Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №3**

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P32092

Голиков А.С.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задача I «Машинки»

Пояснение к примененному алгоритму:

Нам нужно знать, что сейчас лежит на полу и порядок, в котором стоит убирать машинки.

Для ответа на первый вопрос мы заводим unordered\_set, а для второго - priority\_queue, причем в очереди тот номер машинки будет лежать раньше, который понадобится позже всех.

Докажем, что такой компаратор хороший - предположим противное и пусть есть какой-то другой порядок, при котором в итоге понадобится сделать меньше операций.

Найдем первое расхождение в очередях - мой приоритет говорит, что нужно убрать x, а какой-то другой - что нужно убрать y.

Тогда есть два варианта - нам элемент y когда-то понадобится, а мы его убрали, руководствуясь вторым приоритетом(плохо...), либо мы убрали y в обоих случаях, тогда подходы эквивалентны.

Таким образом видим, что предложенный авторитет не хуже, чем любой другой.

Для решения осталось только провести симуляцию игры мальчика и посчитать, сколько операций замены машинок нужно совершить.

Асимптотика - O(plogp)

Код решения:

#include <iostream>

#include <queue>

#include <unordered\_set>

#include <unordered\_map>

#include <vector>

using namespace std;

int main() {

unordered\_set<int> floor\_set;

unordered\_map<int, queue<int>> que\_map;

int ans = 0;

struct ptr\_less {

bool operator()(pair<int, int> left, pair<int, int> right) const {

return (left.second < right.second);

}

};

priority\_queue<pair<int, int>, vector<pair<int, int>>, ptr\_less> floor\_queue;

int n, k, p;

cin >> n >> k >> p;

vector<int> que\_vec(p);

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

int x;

cin >> x;

que\_vec[i] = x;

if (que\_map.find(x) == que\_map.end()) {

queue<int> x\_queue;

que\_map[x] = x\_queue;

}

que\_map[x].push(i);

}

for (auto &i: que\_vec) {

que\_map[i].pop();

if (floor\_set.find(i) == floor\_set.end()) {

if (floor\_set.size() == k) {

int car\_to\_del = floor\_queue.top().first;

floor\_set.erase(car\_to\_del);

floor\_queue.pop();

}

floor\_set.insert(i);

++ans;

}

if (que\_map[i].empty()) que\_map[i].push(p + 1);

floor\_queue.emplace(i, que\_map[i].front());

}

cout << ans;

return 0;

}

Задача J «Гоблины и очереди»

Пояснение к примененному алгоритму:

Нужна структура с быстрым доступом к концу, началу и середине, быстрой вставкой и удалением.

Идеальный кандидат - двусвязный список с указателями на начало, конец и середину (будем двигать каждую вторую вставку)

В общем, все..

Асимптотика - O(n)

Код решения:

#include <iostream>

using namespace std;

int parse\_int(const string &s) {

int ans = 0;

for (char i: s) {

ans \*= 10;

ans += i - '0';

}

return ans;

}

struct list {

int value;

struct list \*next;

struct list \*prev;

};

struct list \*node\_create(int value) {

auto \*a = (list \*) malloc(sizeof(struct list));

\*a = (struct list) {.value = value, .next = nullptr, .prev = nullptr};

return a;

}

void add\_front(struct list \*\*old, struct list \*\*mid, struct list \*\*start, int value, bool \*count) {

struct list \*a = node\_create(value);

if (!(\*start)) {

\*start = a;

}

a->next = \*old;

if (\*old) {

(\*old)->prev = a;

}

\*old = a;

(\*count) ^= 1;

if (\*count) {

if (\*mid) {

\*mid = (\*mid)->prev;

} else {

\*mid = \*old;

}

}

}

void insert\_front(struct list \*\*end, struct list \*\*mid, struct list \*\*start, int value, bool \*count) {

if ((\*end) == (\*start)) {

return add\_front(end, mid, start, value, count);

}

struct list \*a = node\_create(value);

a->next = \*mid;

a->prev = (\*mid)->prev;

if ((\*mid)->prev) {

(\*mid)->prev->next = a;

}

(\*mid)->prev = a;

(\*count) ^= 1;

if (\*count) {

\*mid = a;

}

}

void delete\_back(struct list \*\*mid, struct list \*\*start, bool \*count) {

cout << (\*start)->value << "\n";

(\*start) = (\*start)->prev;

(\*count) ^= 1;

if (\*count) {

(\*mid) = (\*mid)->prev;

}

}

int main() {

struct list \*end = nullptr;

struct list \*mid = end;

struct list \*start = end;

bool count = false;

int n;

cin >> n;

cin.ignore();

for (size\_t i = 0; i < n; ++i) {

string s;

getline(cin, s);

switch (s[0]) {

case '+':

add\_front(&end, &mid, &start, parse\_int(s.substr(2)), &count);

break;

case '\*':

insert\_front(&end, &mid, &start, parse\_int(s.substr(2)), &count);

break;

default:

delete\_back(&mid, &start, &count);

}

}

return 0;

}

Задача K «Менеджер памяти-1»

Пояснение к примененному алгоритму:

Давайте выделять память в минимальном куске, в котором можно и при удалении мержить с соседними пустыми кусками памяти.

Для этого заведем мапы, дающие по началу свободного простраства конец, по концу - начало, мультимапу длин, дающих адреса кусков и вектор, помнящий куда когда чо выделили

Конец :)

Асимптотика - O(mlogn)

Код решения:

#include <iostream>

#include <map>

#include <vector>

using namespace std;

int main() {

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

cin.tie(nullptr);

cout.tie(nullptr);

int n, m;

cin >> n >> m;

multimap<size\_t, pair<size\_t, size\_t>> blocks\_length = **{**{n, {1, n}}**}**;

map<size\_t, size\_t> blocks\_fi\_by\_st = **{**{1, n}**}**;

map<size\_t, size\_t> blocks\_st\_by\_fi = **{**{n, 1}**}**;

vector<pair<size\_t, size\_t>> allocations(m, {1, n});

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

int req;

cin >> req;

if (req > 0) {

auto it = blocks\_length.lower\_bound(req);

if (it != blocks\_length.end()) {

size\_t st = it->second.first;

size\_t fin = it->second.second;

blocks\_length.erase(it);

blocks\_fi\_by\_st.erase(st);

blocks\_st\_by\_fi.erase(fin);

if (st + req <= fin) {

blocks\_length.emplace(fin - st + 1 - req, make\_pair(st + req, fin));

blocks\_fi\_by\_st.emplace(st + req, fin);

blocks\_st\_by\_fi.emplace(fin, st + req);

}

allocations[i] = make\_pair(st, st + req - 1);

cout << st << "\n";

} else {

cout << "-1\n";

}

} else {

pair<size\_t, size\_t> alloc = allocations[-1 - req];

size\_t st = alloc.first;

size\_t fn = alloc.second;

if (blocks\_fi\_by\_st.find(fn + 1) != blocks\_fi\_by\_st.end()) {

size\_t x = blocks\_fi\_by\_st[fn + 1];

blocks\_fi\_by\_st.erase(fn + 1);

blocks\_st\_by\_fi.erase(x);

for (auto it = blocks\_length.find(x - fn);

it != blocks\_length.end() && it->first == x - fn; ++it) {

if (it->second.first == fn + 1) {

blocks\_length.erase(it);

break;

}

}

fn = x;

}

if (blocks\_st\_by\_fi.find(st - 1) != blocks\_st\_by\_fi.end()) {

size\_t x = blocks\_st\_by\_fi[st - 1];

blocks\_fi\_by\_st.erase(x);

blocks\_st\_by\_fi.erase(st - 1);

for (auto it = blocks\_length.find(st - x);

it != blocks\_length.end() && it->first == st - x; ++it) {

if (it->second.first == x) {

blocks\_length.erase(it);

break;

}

}

st = x;

}

blocks\_fi\_by\_st.emplace(st, fn);

blocks\_st\_by\_fi.emplace(fn, st);

blocks\_length.emplace(fn - st + 1, make\_pair(st, fn));

}

}

return 0;

}

Задача L «Минимум на отрезке»

Пояснение к примененному алгоритму:

Дерево отрезков...

Асимптотика - O(nlogn)

Код решения:

#include <iostream>

#include <cmath>

#include <vector>

using namespace std;

int minim(int l, int r, int x, int lx, int rx, vector<int> \*tree) {

if ((l >= rx) || (lx >= r)) {

return 100001;

}

if ((lx >= l) && (rx <= r)) {

return (\*tree)[x];

}

int mid = (lx + rx) / 2;

int minl = minim(l, r, 2 \* x + 1, lx, mid, tree);

int minr = minim(l, r, 2 \* x + 2, mid, rx, tree);

return min(minl, minr);

}

int main() {

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

int n, k;

cin >> n >> k;

int l\_index = (int) pow(2, (int) round(log2(n) + 0.5)) - 1;

vector<int> tree(l\_index \* 2 + 1, 100001);

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

cin >> tree[i + l\_index];

}

for (int i = l\_index - 1; i >= 0; --i) {

tree[i] = min(tree[2 \* i + 1], tree[2 \* i + 2]);

}

int l = l\_index;

int r = l + k;

while (r <= l\_index + n) {

cout << minim(l, r, 0, l\_index, l\_index \* 2 + 1, &tree) << " ";

++l;

++r;

}

return 0;

}