Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3213

Поленов К.А.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2025

Задача №I «Машинки»

```cpp
#include <climits>
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <vector>

using namespace std;

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

  vector<int> cars_to_play_order;
  vector<queue<int>> next_cars_usage(n + 1);

  int car;
  for (int i = 0; i < p; i++) {
    cin >> car;
    cars_to_play_order.push_back(car);
    next_cars_usage[car].push(i);
  }

  set<pair<int, int>> cars_on_floor_next_usages;
  map<int, int> cars_on_floor;
  int operations = 0;

  for (int i = 0; i < p; i++) {
    int curr_car = cars_to_play_order[i];
    next_cars_usage[curr_car].pop();

    int next_usage;
    if (next_cars_usage[curr_car].empty())
      next_usage = INT_MAX;
    else
      next_usage = next_cars_usage[curr_car].front();

    if (cars_on_floor.count(curr_car)) {
      cars_on_floor_next_usages.erase({cars_on_floor[curr_car], curr_car});
      cars_on_floor_next_usages.insert({next_usage, curr_car});

      cars_on_floor[curr_car] = next_usage;
      continue;
    }

    operations++;

    if ((int)cars_on_floor_next_usages.size() == k) {
      auto pair_to_remove = prev(cars_on_floor_next_usages.end());
      int car_to_remove = pair_to_remove->second;
      cars_on_floor.erase(car_to_remove);

      cars_on_floor_next_usages.erase(pair_to_remove);
    }

    cars_on_floor_next_usages.insert({next_usage, curr_car});
    cars_on_floor[curr_car] = next_usage;
  }

  cout << operations << endl;
  return 0;
}

```

Пояснение к примененному алгоритму:

Сначала составляется вектор из порядка игры с машинками и параллельно с ним вектор очередей, в которых находятся индексы, которые представляют собой последовательность игры для каждой машинки. Далее задаем множество пар

<следующее использование машинки, ее номер>, чтобы машинки были отсортированы по возрастанию использования, и словарь, который позволит сохранять предыдущее использование и получать использование по ключу <номер машинки>. Затем в цикле достаем следующее использование машинки. Если очередь использований пуста, то машинка не используется и next_usage приравниваем к INT_MAX, чтобы не итерироваться по ней, но при этом иметь возможность увеличить операции, если следующего использования у машинки нет. Далее, если машинка уже на полу, меняем ее следующее использование в словаре и множестве и переходим на следующую итерацию. Иначе увеличиваем операции и проверяем количество машинок на полу на равенство k. Если количество машинок на полу уже равно максимальному, то удаляем крайний элемент множества, тк там самая не нужная в будущем машинка. Затем добавляем на пол текущую машинку.

Задача №J «Гоблины и очереди»

```cpp
#include <deque>
#include <iostream>

using namespace std;

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

  deque<int> left_half, right_half;
  // left - начало, right - конец очереди
  // left >= right всегда!!!

  string command;
  int goblin;

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

    if (command[0] == '+') {
      cin >> goblin;
      // добавляем в конец, то есть в конец right
      right_half.push_back(goblin);
    } else if (command[0] == '*') {
      cin >> goblin;
      // добавляем в середину, то есть в конец left
      left_half.push_back(goblin);

    } else {
      cout << left_half.front() << endl;
      // убираем из начала, то есть из начала left
      left_half.pop_front();
    }

    // чтобы середина очереди, всегда оставалась по центру

    // если в left пусто, а в right один гоблин, его нужно сдвинуть в начало или
    // в очереди нечетное число гоблачей, то, чтобы вставить привелигированного после цент
    // нужно сдвинуть очередь влево
    while (left_half.size() < right_half.size()) {
      left_half.push_back(right_half.front());
      right_half.pop_front();
    }

    // когда к примеру два раза подряд добавили в середину, ее нужно сдвинуть вправо
    while (left_half.size() > right_half.size() + 1) {
      right_half.push_front(left_half.back());
      left_half.pop_back();
    }
  }

  return 0;
}

```

Пояснение к примененному алгоритму:

Задача с одной deque не проходит по времени, потому решено было использовать две очереди. Парсим команды. Если + -- добавляем гоблина в конец очереди, то есть в конец правой половины, если * -- добавляем гоблина в середину очереди, то есть в конец левой половины, если - -- убираем гоблина из начала очереди, то есть из начала левой половины. После выполнения команды нужно сдвинуть очередь так, чтобы середина была на конце левой половины, и чтобы на следующей итерации

можно было достать гоблина из начала левой половины или вставить его в середину в случае четного количества гоблинов или после центрального элемента в случае нечетного.

Задача №L «Минимум на отрезке»

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

using namespace std;

int main() {
  int n, k, num;
  vector<int> nums;

  cin >> n >> k;

  while (cin >> num) {
    nums.push_back(num);
  }

  map<int, int> frame;  // key=num; value=isInFrame

  for (int i = 0; i < k; i++) {
    frame[nums[i]]++;
  }

  int left = 0;
  int right = k - 1;

  while (right < n) {
    cout << frame.begin()->first << " ";

    frame[nums[left]]--;
    if (!frame[nums[left]])
      frame.erase(nums[left]);

    left++;
    right++;

    if (right == n)
      break;
    frame[nums[right]]++;
  }

  return 0;
}
```

Пояснение к примененному алгоритму:

Считываем все числа. Затем есть словарь, где ключ - это число, а значение это 1 или 0, представляющее собой наличие в «окне». В цикле задаем окну начальное состояние. Затем вводим левый и правый индексы концов окна. В цикле пока правый индекс меньше количества чисел выводим минимум (тк в словарь отсортирован по ключам, минимум всегда в начале), задаем крайнему числу слева 0 и убираем его из окна. Смещаем индексы на 1 вправо, проверяем не стал ли крайний правый индекс окна равен n, если да, то выходим из цикла, если нет, то добавляем в окно число из массива, соответствующее текущему крайнему правому индексу окна