# ICPC Team Reference Document
## HSE-NN 2

## Содержание

# 1 Шаблон

```
#define _USE_MATH_DEFINES
//#include <bits/stdc++.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
#include <set>
#include <queue>
#include <utility>
#include <iomanip>
#include <cstdio>
#include <cstdlib>
#include <numeric>
#include <cmath>
#include <stack>
#include <map>
#include <deque>
#include <sstream>
using namespace std;
#define int long long
typedef vector<int> vi;
typedef vector<pair<int, int>> vii;
typedef long long ll;
typedef long double ld;
//#define pi M_PI
#define all(x) (x).begin(), (x).end()
#define pb push_back
#define re return
#define fr(x) for(int i = 0; i <(x); i++)
const int inf = 1000000000 + 7;
signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
}
```

# 2 Алгоритмы на строки

## 2.1 Префикс-функция

```
vector<int> prefix_function (string s) {
        int n = (int) s.length();
        vector<int> pi (n);
        for (int i=1; i<n; ++i) {
                int j = pi[i-1];
                while (j > 0 && s[i] != s[j])
                        j = pi[j-1];
                if (s[i] == s[j]) ++j;
                pi[i] = j;
        }
        return pi;
}
```

## 2.2 Z-функция

```
vector<int> z_function (string s) {
        int n = (int) s.length();
        vector<int> z (n);
        for (int i=1, l=0, r=0; i<n; ++i) {
                if (i <= r)
                        z[i] = min (r-i+1, z[i-l]);
                while (i+z[i] < n && s[z[i]] == s[i+z[i]])
                        ++z[i];
                if (i+z[i]-1 > r)
                        l = i, r = i+z[i]-1;
        }
        return z;
}
```

## 2.3 Хеширование

```
const int mod = 1000000000 + 7;
const int q = 1009;
vector <ll> ph;
vector <ll> pq;
void pq_put()
{
        pq.pb(1);
        for (size_t i = 1; i < 100000; ++i)
                pq.pb((pq[i - 1] * q) % mod);
}
ll hashing(string s)
{
        ll h = 0;
        if (ph.size())h = ph.back();
        for (int i = 0; i < s.size(); i++)
        {
                h = (h * q + s[i]) % mod;
                ph.pb(h);
        }
        re h;
}

ll get(int l, int r)
{
        ll ans = ph[r];
        if (l) {
                ans -= ph[l - 1] * pq[r - l + 1] % mod;
                if (ans < 0)ans += mod;
        }
        re ans;
}
```

# 3 Алгоритмы на графах

## 3.1 Алгоритм Дейкстры $O(n^2)$

was - брали вершину или нет v - список смежности d - массив расстояний для точки x

```
int d[2001];
int was[2001];
vector<pair<int, int>> v[2001];
int n;
void dijkstra(int x) {
        for (int i = 0; i < n; i++)
                d[i] = inf;
        d[x] = 0;
        for (int it = 0; it < n; it++)
        {
                int id = -1;
                for (int i = 0; i < n; i++)
                        if (!was[i])if (id == -1 || d[id] > d[i])
                                id = i;
                was[id] = -1;
                for (auto p : v[id]) {
                        int y = p.first;
                        int t = p.second;
                        d[y] = min(d[y], d[id] + t);
                }
        }
}
```

## 3.2 Алгоритм Дейкстры $O(log(n) \cdot m)$

d - массив расстояний для точки x

```
int d[3001];
vector<pair<int, int>> v[3001];
bool f(int x, int y) {
        if (d[x] != d[y])
                return d[x] < d[y];
        return x < y;
}
set <int, bool(*)(int, int)> s(f);
void dijkstra(int x) {
        x--;
        for (int i = 0; i <= n; i++)
        {
                d[i] = inf;
        }
        d[x] = 0;
        s.insert(x);
        while (!s.empty()) {
                int x = *s.begin();
                s.erase(x);
                for (auto p : v[x]) {
                        int y = p.first;
                        int t = p.second;
                        if (d[y] > d[x] + t) {
                                s.erase(y);
                                d[y] = d[x] + t;
                                s.insert(y);
                        }
                }
        }
}
```

## 3.3 Поток

```
ll c[102][102];
ll f[102][102];
int was[102];
int p[102];
vector <vector<int>> v(102);
int t;
ll dfs(int x, ll capacity) {
        if (x == t) {
                return capacity;
        }
        was[x] = 1;
        for (auto y : v[x]) {
                ll flow = min(c[x][y] - f[x][y], capacity);
                if (!was[y] && flow > 0) {
                        ll delta = dfs(y, flow);
                        if (delta == 0)
                                continue;
                        p[x] = y;
                        return delta;
                }
        }
        return 0;
}


void calc(int x, ll cap) {

        int y = x;

        while (y != t) {
                f[y][p[y]] += cap;
                f[p[y]][y] -= cap;
                y = p[y];
        }

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

        for (int i = 0; i < k; i++) {
                int a, b; ll w;
                cin >> a >> b >> w;
                c[a][b] = w;
                c[b][a] = w;
                v[a].push_back(b);
                v[b].push_back(a);
        }

        ll ans = 0;
        t = n;
        while (ll cap = dfs(1, 1000000000000000000)) {
                calc(1, cap);
                ans += cap;
                memset(was, 0, sizeof(was));
                memset(p, 0, sizeof(p));
        }

        cout << ans;
        return 0;
}
```

## 3.4 Поиск компонент сильной связности, построение конденсации графа $O(N+M)$

Дан ориентированный граф G, множество вершин которого V и множество рёбер — E. Петли и кратные рёбра допускаются. Обозначим через n количество вершин графа, через m — количество рёбер.

Компонентой сильной связности (strongly connected component) называется такое (максимальное по включению) подмножество вершин C, что любые две вершины этого подмножества достижимы друг из друга, т.е. для $\forall u,v \in C$ :

$u \mapsto v, v \mapsto u$

```
vector < vector<int> > g, gr;
vector<char> used;
vector<int> order, component;

void dfs1 (int v) {
        used[v] = true;
        for (size_t i=0; i<g[v].size(); ++i)
                if (!used[ g[v][i] ])
                        dfs1 (g[v][i]);
        order.push_back (v);
}

void dfs2 (int v) {
        used[v] = true;
        component.push_back (v);
        for (size_t i=0; i<gr[v].size(); ++i)
                if (!used[ gr[v][i] ])
                        dfs2 (gr[v][i]);
}

int main() {
        int n;
        ... read n ...
        for (;;) {
                int a, b;
                ... read edge (a,b) ...
                g[a].push_back (b);
                gr[b].push_back (a);
        }

        used.assign (n, false);
        for (int i=0; i<n; ++i)
                if (!used[i])
                        dfs1 (i);
        used.assign (n, false);
        for (int i=0; i<n; ++i) {
                int v = order[n-1-i];
                if (!used[v]) {
                        dfs2 (v);
                        ... cout component ...
                        component.clear();
                }
        }
}
```

## 3.5 Поиск мостов $O(N+M)$

Пусть дан неориентированный граф. Мостом называется такое ребро, удаление которого делает граф несвязным (или, точнее, увеличивает число компонент связности). Требуется найти все мосты в заданном графе.

```
const int MAXN = ...;
vector<int> g[MAXN];
bool used[MAXN];
int timer, tin[MAXN], fup[MAXN];

void dfs (int v, int p = -1) {
        used[v] = true;
        tin[v] = fup[v] = timer++;
        for (size_t i=0; i<g[v].size(); ++i) {
                int to = g[v][i];
                if (to == p) continue;
                if (used[to])
                        fup[v] = min (fup[v], tin[to]);
                else {
                        dfs (to, v);
                        fup[v] = min (fup[v], fup[to]);
                        if (fup[to] > tin[v])
                                IS_BRIDGE(v,to);
                }
        }
}

void find_bridges() {
        timer = 0;
        for (int i=0; i<n; ++i)
                used[i] = false;
        for (int i=0; i<n; ++i)
                if (!used[i])
                        dfs (i);
}
```

### 3.6 Поиск точек сочленения

Пусть дан связный неориентированный граф. Точкой сочленения (или точкой артикуляции, англ. "cut vertex"или "articulation point") называется такая вершина, удаление которой делает граф несвязным.

```
vector<int> g[MAXN];
bool used[MAXN];
int timer, tin[MAXN], fup[MAXN];

void dfs (int v, int p = -1) {
        used[v] = true;
        tin[v] = fup[v] = timer++;
        int children = 0;
        for (size_t i=0; i<g[v].size(); ++i) {
                int to = g[v][i];
                if (to == p) continue;
                if (used[to])
                        fup[v] = min (fup[v], tin[to]);
                else {
                        dfs (to, v);
                        fup[v] = min (fup[v], fup[to]);
                        if (fup[to] >= tin[v] && p != -1)
                                IS_CUTPOINT(v);
                        ++children;
                }
        }
        if (p == -1 && children > 1)
                IS_CUTPOINT(v);
}

int main() {
        int n;
        ... read n & g ...

        timer = 0;
        for (int i=0; i<n; ++i)
                used[i] = false;
        dfs (0);
}
```

## 4 Простые алгоритмы

### 4.1 Решето Эратосфена $O(n)$

pr - все простые числа до n
lp - минимальный простой делитель числа i

```
const int N = 10001000;
int lp[N + 1];
vector<int> pr;
void pcalc() {
    for (int i = 2; i <= N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            pr.push_back(i);
        }
        for (int j = 0; j < (int) pr.size() &&
                        pr[j] <= lp[i] && i * pr[j] <= N; ++j)
            lp[i * pr[j]] = pr[j];
    }
}
```

### 4.2 Решето Эратосфена $O(n \cdot log(log(n)))$

d[i] == 1 если число i простое

```
long long d[10000000];
void calc_p(int n)
{
        d[0] = 1;
        d[1] = 1;
        for (int i = 2; i <= n; i++)
        {
                if(d[i]==0)
                        for (int j = i + i; j <= n; j += i)
                        {
                                d[j] = 1;
                        }
        }
}
```

### 4.3 Умножение чисел по модулю

```
ll mod;
long long mulmod(long long n, long long p){
    if (p == 0)
        return 0;
    if (p == 1)
        return n % mod;
    long long tmp = mulmod(n, p/2);
    long long ans = (tmp + tmp) % mod;
    if (p % 2 == 1)
        ans = (ans + n) % mod;
    return ans;
}
```

### 4.4 Функция Эйлера

Количество таких чисел в отрезке [1; n], наибольший общий делитель которых с n равен единице.

Если p — простое число, то $\phi$ (p)=p-1. (Это очевидно, т.к. любое число, кроме самого p, взаимно просто с ним.)

Если p — простое, a — натуральное число, то $\phi(p^a) = p^a - p^{a-1}$. (Поскольку с числом $p^a$ не взаимно просты только числа вида $pk (k \in \mathcal{N})$, которых $p^a/p = p^{a-1}$ штук.)

Если a и b взаимно простые, то $\phi(ab) = \phi(a)\phi(b)$

Самое известное и важное свойство функции Эйлера выражается в теореме Эйлера:

$a^{\phi(m)} \equiv 1 \pmod m$,

где $a$ и $m$ взаимно просты. В частном случае, когда $m$ простое, теорема Эйлера превращается в так называемую малую теорему Ферма:

$a^{m-1} \equiv 1 \pmod m$

```
int phi (int n) {
        int result = n;
        for (int i=2; i*i<=n; ++i)
                if (n % i == 0) {
                        while (n % i == 0)
                                n /= i;
                        result -= result / i;
                }
        if (n > 1)
                result -= result / n;
        return result;
}
```

### 4.5 Алгоритм Евклида

```
int gcd (int a, int b) {
        return b ? gcd (b, a % b) : a;
}
```

### 4.6 Расширенный алгоритм Евклида

$a \cdot x + b \cdot y = \gcd(a, b)$.

```
int gcd (int a, int b, int & x, int & y) {
	if (a == 0) {
		x = 0; y = 1;
		return b;
	}
	int x1, y1;
	int d = gcd (b%a, a, x1, y1);
	x = y1 - (b / a) * x1;
	y = x1;
	return d;
}
```

### 4.7 Обратный элемент в кольце по модулю

Обратным к числу a по модулю m называется такое число b, что:

$a \cdot b \equiv 1 \pmod m$

```
int x, y;
int g = gcdex (a, m, x, y);
if (g != 1)
	cout << "no solution";
else {
	x = (x % m + m) % m;
	cout << x;
}
```

### 4.8 Нахождение всех простых по заданному модулю за линейное время

```
r[1] = 1;
for (int i=2; i<m; ++i)
	r[i] = (m - (m/i) * r[m%i] % m) % m;
```

### 4.9 Дискретное логарифмирование

Задача дискретного логарифмирования заключается в том, чтобы по данным целым a, b, m решить уравнение:

$a^x = b \pmod m$,

где a и m — взаимно просты

```
int solve (int a, int b, int m) {
	int n = (int) sqrt (m + .0) + 1;

	int an = 1;
	for (int i=0; i<n; ++i)
		an = (an * a) % m;

	map<int,int> vals;
	for (int i=1, cur=an; i<=n; ++i) {
		if (!vals.count(cur))
			vals[cur] = i;
		cur = (cur * an) % m;
	}

	for (int i=0, cur=b; i<=n; ++i) {
		if (vals.count(cur)) {
			int ans = vals[cur] * n - i;
			if (ans < m)
				return ans;
		}
		cur = (cur * a) % m;
	}
	return -1;
}
```

## 5 Структуры данных

### 5.1 Дерево отрезков

```
ll t[4*100000];
void build(int v, int vl,int vr, vi& a){
    if(vl == vr){
        t[v] = a[vl];
        return;
    }
    int c = vl + (vr- vl)/2;
    build(2*v+1,vl,c,a);
    build(2*v+2,c+1,vr,a);
    t[v] = max(t[2*v+1], t[2*v+2]);
}
ll sum(int v, int vl, int vr, int l, int r){
    if(l > vr || r < vl){
        return -inf - 1;
    }
    if(l <= vl && vr <= r)
        return t[v];
    int c = vl + (vr- vl)/2;
    ll q1 = sum(2*v+1, vl, c, l,r);
    ll q2 = sum(2*v+2,c+1,vr,l,r);
    return max(q1, q2);
}
void modify(int v, int vl, int vr, int pos, int x){
    if(vl == vr){
        t[v] = x;
        return;
    }
    int c = vl + (vr- vl)/2;
    if(c >= pos)
        modify(2*v + 1, vl, c, pos,x);
    else
        modify(2*v + 2,c +1,vr,pos,x);
    t[v] = max(t[2*v+1], t[2*v+2]);
}
```

Прибавление на отрезке

```
void update (int v, int vl, int vr, int l, int r, int add) {
    if (l > r)
        return;
    if (l == vl && vr == r)
        t[v] += add;
    else {
        int c = vl + (vr- vl)/2;
        update (v*2+1, vl, c, l, min(r,c), add);
        update (v*2+2, c+1, vr, max(l,c+1), r, add);
    }
}

int get (int v, int vl, int vr, int pos) {
    if (vl == vr)
        return t[v];
    int c = vl + (vr- vl)/2;
    if (pos <= c)
        return t[v] + get (v*2+1, vl, c, pos);
    else
        return t[v] + get (v*2+2, c+1, vr, pos);
}
```

Присвоение на отрезке

```
void push (int v) {
		if (t[v] != -1) {
			t[v*2+1] = t[v*2+2] = t[v];
			t[v] = -1;
		}
}
void update (int v, int vl, int vr, int l, int r, int color) {
		if (l > r)
			return;
		if (l == vl && vr == r)
			t[v] = color;
		else {
			push (v);
			int c = vl + (vr- vl)/2;
			update (v*2+1, vl, c, l, min(r,c), color);
			update (v*2+2, c+1, vr, max(l,c+1), r, color);
		}
}
int get (int v, int vl, int vr, int pos) {
		if (vl == vr)
			return t[v];
		push (v);
		int c = vl + (vr- vl)/2;
		if (pos <= c)
			return get (v*2+1, vl, c, pos);
		else
			return get (v*2+2, c+1, vr, pos);
}
```

# 6 Геометрия

## 6.1 Полярный угол

```
ld u = atan2(b, a);
if (u < 0) u += 2 * PI;
```

## 6.2 Скалярное произведение, угол между векторами

$\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot cos\varphi$

$\vec{a} \cdot \vec{b} = x_1 \cdot x_2 + y_1 \cdot y_2$

$|\vec{a}| = \sqrt{x^2 + y^2}$

```
double ans = acos((x1 * x2 + y1 * y2)/
sqrt((x1 * x1 + y1 * y1) * (x2 * x2 + y2 * y2)));
```

## 6.3 Площадь многоугольника

```
int n;
cin >> n;
vector<pair<int, int>>a(n);
for (int i = 0; i < n; i++) {
        cin >> a[i].X >> a[i].Y;
}
double s = 0;
for (int i = 0; i < n - 1; i++) {
        s += (a[i + 1].X - a[i].X)*(a[i + 1].Y + a[i].Y);
}
s += (a[0].X - a[n-1].X)*(a[0].Y + a[n-1].Y);
```

## 6.4 Площадь треугольника

```
ld ans = (x2 - x1) * (y3 - y1) - (y2 - y1) * (x3 - x1);
labs(ans) / 2.0;
```

## 6.5 Расстояние от точки до прямой

a b c коэффициенты нормального уравнения прямой

```
ld ans = a*x + b * y + c;
ans /= sqrt(a*a + b*b);
```

## 6.6 Нормальное уравнение по двум точкам

```
int a = y1 - y2; int b = x2 - x1; int c = x1*y2 - x2*y1;
```

# 7 Числа Фибоначчи

## 7.1 Свойства чисел Фибоначчи

Соотношение Кассини:

$F_{n+1}F_{n-1} - F_n^2 = (-1)^n$.

Правило "сложения":

$F_{n+k} = F_kF_{n+1} + F_{k-1}F_n$.

Из предыдущего равенства при k = n вытекает:

$F_{2n} = F_n(F_{n+1} + F_{n-1})$.

Из предыдущего равенста по индукции можно получить, что

$F_{nk}$ всегда кратно $F_n$.

Верно и обратное к предыдущему утверждение:

если $F_m$ кратно $F_n$, то m кратно n.

НОД-равенство:

$\gcd(F_m, F_n) = F_{\gcd(m,n)}$.

Теорема Цекендорфа утверждает, что любое натуральное число n можно представить единственным образом в виде суммы чисел Фибоначчи:

$N = F_{k_1} + F_{k_2} + \ldots + F_{k_r}$

где $k_1 \geq k_2 + 2, k_2 \geq k_3 + 2, \ldots, k_r \geq 2$ (т.е. в записи нельзя использовать два соседних числа Фибоначчи).

Нетрудно получить и правило прибавления единицы к числу в фибоначчиевой системе счисления: если младшая цифра равна 0, то её заменяем на 1, а если равна 1 (т.е. в конце стоит 01), то 01 заменяем на 10. Затем "исправляем"запись, последовательно исправляя везде 011 на 100. В результате за линейное время будет получена запись нового числа.

Перевод числа в фибоначчиеву систему счисления осуществляется простым "жадным"алгоритмом: просто перебираем числа Фибоначчи от больших к меньшим и, если некоторое $F_k \leq n$, то $F_k$ входит в запись числа n, и мы отнимаем $F_k$ от n и продолжаем поиск.