**Структуры данных**

1. Queue

Операции: push(), pop(), size(), front(), back()

2. stack

Операции: push(), pop(), top(), size(), empty()

3. set

Операции: size(), insert(), erase(), find(), count(), clear()

Итераторы: begin(), end(), rbegin(), rend()

4. vector

Операции: size(), push\_back(), pop\_back(), erase(), clear()

Итераторы: begin(), end()

5. graph

Vector <vector <int> > graph;

6.map

Операции: insert(), erase(), swap(), empty(), size(), clear

Итераторы: begin(), end(), rbegin(), rend()

**Алгоритмы**

**Бинарный поиск**: быстрый поиск индекс элемента в отсортированном массиве/векторе, а также место, где он должен стоять, если его нет. Сложность: logN. На каждом шаге проверяет на условие средний элемент, и двигает леву либо правую границу, сужая границы поиска в 2 раза.

Pair <int ,int> binary\_search\_R(int zleft, int zright)

{

int mid = zleft+(zright-zleft)/2;

//cout <<zleft <<" " <<zright <<"\n";

if (zleft>zright) { return make\_pair(0, zleft);}

if (mass[mid] == x) { return make\_pair(1, mid);}

if (mass[mid] < x) { binary\_search\_R(mid+1, zright); }

if (mass[mid] > x) { binary\_search\_R(zleft, mid-1); }

}

**BFS:** обход графа с запуском от всех соседей на каждом шаге, нахождение кратчайшего пути. Используется часто для решения игровых задач. Реализация с использованием структуры очередь. Сложность: О(v+e).

Реализация:

queue<int> q;

q.push (s);

vector<bool> used (n);

vector<int> d (n), p (n);

used[s]= true;

p[s]= -1;

while (!q.empty()) {

int v = q.front();

q.pop();

for (int i=0; i<g[v].size(); ++i) {

int to = g[v][i];

if (!used[to]) {

used[to]= true;

q.push (to);

d[to]= d[v]+ 1;

p[to]= v;

}

}

}

**DFS:**  обход графа, поиск всех путей, перебор (N!), нахождение компонент связности. Реализуется при помощи структуры граф. Запускается от соседней вершины, пока не придет в тупик. Затем возвращается на предыдущую итерацию и запускается от другого соседа. Сложность: О(v+e).

*Наиболее простая реализация:*

vector < vector<int> > g; // граф

int n; // число вершин

vector<char> used;

void dfs (int v) {

used[v] = true;

for (vector<int>::iterator i=g[v].begin(); i!=g[v].end(); ++i)

if (!used[\*i]) {dfs (\*i); used[v] = false;}

used[v] = false;

}