Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Алгоритмам и структурам данных»

Яндекс контест

Выполнил:

Студент группы P3233

Перевозчиков И. С.

Преподаватели:

Косяков М. С.

Тараканов Д. С.

Санкт-Петербург

2022

Задача I. Машинки

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <set>

using namespace std;

int main()
{
	int n, k, p;
	cin >> n >> k >> p;


	vector <pair <int, int>> cars(p);				// <car number, next position>
	map <int, int> old_cars;					// <car number, last position>
	map <int, int, greater<int>> cars_on_floor;		// <next position, car number>
	map <int, int> on_floor;					// <car number, next position>
	int f = p + 1;

	for (size_t i = 0; i < p; i++) {
		int x;
		cin >> x;
		if (old_cars.find(x) == old_cars.end()) {
			old_cars[x] = i;
			cars[i] = { x, f++ };
		}
		else {
			cars[old_cars[x]] = { x, i };
			old_cars[x] = i;
			cars[i] = { x, f++ };
		}
	}

	size_t operations = 0;

	for (size_t i = 0; i < p; i++) {
		if (on_floor.find(cars[i].first) == on_floor.end()) {
			operations++;
			if (on_floor.size() < k) {
				cars_on_floor[cars[i].second] = cars[i].first;
				on_floor[cars[i].first] = cars[i].second;
			}
			else {
				int car = static_cast <int> (cars_on_floor.begin()->second);
				int pos = static_cast <int> (cars_on_floor.begin()->first);
				on_floor.erase(car);
				cars_on_floor.erase(pos);
				on_floor[cars[i].first] = cars[i].second;
				cars_on_floor[cars[i].second] = cars[i].first;
			}
		}
		else {
			cars_on_floor.erase(on_floor[cars[i].first]);
			cars_on_floor[cars[i].second] = cars[i].first;
			on_floor[cars[i].first] = cars[i].second;
		}
	}

	cout << operations;

	return 0;
}
```

До тех пор, пока на полу есть свободное место ставим очередную машинку. Если свободного места нет, то сначала убираем из машинок на полу ту, которой Петя будет играть в последнюю очередь (или вовсе не будет больше играть). Если уберем другую, то ее придется доставать с полки раньше.

Алгоритмическая сложность: $O(n \cdot logn)$.

Задача J. Гоблины и очереди

```
#include <iostream>
#include <list>
#include <queue>

using namespace std;

template <typename T>
class Goblins_Line {
public:
	Goblins_Line() {
		first_half_size = 0;
		second_half_size = 0;
	}

	void middle(T element) {
		if (first_half_size == second_half_size) {
			first_half.emplace(element);
			first_half_size++;
		}
		else {
			second_half.push_front(element);
			second_half_size++;
		}
	}

	void back(T element) {
		if (first_half_size == 0 && second_half_size == 0) {
			first_half.emplace(element);
			first_half_size++;
		}
		else if (first_half_size == second_half_size) {
			T temp = second_half.front();
			second_half.pop_front();
			first_half.emplace(temp);
			second_half.push_back(element);
			first_half_size++;
		}
		else {
			second_half.push_back(element);
			second_half_size++;
		}
	}

	T front() {
		if (first_half_size == second_half_size) {
			T first_element = first_half.front();
			first_half.pop();
			first_half.emplace(second_half.front());
			second_half.pop_front();
			second_half_size--;
			return first_element;
		}
		else {
			T first_element = first_half.front();
			first_half.pop();
			first_half_size--;
			return first_element;
		}
	}

	int size() {
		return first_half_size + second_half_size;
	}
```

```
private:
	int first_half_size;
	int second_half_size;
	queue <T> first_half;
	list <T> second_half;
};


int main()
{
	int n;
	cin >> n;

	Goblins_Line <int> goblins_line;
	for (int i = 0; i < n; i++) {
		char sign;
		cin >> sign;
		if (sign == '-') {
			cout << goblins_line.front() << '\n';
			continue;
		}
		int num;
		cin >> num;
		if (sign == '+') {
			goblins_line.back(num);
		}
		else {
			goblins_line.middle(num);
		}
	}
}
```

Разобьем очередь на две половины. Первая будет quque, вторая – list. В зависимости от входных данных добавляем в середину или в конец очереди, или выводим номер гоблина. Выбор queue обусловлен тем, что в первой половине обращение идет к первому и последнему элементам, удаление первого элемента, добавление в конец. Все эти операции присутствуют в queue. Для второй половины добавляется еще добавление в начало, поэтому выбран list. Все операции выполняются за константное время.

Алгоритмическая сложность: $O(n)$.

Задача K. Менеджер памяти

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
#include <set>

using namespace std;

int malloc(int needed_size, multiset <int>& memory_pos, set <pair<int, int>,
greater<pair<int, int>>>& memory_size) {
	if (memory_pos.size() > 0) {
		int size = static_cast <int> (memory_size.begin()->first);
		int begin = static_cast <int> (memory_size.begin()->second);
		int end = begin + size - 1;

		if (size > needed_size) {
			memory_pos.erase(memory_pos.lower_bound(begin));
			memory_pos.insert(begin + needed_size);
			memory_size.erase({ size, begin });
			memory_size.insert({ size - needed_size, begin + needed_size });
			return begin;
		}
		else if (size == needed_size) {
			memory_pos.erase(memory_pos.lower_bound(begin));
			memory_pos.erase(memory_pos.lower_bound(end));
			memory_size.erase({ size, begin });
			return begin;
		}
	}

	return -1;
}

void free(int begin, int end, multiset <int>& memory_pos, set <pair<int, int>,
greater<pair<int, int>>>& memory_size) {
	auto prev = memory_pos.find(begin - 1);
	auto next = memory_pos.find(end + 1);

	if (prev != memory_pos.end()) {
		if (next != memory_pos.end()) {
			prev = memory_pos.upper_bound(begin - 1);
			prev--;
			int p = static_cast <int> (*prev);
			int n = static_cast <int> (*next);
			prev--;
			next++;
			int p1 = static_cast <int> (*prev);
			int n1 = static_cast <int> (*next);
			memory_size.erase({ p - p1 + 1, p1 });
			memory_size.erase({ n1 - n + 1, n });
			memory_size.insert({ n1 - p1 + 1, p1 });
			memory_pos.erase(memory_pos.lower_bound(p));
			memory_pos.erase(memory_pos.lower_bound(n));
		}
		else {
			prev = memory_pos.upper_bound(begin - 1);
			prev--;
			int p = static_cast <int> (*prev);
			prev--;
			int p1 = static_cast <int> (*prev);
			memory_size.erase({ p - p1 + 1, p1 });
			memory_size.insert({ end - p1 + 1, p1 });
			memory_pos.erase(memory_pos.lower_bound(p));
```

```
                memory_pos.insert(end);
            }
        }
        else if (next != memory_pos.end()) {
            int n = static_cast <int> (*next);
            next++;
            int n1 = static_cast <int> (*next);
            memory_size.erase({ n1 - n + 1, n });
            memory_size.insert({ n1 - begin + 1, begin });
            memory_pos.erase(memory_pos.lower_bound(n));
            memory_pos.insert(begin);
        }
        else {
            memory_size.insert({ end - begin + 1, begin });
            memory_pos.insert(begin);
            memory_pos.insert(end);
        }
}

int main()
{
        int n;
        int m;
        cin >> n >> m;
        multiset <int> memory_pos;
        set <pair<int, int>, greater<pair<int, int>>> memory_size;
        memory_pos.insert(1);
        memory_pos.insert(n);
        memory_size.insert({ n , 1 });
        vector <pair <int, int>> requests(m);

        for (int i = 0; i < m; i++) {
            int x;
            cin >> x;
            if (x < 0) {
                if (requests[-x - 1].first != -1) {
                    free(requests[-x - 1].first, requests[-x - 1].second,
memory_pos, memory_size);
                }
            }
            else {
                int pos = malloc(x, memory_pos, memory_size);
                requests[i] = { pos, pos + x - 1 };
                cout << pos << '\n';
            }
        }

        return 0;
}
```

Память можно представить в виде отрезков с длиной, равной свободной памяти. Изначально это 1 отрезок длины n. При выделении памяти всегда выделяем из самого большого блока. При это этот блок будет либо уменьшаться, либо вовсе исчезать из памяти (если его длина равна выделяемой памяти). При освобождении памяти может быть ситуация, что два блока граничат друг с другом. В этом случае их нужно объединить в один.

Алгоритмическая сложность: $O(n \cdot \log n)$.

Задача L. Минимум на отрезке

```
#include <iostream>
#include <map>
#include <string>
#include <vector>
#include <algorithm>
#include <list>

using namespace std;

int main()
{
	int n, k;
	cin >> n >> k;
	map <int, int> counter;
	vector <int> elements(n);

	for (int i = 0; i < k; i++) {
		int x;
		cin >> x;
		elements[i] = x;
		if (counter.find(x) == counter.end()) {
			counter[x] = 0;
		}
		counter[x]++;
	}

	vector <int> answer;

	answer.push_back(counter.begin()->first);

	for (int i = k; i < n; i++) {
		int x;
		cin >> x;
		elements[i] = x;
		if (counter.find(x) == counter.end()) {
			counter[x] = 0;
		}
		counter[x]++;
		counter[elements[i - k]]--;
		if (counter[elements[i - k]] == 0)
			counter.erase(elements[i - k]);
		answer.push_back(counter.begin()->first);
	}

	for (auto a : answer)
		cout << a << ' ';

}
```

Для начала нужно посчитать минимум на начальной отрезке длины K и сохранить элементы этого отрезка. Удобно сделать это подсчетом количества каждого элемента. Далее а каждом шаге удалять элемент и вставлять новый и исходя из этого находить новый минимум.

Алгоритмическая сложность: на каждом шаге добавляем в map новый элемент. Так как элементов в map не больше, чем K, то в итоге сложность $O(n \cdot \log k)$.