Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3233

Фамилия И.О.

Шикунов Максим Евгеньевич

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2024

Компилятор везде: Clang 17.0.1 C++ 20

Задача №I «Машинки»

Код:

```
#include <iostream>
#include <unordered_map>
#include <unordered_set>
#include <deque>
#include <queue>
using namespace std;


int main() {
    size_t n, k, p, count = 0, sumi = 0;
    cin >> n >> k >> p;
    unordered_map<size_t, deque<size_t>> indexMap;
    unordered_set<size_t> carOnFloor;
    priority_queue<pair<size_t, size_t>> maxIndex;
    size_t carsOperations[p];
    pair<size_t, size_t> deletePair;
    for (int i = 0; i < p; i++) {
        cin >> carsOperations[i];
        indexMap[carsOperations[i]].push_front(i);
    }
    for (int i = 0; i < p; i++) {
        if (sumi < k && carOnFloor.find(carsOperations[i]) ==
carOnFloor.end()) {
            carOnFloor.insert(carsOperations[i]);
            sumi++;
            count++;
        } else if (carOnFloor.find(carsOperations[i]) != carOnFloor.end()) {
            indexMap[carsOperations[i]].pop_back();
            if (indexMap[carsOperations[i]].empty()) {
                maxIndex.emplace(1000000, carsOperations[i]);
            } else {
                maxIndex.emplace(indexMap[carsOperations[i]].back(),
carsOperations[i]);
            }
            continue;
        } else if (sumi == k) {
            if (sumi == 1 || i + 1 == p) {
                count++;
            } else {
                deletePair = maxIndex.top();
                maxIndex.pop();
                carOnFloor.erase(deletePair.second);
                carOnFloor.insert(carsOperations[i]);
                count++;
            }
        }
        indexMap[carsOperations[i]].pop_back();
        if (indexMap[carsOperations[i]].empty()) {
            maxIndex.emplace(1000000, carsOperations[i]);
        } else {
            maxIndex.emplace(indexMap[carsOperations[i]].back(),
carsOperations[i]);
        }
    }
    cout << count;
    return 0;
}
```

Пояснение к примененному алгоритму:

В этой задаче просто изначально выкладываются машинки до отказа, пока это позволяет вместимость “пола”. Дальше, когда это необходимо, мы убираем машинку с пола, сначала те, которые больше не понадобятся нам, если таких нет, то мы смотрим машинку, которая в очереди стоит на самом большом расстоянии.

Сложность: $O(n \log n)$

Задача №J «Гоблины и очереди»

Код:

```cpp
#include <iostream>
#include <list>
using namespace std;

int main() {
    long n;
    char input;
    int number, count = 0;
    list<int> goblinList;
    auto mid = goblinList.begin();
    cin >> n;
    for (long i = 0; i < n; i++) {
        cin >> input;
        if (count == 1) {
            mid = goblinList.begin();
        }
        if (input != '-') {
            cin >> number;
            if (input == '+') {
                goblinList.push_front(number);
                mid--;
                count++;
            } else {
                goblinList.insert(mid, number);
                mid--;
                count++;
            }
            if (count % 2 == 0) {
                mid++;
            }
        } else {
            cout << goblinList.back() << endl;
            goblinList.pop_back();
            count--;
            if (count % 2  != 0) {
                mid--;
            }
        }
    }
    return 0;
}
```

Пояснение к примененному алгоритму:

В данной задаче мы вносим поочередно гоблинов, которые приходят, а если гоблин элитный, то мы вносим по итератору, который указывает на середину листа.

Сложность: $O(n^2)$

Задача №K «Менеджер памяти-1»

Код:

```
#include <iostream>
#include <set>
using namespace std;

struct node {
    size_t startPosition;
    size_t size;
    bool free;
    struct node* next;
    struct node* previous;
};

int main() {
    long long operation, index;
    size_t n, m, size;
    multiset<pair<size_t, struct node*>, greater<>> freeNodes;
    cin >> n >> m;
    struct node* operations[m];
    struct node firstNode = {.startPosition = 1, .size = n, .free = true,
.next = nullptr, .previous = nullptr};
    struct node* firstFreeNode = &firstNode;
    freeNodes.insert(pair(firstFreeNode->size, firstFreeNode));
    for (size_t i = 0; i < m; i++) {
        cin >> operation;
        if (operation > 0) {
            if (freeNodes.begin()->first >= operation) {
                if (freeNodes.begin()->first == operation) {
                    freeNodes.begin()->second->free = false;
                    operations[i] = freeNodes.begin()->second;
                    cout << freeNodes.begin()->second->startPosition << endl;
                    freeNodes.erase(freeNodes.begin());
                } else {
                    node* newNode = new node();
                    newNode->startPosition = freeNodes.begin()->second-
>startPosition;
                    newNode->size = operation;
                    newNode->free = false;
                    newNode->next = freeNodes.begin()->second;
                    newNode->previous = freeNodes.begin()->second->previous;
                    if (freeNodes.begin()->second->previous != nullptr) {
                        freeNodes.begin()->second->previous->next = newNode;
                    }
                    freeNodes.begin()->second->startPosition += operation;
                    freeNodes.begin()->second->previous = newNode;
```

```
                    freeNodes.begin()->second->size -= operation;
                    freeNodes.insert(pair(freeNodes.begin()->first -
operation, freeNodes.begin()->second));
                    freeNodes.erase(freeNodes.begin());
                    operations[i] = newNode;
                    cout << newNode->startPosition << endl;
                }
            } else {
                operations[i] = nullptr;
                cout << -1 << endl;
            }
        } else {
            index = abs(operation) - 1;
            if (operations[index] != nullptr) {
                operations[index]->free = true;
                size = operations[index]->size;
                if (operations[index]->previous != nullptr ||
operations[index]->next != nullptr) {
                    if (operations[index]->previous != nullptr) {
                        if (operations[index]->previous->free) {
                            freeNodes.erase(pair(operations[index]->previous-
>size, operations[index]->previous));
                            size += operations[index]->previous->size;
                            operations[index]->size += operations[index]-
>previous->size;
                            operations[index]->startPosition =
operations[index]->previous->startPosition;
                            if (operations[index]->previous->previous !=
nullptr) {
                                operations[index]->previous->previous->next =
operations[index];
                            }
                            operations[index]->previous = operations[index]-
>previous->previous;
                        }
                    }
                    if (operations[index]->next != nullptr) {
                        if (operations[index]->next->free) {
                            freeNodes.erase(pair(operations[index]->next-
>size, operations[index]->next));
                            size += operations[index]->next->size;
                            operations[index]->size += operations[index]-
>next->size;
                            if (operations[index]->next->next != nullptr) {
                                operations[index]->next->next->previous =
operations[index];
                            }
                            operations[index]->next = operations[index]-
>next->next;
                        }
                    }
                }
                freeNodes.insert(pair(size, operations[index]));
            }
        }
    }
    return 0;
}
```

Пояснения к примененному алгоритму:

Создал структуру node, чтоб легче обрабатывать мои узлы памяти. Пусты узлы хранятся в сете, под видом пары (размер пустого места, ссылка на него). Тем самым мы за O(1) можем узнать, есть ли у нас такой узел для хранения поданных данных. При освобождении рассматриваются все случаи, при необходимости узлы совмещаются.

Сложность: $O(n \log n)$

Задача №L «Минимум на отрезке»

Код:

```
#include <iostream>
#include <queue>
#include <map>
using namespace std;

int main() {
    map<long, long> minMap;
    long n;
    int k;
    cin >> n >> k;
    long numbers[n];
    for (long i = 0; i < n; i++) {
        cin >> numbers[i];
    }
    for (long i = 0; i < n; i++) {
        if (i > k - 1) {
            cout << minMap.begin()->first << " ";
            if (minMap[numbers[i - k]] - 1 == 0) {
                minMap.erase(numbers[i - k]);
            } else {
                minMap[numbers[i - k]]--;
            }
        }
        minMap[numbers[i]]++;
    }
    cout << minMap.begin()->first;
    return 0;
}
```

Пояснение к примененному алгоритму:

Просто вносим в map числа, которые находятся в окне, на каждой итерации выводим максимальное число, когда число выходит из окна удаляет его из map’ы

Сложность: $O(n \log n)$

Задача №1067 «Структура папок»

Код:

```
#include <iostream>
#include <map>
#include <sstream>

using namespace std;


struct directory {
    map<string, struct directory*> childDirectory;
};

void print(struct directory* dir, string spaces) {
    string nextSpace = spaces + " ";
    if (!dir->childDirectory.empty()) {
        for (auto it : dir->childDirectory) {
            cout << spaces << it.first << endl;
            print(it.second, nextSpace);
        }
    }
}

int main() {
    struct directory root = {};
    struct directory* current;
    string path, folderName;
    size_t n;
    cin >> n;
    for (size_t i = 0; i < n; i++) {
        current = &root;
        cin >> path;
        stringstream ss(path);
        while (getline(ss, folderName, '\\')) {
            if (current->childDirectory.find(folderName) != current-
>childDirectory.end()) {
                current = current->childDirectory[folderName];
            } else {
                current->childDirectory[folderName] = new struct directory;
                current = current->childDirectory[folderName];
            }
        }
    }
    print(&root, "");
    return 0;
}
```

Пояснение к примененному алгоритму:

Используем map, чтобы наши имена папок сортировались правильно, а также чтобы по ключу хранить ссылку на подзаголовки данной папки. Каждый итерацию мы принимаем путь, читаем название каждой папки, если нет такого названия, то создаем, иначе передаем существующую.В конце рекурсивно выводим наше дерево папок.

Сложность: $O(nlogn)$

Задача №1494 «Монобильярд»

Код:

```cpp
#include <iostream>
#include <stack>
using namespace std;

int main() {
    stack<size_t> scoredPools;
    size_t topPool = 0;
    size_t n;
    cin >> n;
    size_t pools[n];
    for (size_t i = 0; i < n; i++) {
        cin >> pools[i];
    }
    for (size_t i = 0; i < n; i++) {
        if (pools[i] > topPool) {
            size_t pool = topPool + 1;
            while (pool < pools[i]) {
                scoredPools.push(pool);
                pool++;
            }
            topPool = pools[i];
        } else if (pools[i] == scoredPools.top()) {
            scoredPools.pop();
        } else {
            cout << "Cheater";
            return 0;
        }
    }
    cout << "Not a proof";
    return 0;
}
```

Пояснение к примененному алгоритму:

В данной задаче я проверяю, какой шар вытащили, если этот шар был больше чем “наибольший закатившийся”, то значит между ними должны были закатываться и шары от последнего до этого в порядке возрастания, для этого мы заносим в таком порядке в стек, чтобы они там хранились. Если приходит на проверку шар, который меньше чем наибольший, то он должен быть на вершине стека, иначе он читер.

Сложность: $O(n^2)$