums[k], reverse nums k to the end;

reverse(ans.begin() + i + 1, ans.end());

int k = i + 1;

for (; k < ans.size(); k++) {   
 if (ans[k] > ans[i]) {   
 break;

}

}

swap(ans[i], ans[k]);

/\*

int k = i + 1;

for ( ; k < ans.size(); k++) {

if (ans[k] <= ans[i]) {  
 break;

}

}

k--;

swap(ans[k], ans[i]);

//reverse(ans.begin() + k, ans.end());

\*/

return ans;

}

5, 7, 5, 3, 2 -> 5, 7, -> 7 5 3 2 - 7 2 3 5 - 5 7 5 2

5, 2, 3, 5 ,7 -> 7, 2, 3, 5, 5

5 5 1 5-> 5, 5, 5, 1

5 1 5 5 -> 5, 5, 1, 5

5, 1, 5, 5, -> i = k points to 1 -> swap 1, 5 ->5, 5, 1, 5

i = 1