**Lab 6.**

Solve these problems for the priority queue.

Deadline: Week 8.

<https://leetcode.com/problems/kth-largest-element-in-an-array/>

Find the **k**th largest element in an unsorted array.

Solve: I take priority queue and I will push element of array to the priority queue in descending order, and if size of priority queue larger than k I will delete it, After that I will return top element of priority queue;

class Solution {

public:

int findKthLargest(vector<int>& nums, int k) {

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

for(int num : nums){

q.push(num);

if(q.size() > k)

q.pop();

}

return q.top();

}

};

<https://leetcode.com/problems/kth-smallest-element-in-a-sorted-matrix/>

Given a *n* x *n* matrix where each of the rows and columns are sorted in ascending order, find the kth smallest element in the matrix.

I will take priority queue, then loop through row and column and I will add element of array to the priority queue in ascending order, and if size of priority queue larger than k I will delete it, After that I will return top element of priority queue.

class Solution {

public:

int kthSmallest(vector<vector<int>>& matrix, int k) {

priority\_queue<int> q;

for(auto row : matrix){

for(auto el : row){

q.push(el);

if(q.size() > k)

q.pop();

}

}

return q.top();

}

};

<https://leetcode.com/problems/top-k-frequent-elements/>

Given a non-empty array of integers, return the ***k*** most frequent elements

I firstly count one element occurs how many times while writing to the unordered\_map. Then I will take pair of two int of priority queue and I will add elements of unordered\_map to the priority queue, in format to the first element I will add value(second element) of map and to the second element I will add key (first element ) of map.

And will write top K elements of priority queue to the vector as a result, then return result.

class Solution {

public:

vector<int> topKFrequent(vector<int>& nums, int k) {

unordered\_map<int ,int> count;

for(int x : nums)

count[x]++;

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

for(auto c : count){

q.push({c.second, c.first});

}

vector<int> res;

while(k > 0 && !q.empty()){

res.push\_back(q.top().second);

q.pop();

k--;

}

return res;

}

};

<https://leetcode.com/problems/top-k-frequent-words/>

Given a non-empty list of words, return the *k* most frequent elements.

Your answer should be sorted by frequency from highest to lowest. If two words have the same frequency, then the word with the lower alphabetical order comes first.

I firstly count one element occurs how many times while writing to the unordered\_map. Then I will take pair of two int of priority queue and I will add elements of unordered\_map to the priority queue, and in priority queue I write own comparator myCmp in which takes two pairs String and int. There are

I will compare by frequency of first and second element and return if they are in descending order, else if they frequency equal then I will sort they in lower alphabetical order.

And will write top K elements of priority queue to the vector as a result, then return result.

class Solution {

public:

struct myCmp {

constexpr bool operator()(pair<string, int> &left, pair<string, int> &right) const noexcept

{

if (left.second != right.second)

return left.second < right.second;

return (left.first > right.first);

}

};

vector<string> topKFrequent(vector<string>& words, int k) {

unordered\_map<string, int> F;

for(string word: words){

F[word]++;

}

priority\_queue<pair<string, int>, vector<pair<string, int>>, myCmp > q;

for(auto f : F){

q.push({f.first, f.second});

}

vector<string> res;

while(k > 0 && !q.empty()){

auto w = q.top();

res.push\_back(w.first);

q.pop();

k--;

}

return res;

}

};