KTH Royal Institute of Technology

Omogen Heap

Simon Lindholm, Johan Sannemo, Mårten Wiman

2025-11-06

Contest (1)

template.cpp

14 lines

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

.bashrc

3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc

6 lines

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on   |   im jk <esc>   |   im kj <esc>   |   no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
 \| md5sum \| cut -c-6
```

hash.sh

3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

troubleshoot.txt

52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Mathematics (2)

MillerRabin.h

Description: Primality check for numbers up to $7 \cdot 10^{18}$, extend bases for larger number
Time: $\mathcal{O}(7 * logN)$
Status: Tested on judge.yosupo.jp

c63d96, 35 lines

```
namespace MillerRabin {
    using T = unsigned long long;

    T binpow(T x, T y, T mod) {
        T ans = 1; x %= mod;
        while (y > 0) {
            if (y & 1) ans = (Int)ans * x % mod;
            x = (Int)x * x % mod, y >>= 1;
        }
        return ans;
    }

    bool notPrime(T n, T base, T d, int s) {
        if (base % n == 0) return false;
        T x = binpow(base % n, d, n);
        if (x == 1 || x == n - 1) return false;
```

```
        for (int r = 1; r < s; ++r) {
            x = (Int)x * x % n;
            if (x == n - 1) return false;
        }
        return true;
    }

    bool isPrime(T n) {
        if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
        int s = CTZ(n - 1); T d = n >> s;

        const T bases[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};

        for (T base : bases) {
            if (notPrime(n, base, d, s)) return false;
        }
        return true;
    }
}
```

FastFourierTransform.h

Description: FFT to multiply two polynomials.
Usage: Get the real answer then convert to integer and modulo.
Time: $\mathcal{O}(N * logN)$ where N is the total sizes of two polynomials
Status: Well-tested

28f50c, 76 lines

```
namespace FFT {
    const ld pi = atan2(1, 1) * 4;

    struct cmplx {
        ld a, b;

        cmplx() {
            a = 0, b = 0;
        }

        cmplx(ld a1, ld b1) {
            a = a1, b = b1;
        }
    };

    inline cmplx sum(cmplx& c1, cmplx& c2) {
        return {c1.a + c2.a, c1.b + c2.b};
    }

    inline cmplx sub(cmplx& c1, cmplx& c2) {
        return {c1.a - c2.a, c1.b - c2.b};
    }

    inline cmplx mult(cmplx& c1, cmplx& c2) {
        return {c1.a * c2.a - c1.b * c2.b, c1.a * c2.b + c1.b * c2.a};
    }

    void fft(vector<cmplx> &a, int n) {
        for (int i = 1, j = 0; i < n; ++i) {
            int bit = n >> 1;
            for (; j >= bit; bit >>= 1) j -= bit;
            j += bit;
            if (i < j) swap(a[i], a[j]);
        }
        for (int k = 1; k < n; k *= 2) {
            ld phi = pi / k;
            cmplx wt(cos(phi), sin(phi));
            for (int i = 0; i < n; i += 2 * k) {
                cmplx w(1, 0);
                for (int j = 0; j < k; ++j) {
                    cmplx u = a[i + j], v = mult(a[i + j + k], w);
                    a[i + j] = sum(u, v);
                    a[i + j + k] = sub(u, v);
                    w = mult(w, wt);
                }
            }
        }
    }

    vector<int> multiply(vector<int> &s, vector<int> &t) {
        int need = (int) s.size() + t.size() - 1, sz = 1;
        while (sz < need) sz <<= 1;
        vector<cmplx> a(sz), b(sz);
        for (int i = 0; i < sz; ++i) {
            if(i < (int)s.size()) a[i] = cmplx(s[i], 0);
            else a[i] = cmplx(0, 0);
            if(i < (int)t.size()) b[i] = cmplx(t[i], 0);
            else b[i] = cmplx(0, 0);
        }
        fft(a, sz);
        fft(b, sz);
        for (int i = 0; i < sz; ++i) {
            a[i] = mult(a[i], b[i]);
        }

        fft(a, sz);
        reverse(a.begin() + 1, a.end());

        vector<int> ans(need);
        for (int i = 0; i < need; ++i) ans[i] = (Int)(round(a[i].a / sz)) % MOD;
        return ans;
    }
}
vector<int> operator * (vector<int> &s, vector<int> &t) {
    return FFT::multiply(s, t);
}
```

Data structures (3)

LichaoTree.h

Description: LichaoTree -> use for DP convex hull trick
Time: Both operations are $\mathcal{O}(logN)$
Status: Tested by KickingKun

3ac6f1, 48 lines

```
struct LiChaoTree {
    static const ll lim = 1e9;
    struct Line {
        ll a, b;
        ll operator() (ll x) {
            return a * x + b;
        }
    };

    struct node {
        Line line;
        node *l, *r;
        node() {
            line = {0, INF};
            l = r = NULL;
        }
    };

    node *root = new node();

    ll query(node *i, ll l, ll r, ll x) {
        if (i == NULL || x < l || x > r) return INF;
        ll m = (l + r) >> 1;
        ll ans = i->line(x);
        minimize(ans, query(i->l, l, m, x));
        minimize(ans, query(i->r, m + 1, r, x));
        return ans;
```

```
    }

    ll query(ll x) {
        return query(root, -lim, lim, x);
    }

    void add(Line li, node *&cur, ll l, ll r) {
        if (cur == nullptr){
            cur = new node(); cur->line = li;
            return;
        }
        ll mid = (l + r) >> 1;
        if (li(mid) < cur->line(mid)) swap(li, cur->line);
        if (li(l) < cur->line(l)) add(li, cur->l, l, mid);
        if (li(r) < cur->line(r)) add(li, cur->r, mid + 1, r);
    }

    void add(ll m, ll b) {
        add({m, b}, root, -lim, lim);
    }
} cht;
```

PersistentIT.h

Description: Persistent segment tree with point updates and range sum queries.
Usage: `Remember to call mytree.resize(n) before any operations.`
`Make sure to check for integer overflows in update values.`
Memory: $\mathcal{O}(logN * numUpdates)$
Time: $\mathcal{O}(logN)$ for each operation
Status: Tested on CSES

7b58c2, 55 lines

```
struct Node {
    int le, ri;
    ll sum;

    Node(int _le = 0, int _ri = 0, ll _sum = 0) {
        le = _le, ri = _ri, sum = _sum;
    }
} nodes[N * LG];
int numNode = 0;
vector<int> version;

struct PersistentIT { // 1-indexed
    int n;

    PersistentIT(int _n = 0) {
        resize(_n);
    }

    void resize(int _n) {
        n = _n;
    }

    int update(int oldId, int l, int r, int p, int val) {
        if (l > p || r < p) return oldId;

        int id = ++numNode;
        if (l == r) {
            nodes[id] = nodes[oldId];
            nodes[id].sum += val;
            return id;
        }

        int mid = (l + r) >> 1;
        nodes[id].le = update(nodes[oldId].le, l, mid, p, val);
        nodes[id].ri = update(nodes[oldId].ri, mid + 1, r, p, val);

        nodes[id].sum = nodes[nodes[id].le].sum + nodes[nodes[id].ri].sum;
        return id;
    }

    int update(int root, int p, int val) {
        return update(root, 1, n, p, val);
    }

    ll get(int id, int l, int r, int u, int v) {
        if (l > v || r < u) return 0;
        if (u <= l && r <= v) return nodes[id].sum;
        int mid = (l + r) >> 1;
        return get(nodes[id].le, l, mid, u, v) + get(nodes[id].ri, mid + 1, r, u, v);
    }

    ll get(int root, int u, int v) {
        return get(root, 1, n, u, v);
    }
} mytree;
```

PersistentLazyIT.h

Description: Persistent segment tree with range updates and range sum queries.
Usage: `Remember to call mytree.resize(n) before any operations.`
Memory: $\mathcal{O}(4 * logN * numUpdates)$
Time: $\mathcal{O}(logN)$ for each operation
Status: Not tested

a6f504, 73 lines

```
struct Node {
    int le, ri;
    ll lazy, sum;

    Node(int _le = 0, int _ri = 0, ll _lazy = 0, ll _sum = 0) {
        le = _le, ri = _ri, lazy = _lazy, sum = _sum;
    }
} nodes[4 * N * LG];
int numNode = 0;
vector<int> version;

struct PersistentLazyIT {
    int n;

    PersistentLazyIT(int _n = 0) {
        resize(_n);
    }

    void resize(int _n) {
        n = _n;
    }

    int newlazykid(int oldId, int l, int r, ll val) {
        int id = ++numNode;
        nodes[id] = nodes[oldId];

        nodes[id].sum += 1ll * val * (r - l + 1);
        nodes[id].lazy += val;

        return id;
    }

    int newparent(int le, int ri) {
        int id = ++numNode;
        nodes[id].le = le, nodes[id].ri = ri;
        nodes[id].sum = nodes[le].sum + nodes[ri].sum;
        return id;
    }

    void push(int id, int l, int r) {
        if (nodes[id].lazy == 0) return;
        if (l != r) {
            int mid = (l + r) >> 1;
            nodes[id].le = newlazykid(nodes[id].le, l, mid, nodes[id].lazy);
            nodes[id].ri = newlazykid(nodes[id].ri, mid + 1, r, nodes[id].lazy);
```

```
        }
        nodes[id].lazy = 0;
    }

    int update(int oldId, int l, int r, int u, int v, ll val) {
        if (l > v || r < u) return oldId;
        if (u <= l && r <= v) return newlazykid(oldId, l, r, val);
        push(oldId, l, r); int mid = (l + r) >> 1;
        int le = update(nodes[oldId].le, l, mid, u, v, val),
            ri = update(nodes[oldId].ri, mid + 1, r, u, v, val);
        return newparent(le, ri);
    }

    int update(int root, int u, int v, int val) {
        return update(root, 1, n, u, v, val);
    }

    ll get(int id, int l, int r, int u, int v) {
        if (l > v || r < u) return 0;
        if (u <= l && r <= v) return nodes[id].sum;
        push(id, l, r); int mid = (l + r) >> 1;
        return get(nodes[id].le, l, mid, u, v) + get(nodes[id].ri, mid + 1, r, u, v);
    }

    ll get(int root, int u, int v) {
        return get(root, 1, n, u, v);
    }
} mytree;
```

Graph (4)

TwoSat.h

Description: 2sat -> finding an arbitrary equation
Time: $\mathcal{O}(n+m)$
Status: Tested by KickingKun

28a9c5, 66 lines

```
struct TwoSat { // convert vector -> array to get higher speed ....
  int n; vector <int> lab, num, low; vector <vector <int>> adj;
  int timeDfs, scc; stack <int> st;

  TwoSat (int _n) {
    this->n = _n; lab.assign(n * 2 + 1, 0); low.assign(n * 2 + 1, 0);
    num.assign(n * 2 + 1, 0); adj.assign(n * 2 + 1, vector <int>());
    timeDfs = scc = 0; while (!st.empty()) st.pop();
  }

  int NON(int u) {
    return u <= n ? u + n : u - n;
  }

  void add(int u, int v) {
    adj[u].emplace_back(v);
  }

  void add_u_or_v(int u, int v) { // u or v = 1
    adj[NON(u)].emplace_back(v); adj[NON(v)].emplace_back(u);
  }
  void assign(int u, bool val) { // u = val
    if (val == 1) add_u_or_v(u, u);
    else add_u_or_v(NON(u), NON(u));
  }
  void add_equal(int u, int v) { // u = v
    add_u_or_v(u, NON(v)); add_u_or_v(NON(u), v);
  }
  void add_non_equal(int u, int v) { // u != v
    add_u_or_v(u, v); add_u_or_v(NON(u), NON(v));
  }
  void dfs(int u) {
    low[u] = num[u] = ++timeDfs; st.emplace(u);
    for (int v: adj[u]) {
      if (lab[v]) continue;
      if (!num[v]) dfs(v), minimize(low[u], low[v]);
      else minimize(low[u], num[v]);
    }

    if (low[u] == num[u]) {
      ++scc;
      while (true) {
        int v = st.top(); st.pop();
        lab[v] = scc; if (v == u) break;
      }
    }
  }

  bool check_exist() {
    for (int i = 1; i <= n; i++) {
      if (!lab[NON(i)]) dfs(NON(i));
      if (!lab[i]) dfs(i);
    }
    for (int i = 1; i <= n; i++)
      if (lab[i] == lab[NON(i)]) return false;
    return true;
  }

  vector <int> get_equation() {
    if (!check_exist()) return {-1};
    vector <int> ans = {0};
    for (int i = 1; i <= n; i++) ans.emplace_back(lab[i] < lab[NON(i)]);
    return ans;
  }
};
```

BipartieMatching.h

Description: use for max matching
Time: $\mathcal{O}(E * sqrt(V))$
Status: approved by KickingKun

213423, 82 lines

```
struct bipartite_matching {
  const int n, m;
  vector <int> match1, match2, que, dist;
  vector <vector<int>> adj;

  bipartite_matching(int _n, int _m) : n(_n), m(_m) {
    match1.assign(n + 1, -1); match2.assign(m + 1, -1);
    que.assign(n + 1, 0); dist.assign(n + 1, 0); adj.assign(n + 1, vector <int>());
  }

  void add(int u, int v) {
    adj[u].emb(v);
  }

  bool bfs() {
    fill(dist.begin(), dist.end(), -1);
    int queBegin = 1, queEnd = 1;
    for (int u = 1; u <= n; ++u) {
      if (match1[u] == -1) dist[u] = 0, que[queEnd++] = u;
    }
    bool success = false;
    while (queBegin < queEnd) {
      int u = que[queBegin++];
      for (int v : adj[u]) {
        if (match2[v] == -1) {
          success = true;
        }
        else if (dist[match2[v]] == -1) {
          dist[match2[v]] = dist[u] + 1;
```

```
        que[queEnd++] = match2[v];
      }
    }
  }
  return success;
}

bool dfs(int u) {
  for (int v : adj[u]) {
    if (match2[v] == -1 || (dist[match2[v]] == dist[u] + 1 && dfs(match2[v]))) {
      match1[u] = v; match2[v] = u; dist[u] = n + m;
      return true;
    }
  }
  dist[u] = n + m;
  return false;
}

int max_matching() {
  while (bfs()) {
    for (int u = 1; u <= n; ++u) {
      if (match1[u] == -1) dfs(u);
    }
  }
  return n - count(match1.begin() + 1, match1.end(), -1);
}

pair <vector<int>, vector<int>> minimum_vertex_cover() {
        vector <int> L, R;
        for (int u = 1; u <= n; ++u) {
              if (dist[u] == -1) L.emb(u);
              else if (match1[u] != -1) R.emb(match1[u]);
        }
        return {L, R};
    }

    pair <vector <int>, vector <int>> maximum_independent_set() {
  auto [_L, _R] = minimum_vertex_cover();
  vector <int> L, R;
  vector <bool> mark1(n + 2), mark2(m + 2);
  for (int x: _L) mark1[x] = true; for (int x: _R) mark2[x] = true;
  for (int u = 1; u <= n; u++) if (!mark1[u]) L.emb(u);
  for (int u = 1; u <= m; u++) if (!mark2[u]) R.emb(u);
  return {L, R};
}

vector <pii> get_edges() { // get from max matching
  vector <pii> ans;
  for (int u = 1; u <= n; ++u)
    if (match1[u] != -1) ans.emplace_back(u, match1[u]);
  return ans;
}
};
```

MaxFlow.h

Description: Dinitz's algorithm to find max-flow.
Usage: Call myflow.resize(num_vertices, source, sink) to initalize the network.
Time: $\mathcal{O}\left(V^2 * E\right)$ in worst cases
Status: Well-tested

455f5c, 95 lines

```
struct MaxFlow {
    struct FlowEdge {
        int u, v;
        ll capacity, flow = 0;

        FlowEdge(int _u = 0, int _v = 0, ll _capacity = 0) {
            u = _u, v = _v, capacity = _capacity, flow = 0;
        }
    };

const ll FLOW_INF = (ll)2e18;
vector<FlowEdge> edges;
vector<vector<int>> adj;
vector<int> level, ptr;
int n, m;
int source, sink;

MaxFlow(int _n = 0, int _s = 0, int _t = 0) {
    resize(_n, _s, _t);
}

void resize(int _n, int _s, int _t) {
    n = _n, m = 0, source = _s, sink = _t;
    adj.assign(n + 1, vector<int>()), level.assign(n + 1, -1);
    ptr.assign(n + 1, 0); edges.clear();
}

void reset_flow() {
    for (auto &edge : edges) edge.flow = 0;
}

void add(int u, int v, ll capacity) {
    edges.emplace_back(u, v, capacity);
    edges.emplace_back(v, u, 0);
    adj[u].emplace_back(m++), adj[v].emplace_back(m++);
}

bool bfs() {
    fill(level.begin(), level.end(), -1); level[source] = 0;
    queue<int> q; q.emplace(source);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (auto id : adj[u]) {
            if (edges[id].capacity - edges[id].flow <= 0) continue;
            if (level[edges[id].v] != -1) continue;
            level[edges[id].v] = level[u] + 1;
            q.emplace(edges[id].v);
        }
    }
    return level[sink] != -1;
}

ll dfs(int u, ll pushed) {
    if (pushed == 0) return 0;
    if (u == sink) return pushed;
    for (int &cid = ptr[u]; cid < (int)adj[u].size(); ++cid) {
        int id = adj[u][cid], v = edges[id].v;
        if (level[u] + 1 != level[v] || edges[id].capacity - edges[id].flow <= 0) continue;
        ll amount = dfs(v, min(pushed, edges[id].capacity - edges[id].flow));
        if (amount == 0) continue;
        edges[id].flow += amount, edges[id ^ 1].flow -= amount;
        return amount;
    }
    return 0;
}

ll maxFlow() {
    ll flow = 0;
    while (bfs()) {
        fill(ptr.begin(), ptr.end(), 0);
        while (ll pushed = dfs(source, FLOW_INF)) flow += pushed;
    }
    return flow;
}

vii minCut() {
    queue<int> q; q.emplace(source);
    fill(ptr.begin(), ptr.end(), 0); vector<bool> vis(n + 1, false);
```

```
        while (!q.empty()) {
            int u = q.front(); q.pop();
            if (vis[u]) continue;
            vis[u] = true;
            for (int &cid = ptr[u]; cid < (int)(adj[u].size()); ++cid) {
                int id = adj[u][cid], v = edges[id].v;
                if (level[u] + 1 != level[v] || edges[id].capacity - edges[id].flow <= 0)
                    continue;
                q.emplace(v);
            }
        }
        vii cut;
        for (auto it : edges) {
            if (vis[it.u] && !vis[it.v]) cut.emplace_back(it.u, it.v);
        }
        return cut;
    }
} myflow;
```

MinCostMaxFlow.h

Description: Edmond-Karp algorithm with SPFA to find the shortest path.
Usage: `Call myflow.resize(num_vertices, source, sink) to initalize the network.`
Time: $\mathcal{O}(F * V * E)$ in worst cases
Status: Well-tested

0cce52, 117 lines

```
struct MCMF {
    struct FlowEdge {
        int u, v;
        ll capacity, cost, flow = 0;

        FlowEdge(int _u = 0, int _v = 0, ll _capacity = 0, ll _cost = 0) {
            u = _u, v = _v, capacity = _capacity, cost = _cost, flow = 0;
        }
    };

    const ll FLOW_INF = (ll)2e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    vector<int> level, ptr;
    int n, m;
    int source, sink;

    MCMF(int _n = 0, int _s = 0, int _t = 0) {
        resize(_n, _s, _t);
    }

    void resize(int _n, int _s, int _t) {
        n = _n, m = 0, source = _s, sink = _t;
        adj.assign(n + 1, vector<int>()), level.assign(n + 1, -1);
        ptr.assign(n + 1, 0); edges.clear();
    }

    void reset_flow() {
        for (auto &edge : edges) edge.flow = 0;
    }

    void add(int u, int v, ll capacity, ll cost) {
        edges.emplace_back(u, v, capacity, cost);
        edges.emplace_back(v, u, 0, -cost);
        adj[u].emplace_back(m++), adj[v].emplace_back(m++);
    }

    bool bfs() {
        fill(level.begin(), level.end(), -1); level[source] = 0;
        queue<int> q; q.emplace(source);
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (auto id : adj[u]) {
                if (edges[id].capacity - edges[id].flow <= 0) continue;
                if (level[edges[id].v] != -1) continue;
                level[edges[id].v] = level[u] + 1;
                q.emplace(edges[id].v);
            }
        }
        return level[sink] != -1;
    }

    ll dfs(int u, ll pushed) {
        if (pushed == 0) return 0;
        if (u == sink) return pushed;
        for (int &cid = ptr[u]; cid < (int)adj[u].size(); ++cid) {
            int id = adj[u][cid], v = edges[id].v;
            if (level[u] + 1 != level[v] || edges[id].capacity - edges[id].flow <= 0) continue;
            ll amount = dfs(v, min(pushed, edges[id].capacity - edges[id].flow));
            if (amount == 0) continue;
            edges[id].flow += amount, edges[id ^ 1].flow -= amount;
            return amount;
        }
        return 0;
    }

    ll maxFlow() {
        ll flow = 0;
        while (bfs()) {
            fill(ptr.begin(), ptr.end(), 0);
            while (ll pushed = dfs(source, FLOW_INF)) flow += pushed;
        }
        return flow;
    }

    bool spfa(vector<ll> &dist, vector<int> &par) {
        dist.assign(n + 1, INF), par.assign(n + 1, -1);
        vector<bool> inQueue(n + 1, false); queue<int> q;
        dist[source] = 0; inQueue[source] = true; q.emplace(source);
        while (!q.empty()) {
            int u = q.front(); q.pop();
            inQueue[u] = false;
            for (auto id : adj[u]) {
                int v = edges[id].v;
                if (edges[id].capacity - edges[id].flow <= 0) continue;
                if (minimize(dist[v], dist[u] + edges[id].cost)) {
                    par[v] = id;
                    if (!inQueue[v]) {
                        inQueue[v] = true;
                        q.emplace(v);
                    }
                }
            }
        }
        return dist[sink] != INF;
    }

    ll minCost() {
        ll flow = 0, cost = 0; vector<ll> dist; vector<int> par;
        while (spfa(dist, par)) {
            ll amount = FLOW_INF - flow; int cur = sink;
            while (cur != source) {
                minimize(amount, edges[par[cur]].capacity - edges[par[cur]].flow);
                cur = edges[par[cur]].u;
            }

            flow += amount, cost += amount * dist[sink], cur = sink;
            while (cur != source) {
                edges[par[cur]].flow += amount;
                edges[par[cur] ^ 1].flow -= amount;
                cur = edges[par[cur]].u;
            }
        }
```

```
        return cost;
    }
} myflow;
```

4.1 Math

4.1.1 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k).$$

Geometry (5)

Geometry.h
Description: Geometric Template
Time: $\mathcal{O}(1)$
Status: Seems OK
de8384, 122 lines

```
struct Point {
  double x, y;
  Point() {}
  Point (double _x, double _y) {
    this->x = _x; this->y = _y;
  }

  double len() {
    return x * x + y * y;
  }

  Point operator - (Point other) {
    return {x - other.x, y - other.y};
  }

  Point operator - () {
    return Point(-x, -y);
  }

  bool operator == (Point other) {
    return x == other.x && y == other.y;
  }

  bool operator != (Point other) {
    return x != other.x || y != other.y;
  }

  void show() {
    cout << x << ' ' << y << '\n';
  }
};

struct Line {
  double a, b, c;
  Line() {}
  Line (double _a, double _b, double _c) {
    this->a = _a; this->b = _b; this->c = _c;
  }

  void simplify() { // if double = ll
    double d = __gcd(__gcd((ll)a, (ll)b), (ll)c);
    a /= d, b /= d, c /= d;
    // else do nothing
  }
  void show() {
    cout << a << ' ' << b << ' ' << c << '\n';
  }
};

double dot(Point A, Point B) { // dot product
  return A.x * B.x + A.y * B.y;
}

double cross(Point A, Point B) { // cross product
  return A.x * B.y - A.y * B.x;
}

double dis(Point A, Point B) {
  return sqrt((B.x - A.x) * (B.x - A.x) + (B.y - A.y) * (B.y - A.y));
}

double dis(Point m, Line d) { // d(M, delta)
  return abs(d.a * m.x + d.b * m.y + d.c) / sqrt(d.a * d.a + d.b * d.b);
}

double polygon_area(vector <Point> p) {
  double res = 0;
  for (int i = 1; i < p.size() - 1; i++)
    res += cross(p[i] - p[0], p[i + 1] - p[0]);
  return abs(res) / 2.0;
}

bool same_side(Point A, Point B, Line d) { // out of line
  double y1 = d.a * A.x + d.b * A.y + d.c;
  double y2 = d.a * B.x + d.b * B.y + d.c;
  return (y1 > 0 && y2 > 0) || (y1 < 0 && y2 < 0);
}

Line lineVector(Point A, Point n) { // Point + vector n
  return Line(n.x, n.y, -n.x * A.x - n.y * A.y);
}

Line line(Point A, Point B) { // Point A, B
  Point n(A.y - B.y, B.x - A.x);
  return Line(n.x, n.y, -n.x * A.x - n.y * A.y);
}

bool inside_line(Point A, Line d) { // inside of line
  return d.a * A.x + d.b * A.y + d.c == 0;
}

bool inside_segment(Point M, Point A, Point B) { // inside of segment AB
  if (M.x > max(A.x, B.x) || M.x < min(A.x, B.x)) return false;
  return inside_line(M, line(A, B));
}

bool inside_ray(Point M, Point A, Point B) { // inside of ray AB
  if ((A.x < B.x && M.x < A.x) || (A.x > B.x && M.x > A.x)) return false;
  return inside_line(M, line(A, B));
}

Point intersect(Line A, Line B) { // find p(x, y)
  Point res; if (A.b == 0) swap(A, B);
  res.x = (B.c * A.b - A.c * B.b) / (A.a * B.b - B.a * A.b);
  res.y = (-A.c - A.a * res.x) / A.b;
  return res;
}

Line perpendicular(Point A, Line d) {
  Point n(-d.b, d.a);
  return Line(n.x, n.y, -n.x * A.x - n.y * A.y);
}

bool obtuse(Point M, Point A, Point B) { // MAB > 90
```

```
  return (M - A).len() + (B - A).len() < (M - B).len();
}

double dist(Point M, Point A, Point B) { // dis(M, segment AB)
//  check MBA<= 90 && MAB<= 90
  if (obtuse(M, A, B) || obtuse(M, B, A)) return min(dis(M, A), dis(M, B));
  return dis(M, line(A, B));
}
```

Strings (6)

Various (7)