

目次

1	テンプレート	2
2	グラフ	2
3	フロー	2
3.1	dinic	2
3.2	最小費用流	3
4	数学	4
4.1	modint	4
4.2	FFT	5
5	データ構造	6
5.1	セグメント木	6
5.2	遅延評価セグメント木	8
6	幾何	11
6.1	3D Geometry Template	11
6.2	3D Plane	13
6.3	3D Point on the Triangle	14
6.4	3D Libraries for Lines and Segments	14
6.5	3D Intersection of Planes	15

1 テンプレート

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 using ll = long long;
5 using P = pair<int, int>;
6 constexpr ll MOD = 1000000007;
7 constexpr int INF = 1 << 30;
8 #define REP(i, n) for (int i = 0, i_len = (n); i < i_len; i++)
9 #define ALL(v) (v).begin(), (v).end()

```

2 グラフ

3 フロー

3.1 dinic

最大流問題を解くアルゴリズム。計算量は $O(VE^2)$ だが実用上かなり高速なことが多い。

- Dinic flow(V): 構造体の宣言。V は頂点数。
- flow.add_edge(u, v, c): $u \rightarrow v$ に容量 c の辺を追加する
- flow.max_flow(s, t): $s \rightarrow t$ の最大流を返す

```

1 //Dinic法  $O(V^2E)$ 
2 struct Dinic {
3     int V; // 頂点数
4     vector<vector<vector<long long>>> graph; // グラフ
5     vector<int> dis; // 始点からの距離
6     vector<int> next; // 次に処理する頂点のメモ
7     Dinic(int v) : V(v) { graph.resize(V); }
8     void add_edge(int from, int to, long long capacity) {
9         graph[from].push_back({to, capacity, (int)graph[to].size()});
10        graph[to].push_back({from, 0, (int)graph[from].size() - 1});
11    }
12    void bfs(int s) {
13        dis.assign(V, -1);
14        dis[s] = 0;
15        deque<int> pos = {s};
16        while (pos.size()) {
17            int now = pos[0];
18            pos.pop_front();
19            for (auto& to : graph[now]) {
20                if (dis[to[0]] < 0 and to[1] > 0) {
21                    dis[to[0]] = dis[now] + 1;
22                    pos.emplace_back(to[0]);
23                }
24            }
25        }
26    }
27    long long dfs(int v, int t, long long f) {
28        if (v == t) return f;
29        for (int& i = next[v]; i < graph[v].size(); i++) {
30            int to = graph[v][i][0];
31            long long& cap = graph[v][i][1];
32            int rev = graph[v][i][2];
33            if (cap > 0 and dis[v] < dis[to]) {
34                long long d = dfs(to, t, min(f, cap));
35                if (d > 0) {
36                    cap -= d;
37                    graph[to][rev][1] += d;
38                    return d;
39                }
40            }
41            next[v] = i + 1;
42        }
43        return 0;
44    }
45    long long max_flow(int s, int t) {
46        long long flow = 0;
47        while (bfs(s)) {
48            flow += dfs(s, t, INF);
49        }
50        return flow;
51    }
52 };

```

```

39         }
40     }
41 }
42 return 0;
43 }
44 long long max_flow(int s, int t) {
45     long long flow = 0;
46     while (1) {
47         bfs(s);
48         if (dis[t] < 0) return flow;
49         next.assign(V, 0);
50         long long f;
51         while ((f = dfs(s, t, LLONG_MAX)) > 0) flow += f;
52     }
53 }
54 };

```

3.2 最小費用流

最小費用流問題を解くアルゴリズム。計算量は $O(FE \log V)$

- MinCostFlow flow(V): 構造体の宣言。V は頂点数。
- flow.add_edge(u, v, c, d): $u \rightarrow v$ に容量 c, コスト d の辺を追加する
- flow.min_cost_flow(s, t, F): $s \rightarrow t$ に流量 F を流すときの最小コストを返す。流せない場合は-1 を返す。

```

1  // 最小費用流  $O(FE \log V)$ 
2  struct MinCostFlow {
3      int V;
4      vector<vector<vector<long long>>> g; // g[from] = {{to, 容量, コスト, 逆辺のindex} ... }
5      vector<long long> h, dis; // ポテンシャル, 最短距離
6      vector<int> prevv, preve; // 直前の頂点, 辺
7
8      MinCostFlow(int v) : V(v), g(v), dis(v), prevv(v), preve(v) {
9      }
10
11     void add_edge(int u, int v, long long c, long long d) {
12         g[u].push_back({v, c, d, (int)g[v].size()});
13         g[v].push_back({u, 0, -d, (int)g[u].size() - 1});
14     }
15
16     long long min_cost_flow(int s, int t, long long f) {
17         long long res = 0;
18         h.assign(V, 0);
19         using Q = pair<long long, int>;
20         while (f > 0) {
21             priority_queue<Q, vector<Q>, greater<Q>> que;
22             dis.assign(V, LLONG_MAX);
23             dis[s] = 0;
24             que.push({0, s});
25             while (que.size()) {
26                 Q q = que.top();
27                 int v = q.second;
28                 que.pop();
29                 if (dis[v] < q.first) continue;
30                 for (int i = 0; i < g[v].size(); i++) {
31                     auto edge = g[v][i];
32                     int to = edge[0];
33                     long long cap = edge[1], cost = edge[2];
34                     if (cap > 0 and dis[to] > dis[v] + cost + h[v] - h[to]) {
35                         dis[to] = dis[v] + cost + h[v] - h[to];
36                         prevv[to] = v;
37                         preve[to] = i;
38                         que.push({dis[to], to});
39                     }
40                 }

```

```

41     }
42     if (dis[t] == LLONG_MAX) return -1;
43     for (int i = 0; i < V; i++) h[i] += dis[i];
44     long long d = f;
45     for (int i = t; i != s; i = prevv[i]) d = min(d, g[prevv[i]][preve[i]][1]);
46     f -= d;
47     res += d * h[t];
48     for (int i = t; i != s; i = prevv[i]) {
49         auto& edge = g[prevv[i]][preve[i]];
50         edge[1] -= d;
51         g[i][edge[3]][1] += d;
52     }
53 }
54 return res;
55 }
56 };

```

4 数学

4.1 modint

自動的に mod をとる構造体。mod が問題で固定かつ素数であるとき使用できる。

using mint = modint<1000000007>; 等のように定義して使用するのが推奨。

```

1  template <int64_t Modulus>
2  struct Modint {
3      using mint = Modint;
4      long long v;
5      Modint() : v(0) {}
6      Modint(long long x) {
7          x %= Modulus;
8          if (x < 0) x += Modulus;
9          v = x;
10     }
11     const long long& val() const { return v; }
12     // 代入演算子
13     mint& operator+=(const mint rhs) {
14         v += rhs.v;
15         if (v >= Modulus) v -= Modulus;
16         return *this;
17     }
18     mint& operator-=(const mint rhs) {
19         if (v < rhs.v) v += Modulus;
20         v -= rhs.v;
21         return *this;
22     }
23     mint& operator*=(const mint rhs) {
24         v = v * rhs.v % Modulus;
25         return *this;
26     }
27     mint& operator/=(mint rhs) { return *this = *this * rhs.inv(); }
28     // 累乗, 逆元
29     mint pow(long long n) const {
30         mint x = *this, res = 1;
31         while (n) {
32             if (n & 1) res *= x;
33             x *= x;
34             n >>= 1;
35         }
36         return res;
37     }
38     mint inv() const { return pow(Modulus - 2); }
39     // 算術演算子
40     mint operator+(const mint rhs) const { return mint(*this) += rhs; }
41     mint operator-(const mint rhs) const { return mint(*this) -= rhs; }
42     mint operator*(const mint rhs) const { return mint(*this) *= rhs; }

```

```

43     mint operator/(const mint rhs) const { return mint(*this) /= rhs; }
44     mint operator-() const { return mint() - *this; } // 単項
45 };

```

入出力ストリームの実装

```

1  using mint = Modint<MOD>;
2
3  // 入出力ストリーム
4  istream& operator>>(istream& is, mint& x) {
5      long long a;
6      is >> a;
7      x = a;
8      return is;
9  }
10 ostream& operator<<(ostream& os, mint& x) {
11     return os << x.val();
12 }

```

4.2 FFT

- encode(a): 整数型の配列 a を `std::complex` 型に変換。
- decode(a): `std::complex` 型の配列 a を 64bit 整数型に変換。配列の要素毎に実部を丸めて整数に変換している。
- FFT(a): `std::complex` 型で長さ n の配列 a をフーリエ変換する。整数型の配列は引数にとれないため、`encode(a)`, `decode(a)` 等で適宜変換を行うこと。
- convolution(a,b): 長さ n の整数列 a , 長さ m の整数列 b の畳み込みを $O((n+m)\log(n+m))$ で計算する。畳み込み後の配列の要素がすべて `double` に収まる必要がある。

```

1  // 整数配列を複素数へ
2
3  vector<complex<double>> encode(vector<long long> &a){
4      int N = a.size();
5      vector<complex<double>> ret(N);
6      for(int i = 0; i < N; i++){
7          ret[i] = complex<double>(a[i], 0);
8      }
9      return ret;
10 }
11
12 // 複素数配列を整数へ
13 vector<long long> decode(vector<complex<double>> &a){
14     int N = a.size();
15     vector<long long> ret(N);
16     for(int i = 0; i < N; i++){
17         ret[i] = (long long)round(a[i].real());
18     }
19     return ret;
20 }
21
22 // 非再帰
23 void FFT(vector<complex<double>> &a, int reverse = false) {
24     int n = a.size();
25     int h = 0;
26     for (int i = 0; 1 << i < n; i++) h++;
27     for (int i = 0; i < n; i++) {
28         int j = 0;
29         for (int k = 0; k < h; k++) j |= (i >> k & 1) << (h - 1 - k);
30         if (i < j) swap(a[i], a[j]);
31     }
32     for (int b = 1; b < n; b *= 2) {
33         for (int j = 0; j < b; j++) {
34             double p2 = 2*acos(-1);
35             if(reverse)p2 *= -1;

```

```

36         complex<double> w = exp(complex<double>(0,p2*j/(double)(2*b)));
37         for (int k = 0; k < n; k += b * 2) {
38             complex<double> s = a[j+k];
39             complex<double> t = a[j+k+b]*w;
40             a[j+k] = s+t;
41             a[j+k+b] = s-t;
42         }
43     }
44 }
45 if (reverse)for (int i = 0; i < n; i++) a[i] /= (double)n;
46 return;
47 }
48
49 vector<long long> convolution(vector<long long> &a,vector<long long> &b){
50     vector<complex<double>> A = encode(a),B = encode(b);
51     int s = (int)a.size()+(int)b.size()-1;
52     int t = 1;
53     while(t < s)t *= 2;
54     A.resize(t);B.resize(t);
55     FFT(A,0);FFT(B,0);
56     for(int i = 0;i < t;i++){
57         A[i] *= B[i];
58     }
59     FFT(A,1);
60     A.resize(s);
61     return decode(A);
62 }

```

5 データ構造

5.1 セグメント木

モノイドを満たすデータ S に対し使用できるデータ構造。

長さ N の S の配列に対し、要素の 1 点更新、区間クエリを $O(\log N)$ で行える。モノイド S 同士の演算の計算量が $O(f(n))$ とき、すべての計算量が $O(f(n))$ 倍になる。

```

1  template <class S, S (*op)(S, S), S (*e)()>
2  struct SegmentTree {
3      private:
4          int _n, size, log;
5          vector<S> dat;
6          void update(int k) { dat[k] = op(dat[2 * k], dat[2 * k + 1]); }
7
8      public:
9          SegmentTree() : SegmentTree(0) {}
10         SegmentTree(int n) : SegmentTree(vector<S>(n, e())) {}
11         SegmentTree(const vector<S>& v) : _n(int(v.size())) {
12             log = 0;
13             while ((1 << log) < _n) log++;
14             size = 1 << log;
15             dat = vector<S>(2 * size, e());
16             for (int i = 0; i < _n; i++) dat[size + i] = v[i];
17             for (int i = size - 1; i >= 1; i--) {
18                 update(i);
19             }
20         }
21         // a[p] = x
22         void set(int p, S x) {
23             p += size;
24             dat[p] = x;
25             for (int i = 1; i <= log; i++) update(p >> i);
26         }
27         // return a[p]
28         S get(int p) const {
29             return dat[p + size];

```

```

30 }
31 // return op(a[l], ..., a[r-1])
32 S prod(int l, int r) const {
33     S sm1 = e(), smr = e();
34     l += size;
35     r += size;
36     while (l < r) {
37         if (l & 1) sm1 = op(sm1, dat[l++]);
38         if (r & 1) smr = op(dat[--r], smr);
39         l >>= 1;
40         r >>= 1;
41     }
42     return op(sm1, smr);
43 }
44 S all_prod() const { return dat[1]; }
45
46 // SegmentTree上の二分探索 (必要な場合)
47 // return r, f(op(a[l], ..., a[r-1])) == true
48 template <bool (*f)(S)>
49 int max_right(int l) const {
50     return max_right(l, [](S x) { return f(x); });
51 }
52 template <class F>
53 int max_right(int l, F f) const {
54     assert(f(e()));
55     if (l == _n) return _n;
56     l += size;
57     S sm = e();
58     do {
59         while (l % 2 == 0) l >>= 1;
60         if (!f(op(sm, dat[l]))) {
61             while (l < size) {
62                 l = (2 * l);
63                 if (f(op(sm, dat[l]))) {
64                     sm = op(sm, dat[l]);
65                     l++;
66                 }
67             }
68             return l - size;
69         }
70         sm = op(sm, dat[l]);
71         l++;
72     } while ((l & -l) != l);
73     return _n;
74 }
75 // return l, f(op(a[l], ..., a[r-1])) == true
76 template <bool (*f)(S)>
77 int min_left(int r) const {
78     return min_left(r, [](S x) { return f(x); });
79 }
80 template <class F>
81 int min_left(int r, F f) const {
82     assert(f(e()));
83     if (r == 0) return 0;
84     r += size;
85     S sm = e();
86     do {
87         r--;
88         while (r > 1 && (r % 2)) r >>= 1;
89         if (!f(op(dat[r], sm))) {
90             while (r < size) {
91                 r = (2 * r + 1);
92                 if (f(op(dat[r], sm))) {
93                     sm = op(dat[r], sm);
94                     r--;
95                 }
96             }
97             return r + 1 - size;

```

```

98         }
99         sm = op(dat[r], sm);
100     } while ((r & -r) != r);
101     return 0;
102 }
103 };

```

使用例

Range Minimum Query (RMQ)

```

1 int op(int a, int b) { return min(a, b); }
2 int e() { return INT32_MAX; }
3
4 int n;
5 SegmentTree<int, op, e> seg(n);

```

5.2 遅延評価セグメント木

モノイド S と、 S に対する作用素 $f : S \rightarrow S$ に対し利用できるデータ構造。

長さ N の S の配列に対し、

- 区間 $[l, r)$ の要素に一括で f を作用 ($a_i \leftarrow f(a_i), l \leq i < r$)
- 区間 $[l, r)$ の要素の総積の取得

を $O(\log N)$ で行うことができる。

```

1 template <class S,
2         S (*op)(S, S),
3         S (*e)(),
4         class F,
5         S (*mapping)(F, S),
6         F (*composition)(F, F),
7         F (*id)()>
8 struct LazySegmentTree {
9     private:
10         int _n, size, log;
11         vector<S> dat;
12         vector<F> lz;
13         void update(int k) { dat[k] = op(dat[2 * k], dat[2 * k + 1]); }
14         void all_apply(int k, F f) {
15             dat[k] = mapping(f, dat[k]);
16             if (k < size) lz[k] = composition(f, lz[k]);
17         }
18         void push(int k) {
19             all_apply(2 * k, lz[k]);
20             all_apply(2 * k + 1, lz[k]);
21             lz[k] = id();
22         }
23         int lower_bits(int x, int k) { return x & ((1 << k) - 1); }
24
25     public:
26         LazySegmentTree() : LazySegmentTree(0) {}
27         LazySegmentTree(int n) : LazySegmentTree(vector<S>(n, e())) {}
28         LazySegmentTree(const vector<S>& v) : _n(int(v.size())) {
29             log = 0;
30             while ((1 << log) < _n) log++;
31             size = 1 << log;
32             dat = vector<S>(2 * size, e());
33             lz = vector<F>(size, id());
34             for (int i = 0; i < _n; i++) dat[size + i] = v[i];
35             for (int i = size - 1; i >= 1; i--) update(i);
36         }
37         // a[p] = x
38         void set(int p, S x) {

```



```

39     p += size;
40     for (int i = log; i >= 1; i--) push(p >> i);
41     dat[p] = x;
42     for (int i = 1; i <= log; i++) update(p >> i);
43 }
44 // return a[p]
45 S get(int p) {
46     p += size;
47     for (int i = log; i >= 1; i--) push(p >> i);
48     return dat[p];
49 }
50 // return op(a[l], ..., a[r-1])
51 S prod(int l, int r) {
52     if (l == r) return e();
53     l += size;
54     r += size;
55     for (int i = log; i >= 1; i--) {
56         if (lower_bits(l, i) > 0) push(l >> i);
57         if (lower_bits(r, i) > 0) push((r - 1) >> i);
58     }
59     S sml = e(), smr = e();
60     while (l < r) {
61         if (l & 1) sml = op(sml, dat[l++]);
62         if (r & 1) smr = op(dat[--r], smr);
63         l >>= 1;
64         r >>= 1;
65     }
66     return op(sml, smr);
67 }
68 S all_prod() { return dat[1]; }
69 // a[p] = f(a[p])
70 void apply(int p, F f) {
71     p += size;
72     for (int i = log; i >= 1; i--) push(p >> i);
73     dat[p] = mapping(f, dat[p]);
74     for (int i = 1; i <= log; i++) update(p >> i);
75 }
76 // i = l...r-1 について a[i] = f(a[i])
77 void apply(int l, int r, F f) {
78     if (l == r) return;
79     l += size;
80     r += size;
81     for (int i = log; i >= 1; i--) {
82         if (lower_bits(l, i) > 0) push(l >> i);
83         if (lower_bits(r, i) > 0) push((r - 1) >> i);
84     }
85     int l2 = l, r2 = r;
86     while (l < r) {
87         if (l & 1) all_apply(l++, f);
88         if (r & 1) all_apply(--r, f);
89         l >>= 1;
90         r >>= 1;
91     }
92     l = l2;
93     r = r2;
94     for (int i = 1; i <= log; i++) {
95         if (lower_bits(l, i) > 0) update(l >> i);
96         if (lower_bits(r, i) > 0) update((r - 1) >> i);
97     }
98 }
99 // SegmentTree上の二分探索 (必要な場合)
100 // return r, f(op(a[l], ..., a[r-1])) == true
101 template <bool (*g)(S)>
102 int max_right(int l) {
103     return max_right(l, [](S x) { return g(x); });
104 }
105 template <class G>
106 int max_right(int l, G g) {

```

```

107     assert(g(e()));
108     if (l == _n) return _n;
109     l += size;
110     for (int i = log; i >= 1; i--) push(l >> i);
111     S sm = e();
112     do {
113         while (l % 2 == 0) l >>= 1;
114         if (!g(op(sm, dat[l]))) {
115             while (l < size) {
116                 push(l);
117                 l = (2 * l);
118                 if (g(op(sm, dat[l]))) {
119                     sm = op(sm, dat[l]);
120                     l++;
121                 }
122             }
123             return l - size;
124         }
125         sm = op(sm, dat[l]);
126         l++;
127     } while ((l & -l) != l);
128     return _n;
129 }
130 // return l, f(op(a[l], ..., a[r-1])) == true
131 template <bool (*g)(S)>
132 int min_left(int r) {
133     return min_left(r, [](S x) { return g(x); });
134 }
135 template <class G>
136 int min_left(int r, G g) {
137     assert(g(e()));
138     if (r == 0) return 0;
139     r += size;
140     for (int i = log; i >= 1; i--) push((r - 1) >> i);
141     S sm = e();
142     do {
143         r--;
144         while (r > 1 && (r % 2)) r >>= 1;
145         if (!g(op(dat[r], sm))) {
146             while (r < size) {
147                 push(r);
148                 r = (2 * r + 1);
149                 if (g(op(dat[r], sm))) {
150                     sm = op(dat[r], sm);
151                     r--;
152                 }
153             }
154             return r + 1 - size;
155         }
156         sm = op(dat[r], sm);
157     } while ((r & -r) != r);
158     return 0;
159 }
160 };

```

5.2.1 使用例

Range Update & Range Minimum Query

```

1 constexpr int INF = INT32_MAX;
2 constexpr int ID = INT32_MAX;
3
4 int op(int a, int b) { return min(a, b); }
5 int e() { return INF; }
6 int mapping(int f, int a) { return (f == ID ? a : f); }
7 int composition(int f, int g) { return (f == ID ? g : f); }
8 int id() { return ID; }
9

```

```

10 int n;
11 LazySegmentTree<int, op, e, int, mapping, composition, id> seg(n);

```

Range Add & Range Sum Query

```

1 using S = pair<ll, ll>;
2
3 S op(S a, S b) { return S(a.first + b.first, a.second + b.second); }
4 S e() { return P(0, 0); }
5 S mapping(ll f, S x) { return S(x.first + f * x.second, x.second); }
6 ll composition(ll f, ll g) { return f + g; }
7 ll id() { return 0; }
8
9 int n;
10 vector<S> a(n, S(0, 1));
11 LazySegmentTree<S, op, e, ll, mapping, composition, id> seg(a);

```

Range Add & Range Minimum Query

```

1 int op(int a, int b) { return min(a, b); }
2 int e() { return INT32_MAX; }
3 int mapping(int f, int x) { return x + f; }
4 int composition(int f, int g) { return f + g; }
5 int id() { return 0; }
6
7 vector<int> a(n, 0);
8 LazySegmentTree<int, op, e, int, mapping, composition, id> seg(a);

```

Range Update & Range Sum Query

```

1 using S = pair<ll, ll>;
2 constexpr int ID = INT32_MAX;
3
4 S op(S a, S b) { return S(a.first + b.first, a.second + b.second); }
5 S e() { return S(0, 0); }
6 S mapping(int f, S x) { return (f == ID ? x : S(f * x.second, x.second)); }
7 int composition(int f, int g) { return (f == ID ? g : f); }
8 int id() { return ID; }
9
10 int n;
11 vector<S> a(n, S(0, 1));
12 LazySegmentTree<S, op, e, int, mapping, composition, id> seg(a);

```

6 幾何

6.1 3D Geometry Template

三次元幾何のライブラリを利用するために必要となるクラスや関数などをまとめたものです。

```

1 #define EPS (1e-7)
2 #define equals(a, b) (fabs((a) - (b)) < EPS)
3
4 class Point3d {
5 public:
6     double x, y, z;
7
8     Point3d(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}
9
10    Point3d operator+(const Point3d& a) {
11        return Point3d(x + a.x, y + a.y, z + a.z);
12    }
13    Point3d operator-(const Point3d& a) {
14        return Point3d(x - a.x, y - a.y, z - a.z);
15    }
16    Point3d operator*(const double& d) {
17        return Point3d(x * d, y * d, z * d);
18    }

```

```

19 Point3d operator/(const double& d) {
20     return Point3d(x / d, y / d, z / d);
21 }
22
23 bool operator<(const Point3d& p) const {
24     if (!equals(x, p.x)) return x < p.x;
25     if (!equals(y, p.y)) return y < p.y;
26     if (!equals(z, p.z)) return z < p.z;
27     return false;
28 }
29
30 bool operator==(const Point3d& p) const {
31     return equals(x, p.x) && equals(y, p.y) && equals(z, p.z);
32 }
33 };
34
35 struct Segment3d {
36     Point3d p[2];
37     Segment3d(Point3d p1 = Point3d(), Point3d p2 = Point3d()) {
38         p[0] = p1, p[1] = p2;
39     }
40     bool operator==(const Segment3d& seg) const {
41         return (p[0] == seg.p[0] && p[1] == seg.p[1]) || (p[0] == seg.p[1] && p[1] == seg.p[0]);
42     }
43 };
44
45 using Line3d = Segment3d;
46 using Vector3d = Point3d;
47
48 ostream& operator<<(ostream& os, const Point3d& p) {
49     return os << "(" << p.x << "," << p.y << "," << p.z << ")";
50 }
51
52 ostream& operator<<(ostream& os, const Segment3d& seg) {
53     return os << "(" << seg.p[0] << "," << seg.p[1] << ")";
54 }
55
56 double dot(const Point3d& a, const Point3d& b) {
57     return a.x * b.x + a.y * b.y + a.z * b.z;
58 }
59
60 Vector3d cross(const Point3d& a, const Point3d& b) {
61     return Vector3d(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
62 }
63
64 inline double norm(const Point3d& p) {
65     return p.x * p.x + p.y * p.y + p.z * p.z;
66 }
67
68 inline double abs(const Point3d& p) {
69     return sqrt(norm(p));
70 }
71
72 inline double toRad(double theta) {
73     return theta * M_PI / 180.0;
74 }
75
76 double distanceLP(Line3d line, Point3d p) {
77     return abs(cross(line.p[1] - line.p[0], p - line.p[0])) / abs(line.p[1] - line.p[0]);
78 }
79
80 Point3d project(Segment3d seg, Point3d p) {
81     Vector3d base = seg.p[1] - seg.p[0];
82     double t = dot(p - seg.p[0], base) / norm(base);
83     return seg.p[0] + base * t;
84 }
85
86 Point3d reflect(Segment3d seg, Point3d p) {

```

```

87     return p + (project(seg, p) - p) * 2.0;
88 }
89
90 bool on_line3d(Line3d line, Point3d p) {
91     return equals(abs(cross(line.p[1] - p, line.p[0] - p)), 0);
92 }
93
94 bool on_segment3d(Segment3d seg, Point3d p) {
95     if (!on_line3d(seg, p)) return false;
96     double dist[3] = {abs(seg.p[1] - seg.p[0]), abs(p - seg.p[0]), abs(p - seg.p[1])};
97     return on_line3d(seg, p) && equals(dist[0], dist[1] + dist[2]);
98 }
99
100 double distanceSP(Segment3d seg, Point3d p) {
101     Point3d r = project(seg, p);
102     if (on_segment3d(seg, r)) return abs(p - r);
103     return min(abs(seg.p[0] - p), abs(seg.p[1] - p));
104 }

```

6.2 3D Plane

平面に対する操作をまとめたクラスファイルです。

```

1 class Plane3d {
2     public:
3         Point3d normal_vector; // 法線ベクトル
4         double d; // 平面方程式  $normal\_vector = (a,b,c), a*x + b*y + c*z + d = 0$ 
5
6         Plane3d(Point3d normal_vector = Point3d(), double d = 0) : normal_vector(normal_vector), d(d) {}
7         Plane3d(Vector3d a, Vector3d b, Vector3d c) {
8             Vector3d v1 = b - a;
9             Vector3d v2 = c - a;
10            Vector3d tmp = cross(v1, v2);
11            normal_vector = tmp / abs(tmp);
12            set_d(a);
13        }
14
15        // 法線ベクトル  $normal\_vector$  と平面上の 1 点から  $d$  を計算する
16        void set_d(Point3d p) {
17            d = dot(normal_vector, p);
18        }
19
20        // 平面と点  $p$  の距離を求める
21        double distanceP(Point3d p) {
22            Point3d a = normal_vector * d; // 平面上の適当な点をつくる
23            return abs(dot(p - a, normal_vector));
24        }
25
26        // 平面上でもっとも点  $p$  と近い点を求める
27        Point3d nearest_point(Point3d p) {
28            Point3d a = normal_vector * d;
29            return p - (normal_vector * dot(p - a, normal_vector));
30        }
31
32        // 平面と線分が交差するか
33        bool intersectS(Segment3d seg) {
34            Point3d a = normal_vector * d;
35            double res1 = dot(a - seg.p[0], normal_vector);
36            double res2 = dot(a - seg.p[1], normal_vector);
37            if (res1 > res2) swap(res1, res2);
38            if ((equals(res1, 0.0) || res1 < 0) && (equals(res2, 0.0) || res2 > 0)) return true;
39            return false;
40        }
41
42        // 平面と線分の交点を求める
43        Point3d crosspointS(Segment3d seg) {
44            Point3d a = normal_vector * d;

```

```

45     double dot_p0a = fabs(dot(seg.p[0] - a, normal_vector));
46     double dot_p1a = fabs(dot(seg.p[1] - a, normal_vector));
47     if (equals(dot_p0a + dot_p1a, 0)) return seg.p[0];
48     return seg.p[0] + (seg.p[1] - seg.p[0]) * (dot_p0a / (dot_p0a + dot_p1a));
49 }
50 };

```

6.3 3D Point on the Triangle

平面上の三角形 (tri1, tri2, tri3) と点 (p) を入力として受け取り、その点が三角形上に存在するかどうか判定します。

```

1  bool point_on_the_triangle3d(Point3d tri1, Point3d tri2, Point3d tri3, Point3d p) {
2      // 線分上に p があった場合、三角形内とみなす場合は以下のコメントアウトを外す
3      /*
4      if( on_segment3d(Segment3d(tri1,tri2),p) ) return true;
5      if( on_segment3d(Segment3d(tri2,tri3),p) ) return true;
6      if( on_segment3d(Segment3d(tri3,tri1),p) ) return true;
7      */
8
9      vector<Point3d> vec(3);
10     vec[0] = tri1, vec[1] = tri2, vec[2] = tri3;
11     double area = 0;
12     {
13         double a = abs(vec[0] - vec[1]), b = abs(vec[1] - vec[2]), c = abs(vec[2] - vec[0]);
14         double s = (a + b + c) / 2;
15         area = sqrt(s * (s - a) * (s - b) * (s - c));
16     }
17     double sum = 0;
18     for (int i = 0; i < 3; ++i) {
19         double a = abs(vec[i] - vec[(i + 1) % 3]), b = abs(vec[(i + 1) % 3] - p), c = abs(p - vec[i]);
20         double s = (a + b + c) / 2;
21         sum += sqrt(s * (s - a) * (s - b) * (s - c));
22     }
23     return equals(sum, area);
24 }

```

6.4 3D Libraries for Lines and Segments

平面上の直線と線分に関するライブラリです。ライブラリ中では直線は Line3d、線分は Segment3d として表記されます。

```

1  // 直線 l1 と l2 は平行か？
2  bool isParallel(Line3d l1, Line3d l2) {
3      Vector3d A = l1.p[0], B = l1.p[1], C = l2.p[0], D = l2.p[1];
4      Vector3d AB = B - A, CD = D - C;
5      Vector3d n1 = AB / abs(AB), n2 = CD / abs(CD);
6      double tmp = dot(n1, n2);
7      tmp = 1 - tmp * tmp;
8      return equals(tmp, 0.0);
9  }
10
11 // 直線 l1 と l2 を結ぶような線分であって最も距離が短いものを返す
12 // Note: l1 と l2 が平行な時には使用できないので注意
13 Segment3d nearest_segmentLL(Line3d l1, Line3d l2) {
14     assert(!isParallel(l1, l2)); // 平行な場合は使用不可
15     // l1.p[0] = A, l1.p[1] = B, l2.p[0] = C, l2.p[1] = D
16     Vector3d AB = l1.p[1] - l1.p[0];
17     Vector3d CD = l2.p[1] - l2.p[0];
18     Vector3d AC = l2.p[0] - l1.p[0];
19     Vector3d n1 = AB / abs(AB), n2 = CD / abs(CD);
20     double d1 = (dot(n1, AC) - dot(n1, n2) * dot(n2, AC)) / (1.0 - pow(dot(n1, n2), 2));
21     double d2 = (dot(n1, n2) * dot(n1, AC) - dot(n2, AC)) / (1.0 - pow(dot(n1, n2), 2));
22     return Segment3d(l1.p[0] + n1 * d1, l2.p[0] + n2 * d2);
23 }
24

```

```

25 // 直線 l1 と l2 は交差するか?
26 bool intersectLL(Line3d l1, Line3d l2) {
27     Vector3d A = l1.p[0], B = l1.p[1], C = l2.p[0], D = l2.p[1];
28
29     // そもそも l1, l2 が直線じゃない
30     if (equals(abs(B - A), 0.0) || equals(abs(D - C), 0.0)) {
31         // この場合は注意
32         // そもそも与えられた線分が線分になっていないので、交差するかどうかは判定できない
33         return false;
34     }
35
36     Vector3d AB = B - A, CD = D - C;
37     Vector3d n1 = AB / abs(AB), n2 = CD / abs(CD);
38     double tmp = dot(n1, n2);
39     tmp = 1 - tmp * tmp;
40
41     if (equals(tmp, 0.0)) return 0; // 直線が平行
42
43     Segment3d ns = nearest_segmentLL(l1, l2);
44     if (ns.p[0] == ns.p[1]) return true;
45     return false;
46 }
47
48 // 線分 seg1 と seg2 は交差しているか?
49 bool intersectSS(Segment3d seg1, Segment3d seg2) {
50     if (isParallel(seg1, seg2)) return false;
51     Segment3d seg = nearest_segmentLL(seg1, seg2);
52     if (!(seg.p[0] == seg.p[1])) return false;
53     Point3d cp = seg.p[1];
54     return on_segment3d(seg1, cp) && on_segment3d(seg2, cp);
55 }

```

6.5 3D Intersection of Planes

2つの平面の交差判定等を行うライブラリです。

```

1 using P3db = pair<Point3d, bool>;
2
3 /*
4  [*] Input:
5     2つの平面 pl1, pl2
6  [*] Output:
7     2つの平面が交線をもつ場合 -> first:交線上の任意の1点, second:true
8     交線を持たない場合 -> first:empty, second:false
9  */
10 P3db intersectPlPl(const Plane3d& pl1, const Plane3d& pl2) {
11     Vector3d v = cross(pl1.normal_vector, pl2.normal_vector);
12     if (!equals(v.x, 0.0)) {
13         Point3d p(0,
14             (pl1.d * pl2.normal_vector.z - pl2.d * pl1.normal_vector.z) / v.x,
15             (pl1.d * pl2.normal_vector.y - pl2.d * pl1.normal_vector.y) / (-v.x));
16         return P3db(p, true);
17     }
18     if (!equals(v.y, 0.0)) {
19         Point3d p((pl1.d * pl2.normal_vector.z - pl2.d * pl1.normal_vector.z) / (-v.y),
20             0,
21             (pl1.d * pl2.normal_vector.x - pl2.d * pl1.normal_vector.x) / v.y);
22         return P3db(p, true);
23     }
24     if (!equals(v.z, 0.0)) {
25         Point3d p((pl1.d * pl2.normal_vector.y - pl2.d * pl1.normal_vector.y) / v.z,
26             (pl1.d * pl2.normal_vector.x - pl2.d * pl1.normal_vector.x) / (-v.z),
27             0);
28         return P3db(p, true);
29     }
30     return P3db(Point3d(), false); // 平行なのでそのような交線は存在しない
31 }

```

```
32
33 /*
34 [*] Input:
35     2つの平面 plane, plane2 とその交線上の任意の1点
36 [*] Output:
37     2つの平面の交線
38
39 説明:
40 2つの平面の外積から交線の方法ベクトルを得る
41 あとは任意の1点に拡張した方向ベクトルを加えてセグメント化する
42 交線上の任意の1点は intersectPlPl で取得できる
43 面倒な仕様になってしまった
44 */
45 Line3d intersectPlPl_converter(Plane3d plane, Plane3d plane2, Point3d tmp) {
46     Vector3d ve = cross(plane.normal_vector, plane2.normal_vector);
47     return Line3d(tmp, tmp + (ve * 10)); // 任意の倍数で拡張、ここでは10
48 }
```