Fila de Prioridades
Professor Wladimir

Heap/Fila de Prioridade

Uma fila de prioridades é uma estrutura de dados que armazena elementos associados a uma prioridade, permitindo acessar rapidamente o elemento de maior (ou menor) prioridade.

Características principais:

- Elementos são inseridos de forma arbitrária.
- A remoção (pop) sempre retorna o elemento com maior prioridade (em um max-heap) ou menor prioridade (em um min-heap).
- Internamente, é frequentemente implementada como um heap binário, garantindo eficiência nas operações.

Operações comuns:

- push: insere um elemento com prioridade.
- pop : remove e retorna o elemento com maior (ou menor) prioridade.
- top: consulta o elemento com maior (ou menor) prioridade sem removê-lo.

Complexidade típica (com heap):/

- Inserção: O(log n)
- Remoção do elemento de maior prioridade: O(log n)
- Consulta do elemento de maior prioridade: O(1)

Exemplos de uso:

- Algoritmo de Dijkstra (menor caminho).
- Ordenação por heapsort.
- Simulação de eventos com base em tempos.

Last Stone

Dado um vetor de inteiros stones, onde cada elemento representa o peso de uma pedra, o objetivo é simular um jogo no qual, a cada rodada, as duas pedras mais pesadas são escolhidas para serem ”quebradas”uma contra a outra:

- Se os pesos forem iguais, ambas são destruídas.
- Se forem diferentes, a de menor peso é destruída e a maior tem seu peso reduzido pela diferença.

Esse processo se repete até restar no máximo uma pedra. O problema pede que você retorne o peso da última pedra restante ou 0 se todas forem destruídas.

https://leetcode.com/problems/last-stone-weight/description/

```cpp
class Solution {
public:
    int lastStoneWeight(vector<int>& stones) {
        priority_queue <int> q;

        for(int x : stones) q.push(x);
        int x, y;
        while( q.size() >= 2){
            y = q.top(); q.pop();
            x = q.top(); q.pop();

            if(x == y) continue;
            else{
                q.push(y-x);
            }
        }

        if(q.size() == 0) return 0;
        else return q.top();



    }
};
```

Top K Frequent Elements

Dado um vetor de inteiros nums e um inteiro k, o objetivo é retornar os k elementos que ocorrem com mais frequência no vetor. A ordem dos elementos na resposta não importa.

Exemplo:

Entrada: nums = [1,1,1,2,2,3], k = 2
Saída: [1,2] (os dois elementos mais frequentes)
Entrada: nums = [1], k = 1
Saída: [1] (único elemento disponível)
https://leetcode.com/problems/top-k-frequent-elements/description/

```cpp
#define ALL(c) c.begin(), c.end()
typedef pair<int,int> ii;
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        map <int,  int> freq;

        for(auto x : nums)
            freq[x]++;

        vector < ii > pairs;

        for(auto & [k,v] : freq){
            pairs.push_back( make_pair(v, k) );
        }

        priority_queue < ii > pq (ALL(pairs));

        vector <int> res;
        for(int i = 0; i < k ; i++){
            int fx, x;
            tie(fx, x) = pq.top();
            res.push_back(x);
            pq.pop();
        }
        return res;
    }
};
```

K Closest Points to Origin

Dado um vetor de pontos $points$, onde cada ponto é representado por $[x_i, y_i]$ no plano cartesiano, e um inteiro k, o objetivo é retornar os k pontos mais próximos da origem $(0, 0)$ com base na **distância euclidiana**.

Distância Euclidiana

A distância entre um ponto (x, y) e a origem é dada por:

$$d = \sqrt{x^2 + y^2}$$

Exemplo

- **Entrada:** $points = [[1, 3], [-2, 2]]$, $k = 1$
- **Saída:** $[[-2, 2]]$

Observações

- A resposta pode ser retornada em qualquer ordem.
- A resposta é garantidamente única (exceto pela ordem).

https://leetcode.com/problems/k-closest-points-to-origin/description/

```cpp
class Solution {
public:
    vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
        priority_queue < pair<double, vector<int>> > q;

        for(int i = 0; i < points.size(); i++){
            double dist = points[i][0]*points[i][0] + points[i][1]*points[i][1];
            q.push( { -dist, points[i] } );
        }

        vector<vector<int>> res;

        for(int i = 0; i < k; i++){
            auto p = q.top(); q.pop();
            res.push_back(p.second);
        }

        return res;
    }
};
```

Maximum Product of Two Elements in an Array

https://leetcode.com/problems/maximum-product-of-two-elements-in-an-array/description/

```cpp
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        priority_queue <int> q1( nums.begin(), nums.end() );
        int x, y;
        x = q1.top(); q1.pop();
        y = q1.top(); q1.pop();

        x--;
        y--;

        int res1 = x*y;

        if(q1.size()==2) return res1;


        for(auto & x : nums) x = -x;

        priority_queue <int> q2( nums.begin(), nums.end() );

        x = q2.top(); q2.pop();
        y = q2.top(); q2.pop();

        if(x<0 || y <0) return res1;

        x--;
        y--;


        int res2 = x*y;

        return max(res1, res2);


    }
};
```

Telemarketing

Uma empresa de telemarketing possui uma lista de ligações a serem realizadas, cada uma com uma duração específica em minutos. A empresa conta com N vendedores, identificados pelos inteiros de 1 a N. O objetivo é simular a distribuição das ligações entre os vendedores, respeitando as seguintes regras:

Regras de Atribuição

1. Inicialmente, todos os vendedores estão inativos.
2. Quando um vendedor realiza uma ligação, ele fica ocupado por T minutos, onde T é o tempo da ligação.
3. O tempo entre duas ligações consecutivas de um mesmo vendedor é desprezível.
4. Um vendedor não pode realizar mais de uma ligação ao mesmo tempo.
5. A ligação no topo da lista deve ser atribuída ao vendedor inativo com menor identificador.
6. Assim que um vendedor termina uma ligação, ele se torna inativo e pode receber uma nova ligação.
7. A ligação atribuída é removida da lista de ligações pendentes.

Objetivo

Simular o processo de distribuição das ligações entre os vendedores e determinar, para cada ligação:

- Qual vendedor a realizou;
- Em que instante ela foi iniciada.

Exemplo

Considere $N = 4$ vendedores e uma lista de 6 ligações com tempos (em minutos): $[5, 2, 3, 3, 4, 9]$.

- Inicialmente, os 4 vendedores recebem as 4 primeiras ligações:
  - Vendedor 1: ligação de 5 min (termina no tempo 5)
  - Vendedor 2: ligação de 2 min (termina no tempo 2)
  - Vendedor 3: ligação de 3 min (termina no tempo 3)
  - Vendedor 4: ligação de 3 min (termina no tempo 3)
- O vendedor 2 termina primeiro e pega a próxima ligação (de 4 min), iniciando no tempo 2.
- Entre os vendedores 3 e 4, ambos terminam no tempo 3, mas o vendedor 3 tem menor identificador e realiza a última ligação (de 9 min), iniciando no tempo 3.

https://br.spoj.com/problems/TELEMAR7/

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> ii;

int main(){

    int N, M;

    cin >> N >> M;

    vector <int> cnt;

    cnt.assign(N, 0);

    priority_queue < ii, vector <ii>, greater<ii> > q;


    for(int i = 0; i < N; i++){
        q.push({0,i});
    }

    for(int i = 0; i < M; i++){
        int T;
        cin >> T;

        auto p = q.top(); q.pop();

        //printf("atendendo %d livre %d\n", p.second, p.first);

        cnt[p.second]++;

        q.push({p.first + T, p.second});

    }

    for(int i = 0; i < N; i++){
        printf("%d %d\n", i+1, cnt[i]);
    }


}
```

703. Kth Largest Element in a Stream

https://leetcode.com/problems/kth-largest-element-in-a-stream/

Código

```cpp
class KthLargest {
private:
    int k;
    priority_queue<int, vector<int>, greater<int>> minHeap;


public:
    KthLargest(int k, vector<int>& nums) {
        int i;
        this->k = k;

        if( nums.size() < k){
            for(i = 0; i < nums.size() ; i++){
                minHeap.push( nums[i] );
            }
        }else{
            for(i = 0; i < k ; i++){
                minHeap.push( nums[i] );
            }

            for(i = k; i < nums.size(); i++){
                if( nums[i] >= minHeap.top() ){
                    minHeap.push( nums[i] );
                    minHeap.pop();
                }
            }
        }

    }

    int add(int val) {

        if( minHeap.size() < k){
            minHeap.push( val );
            return minHeap.top();
        }else if( val < minHeap.top() )
            return minHeap.top();
        else{
            minHeap.push( val );
            minHeap.pop();
            return minHeap.top();
        }
    }
};

/**
 * Your KthLargest object will be instantiated and called as such:
 * KthLargest* obj = new KthLargest(k, nums);
 * int param_1 = obj->add(val);
 */
```

Haunted

O rei dos fantasmas deseja incentivar sua raça a assustar humanos novamente, organizando uma competição de N dias entre M fantasmas. Cada fantasma possui uma idade única entre 1 e M.

Regras da Competição

- Em cada um dos N dias, um dos fantasmas recebe o título de **"Fantasma do Dia"**, por ter assustado mais humanos.
- Ao final de cada dia, o rei entrega o **"Troféu de Consistência"** ao fantasma que acumulou o maior número de títulos de "Fantasma do Dia" até aquele momento.
- Em caso de empate no número de títulos, o troféu vai para o **fantasma mais velho** entre os empatados.

Entrada

- A primeira linha contém dois inteiros: N $(1 \leq N \leq 10^5)$ e M $(1 \leq M \leq 10^9)$.
- A segunda linha contém N inteiros, onde o i-ésimo inteiro representa a idade do fantasma que ganhou o título de "Fantasma do Dia" no i-ésimo dia.

Saída

Para cada um dos N dias, imprima uma linha com dois inteiros:

- A idade do fantasma que recebeu o "Troféu de Consistência" ao final do dia.
- O número total de títulos de "Fantasma do Dia" que esse fantasma acumulou até então.

Exemplo

Entrada:

```
7 5
1 3 1 3 2 2 2
```

Saída:

```
1 1
3 1
1 2
3 2
3 2
3 2
2 3
```

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef pair<int,int> ii;

struct ComparePair {
    bool operator()(const ii & p1, const ii & p2) {
        if(p1.second != p2.second )
                    return p1.second < p2.second;
                else
                    return p1.first < p2.first;
    };
};

int main(){
    int N, M;
    cin >> N >> M;
    map < int,int > wins;
    priority_queue < ii, vector <ii>,  ComparePair > q;
    for(int i = 0; i < N; i++){
        int x;
        cin >> x;
        x--;
        wins[x]++;
        q.push( make_pair(x, wins[x]) );
        cout << q.top().first+1 << " " << q.top().second << endl;
    }
}
```

Exercícios

1. https://leetcode.com/problems/find-subsequence-of-length-k-with-the-largest-sum/description
2. https://leetcode.com/problems/car-pooling/description/
3. https://leetcode.com/problems/minimum-amount-of-time-to-fill-cups/description/