Факультет программной инженерии и компьютерной техники

Алгоритмы и структуры данныхка

Yandex Contest

Структуры данных

Преподаватель: Косяков Михаил Сергеевич
Выполнил: Ле Чонг Дат
Группа: P3231

2021 г.

Часть I

I. Машинки

1 Решение

Разделим на 2 случая:

- Случай 1: Когда на полу лежит игрушка $x \geq k$ Рассмотрим следующую игрушку.
- Случай 2: Без игрушки x на полу. Здесь мы разделим на 2 небольших случая:
  - Случай 1: Количество игрушек $< k \rightarrow$ Нам просто нужно взять игрушку с полки, поставить на пол и прибавить результат к 1.
  - Случай 2: Количество игрушек $= k \rightarrow$ Мы возьмем игрушку с самым дальним следующим использованием k игрушек на полу и поместим ее на полку, в то же время поместим игрушку x на пол и сложим результат на 1.

Чтобы решить эту проблему, мы будем использовать набор пар (следующая позиция игрушки x, которую необходимо использовать, x), чтобы сохранить следующую самую дальнюю позицию, которую должна использовать игрушка x.

2 Реализация

```
#include <bits/stdc++.h>
using namespace std;
#define eb                  emplace_back
#define llong               long long
#define forn(i, a, b)       for(int i = a; i <= b; ++i)
#define repn(i, a, b)       for(int i = a; i <  b; ++i)

#ifndef LOCAL
#define cerr                if (0) cerr
#endif
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
auto solve() {
    int n, k, p; cin >> n >> k >> p;
    vector<int> a(p);
    vector< vector<int> > positions(n + 1, vector<int>());

    repn(i, 0, p) cin >> a[i];
    forn(i, 0, n) positions[i].eb(p);
    for(int i = p-1; i >= 0; --i) positions[a[i]].eb(i);
    set< pair<int, int> > floor;
    int ans = 0;
    repn(i, 0, p) {
```

```
        positions[a[i]].pop_back();
        if (floor.count({i, a[i]})) {
            floor.erase({i, a[i]});
            floor.emplace(positions[a[i]].back(), a[i]);
            continue;
        }
        ans++;

        if (floor.size() == k) floor.erase(--floor.end());
        floor.emplace(positions[a[i]].back(), a[i]);
    }
    return ans;
}
int main(void) {
#ifdef LOCAL
    freopen("test.inp", "r", stdin);
    freopen("test.out", "w", stdout);
#endif
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int ntest = 1;
    for(; ntest --;) {
        auto ans = solve();
        cout << ans << '\n';
    }
    return 0;
}
```

Часть II

J. Гоблины и очереди

1 Решение

Мы используем 2 двухсторонних очереди для хранения элементов в массиве. Пусть используются 2 двухсторонней очереди A и B, где A - двухсторонняя очередь, хранящая первую $\frac{1}{2}$ элемента последовательности, а B - двухсторонняя очередь, хранящая последнюю $\frac{1}{2}$ элемента последовательности.

- Каждый раз, когда мы добавляем элемент в конец строки, мы добавляем этот элемент в конец двухсторонней очереди B.
- Каждый раз, когда мы добавляем элемент в середину строки, мы добавляем этот элемент в начало двухсторонней очереди B.
- Каждый раз, когда мы вынимаем элемент из ряда, мы берем первый элемент двухсторонней очереди A.

После каждого удаления или добавления, если количество элементов двухсторонней очереди $A < \frac{1}{2}$ от текущего общего количества, мы возьмем первые

элементы двухсторонней очереди B и продвинемся в конец двухсторонней очереди A до тех пор, пока количество частей не будет элемента двухсторонней очереди $A = \frac{1}{2}$ текущего общего количества элементов.

2 Реализация

```
#include <bits/stdc++.h>
using namespace std;
#define eb                  emplace_back
#define llong               long long
#define forn(i, a, b)       for(int i = a; i <= b; ++i)
#define repn(i, a, b)       for(int i = a; i <  b; ++i)

#ifndef LOCAL
#define cerr                if (0) cerr
#endif
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
auto solve() {
    deque<int> prefix, suffix;
    int n; cin >> n;
    repn(i, 0, n) {
        char type; int num;
        cin >> type;
        switch (type) {
            case '+':
                cin >> num;
                suffix.eb(num);
                break;
            case '*':
                cin >> num;
                suffix.emplace_front(num);
                break;
            case '-':
                cout << prefix.front() << '\n';
                prefix.pop_front();
                break;
        }
        while (prefix.size() < suffix.size()) {
            prefix.eb(suffix.front());
            suffix.pop_front();
        }
    }
    return 0;
}
int main(void) {
#ifdef LOCAL
    freopen("test.inp", "r", stdin);
    freopen("test.out", "w", stdout);
#endif
```

```
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    int ntest = 1;
    for(; ntest --;) {
        auto ans = solve();
    }
    return 0;
}
```

Часть III

K. Менеджер памяти-1

1 Решение

Мы будем использовать набор A для хранения элементов, каждый из которых является пустым блоком в памяти, хранящейся как пара (длина блока, начальная позиция). Изначально A состоит из 1 уникального элемента $(n, 1)$.

- Когда нам нужно выделить память, мы найдем первый блок, длина которого больше или равна длине выделенной памяти. Если его нет, выведите -1.
- Если существует свободный блок, длина которого больше или равна требуемой длине, мы удаляем этот свободный блок из набора A. Вызов свободного блока (m, x), где m - длина, а x - начальная позиция. n - длина выделяемой ячейки памяти, если m > n, мы добавляем к набору A свободный блок длины m - n, а начальная точка равна x + n.
- Когда нам нужно освободить область памяти, мы добавляем этот свободный блок для установки A (в случае успешного выделения) и проверяем, что до и после этого свободного блока есть другие свободные блоки, если это так, мы объединяем их в 1 свободный блок.

2 Реализация

```
#include <bits/stdc++.h>
using namespace std;
#define eb                  emplace_back
#define llong               long long
#define forn(i, a, b)       for(llong i = a; i <= b; ++i)
#define repn(i, a, b)       for(llong i = a; i <  b; ++i)

#ifndef LOCAL
#define cerr                if (0) cerr
#endif
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
auto solve() {
    llong n, m; cin >> n >> m;
    set< pair<llong, llong> > mem;
```

```
    vector< pair<llong, llong> > ans(m + 1);
    map<llong, llong> nxt, prv;
    nxt[0] = prv[0] = -1;
    nxt[n+1] = prv[n+1] = -1;
    nxt[1] = n; prv[n] = 1;
    mem.emplace(n, 1);
    forn(i, 1, m) {
        llong t; cin >> t;
        if (t >= 0) {
            auto itr = mem.lower_bound({t, 0});
            if (itr == mem.end()) {
                ans[i] = {-1, -1};
                cout << -1 << '\n';
                continue;
            }
            auto block = *itr;
            if (block.first > t) {
                nxt[block.second+t] = nxt[block.second];
                prv[nxt[block.second]] = block.second+t;
            }
            nxt[block.second] = prv[block.second+t-1] = -1;
            cout << block.second << '\n';
            mem.erase(itr);
            if (block.first > t) mem.emplace(block.first - t, block.second + t);
            ans[i] = {t, block.second};
            continue;
        }
        t = -t;
        if (ans[t].first == -1) continue;
        auto block = ans[t];
        llong lo = block.second;
        llong hi = block.second + block.first - 1;
        if (prv[lo-1] != -1) {
            llong sz = lo - prv[lo-1];
            mem.erase({sz, prv[lo-1]});
            lo = prv[lo-1];
        }
        if  (nxt[hi+1] != -1) {
            llong sz = nxt[hi+1] - hi;
            mem.erase({sz, hi+1});
            hi = nxt[hi+1];
        }
        nxt[lo] = hi;
        prv[hi] = lo;
        mem.emplace(hi-lo+1, lo);
    }
    return 0;
}
int main(void) {
#ifdef LOCAL
```

```
    freopen("test.inp", "r", stdin);
    freopen("test.out", "w", stdout);
#endif
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    llong ntest = 1;
    for(; ntest --;) {
        auto ans = solve();
    }
    return 0;
}
```

Часть IV

L. Минимум на отрезке

1 Решение

Мы будем использовать двухстороннюю очередь для хранения индекса самых последних k элементов, отсортированных по возрастанию. Рассмотрим i-й элемент:

- Если i-й элемент все еще меньше, чем последний элемент в deque, вытолкнуть последний элемент в deque. Делайте это до тех пор, пока последний элемент deque не станет меньше или равен x_i
- Вставить i в дек.
- Если первый элемент не находится в диапазоне [i-k + 1, i], вытолкнуть первый элемент deque. Делайте это до тех пор, пока первый элемент двухсторонней очереди не окажется в диапазоне [i-k + 1, i].

→ Минимальное значение - текущий первый элемент двухсторонней очереди.

2 Реализация

```
#include <bits/stdc++.h>
using namespace std;
deque<int> dq;
vector<int> a;
int n, k;
void push(int i) {
    while (dq.size() and a[i] < a[dq.back()])
        dq.pop_back();
    dq.push_back(i);
}
void pop(int i) {
    while (dq.size() and i - dq.front() > k-1)
        dq.pop_front();
}
```

```
int main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    cin >> n >> k;
    for(int i = 0; i < n; ++i) {
        int x; cin >> x;
        a.emplace_back(x);
        push(i);
        pop(i);
        if (i >= k - 1) cout << a[dq.front()] << ' ';
    }

}
```

Часть V
Вывод

Я научился использовать базовые структуры данных, такие как deque, queue и set, в качестве основы для моих навыков программирования в будущем.