1. Реализуйте АТД Очередь на vector, используя [циклический буфер](https://ipc.susu.ru/viz/queue.html), с расширением размера в 2 раза при переполнении. Операция pop() должна выполняться за O(1), push() за O(1) (амортизированная оценка).

template <typename T>

class Queue

{

    size\_t max\_sz, sz, fr, back;

    vector<T> q;

public:

    Queue() : max\_sz(2), sz(0), q(2), fr(0), back(0) {}

    void push(T el);

    void pop();

    T front();

    size\_t size() const { return sz; }

};

template <typename T>

void Queue<T>::push(T el)

{

    if (sz == max\_sz)

    {

        max\_sz \*= 2;

        q.resize(max\_sz);

        if (fr != 0)

        {

            for (int i = fr; i < sz; i++)

                q[sz + i] = q[i];

            fr += sz;

        }

    }

    if (back >= max\_sz)

        back = 0;

    q[back++] = el;

    sz++;

}

template <typename T>

void Queue<T>::pop()

{

    if (sz == 0)

        return;

    fr++;

    if (fr >= max\_sz)

        fr = 0;

    sz--;

}

template <typename T>

T Queue<T>::front()

{

    return q[fr];

}

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

Ввод содержит одно целое число N𝑁 (2 ≤ N ≤ 100000).

Отправил на ipc.susu.ru

Напишите функцию для симметричного (in-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 inorder(node \*n, void (\*f)(int));

void inorder(node \*n, void (\*f)(int))

{

    if (!n)

        return;

    inorder(n->left, f);

    f(n->value);

    inorder(n->right, f);

}

1. 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.

#include <iostream>

#include <vector>

#include <map>

using namespace std;

int main()

{

    int n;

    cin >> n;

    vector<int> arr(n);

    for (int i = 0; i < n; i++)

        cin >> arr[i];

    map<int, int> m;

    pair<int, int> ans = {1, 0};

    int i = 0, j = 0;

    while (j < n)

    {

        if (m[arr[j]] == 0)

        {

            m[arr[j++]]++;

            if (j - i > ans.first)

                ans = {j - i, i};

        }

        else

            m[arr[i++]]--;

    }

    cout << ans.first << ' ' << ans.second << '\n';

    return 0;

}

1. Сравните время работы set и unordered\_set из STL для операций поиска с количеством элементов N=100,10000,106,107. 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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | 100 | 1e4 | 1e6 | 1e7 |
| set | 201 | 328 | 1431 | 2459 |
| unordered\_set | 116 | 139 | 459 | 627 |

(Средние значения в std::chrono::nanoseconds)

void create(int size, set<string> &st, unordered\_set<string> &u\_st, vector<string> &tests)

{

    string s = "";

    for (int i = 0; i < size; i++)

    {

        for (int j = 0; j < 16; j++)

            s.push\_back(char('a' + rand() % ('z' - 'a' + 1)));

        st.insert(s);

        u\_st.insert(s);

        if (tests.size() < 100)

            tests.push\_back(s);

        s.clear();

    }

}

void check\_speed(set<string> &st, unordered\_set<string> &u\_st, vector<string> &tests)

{

    std::chrono::steady\_clock::time\_point begin, end;

    long long unset\_count = 0, set\_count = 0;

    for (int i = 0; i < tests.size(); i++)

    {

        begin = std::chrono::steady\_clock::now();

        st.find(tests[i]);

        end = std::chrono::steady\_clock::now();

        set\_count += chrono::duration\_cast<std::chrono::nanoseconds>(end - begin).count();

    }

    for (int i = 0; i < tests.size(); i++)

    {

        begin = std::chrono::steady\_clock::now();

        u\_st.find(tests[i]);

        end = std::chrono::steady\_clock::now();

        unset\_count += chrono::duration\_cast<std::chrono::nanoseconds>(end - begin).count();

    }

    cout << "set: " << set\_count / tests.size() << '\n';

    cout << "unordered\_set: " << unset\_count / tests.size() << '\n';

}

int main()

{

    srand(time(NULL));

    set<string> st;

    unordered\_set<string> u\_st;

    vector<string> tests;

    int size = 100;

    create(size, st, u\_st, tests);

    cout << "size: " << size << '\n';

    check\_speed(st, u\_st, tests);

    st.clear();

    u\_st.clear();

    tests.clear();

size = (int)1e4;

// . . .

return 0;

}

1. Определить АТД Полином, обеспечивающий метод calc для вычисления значения полинома в точке x.

Реализовать полином через представление в виде вектора коэффициентов. В конструкторе задается набор коэффициентов. Определить операции + и \*.

class Polynom

{

    vector<long double> coefs;

public:

    Polynom();

    Polynom(vector<long double> coefs) : coefs(coefs){};

    long double calc(long double x);

    Polynom operator+(Polynom b);

    Polynom operator\*(Polynom b);

};

long double Polynom::calc(long double x)

{

    long double c = 1, ans = 0;

    for (int i = coefs.size() - 1; i >= 0; i--)

    {

        ans += c \* coefs[i];

        c \*= x;

    }

    return ans;

}

Polynom Polynom::operator+(Polynom b)

{

    vector<long double> c = coefs;

    if (b.coefs.size() > c.size())

        c.resize(b.coefs.size());

    for (int i = 0; i < b.coefs.size(); i++)

        c[i] += b.coefs[i];

    return Polynom(c);

}

Polynom Polynom::operator\*(Polynom b)

{

    vector<long double> c(coefs.size() + b.coefs.size() - 1);

    for (int i = 0; i < coefs.size(); i++)

        for (int j = 0; j < b.coefs.size(); j++)

            c[i + j] += coefs[i] \* b.coefs[j];

    return Polynom(c);

}

1. Используя поиск в ширину, решите задачу.  
   (1343) На доске 8x8 некоторые клетки произвольным образом покрашены в черный цвет (кроме верхнего левого и правого нижнего угла доски). Требуется определить имеется ли путь для шахматного коня из верхнего левого в правый нижний угол доски, не проходящий по черным клеткам, и минимальное количество ходов, требующееся для этого.

Отправил на ipc.susu.ru

1. Напишите функцию для проверки отсутствия циклов в орграфе, заданном через списки смежных вершин.

// Вершины [0, N - 1], N - размер graph

bool checkCycle(vector<vector<int>> &graph)

{

    vector<bool> visited(graph.size());

    deque<int> q;

    for (int i = 0; i < graph.size(); i++)

    {

        if (!visited[i])

        {

            q.push\_back(i);

            while (q.size())

            {

                int p = q.front();

                q.pop\_front();

                if (visited[p])

                    return false;

                visited[p] = true;

                for (int j = 0; j < graph[p].size(); j++)

                    q.push\_back(graph[p][j]);

            }

        }

    }

    return false;

}

1. Модифицируйте алгоритм Дейкстры для решения задачи:  
   В городе есть N площадей, соединенных M дорогами. Известна длина каждой дороги и номера площадей ai, bi (1 ≤ ai,bi ≤N), соединенных этой дорогой. Посчитайте количество способов добраться с площади A𝐴 до площади B так, чтобы пройденный путь был минимален.

#include <iostream>

#include <vector>

#include <set>

#include <climits>

using namespace std;

typedef long long ll;

vector<vector<pair<ll, ll>>> graph;

vector<pair<ll, ll>> visited;

int main()

{

    int n, m;

    cin >> n >> m;

    graph.resize(n + 1);

    visited.resize(n + 1, {LONG\_LONG\_MAX, 0});

    int a, b, c;

    for (int i = 0; i < m; i++)

    {

        cin >> a >> b >> c;

        graph[a].push\_back({b, c});

        graph[b].push\_back({a, c});

    }

    cin >> a >> b;

    multiset<pair<ll, ll>> st;

    st.insert({0, a});

    while (st.size())

    {

        auto it = st.begin();

        ll curTime = it->first;

        ll curVert = it->second;

        st.erase(it);

        if (visited[curVert].first < curTime){

st.erase(\*it);

            continue;

}

        else if (visited[curVert].first == curTime)

            visited[curVert].second++;

        else

            visited[curVert] = {curTime, 1};

        if (curVert == b)

            continue;

        for (int i = 0; i < graph[curVert].size(); i++)

        {

            ll vert = graph[curVert][i].first;

            ll time = graph[curVert][i].second;

            if (visited[vert].first >= curTime + time)

                st.insert({curTime + time, vert});

        }

    }

    cout << visited[b].second;

    return 0;

}

1. Напишите функцию разложения числа на простые множители.

vector<int> toPrimeDivisors(int number)

{

    vector<int> ans;

    for (int i = 2; i \* i <= number; i++)

        while (number % i == 0)

        {

            number /= i;

            ans.push\_back(i);

        }

    if (number != 1)

        ans.push\_back(number);

    return ans;

}