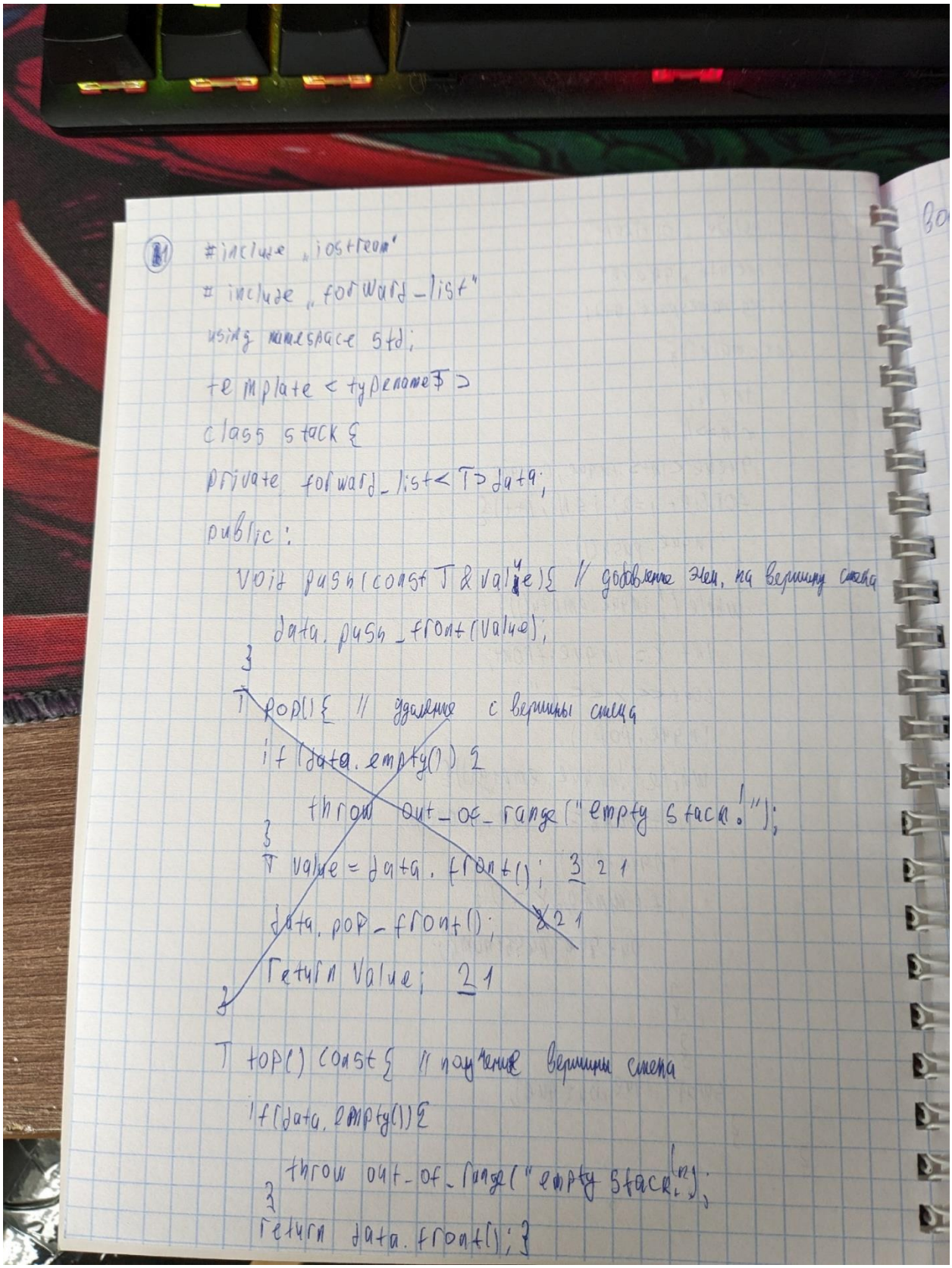
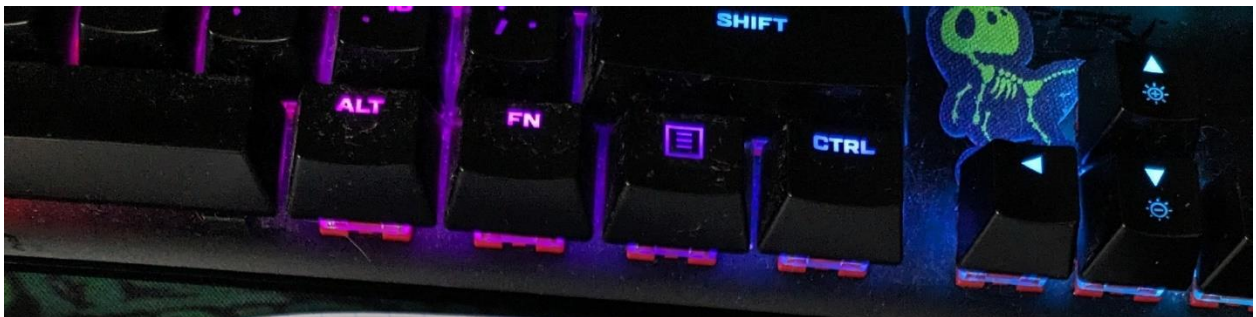
Задание 1) Реализуйте АТД Стек на односвязном списке (forward_list).



```
#include „iostream"
# include „forward_list"
using namespace std;
template <typename T>
class stack {
private forward_list<T> data;
public:
  void push(const T &value){ // добавление эле, на вершину стека
     data.push_front(value);
  }
  pop(){ // удаление с вершины стека
    if(data.empty()) {
       throw out_of_range("empty stack!");
    }
    T value = data.front(); 3 2 1
    data.pop_front(); 2 1
    return value; 2 1
  }
  T top() const{ // получение вершины стека
    if(data.empty()){
       throw out_of_range("empty stack!");
    }
    return data.front(); }
```




```
bool empty() const{
    return data.empty();
}
void pop(){
    if(data.empty()){
        throw out_of_range("empty stack!");
    }
    data.pop_front();
}

size_t size() const{ // forward_list не определяет ф-ии,
                     которые позволяют получить размер
                     контейнера, поэтому
                     метод проходится по
                     каждому элементу списка
                     и увеличивает счетчик.
                     Думаю это не очень эффективно,
                     можно было бы использовать
                     итераторы, но не уверен,
                     что это было бы лучше.)
    size_t count=0
    for(const auto& elem: data){
        count++;
    }
    return count;
}

};
```

Задание 2) Используя класс queue из STL решите следующую задачу.
(2736) Вывести простые числа среди чисел от 2 до N, используя следующий алгоритм:
Первоначально очередь все числа от 2 до N.

1. Взять первый элемент X из входной очереди и напечатать.
2. В выходную очередь поместить числа из очереди, которые не кратны X.
3. Поменять входную и выходную очередь (swap).
4. Пока очередь не пуста, то повторять дествия с шага 2.

Ввод содержит одно целое число N ($2 \le N \le 100000$).

②

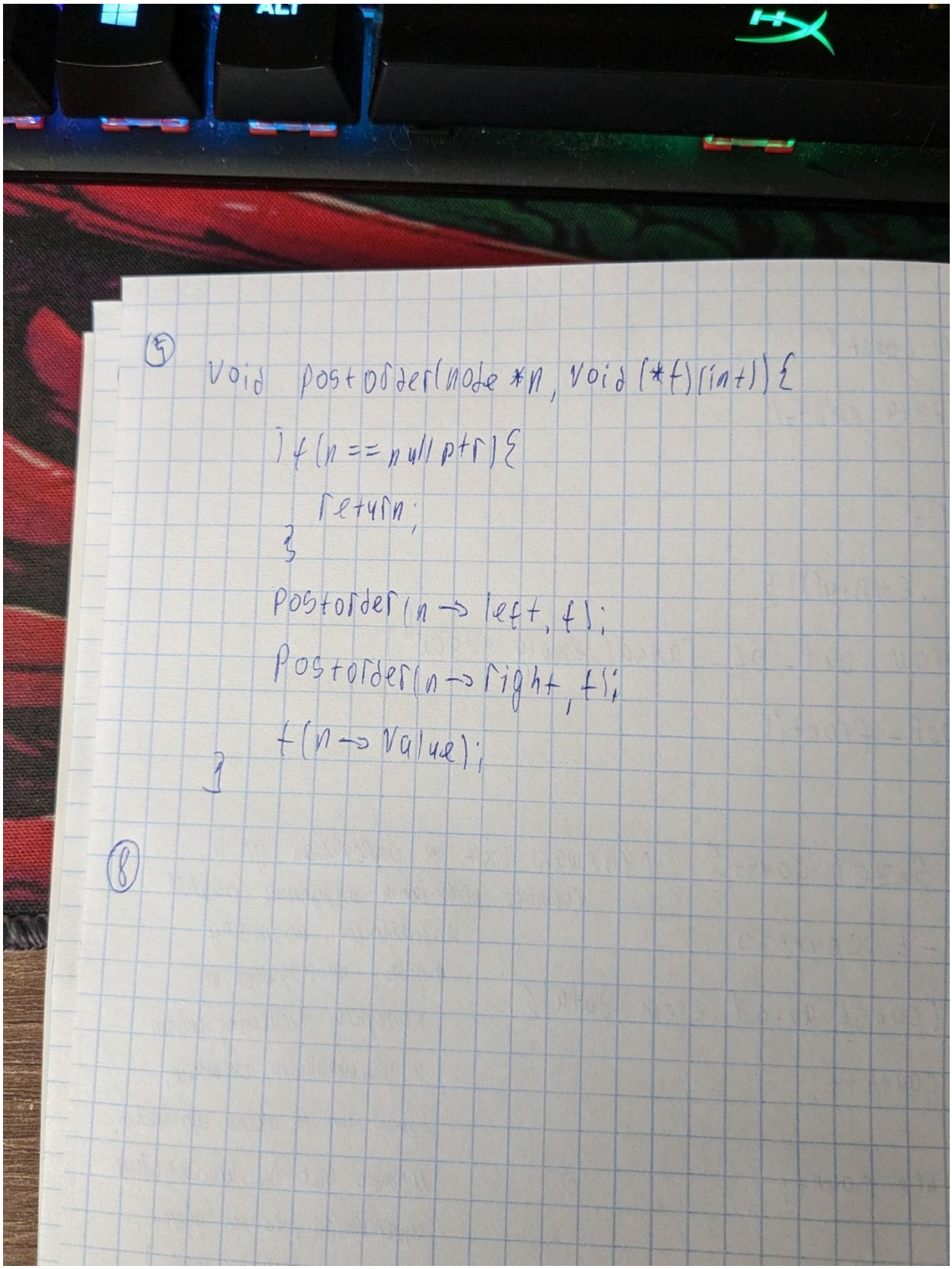
```
#include "iostream"
#include "queue"
using namespace std;
int main(){
    int N;
    cin>>N;
    queue<int> inque, outque;
    for(int i=2; i<=N; i++){
        inque.push(i);
    }
    while(!inque.empty()){
        int X = inque.front;
        cout<<X<<" ";
        inque.pop();
        while(!inque.empty()){
            int numb = inque.front();
            inque.pop();
            if(numb%X!=0){
                outque.push(numb);
            }
        }
        swap(inque, outque);
    }
}
```

Задание 5) Напишите функцию для обратного (post-order) обхода бинарного дерева, заданного следующей структурой:

```
struct node { int value; node *left, *right; };
```

К каждому значению, хранящемуся в дереве, функция применяет функцию, указанную в качестве аргумента:

```
void postorder(node *n, void (*f)(int))
```



Задание 8) Используя map из STL напишите решение следующей задачи с эффективностью O(NlogN).
Дана последовательность из n целых чисел. Найти непрерывную подпоследовательность максимальной длины, в которой нет одинаковых элементов. Вывести длину и начальный индекс найденной подпоследовательности.

1 prac.cpp *

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

using namespace std;

pair<int, int> findlong(const vector<int>& nums) {
    map<int, int> lastIndex;
    int maxLength = 0;
    int startIdx = 0;
    int left = 0;

    for (int right = 0; right < nums.size(); ++right) {
        if (lastIndex.find(nums[right]) != lastIndex.end() && lastIndex[nums[right]] >= left) {
            left = lastIndex[nums[right]] + 1; // сдвиг левого указателя вправо
        }
        lastIndex[nums[right]] = right; // обновляем последний индекс появления элемента
        if (right - left + 1 > maxLength) {
            maxLength = right - left + 1;
            startIdx = left;
        }
    }

    return {maxLength, startIdx};
}

int main() {
    vector<int> nums = {5, 1, 3, 5, 2, 3, 4, 1}; // exmp
    pair<int, int> result = findlong(nums);
    cout << "Длина: " << result.first << "\nНачальный индекс: " << result.second << endl;
    return 0;
}
```

Задание 9) Сравните время работы set и unordered_set из STL для операций поиска с количеством элементов N=100,10000,10^6,10^7 (например, измерить время поиска 10 существующих значений в наборе и 10 несуществующих). Ключами являются строки из случайных букв от a до z длиной ровно 16. Результат оформить в виде таблицы, время в ns. Привести код, использованный для измерения времени для одного значения N

```cpp
#include <iostream>
#include <set>
#include <unordered_set>
#include <string>
#include <chrono>
#include <random>

using namespace std;
using namespace std::chrono;

string get_random_string() {
    const string str = "abcdefghijklmnopqrstuvwxyz";
    string key;
    for (int i = 0; i < 16; i++)
        key += str[rand() % str.size()];
    return key;
}

int main() {
    const int N = 100;
    set<string> set;
    unordered_set<string> unordered_set;
    srand(time(NULL));
    for (int i = 0; i < N; i++) {
        string randomString = get_random_string();
        set.insert(randomString);
        unordered_set.insert(randomString);
    }
    auto start_t = high_resolution_clock::now();
    set.find("string");
    auto end_t = high_resolution_clock::now();
    auto time_set = duration_cast<nanoseconds>(end_t - start_t);
    start_t = high_resolution_clock::now();
    unordered_set.find("string");
    end_t = high_resolution_clock::now();
    auto time_u_set = duration_cast<nanoseconds>(end_t - start_t);
    cout << "Время работы set: " << time_set << endl;
    cout << "Время работы unordered_set: " << time_u_set << endl;

    return 0;
}
```

```
make: 'release/prac.exe' is up to date.
>Запуск программы...
Время работы set: 2700ns
Время работы unordered_set: 200ns
>Код завершения: 0 Время выполнения: 20 ms
Для продолжения нажмите любую клавишу . . .
```

```
>Запуск программы...
Время работы set: 3400ns
Время работы unordered_set: 300ns
>Код завершения: 0 Время выполнения: 20 ms
Для продолжения нажмите любую клавишу . . .
```

```
>Запуск программы...
Время работы set: 9100ns
Время работы unordered_set: 500ns
>Код завершения: 0 Время выполнения: 1910 ms
Для продолжения нажмите любую клавишу . . .
```

```
make: 'release/prac.exe' is up to date.
>Запуск программы...
Время работы set: 9800ns
Время работы unordered_set: 500ns
>Код завершения: 0 Время выполнения: 29040 ms
Для продолжения нажмите любую клавишу . . .
```

A	B	C	D	E
N	Время работы set	Время работы unordered set	Спеки компа	
100	2700ns	200ns	Gpu: i3-12100f	4 ядра 8 потоков 3.3ghz
10000	3400ns	300ns		кэш l1 = 320 kb
10^6	9100ns	500ns		кэш l2 = 5mb
10^7	9800ns	500ns		кэш l3 = 12mb
			Ram: 3200mhz 8x2 GB	CAS latency = 16

// До этого момента все проверено

Задание 15 исправленное) Напишите функцию для проверки отсутствия циклов в орграфе, заданном через списки смежных вершин.

1 prac.cpp *

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

using namespace std;

// Функция для выполнения топологической сортировки (циклов в графе нет, если можно сделать топологическую сортировку)
bool topologicalSort(const vector<vector<int>>& adjList, vector<int>& result) {
    int n = adjList.size();
    vector<int> inDegree(n, 0);

    // Вычисляем входную степень для каждой вершины
    for (int u = 0; u < n; ++u) {
        for (int v : adjList[u]) {
            inDegree[v]++;
        }
    }

    queue<int> q;

    // Добавляем все вершины с нулевой входной степенью в очередь
    for (int i = 0; i < n; ++i) {
        if (inDegree[i] == 0) {
            q.push(i);
        }
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        result.push_back(u);

        // Уменьшаем входную степень для всех соседей текущей вершины
        for (int v : adjList[u]) {
            inDegree[v]--;
            if (inDegree[v] == 0) {
                q.push(v);
            }
        }
    }

    // Если результат содержит все вершины, то граф ацикличен
    return result.size() == n;
}
```

Задание 7) Используя дерево отрезков решите следующую задачу.
Есть массив, содержащий 100000 элементов, первоначально все элементы равны 0. В массиве производятся изменения элементов и требуется находить суммы части массива с ii-го по jj-ый элементы.
В первой строке содержится число KK ($1 \le K \le 500001 \le K \le 50000$) -- количество запросов. Далее следует KK строк, в каждой строке содержится либо команда "S ii jj vv", где $1 \le i \le j \le 1000001 \le i \le j \le 100000$, $-100 \le v \le 100$ $-100 \le v \le 100$, заменяющая значения элементов с ii-го по jj-й массива на vv, либо команда "Q ii jj", где $1 \le i \le j \le 1000001 \le i \le j \le 100000$, требующая вывести сумму части массива с ii-го по jj-ый элементы.
Для каждой команды 'Q' вывести на отдельной строке результат запроса

1 prac.cpp *

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

using namespace std;

const int MAX_N = 100000;
const int NEUTRAL = 0;


class SegmentTree {
private:
  vector<int> tree;
  int n;

  void build(const vector<int>& arr, int v, int tl, int tr) {
    if (tl == tr) {
      tree[v] = arr[tl];
    } else {
      int tm = (tl + tr) / 2;
      build(arr, v * 2, tl, tm);
      build(arr, v * 2 + 1, tm + 1, tr);
      tree[v] = tree[v * 2] + tree[v * 2 + 1];
    }
  }

  int query(int v, int tl, int tr, int l, int r) {
    if (l > r) {
      return NEUTRAL;
    }
    if (l == tl && r == tr) {
      return tree[v];
    }
    int tm = (tl + tr) / 2;
    return query(v * 2, tl, tm, l, min(r, tm)) +
           query(v * 2 + 1, tm + 1, tr, max(l, tm + 1), r);
  }

  void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
      tree[v] = new_val;
    } else {
      int tm = (tl + tr) / 2;
      if (pos <= tm) {
        update(v * 2, tl, tm, pos, new_val);
      } else {
        update(v * 2 + 1, tm + 1, tr, pos, new_val);
      }
      tree[v] = tree[v * 2] + tree[v * 2 + 1];
    }
  }

public:
  // Конструктор
```

```
public:
  // Конструктор
  SegmentTree(const vector<int>& arr) : n(arr.size()) {
    tree.resize(4 * n);
    build(arr, 1, 0, n - 1);
  }


  int getSum(int i, int j) {// Поиск суммы на отрезке с i-го по j-ый
    return query(1, 0, n - 1, i - 1, j - 1);
  }


  void update(int pos, int new_val) {// Обновление значения в позиции pos на new_val
    update(1, 0, n - 1, pos - 1, new_val);
  }
};

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


  vector<int> arr(MAX_N, 0);  // Инициализация массива (все элементы равны 0)

  SegmentTree tree(arr);


  for (int i = 0; i < k; i++) {
    string command;
    cin >> command;
    if (command == "S") {
      int i, j, v;
      cin >> i >> j >> v;
      for (int pos = i; pos <= j; pos++) {
        tree.update(pos, v);
      }
    } else if (command == "Q") {
      int i, j;
      cin >> i >> j;
      cout << tree.getSum(i, j) << endl;
    }
  }

  return 0;
}
```

Задание 14) Используя поиск в ширину, решите задачу.

(1716) Стартовав с числа 1, нужно получить некоторое заданное число N. На каждом шаге можно добавлять к текущему числу один из его делителей, чтобы получить новое число. Например, для первого шага у нас только один вариант: добавить 1 к 1 и получить 2. На втором шаге можно выбрать один из двух делителей и получить число 2+1=3 или 2+2=4. От числа 4 на третьем шаге можно перейти к числам 5, 6 или 8 в зависимости от выбранного делителя.

Напишите программу, определяющую минимальное количество шагов для получения заданного числа N ($2 \le N \le 10^5$).

Файл Правка Поиск Вид Действия Отладка Тестирование Изменения Настройки Подсветка Вкладки Спр

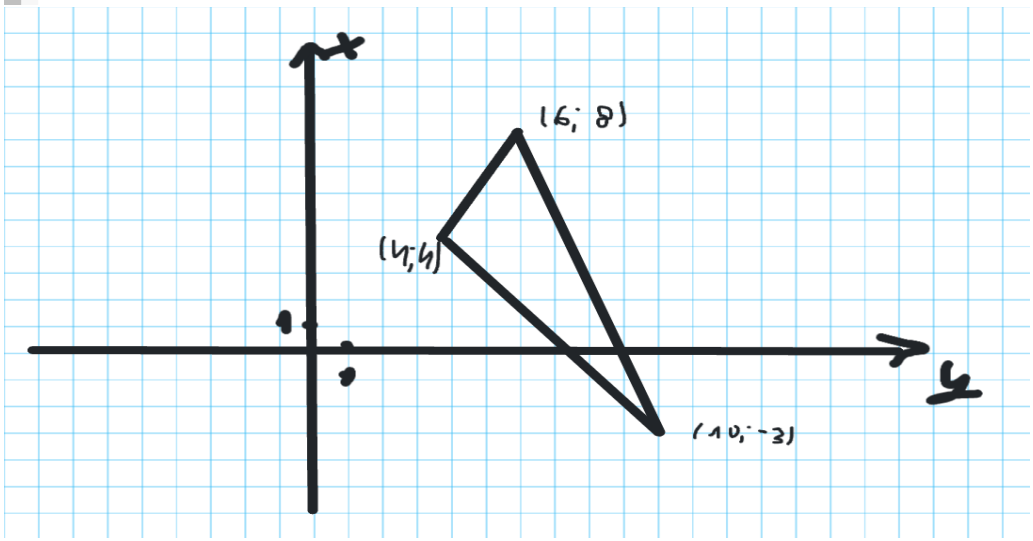
1 prac.cpp *

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_set>

using namespace std;


vector<int> getDivisors(int n) {
    vector<int> divisors;
    for (int i = 1; i * i <= n; ++i) {
        if (n % i == 0) {
            divisors.push_back(i);
            if (i != n / i) {
                divisors.push_back(n / i);
            }
        }
    }
    return divisors;
}


int minStepsToN(int N) {
    if (N == 1) return 0;

    queue<pair<int, int>> q;
    unordered_set<int> visited;

    q.push({1, 0}); // начинаем с числа 1 и 0 шагов
    visited.insert(1);

    while (!q.empty()) {
        auto [current, steps] = q.front();
        q.pop();

        for (int divisor : getDivisors(current)) {
            int nextNum = current + divisor;
            if (nextNum == N) return steps + 1;
            if (nextNum < N && visited.find(nextNum) == visited.end()) {
                visited.insert(nextNum);
                q.push({nextNum, steps + 1});
            }
        }
    }
    return 0;
}

int main() {
    int N;
    cout << "Введите число N\n";
    cin >> N;
    cout << "Минимальное количество шагов: " << minStepsToN(N) <<endl;
    return 0;
}
```

Задание 23) Определите необходимые геометрические объекты и напишите следующую функцию
В декартовой системе координат на плоскости заданы координаты вершин треугольника и ещё одной точки. Определить, принадлежит ли эта точка треугольнику.
Для точки использовать класс из лекций и его методы.

1 prac.cpp *

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

using namespace std;

struct Point { //структура из лекций
  double x,y;
  double len() const { return hypot(x,y); } // расстояние от начала координат
  double phi() const { return atan2(y,x); } // угол
  Point operator+(Point p) const { return {x+p.x,y+p.y}; }
  Point operator-(Point p) const { return {x-p.x,y-p.y}; }
  double operator*(Point p) const { return x*p.x+y*p.y; } // скалярное произведение
  double operator^(Point p) const { return x*p.y-y*p.x; } // векторное произведение
  Point operator*(double a) const { return  {a*x,a*y};} // "масштабирование"
  Point turn(double a) const { double ca=cos(a),sa=sin(a); return {x*ca-y*sa,-x*sa+y*ca}; } // поворот
  Point turn() const { return {-y,x}; } // поворот на п/2
  Point operator-() const { return {-x,-y}; } // поворот на п
};

struct Triangle{
    Point a1,a2,a3;
    void isPointInTriangle(Point a0){
      if( ((a1.x - a0.x)*(a2.y - a1.y) - (a2.x - a1.x)*(a1.y - a0.y))*((a2.x - a0.x)*(a3.y - a2.y) - (a3.x - a2.x)*(a2.y - a0.y)) >= 0){
        if(((a2.x - a0.x)*(a3.y - a2.y) - (a3.x - a2.x)*(a2.y - a0.y))*((a3.x - a0.x)*(a1.y-a3.y) - (a1.x - a3.x)*(a3.y - a0.y)) >= 0){
          cout << "In triangle";
        }
        else cout << "Not in Triangle";
      }
      else cout << "Not in Triangle";
    }
};

int main(){
    Point a,b,c,d;//очевидно, работает для любого треугольника в действительных плоскостях,
                  //не стал делать ввод с клавиатуры для большей наглядности
    a.x = 4; //первая координата треугольника(x,y)
    a.y = 4;

    b.x = 6;//вторая координата треугольника
    b.y = 8;

    c.x = 10;//третья точка треугольника
    c.y = -3;

    d.x = 2;//точка, которую проверяем, принадлежит она треугольнику или нет
    d.y = 4;

    Triangle abc;
    abc.a1 = a;
    abc.a2 = b;
    abc.a3 = c;

    abc.isPointInTriangle(d);
}
```



Задание 3) Определите АТД для хранения информации о таблице базы данных: столбцы (количество и названия столбцов), строки со значениями (все значения в строке таблицы имеют тип string), ключи для получения строк таблицы в некотором порядке (для упрощения ключ включает только один столбец, т.е. ключ - это имя или номер столбца, а строки таблицы можно получать последовательно по одной в порядке возрастания значения в указанном столбце). Перечислите методы АТД, обеспечивающие последовательный доступ к информации и её изменение, аргументы и возвращаемые значения каждого метода с комментариями.

Методы АТД:

CreateTable(CountStolb:int,StolbName:array<string>)->None

Описание: Создает новую пустую таблицу с заданным количеством и названиями столбцов.

- Аргументы:

CountStolb - количество столбцов в таблице.

StolbName - массив названий столбцов.

- Возвращаемое значение: None

Stradd(Values:array<string>) -> None

Описание: Добавляет новую строку с заданными значениями в таблицу.

-Аргументы:

Values - массив значений для новой строки. Длина массива должна совпадать с количеством столбцов в таблице.

- Возвращаемое значение: None

StrDel(index:int)->None

Описание: Удаляет строку по указанному индексу

-Аргументы:

Index – индекс строки, которую надо получить

-Возвращаемое значение: None

StrGet(index:int)->array<string>

Описание: Возвращает значение строки по индексу

-Аргументы:

Index – индекс строки, которую надо получить

-Возвращаемое значение: Массив значений строки

ValueUpd(indexstr:int,indexstolb:int,NewValue:string) ->None

Описание: Обновляет значение в указанной ячейке.

- Аргументы:

  - indexstr - индекс строки, в которой нужно обновить значение.

  - индексstolb - индекс столбца, в котором нужно обновить значение.

  - NewValue - новое значение для ячейки.

- Возвращаемое значение: None

SetupSortKey(StolbName:string)-> None

Описание: Устанавливает столбец, по которому строки будут сортироваться.

- Аргументы:

  StolbName - название столбца, по которому будет осуществляться сортировка.

- Возвращаемое значение: None

GetNextStr()->array<string>

Описание: Возвращает следующую строку в отсортированном порядке по ключевому столбцу. Если достигнут конец таблицы, возвращает пустой массив или специальное значение.

- Аргументы:None

- Возвращаемое значение: Массив значений следующей строки в порядке возрастания значений ключевого столбца.

ResetIterator()->None

Описание: Сбрасывает итератор для последовательного доступа по ключевому столбцу к началу таблицы.

- Аргументы: None

- Возвращаемое значение: None

Задание 4) Предложите структуры данных для представления АТД из задания 3. Перечислите поля, их типы и комментарии к каждому полю. Укажите оценку эффективности (амортизированную или среднюю) для каждого метода с учетом использованных структур данных. Хранимая в структуре информация не должна дублироваться.

Структуры данных и их поля:

Таблица

StolbName: array<string>— Массив названий столбцов

Stroki: array<array<string>> — Массив массивов строк, где каждая строка представляет собой массив значений.

StolbKey: string — Название столбца, по которому осуществляется сортировка.

indexStolbKey: int — Индекс ключевого столбца.

SortedIndex: array<int> — Отсортированный массив индексов строк для итерации по ключевому столбцу.

Iterator: int — Текущий индекс для итерации по отсортированным индексам.

Оценка эффективности методов:

CreateTable:O(1)

Stradd:O(1) // При условии динамического массива

StrDel:O(n)

StrGet:O(1)

ValueUpd:O(1)

SetupSortKey:O(nlogn)

GetNextStr:O(1)

ResetIterator:O(1)

Задание 11) Определить АТД Разреженная матрица, обеспечивающий метод get(i,ji,j) для получения элемента матрицы и set(i,j,vi,j,v) для изменения (добавления) ненулевого элемента. В конструкторе задаются размеры матрицы. Реализовать АТД через словарь по ключам map<pair<int,int>, double>. Определить эффективность операций + и * в зависимости от количества ненулевых элементов KK.

Файл Правка Поиск Вид Действия Отладка Тестирование Изменения Настройки Подсветка Вкладки Справка

1 prac.cpp *

```cpp
#include <iostream>
#include <map>
using namespace std;
class SparseMatrix
{
private:
  int rows; // Количество строк
  int cols; // Количество столбцов
  map<pair<int, int>, double> elements; // Словарь для хранения ненулевых элементов

public:

  SparseMatrix(int numRows, int numCols) : rows(numRows), cols(numCols) {} // Конструктор с заданием размеров матрицы

  void set(int i, int j, double v)// Метод для установки значения элемента
  {
  if (i < 0 || i >= rows || j < 0 || j >= cols){
    throw out_of_range("Out of bounds");
  }

  elements[{i, j}] = v;
  }

  double get(int i, int j) const// Метод для получения значения элемента
{

  if (i < 0 || i >= rows || j < 0 || j >= cols){
    throw out_of_range("Out of bounds");
}

  auto it = elements.find({ i, j });// Поиск значения элемента в словаре
  if (it != elements.end()){
    return it->second;
  }
  else{
    return 0;
  }
}
};
```

Операция +: O(K1 + K2)

Операция *: O(K1 * K2)

Задание 12) Определить АТД Матрица, обеспечивающий метод [i,j*i*,*j*] для доступа к элементам матрицы. В конструкторе задаются размеры матрицы.
Определить операцию *. Выполнить возведение матрицы N×N*N*×*N* из целых чисел 0 и 1 в степень K*K* по модулю 2: AK(mod2)*AK*(mod2). Для возведения матрицы в степень использовать метод уменьшения размера задачи в 2 раза.

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

class Matrix {
public:
    // Конструктор, инициализирующий матрицу заданного размера
    Matrix(int size) : size(size), data(size, std::vector<int>(size, 0)) {}

    // Оператор доступа для получения и установки элементов матрицы
    int& operator()(int i, int j) {
        return data[i][j];
    }

    // Константный оператор доступа для получения элементов матрицы (для использования в константных контекстах)
    const int& operator()(int i, int j) const {
        return data[i][j];
    }

    // Оператор умножения матриц по модулю 2
    Matrix operator*(const Matrix& other) const {
        Matrix result(size);
        for (int i = 0; i < size; ++i) {
            for (int j = 0; j < size; ++j) {
                result(i, j) = 0;
                for (int k = 0; k < size; ++k) {
                    result(i, j) ^= (data[i][k] & other(k, j));
                }
            }
        }
        return result;
    }

    // Статический метод для создания единичной матрицы заданного размера
    static Matrix identityMatrix(int size) {
        Matrix identity(size);
        for (int i = 0; i < size; ++i) {
            identity(i, i) = 1;
        }
        return identity;
    }

    // Статический метод для возведения матрицы в степень
    static Matrix power(const Matrix& matrix, int exponent) {
        if (exponent == 0) {//Если степень равна нулю
            return identityMatrix(matrix.size);
        }
        if (exponent % 2 == 1) {//Если степень нечетная

            return matrix * power(matrix, exponent - 1);
        }
        Matrix halfPower = power(matrix, exponent / 2);// Если степень четная
        return halfPower * halfPower;
    }

    // Метод для вывода матрицы на экран
    void print() const {
```

```
    // Метод для вывода матрицы на экран
    void print() const {
        for (const auto& row : data) {
            for (int element : row) {
                std::cout << element << " ";
            }
            std::cout << std::endl;
        }
    }

private:
    int size;
    std::vector<std::vector<int>> data;
};

int main() {
    int size = 3; // Размер матрицы
    Matrix matrix(size);

    // Условное заполнение матрицы
    matrix(0, 0) = 1; matrix(0, 1) = 1; matrix(0, 2) = 0;//matrix(i,j) = число, стоящее в i строке и j столбце
    matrix(1, 0) = 0; matrix(1, 1) = 1; matrix(1, 2) = 1;
    matrix(2, 0) = 1; matrix(2, 1) = 0; matrix(2, 2) = 1;

    int exponent = 1; // Степень
    Matrix result = Matrix::power(matrix, exponent); // Возведение матрицы в степень

    std::cout << "Матрица возведенная в степень: " << exponent << "По модулю 2" << std::endl;
    result.print(); // Вывод результата

    return 0;
}
```

Задание 13) Используя поиск в глубину, определите число компонент связности в графе, задаваемом следующим образом:
(945) Как матрица $N \times M N \times M$ из клеток черного ('B') и белого ('W') цветов. Клетки считаются связными, если они имеют общую границу и цвета клеток *отличаются* (компонента раскрашена в черно-белую клетку как шахматная доска).

1 prac.cpp

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

using namespace std;
//поиск в глубину
void dfs(const vector<vector<char>>& grid, vector<vector<bool>>& visited, int x, int y, int N, int M) {
    // Массивы для перемещения по соседним клеткам(вверх,вниз,влево,вправо)
    int dx[] = {-1, 1, 0, 0};
    int dy[] = {0, 0, -1, 1};

    visited[x][y] = true;//метка текущей клетки, что мы ее уже посетили

    for (int i = 0; i < 4; ++i) {//4 направления
        int nx = x + dx[i];
        int ny = y + dy[i];

        if (nx >= 0 && nx < N && ny >= 0 && ny < M && !visited[nx][ny] && grid[nx][ny] != grid[x][y]) {//проверка границ матрицы + условие для перехода
            dfs(grid, visited, nx, ny, N, M);
        }
    }
}
// Подсчет числа компонент связности
int countComponents(const vector<vector<char>>& grid) {
    int N = grid.size();//строки
    int M = grid[0].size();//столбцы
    vector<vector<bool>> visited(N, vector<bool>(M, false));
    int components = 0;

    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            if (!visited[i][j]) {
                dfs(grid, visited, i, j, N, M);
                components++;
            }
        }
    }

    return components;
}

int main() {
    int N = 3;
    int M = 3;
    vector<vector<char>> grid = {
        {'B', 'W', 'B'},
        {'W', 'B', 'W'},//1 компонента связности должна быть
        {'B', 'W', 'B'}
    };

    cout << "Number of components: " << countComponents(grid) <<endl;

    return 0;
}
```

Задание 20) Сравните время сортировки с помощью sort, stable_sort, make_heap/sort_heap для вектора из $10^6$106 случайных чисел. Результаты оформить в виде таблицы.

1 prac.cpp

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <random>
using namespace std;
int main()
{
 const int n = 1e6; // Размер вектора
 // Создание вектора и заполнение его случайными числами
 vector<int> numbers(n);
 random_device rd;
 mt19937 gen(rd());
 uniform_int_distribution<int> dis(INT_MIN, INT_MAX);
 for (int& i : numbers)
 {
 i = dis(gen);
 }
 // создаём копию numbers
 vector<int> numbers_copy(numbers);
 // Сортировки с помощью sort
 auto start = chrono::high_resolution_clock::now();
 sort(numbers.begin(), numbers.end());
 auto end = chrono::high_resolution_clock::now();
 auto sort_duration = chrono::duration_cast<chrono::milliseconds>(end -
start).count();
 numbers = numbers_copy;
 // Сортировка с помощью stable_sort
 start = chrono::high_resolution_clock::now();
 stable_sort(numbers.begin(), numbers.end());
 end = chrono::high_resolution_clock::now();
 auto stable_sort_duration = chrono::duration_cast<chrono::milliseconds>(end -
start).count();
 numbers = numbers_copy;
 // Сортировки с помощью make_heap и sort_heap
 start = chrono::high_resolution_clock::now();
 make_heap(numbers.begin(), numbers.end());
 sort_heap(numbers.begin(), numbers.end());
 end = chrono::high_resolution_clock::now();
 auto make_heap_sort_duration = chrono::duration_cast<chrono::milliseconds>(end -
start).count();
 cout << "sort: " << sort_duration << " ms" << endl;
 cout << "stable_sort: " << stable_sort_duration << " ms" << endl;
 cout << "make_heap/sort_heap: " << make_heap_sort_duration << " ms" << endl;
 return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
make: 'release/prac.exe' is up to date.
>Запуск программы...
sort: 58 ms
stable_sort: 66 ms
make_heap/sort_heap: 54 ms
>Код завершения: 0 Время выполнения: 210 ms
Для продолжения нажмите любую клавишу . . .
```

A	B	C	D	E
Вид Сортировки	Время		Спеки компа	
sort	58ms		Cpu: i3-12100f	4 ядра 8 потоков 3.3ghz
stable_sort	66ms			кэш l1 = 320 kb
make_heap/sort_heap	54ms			кэш l2 = 5mb
				кэш l3 = 12mb
			Ram: 3200mhz 8x2 GB	CAS latency = 16
			Компилятор:MinIde	

Замечание по поводу выполнения нескольки поисков учел, данное время является +- средним, относительно всех замеров, дисперсия около 1-3ms.