Three Musketeers

IIT Bhubaneswar

Sarthak Gupta, Akshat Gupta, Arihant Garg

August 26, 2024

Contents

1   NumberTheory

1.1   extended_euclidean

```cpp
// Find a solution to ax + by = 1
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}

// find any solution to ax + by = c (g will store their
    gcd).
// Generalized x and y can be given by: x = x0 +
    r*lcm(a,b) and y = y0 - r*lcm(a,b)
bool find_any_solution(long long a, long long b, long
    long c, long long &x0, long long &y0, long long
    &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
            return false;
    }
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

1.2   factorizer

```cpp
#define mp make_pair
using i32 = int32_t;
using i64 = int64_t;
using i128 = __int128_t;
namespace factorizer {
constexpr i128 mult(i128 a, i128 b, i128 mod) {
    i128 ans = 0;
    while (b) {
        if (b & 1) {
            ans += a;
            if (ans >= mod) ans -= mod;
        }
        a <<= 1;
        if (a >= mod) a -= mod;
        b >>= 1;
    }
    return ans;
}
constexpr i64 mult(i64 a, i64 b, i64 mod) {
    return (i128)a * b % mod;
}
constexpr i32 mult(i32 a, i32 b, i32 mod) {
    return (i64)a * b % mod;
}

template <typename T>
constexpr T f(T x, T c, T mod) {
```

```cpp
    T ans = mult(x, x, mod) + c;
    if (ans >= mod) ans -= mod;
    return ans;
}

template <typename T>
constexpr T brent(T n, T x0 = 2, T c = 1) {
    T x = x0;
    T g = 1;
    T q = 1;
    T xs, y;

    int m = 128;
    int l = 1;
    while (g == 1) {
        y = x;
        for (int i = 1; i < l; i++) x = f(x, c, n);
        int k = 0;
        while (k < l && g == 1) {
            xs = x;
            for (int i = 0; i < m && i < l - k; i++) {
                x = f(x, c, n);
                q = mult(q, abs(y - x), n);
            }
            g = __gcd(q, n);
            k += m;
        }
        l *= 2;
    }
    if (g == n) {
        do {
            xs = f(xs, c, n);
            g = __gcd(abs(xs - y), n);
        } while (g == 1);
    }
    return g;
}

template <typename T>
T binpower(T base, T e, T mod) {
    T result = 1;
    base %= mod;
    while (e) {
        if (e & 1) result = mult(result, base, mod);
        base = mult(base, base, mod);
        e >>= 1;
    }
    return result;
}

template <typename T>
bool check_composite(T n, T a, T d, int s) {
    T x = binpower(a, d, n);
    if (x == 1 || x == n - 1) return false;
    for (int r = 1; r < s; r++) {
        x = mult(x, x, n);
        if (x == n - 1) return false;
    }
    return true;
};

template <typename T>
bool MillerRabin(T n, const vector<T>& bases) {
    // returns true if n is prime,
    // else returns false.

    if (n < 2) return false;
```

```cpp
    int r = 0;
    T d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (T a : bases) {
        if (n == a) return true;
        if (check_composite(n, a, d, r)) return false;
    }
    return true;
}

template <typename T>
bool IsPrime(T n, const vector<T>& bases) {
    if (n < 2) {
        return false;
    }
    vector<T> small_primes = {2, 3, 5, 7, 11, 13, 17,
        19, 23, 29};
    for (const auto& x : small_primes) {
        if (n % x == 0) {
            return n == x;
        }
    }
    if (n < 31 * 31) {
        return true;
    }

    return MillerRabin(n, bases);
}

bool IsPrime(i64 n) {
    return IsPrime(n, {2, 3, 5, 7, 11, 13, 17, 19, 23,
        29, 31, 37});
}

bool IsPrime(i32 n) { return IsPrime(n, {2, 7, 61}); }

template <typename T>
vector<pair<T, int>> MergeFactors(const vector<pair<T,
    int>>& a,
                                  const vector<pair<T,
                                      int>>& b) {
    vector<pair<T, int>> c;
    int i = 0;
    int j = 0;
    while (i < (int)a.size() || j < (int)b.size()) {
        if (i < (int)a.size() && j < (int)b.size() &&
            a[i].first == b[j].first) {
            c.emplace_back(a[i].first, a[i].second +
                b[j].second);
            ++i;
            ++j;
            continue;
        }
        if (j == (int)b.size() ||
            (i < (int)a.size() && a[i].first <
                b[j].first)) {
            c.push_back(a[i++]);
        } else {
            c.push_back(b[j++]);
        }
    }
    return c;
}
```

```cpp
}

template <typename T>
vector<pair<T, int>> RhoC(T n, T c) {
    if (n <= 1) return {};
    if (!(n & 1))
        return MergeFactors({mp(static_cast<T>(2), 1)},
            RhoC(n >> 1, c));
    if (IsPrime(n)) return {mp(n, 1)};
    T g = brent(n, static_cast<T>(2), c);
    return MergeFactors(RhoC(g, c + 1), RhoC(n / g, c +
        1));
}

template <typename T>
vector<pair<T, int>> Factorize(T n) {
    if (n <= 1) return {};
    return RhoC(n, static_cast<T>(1));
}
} // example factorizer::Factorize(35)
```

1.3 floor_sum

```cpp
// f(a,b,c,n) = \sigma ( (a*x + b)/c ) from x=0 to n in
//O(logn) where value is floor of the value.
 long long FloorSumAP(long long a, long long b,
  long long c, long long n){
  if(!a) return (b / c) * (n + 1);
  if(a >= c or b >= c) return ( ( n * (n + 1) ) / 2) *
      (a / c)
  + (n + 1) * (b / c) + FloorSumAP(a % c, b % c, c, n);
  long long m = (a * n + b) / c;
  return m * n - FloorSumAP(c, c - b - 1, a, m - 1);
}
```

1.4 cipolla_sqrt

```cpp
template<int p>
class cipolla_sqrt{
 private:
    static int const phim = p-2;
    static int const phi = p-1;
    static int const phihalf = phi>>1;
    static int const phalf = (p+1)>>1;
 public:
    static int pow(int a,int po=phim){
        int res = 1;
        for(;po;po>>=1,a=normalise(a*1ll*a))
        if(po&1){
            res=normalise(a*1ll*res);
        }
        return res;
    }
    static int normalise(ll x){
        if(x>=p){
            x-=p;if(x>=p)x%=p;return x;
        }
        return x;
    }
    static int get(int x){
        int a = 0,b;
        while(
            pow(b = normalise(a*1ll*a-x+p),
```

```cpp
                phihalf)==1
        )++a;
        int po = phalf;
        pair<int,int> v = {a,1}, res = {1,1};
        while(po){
            if(po&1){
                int temp = normalise(res.S*1ll*v.S);
                res.S = normalise((v.F*1ll*res.S)
                        +(res.F*1ll*v.S));
                res.F = normalise((v.F*1ll*res.F)
                        +(b*1ll*temp));
            }
            po>>=1;
            int temp = normalise(v.S*1ll*v.S);
            v.S = normalise((v.F*1ll*v.S)
                            +(v.F*1ll*v.S));
            v.F = normalise((v.F*1ll*v.F)
                            +(b*1ll*temp));
        }
        if(res.F==1)res.F = p - res.F;
        return res.F;
    }
};
```

2   Strings

2.1   manacher

```cpp
vector<vector<int>> d;
// d[0] contains half the length of max palindrome of
    odd length with centre at i
// d[1] contains half the length of max palindrome of
    even length with right centre at i

void manacher(string s){
    int n = s.size();
    d.resize(n,vector<int>(2));
    for (int t = 0; t < 2; t++) {
        int l = 0, r = -1, j;
        for (int i = 0; i < n; i++) {
            j = (i > r ) ? 1 :
                min(d[l+r-i+t][t],r-i+t) + 1;

            while (
            i + j - t < n && i - j >= 0
            && s[i + j - t] == s[i - j]) j++;

            d[i][t] = --j;
            if (i + j + t > r) {
            l = i - j; r = i + j - t;
            }
        }
    }
}
```

2.2   suffix_array

```cpp
// Time Complexity: O(nlogn)

void count_sort(vector<int> &p,vector<int> &c){ //
    Sorts values in p by keys in c i.e, if c[p[i]] <
    c[p[j]] then i appears before j in p.
    int n = p.size();
```

```cpp
    vector<int> cnt(n);
    for(auto x : c) cnt[x]++;
    vector<int> p_new(n),pos(n);
    pos[0] = 0;
    for(int i=1;i<n;++i) pos[i] = pos[i-1] + cnt[i-1];
    for(auto x : p){
        int i = c[x];
        p_new[pos[i]] = x;
        pos[i]++;
    }
    p = p_new;
}

// Returns sorted suffix_array of size (n+1) with first
    element = n (empty suffix).
vector<int> suffix_array(string s){
    s += '$';
    int n = s.size();
    vector<int> p(n),c(n); // p stores suffix array and
        c stroes its equivalence class
    {
        // k = 0 phase
        vector<pair<char,int>> a(n);
        for(int i=0;i<n;++i) a[i] = {s[i],i};
        sort(all(a));
        for(int i=0;i<n;++i) p[i] = a[i].S;
        c[p[0]] = 0;
        for(int i=1;i<n;++i){
            if(a[i].F == a[i-1].F) c[p[i]] = c[p[i-1]];
            else c[p[i]] = c[p[i-1]] + 1;
        }
    }
    int k=0;
    while((1<<k) < n ){ // k -> k + 1
        // We require to sort p by {c[i], c[i+(1<<k)]}
            value. So we use radix sort: Sort the
            second element of pair then count sort
            first element in a stable fashion.
        // Sort by second element: p[i] -= (1<<k). As
            p[i]'s are already sorted by c[i] values.
        for(int i = 0; i < n ; ++i) p[i] = (p[i] -
            (1<<k) + n )%n;
        count_sort(p,c);

        vector<int> c_new(n);
        c_new[p[0]] = 0;
        for(int i=1;i<n;++i){
            pii prev = {c[p[i-1]] , c[(p[i-1] +
                (1<<k))%n] } ;
            pii now = {c[p[i]] , c[(p[i] + (1<<k))%n] } ;
            if( now == prev) c_new[p[i]] = c_new[p[i-1]];
            else c_new[p[i]] = c_new[p[i-1]] + 1;
        }
        c = c_new;
        ++k;
    }
    return p;
}

vector<int> lcp_array(vector<int> &p,vector<int>
    &c,string s){
    int n = p.size();
    vector<int> lcp(n-1);
    int k=0;
    for(int i=0;i<n-1;++i){
        int pi = c[i];
```

```
        int j = p[pi-1];
        //lcp[i] = lcp[s[i...],s[j...]]
        while(s[i+k] == s[j+k]) ++k;
        lcp[pi-1] = k;
        k = max(k-1,0);
    }
    return lcp;
}
// Finds lcp using p and
// c array defined in suffix array
```

2.3 aho_corasick

```
const static int K = 26;
// Will change if input is not just lowercase alphabets
struct Vertex {
    int next[K];
    int leaf = 0;
    // It actually denotes number of leafs reachable
    // from current vertexes using links.
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};
vector<Vertex> t;
// Automation is stored in form of vector.

// Add String s to the automaton.
void add_s(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].leaf += 1;
}

// Forward declaration of functions
int go(int v, char ch);

// gets the link from vertex v.
int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    // May change if not lowercase alphabet
    if (t[v].go[c] == -1) {
```

```
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 :
                            go(get_link(v), ch);
    }
    return t[v].go[c];
}

// To calculate links and leafs(exit link) for nodes.
void bfs() {
    queue<int> order;
    order.push(0);
    while(!order.empty()) {
        int cur = order.front(); order.pop();
        t[cur].link = get_link(cur);
        t[cur].leaf += t[t[cur].link].leaf;
        for(int i=0;i<K;++i) {
            if(t[cur].next[i] != -1) {
                order.push(t[cur].next[i]);
            }
        }
    }
}
```

2.4 suffix_automaton

```
struct state {
    int len, link;
    map<char, int> next;
};

const int MAXLEN = 100000;
state st[MAXLEN * 2];
int sz, last;

void sa_init() {
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}

void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
```

```
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

3   Polynomials

3.1   poly_mono

```cpp
const int P1 = 880803841, G1 = 26;//(105*2^23)+1
const int P2 = 897581057, G2 = 3;//(107*2^23)+1
const int P3 = 998244353, G3 = 3;//(119*2^23)+1
const int primitive = 3;

#define u32       __uint32_t
#define u64       __uint64_t
#define vi        vector<int>
#define v64       vector<u64>
#define all(x)    x.begin(),x.end()
template<int mod>
constexpr int powmod(int a,int p = mod -2){
    int res = 1;
    while(p){
        if(p&1)res = (res*1ll*a)%mod;
        p>>=1;
        a = (a*1ll*a)%mod;
    }
    return res;
}
template<int max_base,int mod,int primitive>
class Ntt{
 private:
    constexpr static v64 fill(const int o){
        vector<u64> res(max_base);
        const int m = max_base>>1;
        res[m] = (1ll<<32)%mod;
        for(int i = m+1;i<max_base;++i)
            res[i] = reduce(res[i-1]*1ll*o);
        for(int i = m-1;i;--i) res[i] = res[i<<1];
        res[0] = (1ll<<32)%mod;
        return res;
    }
    constexpr static v64 init_omegas(){
        const int omega =
            (powmod<mod>(primitive,(mod-1)/
                max_base)*(1ll<<32))%mod;
        return fill(omega);
    }
    constexpr static v64 init_iomegas(){
        const int omega =
            powmod<mod>(primitive,(mod-1)/max_base);
        const int iomega =
            (powmod<mod>(omega)*(1ll<<32))%mod;
        return fill(iomega);
    }
    static const v64 omegas, iomegas;
    constexpr static u32 init_mod_inv(){
        u32 inv = mod;
        for(int i=0;i<4;++i)inv*=(2-(mod*inv));
        return -inv;
    }
    static const u32 mod_inv = init_mod_inv(),
        mod32 = mod;
```

```cpp
    static const u64 mod64 = mod;
public:
    // use to combine if using fft explicitly
    // see mul for details
    static const inline u32 reduce(u64 x){
        u32 m = static_cast<u32>(x)*mod_inv;
        u32 t = (x+m*mod64)>>32;
        if(t>=mod)t-=mod;
        return t;
    }
    static void fft(vi &a){
        int n = a.size();
        for(int m=n>>1;m;m>>=1){
            auto it_start = omegas.begin()+m;
            auto it_end = it_start+m;
            for(auto l = a.begin();l!=a.end();l+=m){
                for(auto it = it_start;
                    it!=it_end;++it,++l){
                    int e = *l-l[m];
                    if(e<0)e+=mod;
                    *l+=l[m];
                    if(*l>=mod)*l-=mod;
                    l[m] = reduce(e* *it);
                }
            }
        }
    }
    static void ifft(vi &a){
        int n = a.size();
        for(int m=1;m<n;m<<=1){
            auto it_start = iomegas.begin()+m;
            auto it_end = it_start+m;
            for(auto l = a.begin();l!=a.end();l+=m){
                for(auto it = it_start;
                    it!=it_end;++it,++l){
                    l[m] = reduce(l[m]* *it);
                    int e = *l-l[m];
                    if(e<0)e+=mod;
                    *l+=l[m];
                    if(*l>=mod)*l-=mod;
                    l[m] = e;
                }
            }
        }
        u64 f = (((1ll<<32)*omegas[0])/a.size())%mod;
        for(int i=0;i<a.size();++i)
            a[i] = reduce(a[i]*f);
    }
    static vi mul(vi a,vi b){
        int need = a.size()+b.size()-1;
        int nbase = 1<<(32-__builtin_clz(need-1));
        a.resize(nbase);b.resize(nbase);
        fft(a);fft(b);
        for(int i=0;i<nbase;++i)
            a[i]=reduce(a[i]*1ll*b[i]);
        ifft(a);
        a.resize(need);
        return a;
    }
    static vi inv(vi &a){
        int n = a.size(), k=1;
        vi res(1,powmod<mod>(a[0]));
        while(k<n){
            int l = k<<1;
            int need = l<<1;
            if(l>n)a.resize(l);
            res.resize(need);
```

```cpp
            vi temp(a.begin(),a.begin()+l);
            temp.resize(need);fft(res);fft(temp);
            for(int i=0;i<need;++i)
                res[i] = reduce(temp[i]*1ll*
                        reduce(res[i]*1ll*res[i]));
            ifft(res);
            for(int i=k;i<l;++i)
                if(res[i])res[i]=mod-res[i];
            k = l;
        }
        a.resize(n);res.resize(n);
        return res;
    }
};

template<int max_base,int mod,int primitive> const v64
    Ntt<max_base,mod,primitive>::omegas =
    init_omegas();
template<int max_base,int mod,int primitive> const v64
    Ntt<max_base,mod,primitive>::iomegas =
    init_iomegas();

const int mod = 998244353;
const int base = 1<<20;
vi& operator *= (vi& a,const vi& b){
    if(a.empty()||b.empty())a.clear();
    else a = Ntt<base,mod,primitive>::mul(a,b);
    return a;
}
vi operator * (const vi& a,const vi& b){
    vi c = a;return c*=b;
}
vi& operator /= (vi& a,const vi& b){
    if(a.size()<b.size())a.clear();
    else{
        vi d = b;
        reverse(d.begin(),d.end());
        reverse(a.begin(),a.end());
        int deg = a.size()-b.size();
        a.resize(deg+1);
        d.resize(deg+1);
        d = Ntt<base,mod,primitive>::inv(d);
        a*=d;a.resize(deg+1);
        reverse(a.begin(),a.end());
    }
    return a;
}
vi operator / (vi& a,const vi &b){
    vi c = a;return c/=b;
}
vi& operator += (vi& a,const vi& b){
    if(a.size()<b.size())a.resize(b.size());
    for(int i=0;i<b.size();++i){
        a[i]+=b[i];
        if(a[i]>=mod)a[i]-=mod;
    }
    return a;
}
vi operator + (const vi& a,const vi& b){
    vi c = a;return c+=b;
}
vi& operator -= (vi& a,const vi& b){
    if(a.size()<b.size())a.resize(b.size());
    for(int i=0;i<b.size();++i){
        a[i]-=b[i];
        if(a[i]<0)a[i]+=mod;
    }
}
```

```cpp
    return a;
}
vi operator - (const vi& a,const vi& b){
    vi c = a;
    return c-=b;
}
vi& operator %= (vi& a,const vi& b){
    if(a.size()<b.size())return a;
    vi c = (a/b)*b;
    a -= c;
    a.resize(b.size()-1);
    return a;
}
vi operator % (const vi& a,const vi& b){
    vi c = a;return c%=b;
}
```

3.2 poly_arbitrary

```cpp
typedef long long i64;
typedef complex<double> im;

const int o = 20, len = 1 << o, mod = 1e9 + 7;
const double pi = acos(-1.0);

namespace poly {
int r[len], up, l;
im w[len], iw[len], I(0, 1);

void init() {
  w[1] = iw[1] = im(1, 0);
  for (int l = 2; l != len; l <<= 1) {
    double x = cos(pi / l), y = sin(pi / l);
    im p(x, y), ip(x, -y);
    for (int i = 0; i != l; i += 2) {
      w[l + i] = w[(l + i) >> 1], iw[l + i] = iw[(l +
          i) >> 1];
      w[l + i + 1] = w[l + i] * p, iw[l + i + 1] = iw[l
          + i] * ip;
    }
  }
  for (int i = (len >> 1) - 1; i; i--)
    w[i] = w[i << 1], iw[i] = iw[i << 1];
  for (int i = 0; i != len; i++)
    r[i] = (r[i >> 1] >> 1) | ((i & 1) << (o - 1));
}

int pre(int n) {
  l = 32 - __builtin_clz(n), up = 1 << l;
  return up;
}

void fft(im *a, int n, bool op, im *w) {
  for (int i = 0; i != n; i++) {
    int t = r[i] >> (o - l);
    if (i < t)
      swap(a[i], a[t]);
  }
  for (int l = 1; l != n; l <<= 1) {
    im *k = w + l;
    for (im *f = a; f != a + n; f += l)
      for (im *j = k; j != k + l; j++, f++) {
        im x = *f, y = f[l] * *j;
        f[l] = x - y, *f += y;
      }
  }
}
```

```cpp
    }
  if (op)
    for (int i = 0; i != n; i++)
      a[i] /= n;
}

vector<int> mul(vector<int> &f, vector<int> &g) {
    int n = f.size()-1,m = g.size()-1;
  static im a[len], b[len], c[len], d[len];
  pre(n + m);
  int mm = sqrt(mod);
  for (int i = 0; i <= n; i++)
    a[i] = im(f[i] % mm, f[i] / mm);
  for (int i = 0; i <= m; i++)
    b[i] = im(g[i] % mm, g[i] / mm);
  fft(a, up, 0, w), fft(b, up, 0, w);
  for (int i = 0; i != up; i++) {
    a[i] /= 2, b[i] /= 2;
    c[i] = im(a[i].real(), -a[i].imag()),
    d[i] = im(b[i].real(), -b[i].imag());
  }
  reverse(c + 1, c + up), reverse(d + 1, d + up);
  for (int i = 0; i != up; i++) {
    im a0 = a[i] + c[i], a1 = (c[i] - a[i]) * I;
    im b0 = b[i] + d[i], b1 = (d[i] - b[i]) * I;
    a[i] = a0 * b0 + I * a1 * b1,
    b[i] = a0 * b1 + I * a1 * b0;
  }
  fft(a, up, 1, iw), fft(b, up, 1, iw);
  auto num = [](double x) {
    return (i64)round(x) % mod;
  };
  vector<int> h(n+m+1);
  for (int i = 0; i <= n + m; i++){
    h[i] = (num(a[i].real())
              + num(a[i].imag()) * mm * mm
              + (num(b[i].real())
              + num(b[i].imag())) * mm) % mod;
  }
  for (int i = 0; i != up; i++)
    a[i] = b[i] = c[i] = d[i] = 0;
  return h;
}
}
// namespace poly please call poly::init()
// and set o,len,mod accordingly
```

3.3   poly_bitwise

```cpp
void muland(vector<int> &a, int inv) {
    int n = a.size();
    for(int len=2,hlen=1;len<=n;hlen<<=1,len<<=1) {
        for(int i=0;i<n;i+=len) {
            for(int j=0;j<hlen;++j) {
                int str = a[i+j];
                if(!inv) {
                    a[i+j] = a[i+j+hlen];
                    a[i+j+hlen] = a[i+j+hlen]+str;
                } else {
                    a[i+j] = -str+a[i+j+hlen];
                    a[i+j+hlen] = str;
                }
            }
        }
    }
}
```

```cpp
}

void mulor(vector<int> &a, int inv) {
    int n = a.size();
    for(int len=2,hlen=1;len<=n;hlen<<=1,len<<=1) {
        for(int i=0;i<n;i+=len) {
            for(int j=0;j<hlen;++j) {
                int str = a[i+j];
                if(!inv) {
                    a[i+j] += a[i+j+hlen];
                    a[i+j+hlen] = str;
                } else {
                    a[i+j] = a[i+j+hlen];
                    a[i+j+hlen] = str-a[i+j+hlen];
                }
            }
        }
    }
}

// f[j] = sum_{0}^{n} a[i]*(-1^(bc(i&j)))
// where bc is bitcount
void mulxor(vector<int> &a, int inv) {
    int n = a.size();
    for(int len=2,hlen=1;len<=n;hlen<<=1,len<<=1) {
        for(int i=0;i<n;i+=len) {
            for(int j=0;j<hlen;++j) {
                int str = a[i+j];
                a[i+j] = a[i+j] + a[i+j+hlen];
                a[i+j+hlen] = str - a[i+j+hlen];
            }
        }
    }
    if(inv) for(int i=0;i<n;++i) a[i]/=n;
}

vector<int> Multiand(vector<int> &a, vector<int> &b) {
    int n = max(b.size(), a.size());
    int i=1;
    while(n > i) {
        i*=2;
    } n = i;
    while(a.size()!=n) a.pb(0);
    while(b.size()!=n) b.pb(0);

    muland(a,0); muland(b,0);
    vector<int> ans;
    for(int i=0;i<a.size();++i)
      ans.pb(a[i]*b[i]);
    muland(ans,1);
    return ans;
}
```

4   Trees

4.1   centroid_decomposition

```cpp
const int N=200005; // Change based on constraint
set<int> G[N]; // adjacency list (note that this is
    stored in set, not vector)
int sz[N], pa[N];

int dfs(int u, int p) {
  sz[u] = 1;
```

```cpp
    for(auto v : G[u]) if(v != p) {
      sz[u] += dfs(v, u);
    }
    return sz[u];
}
int centroid(int u, int p, int n) {
    for(auto v : G[u]) if(v != p) {
      if(sz[v] > n / 2) return centroid(v, u, n);
    }
    return u;
}
void build(int u, int p) {
    int n = dfs(u, p);
    int c = centroid(u, p, n);
    if(p == -1) p = c;
    pa[c] = p;

    vector<int> tmp(G[c].begin(), G[c].end());
    for(auto v : tmp) {
      G[c].erase(v); G[v].erase(c);
      build(v, c);
    }
}
```

4.2 top_tree

```cpp
// should have identity transform
struct Transform{
    long long val;
    Transform():val(0ll){}
    Transform(long long _val):val(_val){}
    Transform& operator +=(Transform &other){
        val+=other.val;
        return *this;
    }
    bool isLazy()const{return val;}
};
// transforming values and summing = summing and
    transforming
// sum is commutative and associative
// transforming identity = identity
struct Val{
    int n;
    long long val;
    Val(int _n, long long _val):n(_n),val(_val){}
    Val(long long _val):n(1),val(_val){}
    Val():n(0),val(0){}
    Val operator +(Val &other)const{
        return Val(n+other.n, val+other.val);};
    Val& operator +=(Transform
        &T){val+=(n*T.val);return *this;}
    bool isIdentity()const{return n==0;}
};

template<typename val, typename transform>
class TopTree{
public:
    struct Splay{
        struct node{
            int l,r,ar,p;
            bool flip;
            val self, path, sub, all;
            transform lazyPath, lazySub;
            node():
                l(0),r(0),ar(0),p(0),flip(false){}
```

```cpp
            node(int _val):
                l(0),r(0),ar(0),p(0),flip(false),
                self(_val), path(_val), all(_val){}
        };
        int stx;
        vector<node> nodes;
        Splay(int n,int q){
            nodes.assign(n+q+1,node(0));
            nodes[0] = node();
            for(int i=n+1;i<nodes.size();++i)
                nodes[i] = nodes[0];
            stx = n;
        }
        inline void lazyApplyPath(int u, transform &T){
            if(!nodes[u].path.isIdentity()){
                nodes[u].self+=T,nodes[u].path+=T,
                nodes[u].lazyPath+=T;
                nodes[u].all =
                    nodes[u].path+nodes[u].sub;
            }
        }
        inline void lazyApplySub(int u, transform &T){
            if(!nodes[u].sub.isIdentity()){
                nodes[u].sub+=T,nodes[u].lazySub+=T;
                nodes[u].all =
                    nodes[u].path+nodes[u].sub;
            }
        }
        inline void flip(int u){
            swap(nodes[u].l,nodes[u].r);
            nodes[u].flip^=1;
        }
        inline void push(int u){
            if(nodes[u].lazyPath.isLazy()){
                lazyApplyPath(nodes[u].l,
                    nodes[u].lazyPath),
                lazyApplyPath(nodes[u].r,
                    nodes[u].lazyPath);
                nodes[u].lazyPath = transform();
            }
            if(nodes[u].lazySub.isLazy()){
                lazyApplySub(nodes[u].l,
                    nodes[u].lazySub),
                lazyApplySub(nodes[u].r,
                    nodes[u].lazySub),
                lazyApplySub(nodes[u].ar,
                    nodes[u].lazySub),
                lazyApplyPath(nodes[u].ar,
                    nodes[u].lazySub);
                nodes[u].lazySub = transform();
            }
            if(nodes[u].flip){
                nodes[u].flip = false;
                flip(nodes[u].l);
                flip(nodes[u].r);
            }
        }
        inline void pull(int u){
            if(!u)return;
            int lc = nodes[u].l, rc = nodes[u].r,
                    ar = nodes[u].ar;
            nodes[u].path = nodes[lc].path
                        +nodes[u].self
                        +nodes[rc].path;
            nodes[u].sub = nodes[lc].sub
                        +nodes[rc].sub
                        +nodes[ar].all;
```

```cpp
        nodes[u].all = nodes[u].path+nodes[u].sub;
    }
    inline void rotate(int u){
        int p = nodes[u].p;
        if(nodes[p].r==u){
            nodes[p].r = nodes[u].l;
            if(nodes[u].l)
                nodes[nodes[u].l].p = p;
            nodes[u].l = p;
        }
        else{
            nodes[p].l = nodes[u].r;
            if(nodes[u].r)
                nodes[nodes[u].r].p = p;
            nodes[u].r = p;
        }
        nodes[u].p = nodes[p].p;
        nodes[p].p = u;
        if(nodes[nodes[u].p].l == p)
            nodes[nodes[u].p].l = u;
        else if(nodes[nodes[u].p].r == p)
            nodes[nodes[u].p].r = u;
        else if(nodes[nodes[u].p].ar == p)
            nodes[nodes[u].p].ar = u;
    }
    inline void splay(int x){
        while((nodes[nodes[x].p].l==x)||
                (nodes[nodes[x].p].r==x)){
            int y = nodes[x].p;
            int z = nodes[y].p;
            if((nodes[z].l==y)||(nodes[z].r==y)){
                push(z);push(y);push(x);
                if(((nodes[z].l==y)&&(
                    nodes[y].l==x))||
                    ((nodes[z].r==y)&&(
                        nodes[y].r==x)))
                    rotate(y);
                else
                    rotate(x);
                rotate(x);
                pull(z);pull(y);pull(x);
            }
            else{
                push(y);push(x);
                rotate(x);
                pull(y);pull(x);
            }
        }
        push(x);
    }
    inline void detach(int u){
        push(u);
        if(nodes[u].r){
            if(nodes[nodes[u].ar].ar
                    ||(!nodes[u].ar)){
                nodes[++stx].r = nodes[u].ar;
                nodes[stx].p = u;
                if(nodes[stx].r)
                    nodes[nodes[stx].r].p = stx;
                nodes[u].ar = stx;
            }
            else
                push(nodes[u].ar);
            nodes[nodes[u].ar].ar = nodes[u].r;
            nodes[nodes[u].r].p = nodes[u].ar;
            nodes[u].r = 0;
            pull(nodes[u].ar);
```

```cpp
            pull(u);
        }
    }
    inline int access(int u){
        int x = u;
        int v = u;
        while(x){
            splay(x);
            if(u!=x){
                push(nodes[x].ar);
                swap(nodes[x].r,
                    nodes[nodes[x].ar].ar);
                if(nodes[x].r)
                    nodes[nodes[x].r].p = x;
                if(nodes[nodes[x].ar].ar)
                    nodes[nodes[nodes[x].ar].ar].p =
                        nodes[x].ar;
                pull(nodes[x].ar);
                pull(x);
            }
            else
                detach(x);
            v = x;
            x = nodes[x].p;
            if(x){
                splay(x);
                x = nodes[x].p;
            }
        }
        splay(u);
        return v;
    }
    void root(int x){
        access(x);flip(x);push(x);
    }
};
Splay S;
int root;
TopTree(int _n, int _q,int
    _root):S(_n,_q*2),root(_root){}
void updateSub(int x,transform T){
    S.root(root);S.access(x);
    int y = S.nodes[x].l;
    S.nodes[x].l = 0;
    S.pull(x);
    S.lazyApplyPath(x, T),S.lazyApplySub(x, T);
    S.push(x);S.nodes[x].l = y;S.pull(x);
}
void updatePath(int x,int y,transform T){
    S.root(x);S.access(y);S.lazyApplyPath(y, T);
}
void reroot(int r){root = r;}
val getPath(int x,int y){
    S.root(x);S.access(y);
    return S.nodes[y].path;
}
val getSub(int x){
    S.root(root);S.access(x);
    return S.nodes[x].self
        +S.nodes[S.nodes[x].r].path
        +S.nodes[S.nodes[x].ar].all;
}
void link(int x,int y){
    S.root(x);S.access(y);
    S.nodes[y].r = x,S.nodes[x].p = y;
    S.pull(y);
}
```

```cpp
    int lca(int x,int y){
        S.root(root);S.access(y);
        return S.access(x);
    }
    void changePar(int x,int y){
        if(lca(x,y)!=x){
            S.nodes[S.nodes[x].l].p = 0;
            S.nodes[x].l = 0;S.pull(x);
            link(x,y);
        }
    }
    void cut(int u){
        S.root(root);S.access(u);
        S.nodes[S.nodes[u].l].p = 0;
        S.nodes[u].l = 0;S.pull(u);
    }
};
```

4.3   hld

```cpp
#include<bits/stdc++.h>
using namespace std;
class HLD {
    vector<int> segtree, parent, depth, heavy, head,
        pos, maxdepth;
    int cur_pos;

    int combine(int lval, int rval) {
        return max(lval, rval);
    }

    void segtree_update(int v, int l, int r, int x, int
        val) {
        if (x < l || r < x)return;
        if (l == r) {
            segtree[v] = val;
            return;
        }
        int mid = (l + r) >> 1;
        int z = (mid + 1 - l) << 1;
        if (x <= mid)
            segtree_update(v + 1, l, mid, x, val);
        else
            segtree_update(v + z, mid + 1, r, x, val);

        segtree[v] = combine(segtree[v + 1], segtree[v
            + z]);
    }

    int segtree_query(int v, int l, int r, int L, int
        R) {
        if (R < l || r < L)return 0;
        if (L <= l && r <= R) return segtree[v];
        int mid = (l + r) >> 1;
        int z = (mid + 1 - l) << 1;
        return combine(segtree_query(v + 1, l, mid, L,
            R), segtree_query(v + z, mid + 1, r, L, R));
    }

    // HLD Template
    // Taken from cp-algotithms


    int dfs(int v, vector<vector<int>> const& adj) {
```

```cpp
        int size = 1;
        int max_c_size = 0;
        maxdepth[v] = depth[v];
        for (int c : adj[v]) {
            if (c != parent[v]) {
                parent[c] = v, depth[c] = depth[v] + 1;
                int c_size = dfs(c, adj);
                size += c_size;
                if (c_size > max_c_size) {
                    max_c_size = c_size,
                        heavy[v] = c, maxdepth[v] =
                            maxdepth[c];
                }
            }
        }
        return size;
    }

    void decompose(int v, int h, vector<vector<int>>
        const& adj) {
        head[v] = h, pos[v] = cur_pos++;    // Now v is
            present at pos[v]
        if (heavy[v] != -1)
            decompose(heavy[v], h, adj);
        for (int c : adj[v]) {
            if (c != parent[v] && c != heavy[v])
                decompose(c, c, adj);
        }
    }

public:
    HLD(vector<vector<int>> const& adj) {
        int n = adj.size();
        parent = depth = head = pos = maxdepth =
            vector<int>(n);
        heavy = vector<int>(n, -1);
        segtree = vector<int>(n << 1);
        cur_pos = 0;

        dfs(0, adj);
        decompose(0, 0, adj);
    }

    int query(int a, int b) {
        int res = 0;
        for (; head[a] != head[b]; b = parent[head[b]])
            {

            if (depth[head[a]] > depth[head[b]])
                swap(a, b);
            int cur_heavy_path_res =
                segtree_query(pos[head[b]] << 1,
                depth[head[b]], maxdepth[b],
                depth[head[b]], depth[b]);
            res = combine(res, cur_heavy_path_res);
        }

        if (depth[a] > depth[b])
            swap(a, b);
        int last_heavy_path_max =
            segtree_query(pos[head[b]] << 1,
            depth[head[b]], maxdepth[b], depth[a],
            depth[b]);
        res = combine(res, last_heavy_path_max);
        return res;
    }
```

```
        void update(int x, int val) { // 0 based indexing
            segtree_update(pos[head[x]] << 1,
                depth[head[x]], maxdepth[x], depth[x], val);
        }
    };
```

5   Graphs

5.1   max_flow

```
/*
Implementation of Dinic's blocking algorithm
for the maximum flow.
Complexity: V^2 E (faster on real graphs).

please add edges not related to input first
to improve constants

This class accepts a graph
(costructed calling AddEdge) and then solves
the maximum flow problem for any source and sink.

Both directed and undirected graphs are supported.
In case of undirected graphs,
each edge must be added twice.

To compute the maximum flow just call
GetMaxFlowValue(source, sink).
*/

template <typename T>
struct Dinic {
    struct Edge {
        int u, v;
        T cap, flow;
        Edge() {}
        Edge(int u, int v, T cap): u(u), v(v),
            cap(cap), flow(0) {}
    };

    int N;
    vector<Edge> edges; // The "inverse" edge of
        edges[i] is edges[i^1].
    vector<vector<int>> aa; // Stores the index of the
        edge in the edges vector.
    // dist is the distance in the bfs.
    // pt is used internally to save time in the dfs.
    vector<int> dist, pt;

    Dinic(int N): N(N), edges(0), aa(N), dist(N), pt(N)
        {}

    void AddEdge(int u, int v, T cap) {
        assert(0 <= u and u < N);
        assert(0 <= v and v < N);
        // dbg(u, v, cap);
        if (u != v) {
            edges.push_back(Edge(u, v, cap));
            aa[u].push_back(edges.size() - 1);
            // The inverse edge has 0 capacity.
            edges.push_back(Edge(v, u, 0));
            aa[v].push_back(edges.size() - 1);
        }
    }
```

```
    // Computes all distances from source and stores
        them in dist.
    // It returns true if sink is reachable from source.
    bool BFS(int source, int sink) {
        queue<int> q({source});
        fill(dist.begin(), dist.end(), N + 1);
        dist[source] = 0;
        while (!q.empty()) {
            int u = q.front(); q.pop();
            if (u == sink) break;
            for (int k : aa[u]) {
                Edge &e = edges[k];
                if (e.flow < e.cap && dist[e.v] >
                    dist[e.u] + 1) {
                    dist[e.v] = dist[e.u] + 1;
                    q.push(e.v);
                }
            }
        }
        return dist[sink] != N + 1;
    }

    T DFS(int u, int sink, T flow = -1) {
        if (u == sink || flow == 0) return flow;
        // ACHTUNG: Be careful of using references (&)
            where needed!
        for (int &i = pt[u]; i < (int)aa[u].size();
            i++) {
            Edge &e = edges[aa[u][i]];
            Edge &oe = edges[aa[u][i] ^ 1];
            if (dist[e.v] == dist[e.u] + 1) {
                T amt = e.cap - e.flow;
                if (flow != -1 && amt > flow) amt = flow;
                if (T pushed = DFS(e.v, sink, amt)) {
                    e.flow += pushed;
                    oe.flow -= pushed;
                    return pushed;
                }
            }
        }
        return 0;
    }

    T GetMaxFlowValue(int source, int sink) {
        for (Edge& e : edges) e.flow = 0;
        T res = 0;
        while (BFS(source, sink)) {
            fill(pt.begin(), pt.end(), 0);
            while (T flow = DFS(source, sink)) res +=
                flow;
        }
        return res;
    }
};
```

5.2   edmond_blossom_unweighted

```
struct edmond {
    int n, m , nE, n_matches,q_n,book_mark;
    vector<int> adj,nxt,go,mate,q,book,type,fa,bel;

    edmond(int n,int m):n(n),m(m){
        nE=0;
        n_matches=0;
```

```cpp
        adj.resize(n+1);
        mate.resize(n+1);
        q.resize(n+1);
        book.resize(n+1);
        type.resize(n+1);
        fa.resize(n+1);
        bel.resize(n+1);
        go.resize((m<<1)|1);
        nxt.resize((m<<1)|1);
    }

    void addEdge(const int &u, const int &v) {
        nxt[++nE] = adj[u], go[adj[u] = nE] = v;
        nxt[++nE] = adj[v], go[adj[v] = nE] = u;
    }

    void augment(int u) {
        while (u) {
            int nu = mate[fa[u]];
            mate[mate[u] = fa[u]] = u;
            u = nu;
        }
    }

    int get_lca(int u, int v) {
        ++book_mark;
        while (true) {
            if (u) {
                if (book[u] == book_mark) return u;
                book[u] = book_mark;
                u = bel[fa[mate[u]]];
            }
            swap(u, v);
        }
    }

    void go_up(int u, int v, const int &mv) {
        while (bel[u] != mv) {
            fa[u] = v;
            v = mate[u];
            if (type[v] == 1) type[q[++q_n] = v] = 0;
            bel[u] = bel[v] = mv;
            u = fa[v];
        }
    }
    void after_go_up() {
        for (int u = 1; u <= n; ++u) bel[u] =
            bel[bel[u]];
    }

    bool match(const int &sv) {
        for (int u = 1; u <= n; ++u) bel[u] = u,
            type[u] = -1;
        type[q[q_n = 1] = sv] = 0;
        for (int i = 1; i <= q_n; ++i) {
            int u = q[i];
            for (int e = adj[u]; e; e = nxt[e]) {
                int v = go[e];
                if (!~type[v]) {
                    fa[v] = u, type[v] = 1;
                    int nu = mate[v];
                    if (!nu) {
                        augment(v);
                        return true;
                    }
                    type[q[++q_n] = nu] = 0;
```

```cpp
                } else if (!type[v] && bel[u] != bel[v])
                    {
                    int lca = get_lca(u, v);
                    go_up(u, v, lca);
                    go_up(v, u, lca);
                    after_go_up();
                }
            }
        }
        return false;
    }

    void calc_max_match() {
        n_matches = 0;
        for (int u = 1; u <= n; ++u)
            if (!mate[u] && match(u)) ++n_matches;
    }
};

/*
int main() {
    int n,m;
    cin >> n >> m;
    edmond er(n, m);
    while(m--) {
        int x,y; cin >> x >> y;
        er.addEdge(x+1, y+1); // Input should be
            strictly 1-based indexed node.
    }
    er.calc_max_match();
    cout << er.n_matches << endl;
    for(int u = 1;u <= er.n; ++u)
        if(er.mate[u] > u) cout << er.mate[u]-1 << ' '
            << u-1 << '\n';
    return 0;
}
*/
```

5.3 edmond_blossom_weighted

```cpp
struct Blossom {
    const long long inf = 1e18;
    static const int N = 105; // > Max number of
        vertices.
    struct edge {
        int u, v;
        long long w;
    } g[N * 2][N * 2];
    int n, n_x, match[N * 2], slack[N * 2], st[N * 2],
        pa[N * 2], flower_from[N * 2][N * 2], S[N * 2],
        vis[N * 2];
    long long lab[N * 2];
    long long dist(edge const& e) { return lab[e.u] +
        lab[e.v] - g[e.u][e.v].w * 2; }
    vector<int> flower[N * 2];
    deque<int> q;
    Blossom(int _n) {
        n = _n;
        q = deque<int>();
        for (int u = 1; u <= n * 2; ++u) {
            match[u] = slack[u] = st[u] = pa[u] = S[u] =
                vis[u] = lab[u] = 0;
            for (int v = 1; v <= n * 2; ++v) {
                g[u][v] = edge{u, v, 0};
                flower_from[u][v] = 0;
```

```
            }
            flower[u].clear();
        }
    }
    void add_edge(int u, int v, long long w) {
        ++u; ++v;
        g[u][v].w = max(g[u][v].w, w);
        g[v][u].w = max(g[v][u].w, w);
    }
    inline void update_slack(int u, int x) {
        if (!slack[x] || dist(g[u][x]) <
            dist(g[slack[x]][x])) slack[x] = u;
    }
    inline void set_slack(int x) {
        slack[x] = 0;
        for (int u = 1; u <= n; ++u) {
            if (g[u][x].w > 0 && st[u] != x && S[st[u]]
                == 0) update_slack(u, x);
        }
    }
    inline void q_push(int x) {
        if (x <= n) return q.push_back(x);
        for (int i = 0; i < (int)flower[x].size(); i++)
            q_push(flower[x][i]);
    }
    inline void set_st(int x, int b) {
        st[x] = b;
        if (x <= n) return;
        for (int i = 0; i < (int)flower[x].size(); ++i)
            set_st(flower[x][i], b);
    }
    inline int get_pr(int b, int xr) {
        int pr = find(flower[b].begin(),
            flower[b].end(), xr) - flower[b].begin();
        if (pr % 2 == 1) {
            reverse(flower[b].begin() + 1,
                flower[b].end());
            return (int)flower[b].size() - pr;
        } else return pr;
    }
    inline void set_match(int u, int v) {
        match[u] = g[u][v].v;
        if (u <= n) return;
        edge e = g[u][v];
        int xr = flower_from[u][e.u], pr = get_pr(u,
            xr);
        for (int i = 0; i < pr; ++i)
            set_match(flower[u][i], flower[u][i ^ 1]);
        set_match(xr, v);
        rotate(flower[u].begin(), flower[u].begin() +
            pr, flower[u].end());
    }
    inline void augment(int u, int v) {
        int xnv = st[match[u]];
        set_match(u, v);
        if (!xnv) return;
        set_match(xnv, st[pa[xnv]]);
        augment(st[pa[xnv]], xnv);
    }
    inline int get_lca(int u, int v) {
        static int t = 0;
        for (++t; u || v; swap(u, v)) {
            if (u == 0) continue;
            if (vis[u] == t) return u;
            vis[u] = t;
            u = st[match[u]];
            if (u) u = st[pa[u]];
        }
    }
        }
        return 0;
    }
    inline void add_blossom(int u, int lca, int v) {
        int b = n + 1;
        while (b <= n_x && st[b]) ++b;
        if (b > n_x) ++n_x;
        lab[b] = 0, S[b] = 0;
        match[b] = match[lca];
        flower[b].clear();
        flower[b].push_back(lca);
        for (int x = u, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
                flower[b].push_back(y = st[match[x]]),
                q_push(y);
        }
        reverse(flower[b].begin() + 1, flower[b].end());
        for (int x = v, y; x != lca; x = st[pa[y]]) {
            flower[b].push_back(x),
                flower[b].push_back(y = st[match[x]]),
                q_push(y);
        }
        set_st(b, b);
        for (int x = 1; x <= n_x; ++x) g[b][x].w =
            g[x][b].w = 0;
        for (int x = 1; x <= n; ++x) flower_from[b][x]
            = 0;
        for (int i = 0; i < (int)flower[b].size(); ++i)
            {
            int xs = flower[b][i];
            for (int x = 1; x <= n_x; ++x) {
                if (g[b][x].w == 0 || dist(g[xs][x]) <
                    dist(g[b][x]))
                    g[b][x] = g[xs][x], g[x][b] =
                        g[x][xs];
            }
            for (int x = 1; x <= n; ++x) {
                if (flower_from[xs][x])
                    flower_from[b][x] = xs;
            }
        }
        set_slack(b);
    }
    inline void expand_blossom(int b) { // S[b] == 1
        for (int i = 0; i < (int)flower[b].size(); ++i)
            set_st(flower[b][i], flower[b][i]);
        int xr = flower_from[b][g[b][pa[b]].u], pr =
            get_pr(b, xr);
        for (int i = 0; i < pr; i += 2) {
            int xs = flower[b][i], xns = flower[b][i +
                1];
            pa[xs] = g[xns][xs].u;
            S[xs] = 1, S[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        S[xr] = 1, pa[xr] = pa[b];
        for (int i = pr + 1; i < (int)flower[b].size();
            ++i) {
            int xs = flower[b][i];
            S[xs] = -1, set_slack(xs);
        }
        st[b] = 0;
    }
    inline bool on_found_edge(const edge &e) {
        int u = st[e.u], v = st[e.v];
        if (S[v] == -1) {
```

```cpp
                pa[v] = e.u, S[v] = 1;
                int nu = st[match[v]];
                slack[v] = slack[nu] = 0;
                S[nu] = 0, q_push(nu);
            } else if (S[v] == 0) {
                int lca = get_lca(u, v);
                if (!lca) return augment(u, v), augment(v,
                    u), 1;
                else add_blossom(u, lca, v);
            }
        }
        return 0;
    }
    inline bool matching() {
        fill(S, S + n_x + 1, -1), fill(slack, slack +
            n_x + 1, 0);
        q.clear();
        for (int x = 1; x <= n_x; ++x) {
            if (st[x] == x && !match[x]) pa[x] = 0, S[x]
                = 0, q_push(x);
        }
        if (q.empty()) return 0;
        for (;;) {
            while ((int)q.size()) {
                int u = q.front();
                q.pop_front();
                if (S[st[u]] == 1) continue;
                for (int v = 1; v <= n; ++v) {
                    if (g[u][v].w > 0 && st[u] != st[v]) {
                        if (dist(g[u][v]) == 0) {
                            if (on_found_edge(g[u][v]))
                                return 1;
                        } else update_slack(u, st[v]);
                    }
                }
            }
            long long d = inf;
            for (int b = n + 1; b <= n_x; ++b) {
                if (st[b] == b && S[b] == 1) d = min(d,
                    lab[b] / 2);
            }
            for (int x = 1; x <= n_x; ++x) {
                if (st[x] == x && slack[x]) {
                    if (S[x] == -1) d = min(d,
                        dist(g[slack[x]][x]));
                    else if (S[x] == 0) d = min(d,
                        dist(g[slack[x]][x]) / 2);
                }
            }
            for (int u = 1; u <= n; ++u) {
                if (S[st[u]] == 0) {
                    if (lab[u] <= d) return 0;
                    lab[u] -= d;
                } else if (S[st[u]] == 1) lab[u] += d;
            }
            for (int b = n + 1; b <= n_x; ++b) {
                if (st[b] == b) {
                    if (S[st[b]] == 0) lab[b] += d * 2;
                    else if (S[st[b]] == 1) lab[b] -= d *
                        2;
                }
            }
            q.clear();
            for (int x = 1; x <= n_x; ++x) {
                if (st[x] == x && slack[x] &&
                    st[slack[x]] != x &&
                    dist(g[slack[x]][x]) == 0)
```

```cpp
                    if (on_found_edge(g[slack[x]][x]))
                        return 1;
                }
                for (int b = n + 1; b <= n_x; ++b) {
                    if (st[b] == b && S[b] == 1 && lab[b] ==
                        0) expand_blossom(b);
                }
            }
        }
        return 0;
    }
    pair<long long, int> solve() {
        fill(match, match + n + 1, 0);
        n_x = n;
        int cnt = 0;
        long long ans = 0;
        for (int u = 0; u <= n; ++u) st[u] = u,
            flower[u].clear();
        long long w_max = 0;
        for (int u = 1; u <= n; ++u) {
            for (int v = 1; v <= n; ++v) {
                flower_from[u][v] = (u == v ? u : 0);
                w_max = max(w_max, g[u][v].w);
            }
        }
        for (int u = 1; u <= n; ++u) lab[u] = w_max;
        while (matching()) ++cnt;
        for (int u = 1; u <= n; ++u) {
            if (match[u] && match[u] < u) ans +=
                g[u][match[u]].w;
        }
        for (int i = 0; i < n; ++i)
            match[i] = match[i + 1] - 1;
        return make_pair(ans, cnt);
    }
};

/*
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    Blossom g(n);
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        g.add_edge(u-1, v-1, w);
        // pass 0-based index here.
    }
    auto ans = g.solve();
    // {max_weight, matching size}
    cout << ans.second << ' ' << ans.first << '\n';
    for (int i = 0; i < n; ++i) {
        if (g.match[i] > i) cout << i+1 << ' ' <<
            g.match[i]+1 << '\n'; // print 1-based
            index.
    }
}
*/
```

5.4   minimum_directed_spanning_tree

```cpp
struct E { int s, t; ll w; }; // 0-base
struct PQ {
  struct P {
```

```
  ll v; int i;
  bool operator>(const P &b) const { return v > b.v; }
};
priority_queue<P, vector<P>, greater<>> pq; ll tag;
    // min heap
void push(P p) { p.v -= tag; pq.emplace(p); }
P top() { P p = pq.top(); p.v += tag; return p; }
void join(PQ &b) {
  if (pq.size() < b.pq.size())
    swap(pq, b.pq), swap(tag, b.tag);
  while (!b.pq.empty()) push(b.top()), b.pq.pop();
}
};
vector<int> dmst(const vector<E> &e, int n, int root) {
  vector<PQ> h(n * 2);
  for (int i = 0; i < int(e.size()); ++i)
    h[e[i].t].push({e[i].w, i});
  vector<int> a(n * 2); iota(all(a), 0);
  vector<int> v(n * 2, -1), pa(n * 2, -1), r(n * 2);
  auto o = [&](auto Y, int x) -> int {
    return x==a[x] ? x : a[x] = Y(Y, a[x]); };
  auto S = [&](int i) { return o(o, e[i].s); };
  int pc = v[root] = n;
  for (int i = 0; i < n; ++i) if (v[i] == -1)
    for (int p = i; v[p]<0 || v[p]==i; p = S(r[p])) {
      if (v[p] == i)
        for (int q = pc++; p != q; p = S(r[p])) {
          h[p].tag -= h[p].top().v; h[q].join(h[p]);
          pa[p] = a[p] = q;
        }
      while (S(h[p].top().i) == p) h[p].pq.pop();
      v[p] = i; r[p] = h[p].top().i;
    }
  vector<int> ans;
  for (int i = pc - 1; i >= 0; i--) if (v[i] != n) {
    for (int f = e[r[i]].t; f!=-1 && v[f]!=n; f = pa[f])
      v[f] = n;
    ans.push_back(r[i]);
  }
  return ans; // default minimize, returns edgeid array
}
// return ids of edges in mdst
```

5.5 directed_eulerian_cycle

```
class DirectedEulerianCircuit{
    int n;
    vector<stack<int>> adj;
    int get(int u){
        return adj[u].top();
    }
public:
    DirectedEulerianCircuit(int _n):n(_n){
        adj.resize(n);
        for(int i=0;i<n;++i)
            adj[i].push(-1);
    }
    void addEdge(int u,int v){
        adj[u].push(v);
    }
    int findFirst(){
        int u = 0;
        for(int i=0;i<n;++i){
            if(adj[i].size()==1)continue;
            u = i;
```

```
            if(!(adj[i].size()&1))break;
        }
        return u;
    }
    vector<int> eulerCircuit(){
        stack<int> st;
        int u = 0;
        for(int i=0;i<n;++i){
            if(adj[i].size())u = i;
        }
        vector<int> circuit;
        st.push(u);
        while(!st.empty()){
            int x = get(st.top());
            if(x!=-1){
                adj[st.top()].pop();
                st.push(x);
            }
            else{
                circuit.push_back(st.top());
                st.pop();
            }
        }
        reverse(circuit.begin(),circuit.end());
        return circuit;
    }
};
```

5.6 eulerian_cycle

```
class EulerianCircuit{
    int n,e;

    vector<vector<int>> adj;
    vector<bool> visit;
    vector<int> edges;
    int get(int u){
        while(visit[adj[u].back()])adj[u].pop_back();
        return adj[u].back();
    }
public:
    EulerianCircuit(int _n):n(_n),e(0){
        adj.resize(n);
        for(int i=0;i<n;++i)
            adj[i].push_back(0);
        edges.resize(2);
        visit.resize(2);
    }
    void addEdge(int u,int v){
        ++e;
        adj[u].push_back(e<<1);
        edges.push_back(v);
        adj[v].push_back((e<<1)|1);
        edges.push_back(u);
        visit.push_back(false);
        visit.push_back(false);
    }
    int findFirst(){
        int u = 0;
        for(int i=0;i<n;++i){
            if(adj[i].size()==1)continue;
            u = i;
            if(!(adj[i].size()&1))break;
        }
        return u;
```

```cpp
    }
    vector<int> eulerCircuit(int u){
        if(!e)return vector<int>();
        stack<int> st;
        vector<int> circuit;
        st.push(u);
        while(!st.empty()){
            int x = get(st.top());
            if(x){
                visit[x] = visit[x^1] = true;
                st.push(edges[x]);
            }
            else{
                circuit.push_back(st.top());
                st.pop();
            }
        }
        return circuit;
    }
    vector<int> eulerCircuit(){
        if(!e)return vector<int>();
        return eulerCircuit(findFirst());
    }
};
```

5.7 tarjan_bridges_articulation_points

```cpp
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> tin, low;
vector<pair<int,int>> bridge;
vector<int> cutpoint;
int timer;

void IS_BRIDGE(int v,int to) {
    bridge.push_back({v,to});
}

void dfs_bridge(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs_bridge(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs_bridge(i);
    }
}
```

```cpp
void IS_CUTPOINT(int v) {
    cutpoint.push_back(v);
}

void dfs_points(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs_points(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs_points(i);
    }
}
```

6 Geometry

6.1 convex_hull

```cpp
#include<bits/stdc++.h>
using namespace std;

struct pt {
    double x, y;
    pt(double x=0, double y=0): x(x),y(y) {}
    bool operator == (const pt &rhs) const {
        return (x == rhs.x && y == rhs.y);
    }
    bool operator < (const pt &rhs) const {
        return y < rhs.y || (y==rhs.y && x < rhs.x);
    }
    bool operator > (const pt &rhs) const {
        return y > rhs.y || (y==rhs.y && x > rhs.x);
    }
};

double area(const vector<pt> &poly) {
    int n = static_cast<int>(poly.size());
    double area = 0;
    for(int i=0;i<n;++i) {
        pt p = i ? poly[i-1] :poly.back();
        pt q = poly[i];

        area += (p.x - q.x) * (p.y - q.y) ;
```

```cpp
    }
    area = fabs(area)/2;
    return area ;
}


int orientation(pt a, pt b, pt c) {
    double v = a.x*(b.y-c.y)+
               b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}


bool cw(
    pt a, pt b, pt c, bool include_collinear=false) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) {
    return orientation(a, b, c) == 0;
}


// Returns square of distance between point a and b
long double square_dist(pt a, pt b) {
    return (a.x-b.x)*1ll*(a.x-b.x)
        + (a.y-b.y)*1ll*(a.y-b.y);
}


// Returns points in clokwise order with a[0] as
// left lowermost point(Minimum point)
void convex_hull(
    vector<pt>& a, bool include_collinear = false) {
    pt p0 = *min_element(a.begin(), a.end(),
        [](pt a,pt b) {
        return make_pair(a.y, a.x)
                < make_pair(b.y, b.x);
    });
    auto square_dist_from_p0 = [&p0](pt& a){
        return square_dist(p0,a);
    };
    sort(a.begin(), a.end(), [&p0](
        const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return square_dist_from_p0(a)
                < square_dist_from_p0(b);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 &&
            collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<pt> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2],
            st.back(), a[i], include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    a = st;
}
```

```cpp
// poly: Contains points of polygon in counter
// clockwise order, with poly[0] as lower
// leftmost point(minimum point) and top is the index
// of top rightmost point(maximum point).
// Min/Max are defined by comparator operator
// defined for points.
// It returns 1: Point outside, 0: Point in boundary,
// -1: Point inside.
int pointVsConvexPolygon(
    pt &point, vector<pt> &poly, int top) {
    if(point < poly[0] || point > poly[top]) return 1;
    int o = orientation(point, poly[top], poly[0]);
    if(o == 0) {
        if(point == poly[0] || point == poly[top])
            return 0;
        return (top == 1 || top+1==poly.size())
            ? 0 : -1;
    } else if(o < 0) {
        auto itLeft = upper_bound(
            poly.rbegin(), poly.rend() - top-1, point);
        return orientation((itLeft == poly.rbegin() ?
            poly[0]:itLeft[-1]), itLeft[0], point);
    } else {
        auto itRight = lower_bound(poly.begin()+1,
                    poly.begin()+top,point);
        return orientation(point,
            itRight[0], itRight[-1]);
    }
}




// a and b represent direction:
// Returns 1 if direction is ccw, 0 is coolinear
// and -1 is direction is cw (clockwise).
int ccw(const pt &a, const pt&b) {
    long double ans = (long double)a.x*b.y
                    - (long double)a.y*b.x;
    if(ans < 0) return -1;
    return (ans > 0);
}

// Maximum distance (squared) between
// given sets of points
// in O(N) or O(N*log(N))
// (first make them a convex polygon)
long double maxSquareDist(vector<pt> &poly) {
    int n = static_cast<int>(poly.size());
    long double res = 0;
    for(int i = 0, j = n < 2 ? 0 : 1; i < j; ++i)
        for(;; j = (j+1)%n) {
            res = max(res, square_dist(
                poly[i], poly[j]));
            pt dir1 = pt(poly[i+1].x-poly[i].x,
                poly[i+1].y-poly[i].y);
            pt dir2 = pt(poly[(j+1)%n].x-poly[j].x,
                poly[(j+1)%n].y-poly[j].y);
            if(ccw(dir1, dir2) <= 0) break;
        }
    return res;
}
```

6.2   3d

```cpp
using ll = long long;
```

```cpp
using ld = long double;
using uint = unsigned int;
template<typename T>
using pair2 = pair<T, T>;
using pii = pair<int, int>;
using pli = pair<ll, int>;
using pll = pair<ll, ll>;

#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second

const ld eps = 1e-8;
bool eq(ld x, ld y) {
        return fabsl(x - y) < eps;
}
bool ls(ld x, ld y) {
        return x < y && !eq(x, y);
}
bool lseq(ld x, ld y) {
        return x < y || eq(x, y);
}

ld readLD() {
        int x;
        scanf("%d", &x);
        return x;
}

struct Point {
        ld x, y, z;

        Point() : x(), y(), z() {}
        Point(ld _x, ld _y, ld _z) : x(_x), y(_y),
            z(_z) {}

        void scan() {
                x = readLD();
                y = readLD();
                z = readLD();
        }

        Point operator + (const Point &a) const {
                return Point(x + a.x, y + a.y, z + a.z);
        }
        Point operator - (const Point &a) const {
                return Point(x - a.x, y - a.y, z - a.z);
        }
        Point operator * (const ld &k) const {
                return Point(x * k, y * k, z * k);
        }
        Point operator / (const ld &k) const {
                return Point(x / k, y / k, z / k);
        }
        ld operator % (const Point &a) const {
                return x * a.x + y * a.y + z * a.z;
        }
        Point operator * (const Point &a) const {
                return Point(
                        y * a.z - z * a.y,
                        z * a.x - x * a.z,
                        x * a.y - y * a.x
                        );
        }
        ld sqrLen() const {
```

```cpp
                return *this % *this;
        }
        ld len() const {
                return sqrtl(sqrLen());
        }
        Point norm() const {
                return *this / len();
        }
};

ld ANS;
//triangle contains 4 points, 4th point is a copy of 1st
Point A[4], B[4];

// P lies on plane, n is perpendicular, return A' such
// that AA' is parallel plane,
// PA' is perpendicular
Point getHToPlane(Point P, Point A, Point n) {
        n = n.norm();
        return P + n * ((A - P) % n);
}

// Checks if point P is in traingle t
bool inTriang(Point P, Point* t) {
        ld S = 0;
        S += ((t[1] - t[0]) * (t[2] - t[0])).len();
        for (int i = 0; i < 3; i++)
                S -= ((t[i] - P) * (t[i + 1] - P)).len();
        return eq(S, 0);
}
// Assuming A,B,C are on same line,
// it finds if C is between B
bool onSegm(Point A, Point B, Point C) {
        return lseq((A - B) % (C - B), 0);
}
//A is a point on line, a is direction of line, B is
// point on plane,
//b is normal to plane, l is used to store result in
// case intersection exists
bool intersectLinePlane(Point A, Point a, Point B,
    Point b, Point &I) {
        if (eq(a % b, 0)) return false;
        ld t = ((B - A) % b) / (a % b);
        I = A + a * t;
        return true;
}
//A is point on line, a is direction of line
//returns projection of P on line
Point getHToLine(Point P, Point A, Point a) {
        a = a.norm();
        return A + a * ((P - A) % a);
}
//get minimum distance of A from PQ
ld getPointSegmDist(Point A, Point P, Point Q) {
        Point H = getHToLine(A, P, Q - P);
        if (onSegm(P, H, Q)) return (A - H).len();
        return min((A - P).len(), (A - Q).len());
}
//
ld getSegmDist(Point A, Point B, Point C, Point D) {
        ld res = min(
                min(getPointSegmDist(A, C, D),
                        getPointSegmDist(B, C, D)),
                min(getPointSegmDist(C, A, B),
                        getPointSegmDist(D, A, B)));
        Point n = (B - A) * (D - C);
        if (eq(n.len(), 0)) return res;
```

```
        n = n.norm();
        Point I1, I2;
        if (!intersectLinePlane(A, B - A, C,
                (D - C) * n, I1)) throw;
        if (!intersectLinePlane(C, D - C, A,
                (B - A) * n, I2)) throw;
        if (!onSegm(A, I1, B)) return res;
        if (!onSegm(C, I2, D)) return res;
        return min(res, (I1 - I2).len());
}
```

6.3   lichao

```
// Important : Here range l,r denotes [l,r). With this
    a bug is avoided which is if we take [l,r] then
    when both l and r are negative the update [l,mid]
    can result in infinite
//          recursion. E.g: l = -7, r = -6. mid =
    (l+r)/2 = -6 so [l,mid] = [-7,-6], resulting in
    infinite recursion.

typedef long long ll;
struct line {
    ll m, c;
    ll operator()(ll x) { return m * x + c; }
};


class Lichao {
    vector<line> lc_tree;
public:
    Lichao(int N) {
        lc_tree = vector<line>(N << 1, { 0, -(ll)(1e18)
            });
    }

    void insert(int v, int l, int r, line cur) {
        if (l + 1 == r) {
            if (lc_tree[v](l) < cur(l)) lc_tree[v] = cur;
            return;
        }
        int mid = (l + r) >> 1;
        int z = (mid - l) << 1;
        if (lc_tree[v].m > cur.m) swap(lc_tree[v], cur);
        if (cur(mid) > lc_tree[v](mid)) {
            swap(lc_tree[v], cur);
            insert(v + 1, l, mid, cur);
        }
        else
            insert(v + z, mid, r, cur);
    }

    ll query(int v, int l, int r, int pt) {  // Finding
        maximum value of function at x = pt
        if (l + 1 == r) return lc_tree[v](pt);
        int mid = (l + r) >> 1;
        int z = (mid - l) << 1;
        if (pt <= mid) return max(lc_tree[v](pt),
            query(v + 1, l, mid, pt));
        else return max(lc_tree[v](pt), query(v + z,
            mid, r, pt));
    }
};
```

7   STL

7.1   pbds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type, less<int>,\
    rb_tree_tag, tree_order_statistics_node_update>

// strict less than is recommended to avoid problems
// with equality
// order_of_key(T key):number of elements less than key
// find_by_order(int): gives the 0 based index elem
```

7.2   pragmas

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```

7.3   random

```
mt19937_64 rng(chrono::steady_clock::now().
            time_since_epoch().count());

// call rng() for random number
```

8   Theory

8.1   Taylor function approximation

$$y_{n+1} = y_n + [A(x) - g(y)/g'(y_n)]$$

8.2   Pentagonal Theorem

$$\prod_{n=1}^{\infty} (1 - x^n) = \sum_{k=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2}$$
$$= 1 + \sum_{k=1}^{\infty} (-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2} \right)$$

8.3   Arithmetic Polynomial Evaluation

$$let\ degree(p(x)) = n;$$

$$\exists\ g(x)\ with\ d(g(x)) = n + 1$$

$$such\ that\ e^x g(x) = \sum_{0}^{\infty} (p(i)x^i/i!)$$

8.4 Lagrange Theorem

$$let f(g(x)) = x, f(0) = g(0) = 0,$$
$$f'(0) = g'(0) = 1;$$

$$[x^n]H(f(x)) = [x^{n-1}]H'(x)/((g(x)/x)^n)$$

8.5 Chirp Z Transform

$$ij = {}^{i+j}C_2 - {}^iC_2 - {}^jC_2$$

8.6 Mobius Transform

$$\mu(p^k) = [k == 0] - [k == 1]$$

$$\sum_{d|n} \mu(d) = [n == 1]$$

8.7 Xor Convolution

$$b_j = \sum a_i(-1)^{bitcount(i\&j)}$$

$$inverse\ is\ the\ same$$

8.8 Or Convolution

$$b_j = \sum a_i[i\&j == i]$$

8.9 And Convolution

$$b_j = \sum a_i[i|j == i]$$

8.10 GCD Convolution

$$b_j = \sum_{j|i} a_i;\ inverse:\ reverse\ of\ code$$

8.11 LCM Convolution

$$b_j = \sum_{i|j} a_j;\ inverse:\ reverse\ of\ code$$

8.12 Picks Theorem

$$A = i + (b/2) - 1$$

8.13 Knuth Dp Optimisation

$$dp(i,j) = \min_{i \leq k \leq j}[dp(i,k) + dp(k+1,j) + C(i,j)]$$

$$use\ for\ finding\ opt(i,j)$$
$$opt(i, j-1) \leq opt(i,j) \leq opt(i+1,j);$$

$$conditions:$$
$$a \leq b \leq c \leq d$$
$$\implies C(b,c) \leq C(a,d)$$
$$\implies C(a,c) + C(b,d) \leq C(a,d) + C(b,c)$$

```
for l in [0..n] do
    for i in [0..n] do
        L ← opt(i, i + l − 1)
        R ← opt(i + 1, i + l − 2)
        for k in [L..R] do
            dp(i, i + l) ← ...
        end for
    end for
end for
```