



Alfred  
代码模版库

# 目录

<b>1 数据结构</b>	<b>3</b>	3.6 一般图最大匹配 (Graph, 带花树算法)	10
1.1 珂朵莉树	3	3.7 2-SAT	12
1.2 树状数组	3	<b>4 Watashi 代码库 (备用)</b>	<b>13</b>
<b>2 比赛配置 and 奇技淫巧</b>	<b>4</b>	4.1 $O(n \log n) - O(1)$ RMQ	13
2.1 多组数据代码模板	4	4.2 $O(n \log n) - O(\log n)$ LCA	14
2.2 快读快写	4	4.3 树状数组	15
2.3 .clang-format	5	4.4 并查集	16
<b>3 jiangly 代码库 (备用, 侵权请提出 issue)</b>	<b>5</b>	4.5 轻重权树剖分	16
3.1 int128 输出流自定义	5	4.6 强连通分量	17
3.2 常用数学运算库函数及 gcd 重载	6	4.7 双连通分量	18
3.3 强连通分量缩点 (SCC)	6	4.8 二分图匹配	20
3.4 割边与割边缩点 (EBCC)	7	4.9 最小费用最大流	21
3.5 二分图最大权匹配 (MaxAssignment, 基于 KM)	8	4.10 AhoCorasick 自动机	23
		4.11 后缀数组	24
		4.12 LU 分解	25
		<b>5 对一类问题的处理方法</b>	<b>26</b>

# 1 数据结构

## 1.1 珂朵莉树

支持区间推平，颜色段统计，在随机数据下期望复杂度为  $O(n \log n)$  的暴力数据结构。

Listing 1: chtholly.cpp

```

1  #include <set>
2
3  struct ChthollyTree {
4      typedef long long ll;
5      struct Node {
6          mutable ll l, r, v;
7          inline bool operator<(const Node &x) const { return l < x.l; }
8      };
9      std::set<Node> tr;
10     typedef std::set<Node>::iterator iterator;
11     ChthollyTree(void) = default;
12     ChthollyTree(int rng, int val) { init(rng, val); }
13     inline void init(ll rng, ll val) noexcept {
14         tr.insert({l, rng, val}), tr.insert({rng + 1, rng + 1, 0});
15     }
16     inline iterator begin(void) const noexcept { return tr.begin(); }
17     inline iterator end(void) const noexcept { return tr.end(); }
18     inline iterator split(ll pos) {
19         auto it = tr.lower_bound({pos, 0, 0});
20         if (it != tr.end() && it->l == pos) return it;
21         ll l = (--it)->l, r = it->r, v = it->v;
22         tr.erase(it), tr.insert({l, pos - 1, v});
23         return tr.insert({pos, r, v}).first;
24     }
25     inline void assign(ll l, ll r, ll v) {
26         auto R = split(r + 1), L = split(l);
27         tr.erase(L, R), tr.insert({l, r, v});
28     }
29     template <class _Functor> // func(iterator)
30     inline void modify(ll l, ll r, _Functor func) {
31         auto R = split(r + 1), L = split(l);
32         for (auto it = L; it != R; it++) func(it);
33     }
34     template <class _Functor> // func(ll &, iterator)
35     inline ll query(ll l, ll r, _Functor func) {
36         ll ans = 0;
37         auto R = split(r + 1);
38         for (auto it = split(l); it != R; it++) func(ans, it);
39         return ans;
40     }
41 };

```

## 1.2 树状数组

维护满足结合律且可差分信息的，常数较小的数据结构。

Listing 2: fenwick.cpp

```

1  #include <vector>
2
3  template <class T>
4  struct Fenwick {
5      std::vector<T> c;
6      inline int lowbit(int x) { return x & -x; }
7      inline void merge(T &x, T y) { x = x + y; } // remember to modify
8      inline T subtract(T x, T y) { return x - y; }
9      inline void update(size_t pos, T x) {
10         for (pos++; pos < c.size(); pos += lowbit(pos)) merge(c[pos], x);

```

```

11     }
12     inline void clear(void) {
13         for (auto &x : c) x = T();
14     }
15     inline T query(size_t pos) {
16         T ans = T();
17         for (pos++; pos; pos ^= lowbit(pos)) merge(ans, c[pos]);
18         return ans;
19     }
20     inline T query(size_t l, size_t r) {
21         return subtract(query(r), query(l - 1));
22     }
23     Fenwick(size_t len) : c(len + 2) {}
24 };

```

## 2 比赛配置 and 奇技淫巧

### 2.1 多组数据代码模板

Listing 3: template.cpp

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using i64 = long long;
4  const i64 N = 1e5 + 10;
5  int t = 1;
6  inline void solve(int Case) {
7      // your code here;
8  }
9  inline void optimizeIO(void) {
10     ios::sync_with_stdio(false);
11     cin.tie(NULL), cout.tie(NULL);
12 }
13 inline void init(void) {}
14 int main(int argc, char const *argv[]) {
15     optimizeIO(), init(), cin >> t;
16     for (int i = 1; i <= t; i++) solve(i);
17     return 0;
18 }

```

### 2.2 快读快写

Listing 4: fast-io.cpp

```

1  namespace fastIO {
2      char c, f, e = 0;
3      namespace usr {
4          template <class _Tp>
5          inline int read(_Tp &x) {
6              x = f = 0, c = getchar();
7              while (!isdigit(c) && !e) f = c == '-', e |= c == EOF, c = getchar();
8              while (isdigit(c) && !e) x = (x << 1) + (x << 3) + (c ^ 48), c = getchar();
9              return (e |= c == EOF) ? 0 : ((f ? x = -x : 0), 1);
10         }
11         template <class _Tp>
12         inline void write(_Tp x) {
13             if (x < 0) putchar('-'), x = -x;
14             if (x > 9) write(x / 10);
15             putchar((x % 10) ^ 48);
16         }
17         template <typename T, typename... V>
18         inline void read(T &t, V &...v) { read(t), read(v...); }
19         template <typename T, typename... V>
20         inline void write(T t, V... v) {

```

```

21         write(t), putchar('_'), write(v...);
22     }
23 }
24 }
25 using namespace fastIO::usr;

```

## 2.3 .clang-format

Listing 5: .clang-format

```

1 BasedOnStyle: LLVM
2 AlignAfterOpenBracket: BlockIndent
3 # AlignConsecutiveAssignments: Consecutive
4 AlignArrayOfStructures: Right
5 UseTab: Never
6 IndentWidth: 4
7 TabWidth: 4
8 BreakBeforeBraces: Attach
9 AllowShortIfStatementsOnASingleLine: AllIfsAndElse
10 AllowShortLoopsOnASingleLine: true
11 AllowShortBlocksOnASingleLine: true
12 IndentCaseLabels: true
13 ColumnLimit: 0
14 AccessModifierOffset: -4
15 NamespaceIndentation: All
16 FixNamespaceComments: false
17 AllowShortCaseLabelsOnASingleLine: true
18 AlwaysBreakTemplateDeclarations: MultiLine
19 BinPackParameters: true
20 BraceWrapping:
21     AfterCaseLabel: true
22     AfterClass: true
23 AlignConsecutiveMacros: AcrossEmptyLinesAndComments
24 AlignTrailingComments: Always

```

## 3 jiangly 代码库 (备用, 侵权请提出 issue)

### 3.1 int128 输出流自定义

Listing 6: others/i128-stream.cpp

```

1 #include <iostream>
2
3 using i128 = __int128;
4
5 std::istream &operator>>(std::istream is, i128 &n) {
6     std::string s;
7     is >> s;
8     for (auto c : s) {
9         n = n * 10 + (c - '0');
10    }
11    return is;
12 }
13
14 std::ostream &operator<<(std::ostream &os, i128 n) {
15     std::string s;
16     while (n) {
17         s += '0' + n % 10;
18         n /= 10;
19     }
20     std::reverse(s.begin(), s.end());
21     return os << s;
22 }

```

### 3.2 常用数学运算库函数及 gcd 重载

Listing 7: others/clf.cpp

```

1  using i64 = long long;
2  using i128 = __int128;
3
4  i64 ceilDiv(i64 n, i64 m) {
5      if (n >= 0) {
6          return (n + m - 1) / m;
7      } else {
8          return n / m;
9      }
10 }
11
12 i64 floorDiv(i64 n, i64 m) {
13     if (n >= 0) {
14         return n / m;
15     } else {
16         return (n - m + 1) / m;
17     }
18 }
19
20 template <class T>
21 void chmax(T &a, T b) {
22     if (a < b) a = b;
23 }
24
25 i128 gcd(i128 a, i128 b) {
26     return b ? gcd(b, a % b) : a;
27 }

```

### 3.3 强连通分量缩点 (SCC)

Listing 8: graph/scc.cpp

```

1  #include <vector>
2
3  struct SCC {
4      int n;
5      std::vector<std::vector<int>> adj;
6      std::vector<int> stk;
7      std::vector<int> dfn, low, bel;
8      int cur, cnt;
9
10     SCC() {}
11     SCC(int n) {
12         init(n);
13     }
14
15     void init(int n) {
16         this->n = n;
17         adj.assign(n, {});
18         dfn.assign(n, -1);
19         low.resize(n);
20         bel.assign(n, -1);
21         stk.clear();
22         cur = cnt = 0;
23     }
24
25     void addEdge(int u, int v) {
26         adj[u].push_back(v);
27     }
28
29     void dfs(int x) {
30         dfn[x] = low[x] = cur++;

```

```

31     stk.push_back(x);
32
33     for (auto y : adj[x]) {
34         if (dfn[y] == -1) {
35             dfs(y);
36             low[x] = std::min(low[x], low[y]);
37         } else if (bel[y] == -1) {
38             low[x] = std::min(low[x], dfn[y]);
39         }
40     }
41
42     if (dfn[x] == low[x]) {
43         int y;
44         do {
45             y = stk.back();
46             bel[y] = cnt;
47             stk.pop_back();
48         } while (y != x);
49         cnt++;
50     }
51 }
52
53 std::vector<int> work() {
54     for (int i = 0; i < n; i++) {
55         if (dfn[i] == -1) {
56             dfs(i);
57         }
58     }
59     return bel;
60 }
61 };

```

### 3.4 割边与割边缩点 (EBCC)

Listing 9: graph/ebcc.cpp

```

1  #include <set>
2  #include <vector>
3
4  std::set<std::pair<int, int>> E;
5
6  struct EBCC {
7      int n;
8      std::vector<std::vector<int>> adj;
9      std::vector<int> stk;
10     std::vector<int> dfn, low, bel;
11     int cur, cnt;
12
13     EBCC() {}
14     EBCC(int n) {
15         init(n);
16     }
17
18     void init(int n) {
19         this->n = n;
20         adj.assign(n, {});
21         dfn.assign(n, -1);
22         low.resize(n);
23         bel.assign(n, -1);
24         stk.clear();
25         cur = cnt = 0;
26     }
27
28     void addEdge(int u, int v) {
29         adj[u].push_back(v);
30         adj[v].push_back(u);

```

```

31     }
32
33     void dfs(int x, int p) {
34         dfn[x] = low[x] = cur++;
35         stk.push_back(x);
36
37         for (auto y : adj[x]) {
38             if (y == p) {
39                 continue;
40             }
41             if (dfn[y] == -1) {
42                 E.emplace(x, y);
43                 dfs(y, x);
44                 low[x] = std::min(low[x], low[y]);
45             } else if (bel[y] == -1 && dfn[y] < dfn[x]) {
46                 E.emplace(x, y);
47                 low[x] = std::min(low[x], dfn[y]);
48             }
49         }
50
51         if (dfn[x] == low[x]) {
52             int y;
53             do {
54                 y = stk.back();
55                 bel[y] = cnt;
56                 stk.pop_back();
57             } while (y != x);
58             cnt++;
59         }
60     }
61
62     std::vector<int> work() {
63         dfs(0, -1);
64         return bel;
65     }
66
67     struct Graph {
68         int n;
69         std::vector<std::pair<int, int>> edges;
70         std::vector<int> siz;
71         std::vector<int> cnte;
72     };
73     Graph compress() {
74         Graph g;
75         g.n = cnt;
76         g.siz.resize(cnt);
77         g.cnte.resize(cnt);
78         for (int i = 0; i < n; i++) {
79             g.siz[bel[i]]++;
80             for (auto j : adj[i]) {
81                 if (bel[i] < bel[j]) {
82                     g.edges.emplace_back(bel[i], bel[j]);
83                 } else if (i < j) {
84                     g.cnte[bel[i]]++;
85                 }
86             }
87         }
88         return g;
89     }
90 };

```

### 3.5 二分图最大权匹配 (MaxAssignment, 基于 KM)

Listing 10: graph/biggraph-weight-match.cpp

```
1 #include <queue>
```



```

2  #include <vector>
3
4  template <class T>
5  struct MaxAssignment {
6  public:
7      T solve(int nx, int ny, std::vector<std::vector<T>> a) {
8          assert(0 <= nx && nx <= ny);
9          assert(int(a.size()) == nx);
10         for (int i = 0; i < nx; ++i) {
11             assert(int(a[i].size()) == ny);
12             for (auto x : a[i])
13                 assert(x >= 0);
14         }
15
16         auto update = [&](int x) {
17             for (int y = 0; y < ny; ++y) {
18                 if (lx[x] + ly[y] - a[x][y] < slack[y]) {
19                     slack[y] = lx[x] + ly[y] - a[x][y];
20                     slackx[y] = x;
21                 }
22             }
23         };
24
25         costs.resize(nx + 1);
26         costs[0] = 0;
27         lx.assign(nx, std::numeric_limits<T>::max());
28         ly.assign(ny, 0);
29         xy.assign(nx, -1);
30         yx.assign(ny, -1);
31         slackx.resize(ny);
32         for (int cur = 0; cur < nx; ++cur) {
33             std::queue<int> que;
34             visx.assign(nx, false);
35             visy.assign(ny, false);
36             slack.assign(ny, std::numeric_limits<T>::max());
37             p.assign(nx, -1);
38
39             for (int x = 0; x < nx; ++x) {
40                 if (xy[x] == -1) {
41                     que.push(x);
42                     visx[x] = true;
43                     update(x);
44                 }
45             }
46
47             int ex, ey;
48             bool found = false;
49             while (!found) {
50                 while (!que.empty() && !found) {
51                     auto x = que.front();
52                     que.pop();
53                     for (int y = 0; y < ny; ++y) {
54                         if (a[x][y] == lx[x] + ly[y] && !visy[y]) {
55                             if (yx[y] == -1) {
56                                 ex = x;
57                                 ey = y;
58                                 found = true;
59                                 break;
60                             }
61                             que.push(yx[y]);
62                             p[yx[y]] = x;
63                             visy[y] = visx[yx[y]] = true;
64                             update(yx[y]);
65                         }
66                     }
67                 }

```

```

68         if (found)
69             break;
70
71         T delta = std::numeric_limits<T>::max();
72         for (int y = 0; y < ny; ++y)
73             if (!visy[y])
74                 delta = std::min(delta, slack[y]);
75         for (int x = 0; x < nx; ++x)
76             if (visx[x])
77                 lx[x] -= delta;
78         for (int y = 0; y < ny; ++y) {
79             if (visy[y]) {
80                 ly[y] += delta;
81             } else {
82                 slack[y] -= delta;
83             }
84         }
85         for (int y = 0; y < ny; ++y) {
86             if (!visy[y] && slack[y] == 0) {
87                 if (yx[y] == -1) {
88                     ex = slackx[y];
89                     ey = y;
90                     found = true;
91                     break;
92                 }
93                 que.push(yx[y]);
94                 p[yx[y]] = slackx[y];
95                 visy[y] = visx[yx[y]] = true;
96                 update(yx[y]);
97             }
98         }
99     }
100
101     costs[cur + 1] = costs[cur];
102     for (int x = ex, y = ey, ty; x != -1; x = p[x], y = ty) {
103         costs[cur + 1] += a[x][y];
104         if (xy[x] != -1)
105             costs[cur + 1] -= a[x][xy[x]];
106         ty = xy[x];
107         xy[x] = y;
108         yx[y] = x;
109     }
110 }
111 return costs[nx];
112 }
113 std::vector<int> assignment() {
114     return xy;
115 }
116 std::pair<std::vector<T>, std::vector<T>> labels() {
117     return std::make_pair(lx, ly);
118 }
119 std::vector<T> weights() {
120     return costs;
121 }
122
123 private:
124     std::vector<T> lx, ly, slack, costs;
125     std::vector<int> xy, yx, p, slackx;
126     std::vector<bool> visx, visy;
127 };

```

### 3.6 一般图最大匹配 (Graph, 带花树算法)

Listing 11: graph/general-match.cpp

```
1 #include <queue>
```

```

2  #include <vector>
3
4  struct Graph {
5      int n;
6      std::vector<std::vector<int>> e;
7      Graph(int n) : n(n), e(n) {}
8      void addEdge(int u, int v) {
9          e[u].push_back(v);
10         e[v].push_back(u);
11     }
12     std::vector<int> findMatching() {
13         std::vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);
14
15         // disjoint set union
16         auto find = [&](int u) {
17             while (f[u] != u)
18                 u = f[u] = f[f[u]];
19             return u;
20         };
21
22         auto lca = [&](int u, int v) {
23             u = find(u);
24             v = find(v);
25             while (u != v) {
26                 if (dep[u] < dep[v])
27                     std::swap(u, v);
28                 u = find(link[match[u]]);
29             }
30             return u;
31         };
32
33         std::queue<int> que;
34         auto blossom = [&](int u, int v, int p) {
35             while (find(u) != p) {
36                 link[u] = v;
37                 v = match[u];
38                 if (vis[v] == 0) {
39                     vis[v] = 1;
40                     que.push(v);
41                 }
42                 f[u] = f[v] = p;
43                 u = link[v];
44             }
45         };
46
47         // find an augmenting path starting from u and augment (if exist)
48         auto augment = [&](int u) {
49             while (!que.empty())
50                 que.pop();
51
52             std::iota(f.begin(), f.end(), 0);
53
54             // vis = 0 corresponds to inner vertices, vis = 1 corresponds to outer vertices
55             std::fill(vis.begin(), vis.end(), -1);
56
57             que.push(u);
58             vis[u] = 1;
59             dep[u] = 0;
60
61             while (!que.empty()) {
62                 int u = que.front();
63                 que.pop();
64                 for (auto v : e[u]) {
65                     if (vis[v] == -1) {
66
67                         vis[v] = 0;

```

```

68         link[v] = u;
69         dep[v] = dep[u] + 1;
70
71         // found an augmenting path
72         if (match[v] == -1) {
73             for (int x = v, y = u, temp; y != -1; x = temp, y = x == -1 ? -1 : link[x]) {
74                 temp = match[y];
75                 match[x] = y;
76                 match[y] = x;
77             }
78             return;
79         }
80
81         vis[match[v]] = 1;
82         dep[match[v]] = dep[u] + 2;
83         que.push(match[v]);
84
85     } else if (vis[v] == 1 && find(v) != find(u)) {
86         // found a blossom
87         int p = lca(u, v);
88         blossom(u, v, p);
89         blossom(v, u, p);
90     }
91 }
92 }
93 };
94
95 // find a maximal matching greedily (decrease constant)
96 auto greedy = [&]() {
97     for (int u = 0; u < n; ++u) {
98         if (match[u] != -1)
99             continue;
100         for (auto v : e[u]) {
101             if (match[v] == -1) {
102                 match[u] = v;
103                 match[v] = u;
104                 break;
105             }
106         }
107     }
108 };
109
110 greedy();
111
112 for (int u = 0; u < n; ++u)
113     if (match[u] == -1)
114         augment(u);
115
116 return match;
117 }
118 };

```

### 3.7 2-SAT

Listing 12: graph/2-sat.cpp

```

1  #include <vector>
2
3  struct TwoSat {
4      int n;
5      std::vector<std::vector<int>>> e;
6      std::vector<bool> ans;
7      TwoSat(int n) : n(n), e(2 * n), ans(n) {}
8      void addClause(int u, bool f, int v, bool g) {
9          e[2 * u + !f].push_back(2 * v + g);
10         e[2 * v + !g].push_back(2 * u + f);

```

```

11     }
12     bool satisfiable() {
13         std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
14         std::vector<int> stk;
15         int now = 0, cnt = 0;
16         std::function<void(int)> tarjan = [&](int u) {
17             stk.push_back(u);
18             dfn[u] = low[u] = now++;
19             for (auto v : e[u]) {
20                 if (dfn[v] == -1) {
21                     tarjan(v);
22                     low[u] = std::min(low[u], low[v]);
23                 } else if (id[v] == -1) {
24                     low[u] = std::min(low[u], dfn[v]);
25                 }
26             }
27             if (dfn[u] == low[u]) {
28                 int v;
29                 do {
30                     v = stk.back();
31                     stk.pop_back();
32                     id[v] = cnt;
33                 } while (v != u);
34                 ++cnt;
35             }
36         };
37         for (int i = 0; i < 2 * n; ++i)
38             if (dfn[i] == -1) tarjan(i);
39         for (int i = 0; i < n; ++i) {
40             if (id[2 * i] == id[2 * i + 1]) return false;
41             ans[i] = id[2 * i] > id[2 * i + 1];
42         }
43         return true;
44     }
45     std::vector<bool> answer() { return ans; }
46 };

```

## 4 Watashi 代码库 (备用)

### 4.1 $O(n \log n) - O(1)$ RMQ

Listing 13: rmq.cpp

```

1  #include <algorithm> // copy
2  #include <climits>   // CHAR_BIT
3
4  using namespace std;
5
6  template <typename T>
7  struct RMQ {
8      int n;
9      vector<T> e;
10     vector<vector<int>> rmq;
11
12     static const int INT_BIT = sizeof(4) * CHAR_BIT;
13     static inline int LG2(int i) { return INT_BIT - 1 - __builtin_clz(i); }
14     static inline int BIN(int i) { return 1 << i; }
15
16     int cmp(int l, int r) const {
17         return e[l] <= e[r] ? l : r;
18     }
19
20     void init(int n, const T e[]) {
21         this->n = n;
22         vector<T> (e, e + n).swap(this->e);

```

```

23
24     int m = 1;
25     while (BIN(m) <= n) {
26         ++m;
27     }
28     vector<vector<int>>(m, vector<int>(n)).swap(rmq);
29
30     for (int i = 0; i < n; ++i) {
31         rmq[0][i] = i;
32     }
33     for (int i = 0; BIN(i + 1) <= n; ++i) {
34         for (int j = 0; j + BIN(i + 1) <= n; ++j) {
35             rmq[i + 1][j] = cmp(rmq[i][j], rmq[i][j + BIN(i)]);
36         }
37     }
38 }
39
40 int index(int l, int r) const {
41     int b = LG2(r - l);
42     return cmp(rmq[b][l], rmq[b][r - (1 << b)]);
43 }
44
45 T value(int l, int r) const {
46     return e[index(l, r)];
47 }
48 };

```

## 4.2 $O(n \log n) - O(\log n)$ LCA

Listing 14: lca.cpp

```

1  #include <algorithm>
2  #include <cstdio>
3  #include <vector>
4
5  using namespace std;
6
7  const int MAXM = 16;
8  const int MAXN = 1 << MAXM;
9
10 // LCA
11 struct LCA {
12     vector<int> e[MAXN];
13     int d[MAXN], p[MAXN][MAXM];
14
15     void dfs_(int v, int f) {
16         p[v][0] = f;
17         for (int i = 1; i < MAXM; ++i) {
18             p[v][i] = p[p[v][i - 1]][i - 1];
19         }
20         for (int i = 0; i < (int)e[v].size(); ++i) {
21             int w = e[v][i];
22             if (w != f) {
23                 d[w] = d[v] + 1;
24                 dfs_(w, v);
25             }
26         }
27     }
28
29     int up_(int v, int m) {
30         for (int i = 0; i < MAXM; ++i) {
31             if (m & (1 << i)) {
32                 v = p[v][i];
33             }
34         }
35         return v;

```

```

36     }
37
38     int lca(int a, int b) {
39         if (d[a] > d[b]) {
40             swap(a, b);
41         }
42         b = up_(b, d[b] - d[a]);
43         if (a == b) {
44             return a;
45         } else {
46             for (int i = MAXM - 1; i >= 0; --i) {
47                 if (p[a][i] != p[b][i]) {
48                     a = p[a][i];
49                     b = p[b][i];
50                 }
51             }
52             return p[a][0];
53         }
54     }
55
56     void init(int n) {
57         for (int i = 0; i < n; ++i) {
58             e[i].clear();
59         }
60     }
61
62     void add(int a, int b) {
63         e[a].push_back(b);
64         e[b].push_back(a);
65     }
66
67     void build() {
68         d[0] = 0;
69         dfs_(0, 0);
70     }
71 } lca;

```

### 4.3 树状数组

Listing 15: bit.cpp

```

1  #include <vector>
2
3  using namespace std;
4
5  template<typename T = int>
6  struct BIT {
7      vector<T> a;
8
9      void init(int n) {
10         vector<T>(n + 1).swap(a);
11     }
12
13     void add(int i, T v) {
14         for (int j = i + 1; j < (int)a.size(); j = (j | (j - 1)) + 1) {
15             a[j] += v;
16         }
17     }
18
19     // [0, i)
20     T sum(int i) const {
21         T ret = T();
22         for (int j = i; j > 0; j = j & (j - 1)) {
23             ret += a[j];
24         }
25         return ret;

```

```

26     }
27
28     T get(int i) const {
29         return sum(i + 1) - sum(i);
30     }
31
32     void set(int i, T v) {
33         add(i, v - get(i));
34     }
35 };

```

## 4.4 并查集

Listing 16: union-find.cpp

```

1  #include <vector>
2
3  using namespace std;
4
5  struct DisjointSet {
6      vector<int> p;
7
8      void init(int n) {
9          p.resize(n);
10         for (int i = 0; i < n; ++i) {
11             p[i] = i;
12         }
13     }
14
15     int getp(int i) {
16         return i == p[i] ? i : (p[i] = getp(p[i]));
17     }
18
19     bool setp(int i, int j) {
20         i = getp(i);
21         j = getp(j);
22         p[i] = j;
23         return i != j;
24     }
25 };

```

## 4.5 轻重权树剖分

Listing 17: chain-decomp.cpp

```

1  #include <cstdio>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int MAXM = 16;
8  const int MAXN = 1 << MAXM;
9
10 // Heavy-Light Decomposition
11 struct TreeDecomposition {
12     vector<int> e[MAXN], c[MAXN];
13     int s[MAXN];    // subtree size
14     int p[MAXN];    // parent id
15     int r[MAXN];    // chain root id
16     int t[MAXN];    // timestamp, index used in segtree
17     int ts;
18
19     void dfs_(int v, int f) {
20         p[v] = f;

```



```

21     s[v] = 1;
22     for (int i = 0; i < (int)e[v].size(); ++i) {
23         int w = e[v][i];
24         if (w != f) {
25             dfs_(w, v);
26             s[v] += s[w];
27         }
28     }
29 }
30
31 void decomp_(int v, int f, int k) {
32     t[v] = ts++;
33     c[k].push_back(v);
34     r[v] = k;
35
36     int x = 0, y = -1;
37     for (int i = 0; i < (int)e[v].size(); ++i) {
38         int w = e[v][i];
39         if (w != f) {
40             if (s[w] > x) {
41                 x = s[w];
42                 y = w;
43             }
44         }
45     }
46     if (y != -1) {
47         decomp_(y, v, k);
48     }
49
50     for (int i = 0; i < (int)e[v].size(); ++i) {
51         int w = e[v][i];
52         if (w != f && w != y) {
53             decomp_(w, v, w);
54         }
55     }
56 }
57
58 void init(int n) {
59     for (int i = 0; i < n; ++i) {
60         e[i].clear();
61     }
62 }
63
64 void add(int a, int b) {
65     e[a].push_back(b);
66     e[b].push_back(a);
67 }
68
69 void build() { // !!
70     ts = 0;
71     dfs_(0, 0);
72     decomp_(0, 0, 0);
73 }
74 } hld;

```

## 4.6 强连通分量

Listing 18: scc.cpp

```

1  #include <algorithm>
2  #include <stack>
3  #include <vector>
4
5  using namespace std;
6
7  struct SCCTarjan {

```

```

8     int n;
9     vector<vector<int>> e;
10
11     vector<int> id;
12     vector<vector<int>> scc;
13
14     void init(int n) {
15         this->n = n;
16         vector<vector<int>> (n).swap(e);
17         id.resize(n);
18         dfn.resize(n);
19         low.resize(n);
20     }
21
22     void add(int a, int b) {
23         e[a].push_back(b);
24     }
25
26     vector<int> dfn, low;
27     int timestamp;
28     stack<int> s;
29
30     void dfs(int v) {
31         dfn[v] = timestamp++;
32         low[v] = dfn[v];
33         s.push(v);
34         for (vector<int>::const_iterator w = e[v].begin(); w != e[v].end(); ++w) {
35             if (dfn[*w] == -1) {
36                 dfs(*w);
37                 low[v] = min(low[v], low[*w]);
38             } else if (dfn[*w] != -2) {
39                 low[v] = min(low[v], dfn[*w]);
40             }
41         }
42
43         if (low[v] == dfn[v]) {
44             vector<int> t;
45             do {
46                 int w = s.top();
47                 s.pop();
48                 id[w] = (int)scc.size();
49                 t.push_back(w);
50                 dfn[w] = -2;
51             } while (t.back() != v);
52             scc.push_back(t);
53         }
54     }
55
56     int gao() {
57         scc.clear();
58         stack<int>().swap(s);
59         timestamp = 0;
60
61         fill(dfn.begin(), dfn.end(), -1);
62         for (int i = 0; i < n; ++i) {
63             if (dfn[i] == -1) {
64                 dfs(i);
65             }
66         }
67         return (int)scc.size();
68     }
69 };

```

## 4.7 双连通分量

Listing 19: bcc.cpp

```

1  #include <algorithm>
2  #include <stack>
3  #include <utility>
4  #include <vector>
5
6  using namespace std;
7
8  // TODO: cannot handle duplicate edges
9  struct Tarjan {
10     int n;
11     vector<vector<int>> e;
12
13     vector<int> cut;
14     vector<pair<int, int>> bridge;
15     vector<vector<pair<int, int>>> bcc;
16
17     void init(int n) {
18         this->n = n;
19         e.clear();
20         e.resize(n);
21         dfn.resize(n);
22         low.resize(n);
23     }
24
25     void add(int a, int b) {
26         // assert(find(e[a].begin(), e[a].end(), b) == e[a].end());
27         e[a].push_back(b);
28         e[b].push_back(a);
29     }
30
31     vector<int> dfn, low;
32     int timestamp;
33     stack<pair<int, int>> s;
34
35     void dfs(int v, int p) {
36         int part = p == -1 ? 0 : 1;
37         dfn[v] = low[v] = timestamp++;
38         for (vector<int>::const_iterator w = e[v].begin(); w != e[v].end(); ++w) {
39             pair<int, int> f = make_pair(min(v, *w), max(v, *w));
40             if (dfn[*w] == -1) {
41                 s.push(f);
42                 dfs(*w, v);
43                 low[v] = min(low[v], low[*w]);
44                 if (dfn[v] <= low[*w]) {
45                     // articulation point
46                     if (++part == 2) {
47                         cut.push_back(v);
48                     }
49                     // articulation edge
50                     if (dfn[v] < low[*w]) {
51                         bridge.push_back(f);
52                     }
53                     // biconnected component (2-vertex-connected)
54                     vector<pair<int, int>> t;
55                     do {
56                         t.push_back(s.top());
57                         s.pop();
58                     } while (t.back() != f);
59                     bcc.push_back(t);
60                 }
61             } else if (*w != p && dfn[*w] < dfn[v]) {
62                 s.push(f);
63                 low[v] = min(low[v], dfn[*w]);
64             }
65         }

```

```

66     }
67
68     void gao() {
69         cut.clear();
70         bridge.clear();
71         bcc.clear();
72
73         timestamp = 0;
74         stack<pair<int, int>>().swap(s);
75         fill(dfn.begin(), dfn.end(), -1);
76
77         for (int i = 0; i < n; ++i) {
78             if (dfn[i] == -1) {
79                 dfs(i, -1);
80             }
81         }
82     }
83 };
84
85 struct BridgeBlockTree {
86     Tarjan<MAXN> bcc;
87     DisjointSet<MAXN> ds;
88     vector<int> e[MAXN];
89
90     void init(int n) {
91         bcc.init(n);
92         ds.init(n);
93     }
94
95     void add(int a, int b) {
96         bcc.add(a, b);
97     }
98
99     void gao() {
100         bcc.gao();
101         for (const auto &i : bcc.bcc) {
102             if (i.size() > 1) {
103                 for (const auto &j : i) {
104                     ds.setp(j.first, j.second);
105                 }
106             }
107         }
108         for (const auto &i : bcc.bridge) {
109             int a = ds.getp(i.first);
110             int b = ds.getp(i.second);
111             e[a].push_back(b);
112             e[b].push_back(a);
113         }
114     }
115
116     int id(int v) {
117         return ds.getp(v);
118     }
119 };

```

## 4.8 二分图匹配

Listing 20: bimatch.cpp

```

1 // maximum matchings in bipartite graphs
2 // maximum cardinality bipartite matching
3 //  $O(|V||E|)$ , generally fast
4
5 #include <algorithm>
6 #include <string>
7 #include <vector>

```

```

8
9 using namespace std;
10
11 struct Hungarian {
12     int nx, ny;
13     vector<int> mx, my;
14     vector<vector<int>> e;
15
16     void init(int nx, int ny) {
17         this->nx = nx;
18         this->ny = ny;
19         mx.resize(nx);
20         my.resize(ny);
21         e.clear();
22         e.resize(nx);
23         mark.resize(nx);
24     }
25
26     void add(int a, int b) {
27         e[a].push_back(b);
28     }
29
30     // vector<bool> is evil!!!
31     basic_string<bool> mark;
32
33     bool augment(int i) {
34         if (!mark[i]) {
35             mark[i] = true;
36             for (vector<int>::const_iterator j = e[i].begin(); j != e[i].end(); ++j) {
37                 if (my[*j] == -1 || augment(my[*j])) {
38                     mx[i] = *j;
39                     my[*j] = i;
40                     return true;
41                 }
42             }
43         }
44         return false;
45     }
46
47     int gao() {
48         int ret = 0;
49         fill(mx.begin(), mx.end(), -1);
50         fill(my.begin(), my.end(), -1);
51         for (int i = 0; i < nx; ++i) {
52             fill(mark.begin(), mark.end(), false);
53             if (augment(i)) {
54                 ++ret;
55             }
56         }
57         return ret;
58     }
59 };

```

## 4.9 最小费用最大流

Listing 21: flow.cpp

```

1 #include <algorithm>
2 #include <cstdio>
3 #include <limits>
4 #include <queue>
5 #include <vector>
6
7 using namespace std;
8
9 template <int MAXN, typename T = int, typename S = T>

```

```

10 struct MinCostMaxFlow {
11     struct NegativeCostCircuitExistsException {
12     };
13
14     struct Edge {
15         int v;
16         T c;
17         S w;
18         int b;
19         Edge(int v, T c, S w, int b) : v(v), c(c), w(w), b(b) {}
20     };
21
22     int n, source, sink;
23     vector<Edge> e[MAXN];
24
25     void init(int n, int source, int sink) {
26         this->n = n;
27         this->source = source;
28         this->sink = sink;
29         for (int i = 0; i < n; ++i) {
30             e[i].clear();
31         }
32     }
33
34     void addEdge(int a, int b, T c, S w) {
35         e[a].push_back(Edge(b, c, w, e[b].size()));
36         e[b].push_back(Edge(a, 0, -w, e[a].size() - 1)); // TODO
37     }
38
39     bool mark[MAXN];
40     T maxc[MAXN];
41     S minw[MAXN];
42     int dist[MAXN];
43     Edge *prev[MAXN];
44
45     bool _spfa() {
46         queue<int> q;
47         fill(mark, mark + n, false);
48         fill(maxc, maxc + n, 0);
49         fill(minw, minw + n, numeric_limits<S>::max());
50         fill(dist, dist + n, 0);
51         fill(prev, prev + n, (Edge *)NULL);
52         mark[source] = true;
53         maxc[source] = numeric_limits<S>::max();
54         minw[source] = 0;
55
56         q.push(source);
57         while (!q.empty()) {
58             int cur = q.front();
59             mark[cur] = false;
60             q.pop();
61             for (typename vector<Edge>::iterator it = e[cur].begin(); it != e[cur].end(); ++it) {
62                 T c = min(maxc[cur], it->c);
63                 if (c == 0) {
64                     continue;
65                 }
66
67                 int v = it->v;
68                 S w = minw[cur] + it->w;
69                 if (minw[v] > w || (minw[v] == w && maxc[v] < c)) { // TODO
70                     maxc[v] = c;
71                     minw[v] = w;
72                     dist[v] = dist[cur] + 1;
73                     if (dist[v] >= n) {
74                         return false;
75                     }

```

```

76         prev[v] = &*it;
77         if (!mark[v]) {
78             mark[v] = true;
79             q.push(v);
80         }
81     }
82 }
83 }
84 return true;
85 }
86
87 pair<T, S> gao() {
88     T sumc = 0;
89     S sumw = 0;
90     while (true) {
91         if (!_spfa()) {
92             throw NegativeCostCircuitExistsException();
93         } else if (maxc[sink] == 0) {
94             break;
95         } else {
96             T c = maxc[sink];
97             sumc += c;
98             sumw += c * minw[sink];
99
100             int cur = sink;
101             while (cur != source) {
102                 Edge *e1 = prev[cur];
103                 e1->c -= c;
104                 Edge *e2 = &e[e1->v][e1->b];
105                 e2->c += c;
106                 cur = e2->v;
107             }
108         }
109     }
110     return make_pair(sumc, sumw);
111 }
112 };

```

## 4.10 AhoCorasick 自动机

Listing 22: ac-automata.cpp

```

1  #include <algorithm>
2  #include <queue>
3
4  using namespace std;
5
6  struct AhoCorasick {
7      static const int NONE = 0;
8      static const int MAXN = 1024;
9      static const int CHARSET = 26;
10
11      int end;
12      int tag[MAXN];
13      int fail[MAXN];
14      int trie[MAXN][CHARSET];
15
16      void init() {
17          tag[0] = NONE;
18          fill(trie[0], trie[0] + CHARSET, -1);
19          end = 1;
20      }
21
22      int add(int m, const int *s) {
23          int p = 0;
24          for (int i = 0; i < m; ++i) {

```

```

25         if (trie[p][*s] == -1) {
26             tag[end] = NONE;
27             fill(trie[end], trie[end] + CHARSET, -1);
28             trie[p][*s] = end++;
29         }
30         p = trie[p][*s];
31         ++s;
32     }
33     return p;
34 }
35
36 void build(void) { // !!
37     queue<int> bfs;
38     fail[0] = 0;
39     for (int i = 0; i < CHARSET; ++i) {
40         if (trie[0][i] != -1) {
41             fail[trie[0][i]] = 0;
42             bfs.push(trie[0][i]);
43         } else {
44             trie[0][i] = 0;
45         }
46     }
47     while (!bfs.empty()) {
48         int p = bfs.front();
49         tag[p] |= tag[fail[p]];
50         bfs.pop();
51         for (int i = 0; i < CHARSET; ++i) {
52             if (trie[p][i] != -1) {
53                 fail[trie[p][i]] = trie[fail[p]][i];
54                 bfs.push(trie[p][i]);
55             } else {
56                 trie[p][i] = trie[fail[p]][i];
57             }
58         }
59     }
60 }
61 } ac;

```

## 4.11 后缀数组

Listing 23: sa.cpp

```

1  #include <algorithm>
2  #include <utility>
3  #include <vector>
4  using namespace std;
5
6  struct SuffixArray {
7      vector<int> sa, rank, height;
8
9      template <typename T>
10     void init(int n, const T a[]) {
11         sa.resize(n);
12         rank.resize(n);
13
14         vector<pair<T, int>> assoc(n);
15         for (int i = 0; i < n; ++i) {
16             assoc[i] = make_pair(a[i], i);
17         }
18         sort(assoc.begin(), assoc.end());
19         for (int i = 0; i < n; ++i) {
20             sa[i] = assoc[i].second;
21             if (i == 0 || assoc[i].first != assoc[i - 1].first) {
22                 rank[sa[i]] = i;
23             } else {
24                 rank[sa[i]] = rank[sa[i - 1]];

```



```

25     }
26 }
27
28 vector<int> tmp(n), cnt(n);
29 vector<pair<int, int>> suffix(n);
30 for (int m = 1; m < n; m <= 1) {
31     // snd
32     for (int i = 0; i < m; ++i) {
33         tmp[i] = n - m + i;
34     }
35     for (int i = 0, j = m; i < n; ++i) {
36         if (sa[i] >= m) {
37             tmp[j++] = sa[i] - m;
38         }
39     }
40     // fst
41     fill(cnt.begin(), cnt.end(), 0);
42     for (int i = 0; i < n; ++i) {
43         ++cnt[rank[i]];
44     }
45     partial_sum(cnt.begin(), cnt.end(), cnt.begin());
46     for (int i = n - 1; i >= 0; --i) {
47         sa[--cnt[rank[tmp[i]]]] = tmp[i];
48     }
49     //
50     for (int i = 0; i < n; ++i) {
51         suffix[i] = make_pair(rank[i], i + m < n ? rank[i + m] : numeric_limits<int>::min());
52     }
53     for (int i = 0; i < n; ++i) {
54         if (i == 0 || suffix[sa[i]] != suffix[sa[i - 1]]) {
55             rank[sa[i]] = i;
56         } else {
57             rank[sa[i]] = rank[sa[i - 1]];
58         }
59     }
60 }
61
62 height.resize(n);
63 for (int i = 0, z = 0; i < n; ++i) {
64     if (rank[i] == 0) {
65         height[0] = z = 0;
66     } else {
67         int x = i, y = sa[rank[i] - 1];
68         z = max(0, z - 1);
69         while (x + z < n && y + z < n && a[x + z] == a[y + z]) {
70             ++z;
71         }
72         height[rank[i]] = z;
73     }
74 }
75 }
76 };

```

## 4.12 LU 分解

Listing 24: lu.cpp

```

1  const int MAXN = 128;
2  const double EPS = 1e-10;
3
4  void LU(int n, double a[MAXN][MAXN], int r[MAXN], int c[MAXN]) {
5      for (int i = 0; i < n; ++i) {
6          r[i] = c[i] = i;
7      }
8      for (int k = 0; k < n; ++k) {
9          int ii = k, jj = k;

```

```

10     for (int i = k; i < n; ++i) {
11         for (int j = k; j < n; ++j) {
12             if (fabs(a[i][j]) > fabs(a[ii][jj])) {
13                 ii = i;
14                 jj = j;
15             }
16         }
17     }
18     swap(r[k], r[ii]);
19     swap(c[k], c[jj]);
20     for (int i = 0; i < n; ++i) {
21         swap(a[i][k], a[i][jj]);
22     }
23     for (int j = 0; j < n; ++j) {
24         swap(a[k][j], a[ii][j]);
25     }
26     if (fabs(a[k][k]) < EPS) {
27         continue;
28     }
29     for (int i = k + 1; i < n; ++i) {
30         a[i][k] = a[i][k] / a[k][k];
31         for (int j = k + 1; j < n; ++j) {
32             a[i][j] -= a[i][k] * a[k][j];
33         }
34     }
35 }
36 }
37
38 void solve(int n, double a[MAXN][MAXN], int r[MAXN], int c[MAXN], double b[MAXN]) {
39     static double x[MAXN];
40     for (int i = 0; i < n; ++i) {
41         x[i] = b[r[i]];
42     }
43     for (int i = 0; i < n; ++i) {
44         for (int j = 0; j < i; ++j) {
45             x[i] -= a[i][j] * x[j];
46         }
47     }
48     for (int i = n - 1; i >= 0; --i) {
49         for (int j = n - 1; j > i; --j) {
50             x[i] -= a[i][j] * x[j];
51         }
52         if (fabs(a[i][i]) >= EPS) {
53             x[i] /= a[i][i];
54         } // else assert(fabs(x[i]) < EPS);
55     }
56     for (int i = 0; i < n; ++i) {
57         b[c[i]] = x[i];
58     }
59 }
60
61 // LU(n - 1, a, r, c);
62 // solve(n - 1, a, r, c, b);

```

## 5 对一类问题的处理方法