Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по «Алгоритмам и структурам данных»

Базовые задачи / Timus

Выполнил:

Студент группы P3234

Баянов Р.Д.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2024

Яндекс контест

Задача I. Машинки

Ход решения:

Для решения данной задачи я использовал LRU алгоритм. Его суть заключается в том, чтобы обновлять каждый раз время использования какого-то элемента и убирать самый долго неиспользуемый. Представим, что пол – это кэш, в который мы будем складывать машинки. Перед тем как подсчитывать наименьшее кол-во операция для мамы Пети, составим приоритет для каждой машинки с помощью вектора priority. Там будут храниться данные о времени использования каждой машинки. Затем, мы будем идти по каждому Петиному запросу и складывать машины в кэш и убирать те машины, которые слабее по приоритету. Всё это время подсчитываем кол-во действий и выводим. Вот и ответ!

Оценка сложности решения:

Время: O(P * log P)

Память: O(N + P + K)

Код решения:

```
#include <iostream>
#include <vector>
#include <unordered_set>
#include <queue>
#include <list>
#include <limits.h>

struct priority_car {
	int priority;
	int car;
	bool operator<(const priority_car& other) const {
		if (priority != other.priority) {
			return priority < other.priority;
		}
		else {
			return car > other.car;
		}
	}
```

```
};

int main()
{
	int N, K, P;
	std::cin >> N >> K >> P;
	std::unordered_set<int> cashe;
	std::priority_queue<struct priority_car> cars;
	std::vector<std::list<int>> priority(N + 1);
	std::vector<int> sequence(P);
	int count = 0;
	for (int i = 0; i < P; i++) {
		std::cin >> sequence[i];
		priority[sequence[i]].push_back(i);
	}
	for (int i = 0; i < P; i++) {
		int current_car = sequence[i];
		priority[current_car].pop_front();
		if (cashe.find(current_car) == cashe.end()) {
			if (cashe.size() >= K) {
				cashe.erase(cars.top().car);
				cars.pop();
			}
			count++;
			cashe.insert(current_car);

		}
		struct priority_car a;
		if (priority[current_car].empty()) {
			a.car = current_car;
			a.priority = INT_MAX;
			cars.push(a);
		}
		else {
			a.car = current_car;
			a.priority = priority[current_car].front();
			cars.push(a);
		}
	}
	std::cout << count;
	return 0;
}
```

Задача J. Гоблины и очереди

Ход решения:

Для решения данной задачи я понял, как можно всегда помнить, где у меня находится середина очереди гоблинов и не тратить на это много времени. Разобьём очередь на две части. И логично предположить, что при “+” мы добавляем гоблина в конец второй очереди, при “-“ мы убираем с начала первой очереди гоблина и при “*” мы добавляем гоблина в конец первой очереди. При добавлении каждого гоблина балансируем центр, перекидывая гоблинов между очередями. И в конце выводим гоблинов, вышедших из очереди. Готово!

Оценка сложности решения:

Время: O(N)

Память: O(N + N) = O(N)

Код решения:

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

int main()
{
	int N;
	std::cin >> N;
	std::list<int> queue1;
	std::list<int> queue2;
	std::vector<int> result;
	for (int i = 0; i < N; i++) {
		char symbol;
		int number;
		std::cin >> symbol;
		if (symbol == '-') {
			result.push_back(queue1.front());
			queue1.pop_front();
		}
		else if (symbol == '+') {
			std::cin >> number;
			queue2.push_back(number);
		}
		else {
```

```
                std::cin >> number;
                queue1.push_back(number);
            }
            if (queue1.size() > queue2.size() + 1) {
                queue2.push_front(queue1.back());
                queue1.pop_back();
            }
            else if (queue1.size() < queue2.size()){
                queue1.push_back(queue2.front());
                queue2.pop_front();
            }
        }
        for (int i = 0; i < result.size(); i++) {
            std::cout << result[i] << std::endl;
        }
        return 0;
}
```

Задача K. Менеджер памяти-1

Ход решения:

Для решения данной задачи заведём мапы для хранений свободных блоков. Одна из низ будет по размеру, другая - наоборот по индексу. После этого проходя по запросам будем освобождать и выделять память. Обязательно с помощью мапы под названием requests мы будем обращаться к нашим старым запросам на выделение памяти при освобождении. Ну и дальше будем стабильно при выделении поддерживать свободные блоки в мапах. А при освобождении блоков будем стараться их максимально соединять, дабы каждый новый блок с большей вероятностью уместился в один из них. Таким образом в вектор result складываем начало каждого блока и выводим ответ. Сделано!

Оценка сложности решения:

Время: O(M * log N)

Память: O(M + N)

Код решения:

```
#include <iostream>
#include <map>
#include <unordered_map>
#include <vector>


int main() {
	int N, M, index_x,  size_x;
	std::cin >> N >> M;
	std::unordered_map<int, std::pair<int, int>> requests;
	std::multimap<int, int> free_blocks_by_size;
	std::map<int, int> free_blocks;
	free_blocks_by_size.insert({N, 1});
	free_blocks.insert({ 1, N });
	int req;
	std::vector<int> results;
	auto it_d = free_blocks.begin();
	for (int i = 1; i <= M; i++) {
		std::cin >> req;
		if (req <= 0) {
			requests.insert({ i, {req, 0} });
			std::pair<int, int> block = requests.at(abs(req));
			int index = block.second;
			int size = block.first;
			if (index == -1) {
				continue;
			}
			auto it_right = free_blocks.lower_bound(index);
			auto it_left = (it_right != free_blocks.begin()) ? std::prev(it_right) : free_blocks.end();

			if (it_right != free_blocks.end() && it_right->first == index + size) {
				if (it_left != free_blocks.end() && it_left->first + it_left->second == index) {
					index_x = it_left->first;
					size_x = it_left->second + it_right->second;
					it_d = free_blocks_by_size.find(it_left->second);
					while (it_d->second != it_left->first) it_d++;
					free_blocks_by_size.erase(it_d);
					free_blocks.erase(it_left);
					it_d = free_blocks_by_size.find(it_right->second);
					while (it_d->second != it_right->first) it_d++;
					free_blocks_by_size.erase(it_d);
					free_blocks.erase(it_right);
					free_blocks.insert({ index_x, size + size_x });
					free_blocks_by_size.insert({ size + size_x, index_x });
				}
				else {
					size_x = it_right->second;
					it_d = free_blocks_by_size.find(it_right->second);
					while (it_d->second != it_right->first) it_d++;
					free_blocks_by_size.erase(it_d);
					free_blocks.erase(it_right);
					free_blocks.insert({ index, size + size_x });
					free_blocks_by_size.insert({ size + size_x, index });
				}
			}
			else {
				if (it_left != free_blocks.end() && it_left->first + it_left->second == index) {
					index_x = it_left->first;
					size_x = it_left->second;
					it_d = free_blocks_by_size.find(it_left->second);
					while (it_d->second != it_left->first) it_d++;
					free_blocks_by_size.erase(it_d);
					free_blocks.erase(it_left);
```

```
                                        free_blocks.insert({index_x, size + size_x});
                                        free_blocks_by_size.insert({size + size_x, index_x});
                                }
                                else {
                                        free_blocks.insert({index, size});
                                        free_blocks_by_size.insert({size, index});
                                }
                        }
                }
                else {

                        auto it = free_blocks_by_size.lower_bound(req);
                        if (it == free_blocks_by_size.end()) {
                                requests.insert({i, {req, -1}});
                                results.push_back(-1);
                        }
                        else {
                                int new_block_size = it->first - req;
                                int index = it->second;
                                requests.insert({ i, { req, index } });
                                results.push_back(index);
                                free_blocks.erase(it->second);
                                free_blocks_by_size.erase(it);
                                if (new_block_size > 0) {
                                        free_blocks.insert({index + req, new_block_size});
                                        free_blocks_by_size.insert({new_block_size, index + req});
                                }
                        }
                }
        }
        for (int val : results) {
                std::cout << val << std::endl;
        }
        return 0;
}
```

Задача L. Минимум на отрезке

Ход решения:

Данную задачу нужно решать с помощью дерева отрезков. Но всё же я не успевал его реализовать и получилось так, что у меня зашло решение в лоб. Расскажу о нём. Здесь мы просто проходимся по всему отрезку и двигаем окошко и в нём ищем минимум. На каждой итерации выводим этот самый минимум.

Оценка сложности решения:

Время: O(N * K)

Память: O(N)

Код решения:

```
#include <iostream>
#include <vector>

int main()
{
	int N, K;
	std::cin >> N >> K;
	std::vector<int> numbers;
	for (int i = 0; i < N; i++) {
		int x;
		std::cin >> x;
		numbers.push_back(x);
	}
	int min;
	for (int i = 0; i < N - K + 1; i++) {
		min = 100000;
		int k = 0;
		int j = i;
		while (k < K) {
			if (numbers[j+k] <= min) {
				min = numbers[j+k];
			}
			k++;
		}
		std::cout << min << " ";
	}
}
```

Тимус

Задача 1628. Белые полосы

Ход решения:

Для решения данной задачи создадим две мапы (матрицу), которые будут зеркальны друг другу и к ним же добавим множество одиноких точек. Пройдёмся по всем неудачным дням Вась-Вася и заполним наши структуры. Затем, пройдёмся по всем строкам и подсчитаем количество белых полос Вась-Вася и заодно подсчитаем одинокие клетки на каждой строке и сложим их координаты в множество одиноких точек. После этого пойдём по матрице

со стороны столбцов и так же будем считать белые полосы. И вместе с этим будем пытаться одинокие точки класть в множество одиноких клеток. Если же при обходе строчек и при обходе столбца мы встречаем одну и ту же точку, эта точка становится изолированной и её тоже нужно посчитать как белую полосу Вась-Вася, потому что только тогда она будет максимальной по включению полосой для себя. И всё!

Оценка сложности решения:

Время: O(K + 2 * M * N) – в худшем случае

Память: O(M + N)

Код решения:

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

int main()
{
	int m, n, k, count = 0;
	std::cin >> m >> n >> k;
	int x, y;
	std::set<std::pair<int, int>> single_elements;
	std::unordered_map<int, std::set<int>> rows;
	std::unordered_map<int, std::set<int>> cols;
	for (int i = 0; i < k; i++) {
		std::cin >> x >> y;
		rows[x].insert(y);
		cols[y].insert(x);
	}
	for (int i = 1; i <= m; i++) {
		int prev = 0;
		for (int cur : rows[i]) {
			if (cur - prev > 2) {
				count++;
			}
			else if (cur - prev == 2){
				single_elements.insert({ i, cur - 1});
			}
			prev = cur;
		}
		if (n - prev > 1) {
			count++;
```

```
                }
                else if (n - prev == 1){
                        single_elements.insert({ i, n } );
                }
        }
        for (int i = 1; i <= n; i++ ) {
                int prev = 0;
                for (int cur : cols[i]) {
                        if (cur - prev > 2) {
                                count++;
                        }
                        else if (cur - prev == 2){
                                if (single_elements.find({cur - 1, i}) != single_elements.end()) {
                                        count++;
                                }
                        }
                        prev = cur;
                }
                if (m - prev > 1) {
                        count++;
                }
                else if (m - prev == 1){
                        if (single_elements.find({m, i}) != single_elements.end()) {
                                count++;
                        }
                }
        }
        std::cout << count;
        return 0;
}
```

Задача 1650. Миллиардеры

Ход решения:

Для решения данной задачи мы используем несколько мап для связи данных между собой. Эта задача напомнила мне СУБД, и именно поэтому каждую мапу я буду называть сущностью. Заведём сущности “город-дни”, “город-деньги”, “деньги-города”, “люди-деньги-город, в котором человек находится”. Сущности “город-дни”, “деньги-город” должны быть мап, чтобы

они автоматически отсортировывались по ключам, это нужно будет для вывода ответа и для удобства нахождения города с максимальной суммой денег. Вот и всё. Теперь пойдём по всем запросам и аккуратно будем перекидывать деньги каждого миллиардера из города в город. Параллельно не будем забывать при каждой итерации прибавлять к городу с максимальной суммой количество дней, которое он лидирует. В целом всё, боле подробно не вижу смысла объяснять.

Оценка сложности решения:

Время: O(N^2) – в худшем случае, O(N log N)

Память: O(N)

Код решения:

```
#include <iostream>
#include <map>
#include <string>
#include <unordered_map>
#include <set>


int main()
{
	std::cin.tie(0);
	std::ios_base::sync_with_stdio(0);
	int n;
	std::cin >> n;
	std::unordered_map<std::string, std::pair<std::string, long long>> people;
	std::map<std::string, int> cities_days;
	std::map<long long, std::set<std::string>> rich_city;
	std::unordered_map<std::string, long long> cities_sums;
	std::string name, city;
	long long money;
	for (int i = 0; i < n; i++) {
		std::cin >> name >> city >> money;
		people[name] = { city, money };
		if (cities_sums.find(city) != cities_sums.end()) {
			long long sum = cities_sums[city];
			rich_city[sum].erase(city);
			if (rich_city[sum].size() == 0) {
				rich_city.erase(sum);
			}
		}
		cities_sums[city] += money;
```

```
                rich_city[cities_sums[city]].insert(city);
        }
        int m, k, day;
        std::cin >> m >> k;
        int cur_day, prev_day = 0;
        for (int i = 0; i <= k; i++) {
                if (i == k) {
                        day = m;
                }
                else {
                        std::cin >> day >> name >> city;
                }
                cur_day = day;
                std::map<long long, std::set<std::string>>::reverse_iterator it = rich_city.r-
begin();
                if (cur_day != prev_day && it->second.size() == 1) cities_days[*(it->sec-
ond.begin())] += cur_day - prev_day;
                if (i < k) {
                        std::string from_city = people[name].first;
                        long long old_money = cities_sums[from_city];
                        rich_city[old_money].erase(from_city);
                        if (rich_city[old_money].size() == 0) {
                                rich_city.erase(old_money);
                        }
                        money = people[name].second;
                        cities_sums[from_city] -= money;
                        rich_city[cities_sums[from_city]].insert(from_city);

                        long long new_money = cities_sums[city];
                        rich_city[new_money].erase(city);
                        if (rich_city[new_money].size() == 0) {
                                rich_city.erase(new_money);
                        }
                        cities_sums[city] += money;
                        rich_city[cities_sums[city]].insert(city);
                        std::move(people[name].first) = city;
                        prev_day = cur_day;
                }
        }
        for (auto it = cities_days.begin(); it != cities_days.end(); it++) {
                std::cout << it->first << " " << it->second << std::endl;
        }
        return 0;
}
```