Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

**Отчёт по задачам блока №3**

**По дисциплине «Алгоритмы и структуры данных»**

Группа: P3212

Выполнил: Балин А. А.

Проверил: Косяков М. С.

# Задача I

## Решение

Если у нас есть k машинок, которые могут стоять на полу, мы возьмём первые k различных машинок и поставим их на пол. Если у нас появляется новая машинка, нам нужно заменить её на ту, которая появится самой последней или не появится вообще в списке мальчика среди тех, которые уже стоят на полу. Самая худшая сложность используемых методов: поиск в map O(log(n)), мы делаем это в худшем случае для каждой машинки, поэтому сложность O(n\*log(n)).

## Код

#include <iostream>

#include <list>

#include <map>

#include <unordered\_set>

#include <vector>

#define INT\_MAX 2147483647

using namespace std;

struct ReverseCompare {

bool operator()(const int &a, const int &b) const { return a > b; }

};

int main() {

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

cout.tie(0);

int n, k, p;

cin >> n >> k >> p;

int s = k;

vector<list<int>> pos(n);

vector<int> values(p);

unordered\_set<int> curr;

for (int i = 0; i < p; i++) {

int x;

cin >> x;

values[i] = x;

pos[values[i] - 1].push\_back(i);

}

map<int, int> prev;

multimap<int, int, ReverseCompare> reverse\_prev;

int i = 0;

while (curr.size() < k && i < p) {

int x = values[i];

curr.insert(x);

if (pos[x - 1].size() > 1) {

pos[x - 1].pop\_front();

if (pos[x - 1].size() > 0) {

prev.emplace(x, pos[x - 1].front());

reverse\_prev.emplace(pos[x - 1].front(), x);

} else {

prev.emplace(x, INT\_MAX);

reverse\_prev.emplace(INT\_MAX, x);

}

} else {

pos[x - 1].pop\_front();

reverse\_prev.emplace(INT\_MAX, x);

}

i++;

}

while (i < p) {

int x = values[i];

if (curr.find(x) == curr.end()) {

int y = reverse\_prev.begin()->second;

reverse\_prev.erase(reverse\_prev.begin());

curr.erase(y);

curr.insert(x);

prev.erase(y);

pos[x - 1].pop\_front();

if (pos[x - 1].size() > 0) {

prev.emplace(x, pos[x - 1].front());

reverse\_prev.emplace(pos[x - 1].front(), x);

} else {

prev.emplace(x, INT\_MAX);

reverse\_prev.emplace(INT\_MAX, x);

}

s++;

} else {

int e = pos[x - 1].front();

pos[x - 1].pop\_front();

if (pos[x - 1].size() > 0) {

prev[x] = pos[x - 1].front();

reverse\_prev.erase(e);

reverse\_prev.emplace(pos[x - 1].front(), x);

} else {

prev[x] = INT\_MAX;

reverse\_prev.erase(e);

reverse\_prev.emplace(INT\_MAX, x);

}

}

i++;

}

cout << s << endl;

}

# Задача J

## Решение

Идейно мы делаем ровно то, что написано в условии задачи: расставляем гоблинов, удаляем их из очереди в соответствии с приоритетами и порядком, нужно только сказать, что лучше всего для вставки в середину некоторой коллекции используем list. Так как вставка и удаление происходит за O(1), пройдя по всем гоблинам, мы имеем O(n).

## Код

#include <iostream>

#include <list>

using namespace std;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(nullptr);

list<int> goblins;

list<int>::iterator mid =

goblins.end(); // Pointer to the middle or middle-left element

int n;

cin >> n;

for (int i = 0; i < n; i++) {

char c;

cin >> c;

if (c == '+') {

int x;

cin >> x;

goblins.push\_back(x);

if (goblins.size() == 1) {

mid = goblins.begin();

}

else if (goblins.size() % 2 != 0) {

if (next(mid) == goblins.end()) {

mid = prev(mid);

}

else {

mid = next(mid);

}

}

}

else if (c == '-') {

cout << goblins.front() << endl;

if (goblins.size() % 2 == 0) {

mid = next(mid);

}

goblins.pop\_front();

}

else if (c == '\*') {

int x;

cin >> x;

if (goblins.size() == 0) {

goblins.push\_back(x);

mid = goblins.begin();

}

else {

goblins.insert(next(mid), x);

if (goblins.size() % 2 != 0) {

mid = next(mid);

}

}

}

}

return 0;

}

# Задача K

## Решение

Изначально решение основывалось на хранении занятых ячеек и свободных справа от нее ячеек в одной структуре, а сами структуры я пытался хранить в куче, немного поиграв с компаратором для своей структуры. Но выяснилось, что я не существует структуры данных, которая бы быстро выполняла как поиск, вставку удаление, так и итерацию. Тогда я вспомнил, каким образом мы писали собственный метод free на С на семинарах/лабе по языкам программирования, а именно использовать структуру по подобию двусвязного списка, то есть структура должна иметь ссылку на следующий и предыдущий элемент в некотором множестве. А в мапе мы храним ссылку на занятый блок и количество свободных ячеек справа от него. Сложность O(nlog(n)).

## Код

#include <iostream>

#include <map>

#include <set>

using namespace std;

#define int long long

struct memory\_task {

int start;

int end;

memory\_task\* left;

memory\_task\* right;

};

signed main() {

int n, m;

cin >> n >> m;

set<pair<int, memory\_task\*>> T;

map<int, memory\_task\*> inds;

memory\_task head = { 0, 0, nullptr, nullptr };

memory\_task tail = { n + 1, n + 1, nullptr, nullptr };

head.right = &tail;

tail.left = &head;

T.insert(make\_pair(0, &head));

T.insert(make\_pair(n + 1, &tail));

for (int t = 1; t <= m; t++) {

int param;

cin >> param;

if (param > 0) {

auto it = T.lower\_bound(make\_pair(param + 1, nullptr));

if (it == T.end()) {

cout << -1 << endl;

}

else {

memory\_task\* task = it->second;

memory\_task\* new\_task = new memory\_task{

task->left->end + 1, task->left->end + param, task->left, task };

task->left->right = new\_task;

task->left = new\_task;

inds[t] = new\_task;

cout << new\_task->start << endl;

T.insert(make\_pair(new\_task->start - new\_task->left->end, new\_task));

T.erase(it);

T.insert(make\_pair(task->start - new\_task->end, task));

}

}

else {

param = -param;

if (inds.find(param) != inds.end()) {

memory\_task\* task = inds[param];

T.erase(make\_pair(task->start - task->left->end, task));

task->left->right = task->right;

task->right->left = task->left;

T.erase(make\_pair(task->right->start - task->end, task->right));

T.insert(make\_pair(task->right->start - task->left->end, task->right));

delete task;

}

}

}

}

# Задача L

## Решение

Пользуясь тем, что list будет сортировать пару значений по значению числа, а затем по его позиции, мы храним list c числами и их позициями. Первый элемент всегда будет минимальным в list, поэтому мы будем удалять элементы с вершины до тех пор, пока не найдётся элемент, влезающий в окно, причём по замечанию выше он будет минимальным. Вставка удаление за O(1), по всем элементам: O(n).

## Код

#include <iostream>

#include <list>

using namespace std;

int main() {

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

cout.tie(0);

int n, k;

cin >> n >> k;

list<pair<int, int>> window;

for (int i = 0; i < n; i++) {

int x;

cin >> x;

while (!window.empty() && window.front().second < i - k + 1) {

window.pop\_front();

}

while (!window.empty() && window.back().first > x) {

window.pop\_back();

}

window.push\_back({ x, i });

if (i > k - 2) {

cout << window.front().first << ' ';

}

}

}