

Contents

1	準備	1
1.1	init.el	1
1.2	tpl.cpp	1
2	文字列	2
2.1	マッチング	2
2.1.1	複数文字列マッチング (Aho-Corasick 法)	2
2.2	Suffix Array	2
3	グラフ	3
3.1	強連結成分分解	3
3.1.1	関節点	3
3.1.2	橋	3
3.1.3	強連結成分分解	4
3.2	フロー	4
3.2.1	最大流	4
3.2.2	二部マッチング	4
3.2.3	最小費用流	5
3.3	木	5
3.3.1	木の直径	5
3.3.2	最小全域木	5
3.3.3	最小シュタイナー木	6
3.4	包除原理	6
3.4.1	彩色数	6
4	数学	6
4.1	整数	6
4.1.1	剰余	6
4.1.2	カタラン数	7
4.2	多項式	7
4.2.1	FFT(complex)	7
4.2.2	FFT(modulo)	7
4.2.3	積 (FMT)	8
4.2.4	逆元 (FMT)	8
4.2.5	平方根 (FMT)	8
4.3	行列	8
4.3.1	線形方程式の解 (Givens 消去法)	9
5	幾何	9
6	データ構造	11
6.1	Union-Find 木	11
6.2	赤黒木	11
6.3	永続赤黒木	12

1 準備

1.1 init.el

linum は emacs24 のみ

```
1 ;key
2 (keyboard-translate ?\C-h ?\C-?)
3 (global-set-key "\M-g" 'goto-line)
4
5 ;tab
6 (setq-default indent-tabs-mode nil)
7 (setq-default tab-width 4)
8 (setq indent-line-function 'insert-tab)
9
10 ;line number
11 (global-linum-mode t)
12 (setq linum-format "%4d ")
```

1.2 tpl.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define rep(i,n) repi(i,0,n)
5 #define repi(i,a,b) for(int i=int(a);i<int(b);++i)
6 #define repit(it,u) for(auto it=begin(u);it!=end(u);++it)
7 #define all(u) begin(u),end(u)
8 #define uniq(u) (u).erase(unique(all(u)),end(u))
9 #define ll long
10 #define long int64_t
11 #define mp make_pair
12 #define pb push_back
13 #define eb emplace_back
14
15 //
16
17 bool input()
18 {
19     return true;
20 }
21
22 void solve()
23 {
24 }
25
26
27 int main()
28 {
29     cin.tie(0);
30     ios_base::sync_with_stdio(false);
31
32     while (input()) solve();
33 }
```

2 文字列

2.1 マッチング

2.1.1 複数文字列マッチング (Aho-Corasick 法)

$O(N + M)$

```
1 #include "macro.cpp"
2
3 const int C = 128;
4
5 struct pma_node {
6     pma_node *next[C]; // use next[0] as failure link
7     vector<int> match;
8     pma_node() { fill(next, next + C, (pma_node *) NULL); }
9     ~pma_node() { rep(i, C) if (next[i] != NULL) delete next[i]; }
10 };
11
12 pma_node *construct_pma(const vector<string>& pat) {
13     pma_node *const root = new pma_node();
14     root->next[0] = root;
15     // construct trie
16     rep(i, pat.size()) {
17         const string& s = pat[i];
18         pma_node *now = root;
19         for (const char c : s) {
20             if (now->next[int(c)] == NULL) now->next[int(c)] = new pma_node();
21             now = now->next[int(c)];
22         }
23         now->match.pb(i);
24     }
25     // make failure links by BFS
26     queue<pma_node*> q;
27     rep(i, 1, C) {
28         if (root->next[i] == NULL) root->next[i] = root;
29         else {
30             root->next[i]->next[0] = root;
31             q.push(root->next[i]);
32         }
33     }
34     while (not q.empty()) {
35         auto now = q.front();
36         q.pop();
37         rep(i, 1, C) if (now->next[i] != NULL) {
38             auto next = now->next[i];
39             while (next->next[i] == NULL) next = next->next[0];
40             now->next[i]->next[0] = next->next[i];
41             vector<int> tmp;
42             set_union(all(now->next[i]->match), all(next->next[i]->match), back_inserter
43                     (tmp));
44             now->next[i]->match = tmp;
45             q.push(now->next[i]);
46         }
47     }
48     return root;
49 }
50
51 void match(pma_node*& now, const string s, vector<int>& ret) {
52     for (const char c : s) {
53         while (now->next[int(c)] == NULL) now = now->next[0];
54         now = now->next[int(c)];
55         for (const int e : now->match) ret[e] = true;
56     }
57 }
```

2.2 Suffix Array

find_string() : $O(|T| \log |S|)$

S 中に T が含まれないなら -1, 含まれるならその先頭.

LCS() : $O(|S| + |T|)$

最長共通部分文字列. (先頭, 長さ) を返す.

```
1 #include "macro.cpp"
2
3 const int MAX_N = 1000000;
4 int n, k;
5 int rnk[MAX_N+1], tmp[MAX_N+1], sa[MAX_N+1], lcp[MAX_N+1];
6
7 bool compare_sa(int i, int j) {
8     if (rnk[i] != rnk[j]) return rnk[i] < rnk[j];
9     else {
10         int ri = i + k <= n ? rnk[i+k] : -1;
11         int rj = j + k <= n ? rnk[j+k] : -1;
12         return ri < rj;
13     }
14 }
15
16 void construct_sa(string S, int *sa) {
17     n = S.length();
18     for (int i = 0; i <= n; i++) {
19         sa[i] = i;
20         rnk[i] = i < n ? S[i] : -1;
21     }
22     for (k = 1; k <= n; k*=2) {
23         sort(sa, sa+n+1, compare_sa);
24         tmp[sa[0]] = 0;
25         for (int i = 1; i <= n; i++) {
26             tmp[sa[i]] = tmp[sa[i-1]] + (compare_sa(sa[i-1], sa[i]) ? 1 : 0);
27         }
28         for (int i = 0; i <= n; i++) {
29             rnk[i] = tmp[i];
30         }
31     }
32 }
33
34 void construct_lcp(string S, int *sa, int *lcp) {
35     int n = S.length();
36     for (int i = 0; i <= n; i++) rnk[sa[i]] = i;
37     int h = 0;
38     lcp[0] = 0;
39     for (int i = 0; i < n; i++) {
40         int j = sa[rnk[i] - 1];
41         if (h > 0) h--;
42         for (; j + h < n && i + h < n; h++) {
43             if (S[j+h] != S[i+h]) break;
44         }
45         lcp[rnk[i] - 1] = h;
46     }
47 }
48
49 //===== 使用例 =====//
50 // 文字列検索 (雑本 p338 改)  $O(|T| \log |S|)$ 
51 // S 中に T が含まれないなら -1, 含まれるならその先頭
52 int find_string(string S, int *sa, string T) {
53     int a = 0, b = S.length();
54     while (b - a > 1) {
55         int c = (a + b) / 2;
56         if (S.compare(sa[c], T.length(), T) < 0) a = c;
57         else b = c;
58     }
59     return (S.compare(sa[b], T.length(), T) == 0) ? sa[b] : -1;
60 }
```

```

60 }
61
62 // 最長共通部分文字列(蟻本p341 改) construct_sa以外はO(|S+T|)
63 // (先頭, 長さ)を返す
64 pair<int, int> LCS(string S, string T) {
65     int sl = S.length();
66     S += '\0' + T;
67     construct_sa(S, sa);
68     construct_lcp(S, sa, lcp);
69     int len = 0, pos = -1;
70     for(int i = 0; i < S.length(); i++) {
71         if(((sa[i] < sl) != (sa[i+1] < sl)) && (len < lcp[i])) {
72             len = lcp[i];
73             pos = sa[i];
74         }
75     }
76     return make_pair(pos, len);
77 }

```

3 グラフ

```

1 #include "macro.cpp"
2
3 struct edge {
4     int to; long w;
5     edge(int to, long w) : to(to), w(w) {}
6 };
7 typedef vector<vector<edge> > graph;
8
9 graph rev(const graph& G) {
10     const int n = G.size();
11     graph ret(n);
12     rep(i, n) for (const auto& e : G[i]) {
13         ret[e.to].eb(i, e.w);
14     }
15     return ret;
16 }

```

3.1 強連結成分分解

3.1.1 関節点

$O(E)$

ある関節点 u がグラフを k 個に分割するとき art には $k-1$ 個の u が含まれる. 不要な場合は `unique` を忘れないこと.

```

1 #include "macro.cpp"
2
3 typedef vector<vector<int> > graph;
4
5 class articulation {
6     const int n;
7     graph G;
8     int cnt;
9     vector<int> num, low, art;
10     void dfs(int v) {
11         num[v] = low[v] = ++cnt;
12         for (int nv : G[v]) {
13             if (num[nv] == 0) {

```

```

14         dfs(nv);
15         low[v] = min(low[v], low[nv]);
16         if ((num[v] == 1 and num[nv] != 2) or
17             (num[v] != 1 and low[nv] >= num[v])) {
18             art[v] = true;
19         }
20     } else {
21         low[v] = min(low[v], num[nv]);
22     }
23 }
24
25 public:
26 articulation(const graph& G) : n(G.size()), G(G), cnt(0), num(n), low(n), art(n) {
27     rep(i, n) if (num[i] == 0) dfs(i);
28 }
29 vector<int> get() {
30     return art;
31 }
32 };

```

3.1.2 橋

$O(V + E)$

```

1 #include "macro.cpp"
2
3 typedef vector<vector<int> > graph;
4
5 class bridge {
6     const int n;
7     graph G;
8     int cnt;
9     vector<int> num, low, in;
10     stack<int> stk;
11     vector<pair<int, int> > brid;
12     vector<vector<int> > comp;
13     void dfs(int v, int p) {
14         num[v] = low[v] = ++cnt;
15         stk.push(v), in[v] = true;
16         for (const int nv : G[v]) {
17             if (num[nv] == 0) {
18                 dfs(nv, v);
19                 low[v] = min(low[v], low[nv]);
20             } else if (nv != p and in[nv]) {
21                 low[v] = min(low[v], num[nv]);
22             }
23         }
24         if (low[v] == num[v]) {
25             if (p != n) brid.eb(min(v, p), max(v, p));
26             comp.eb();
27             int w;
28             do {
29                 w = stk.top();
30                 stk.pop(), in[w] = false;
31                 comp.back().pb(w);
32             } while (w != v);
33         }
34     }
35 public:
36 bridge(const graph& G) : n(G.size()), G(G), cnt(0), num(n), low(n), in(n) {
37     rep(i, n) if (num[i] == 0) dfs(i, n);
38 }
39 vector<pair<int, int> > get() {
40     return brid;
41 }

```

```

42     vector<vector<int> > components() {
43         return comp;
44     }
45 };

```

3.1.3 強連結成分分解

$O(V + E)$

```

1  #include "macro.cpp"
2
3  typedef vector<vector<int> > graph;
4
5  class scc {
6      const int n;
7      graph G;
8      int cnt;
9      vector<int> num, low, in;
10     stack<int> stk;
11     vector<vector<int> > comp;
12     void dfs(int v) {
13         num[v] = low[v] = ++cnt;
14         stk.push(v), in[v] = true;
15         for (const int nv : G[v]) {
16             if (num[nv] == 0) {
17                 dfs(nv);
18                 low[v] = min(low[v], low[nv]);
19             } else if (in[nv]) {
20                 low[v] = min(low[v], num[nv]);
21             }
22         }
23         if (low[v] == num[v]) {
24             comp.eb();
25             int w;
26             do {
27                 w = stk.top();
28                 stk.pop(), in[w] = false;
29                 comp.back().pb(w);
30             } while (w != v);
31         }
32     }
33 public:
34     scc(const graph& G) : n(G.size()), G(G), cnt(0), num(n), low(n), in(n) {
35         rep(i, n) if (num[i] == 0) dfs(i);
36     }
37     vector<vector<int> > components() {
38         return comp;
39     }
40 };

```

3.2 フロー

3.2.1 最大流

$O(EV^2)$

```

1  #include "macro.cpp"
2
3  const int inf = 1e9;
4  struct edge {
5      int to, cap, rev;
6      edge(int to, int cap, int rev) : to(to), cap(cap), rev(rev) {}

```

```

7  };
8  typedef vector<vector<edge> > graph;
9
10 void add_edge(graph& G, int from, int to, int cap) {
11     G[from].eb(to, cap, G[to].size());
12     G[to].eb(from, 0, G[from].size() - 1);
13 }
14
15 class max_flow {
16     const int n;
17     graph& G;
18     vector<int> level, iter;
19     void bfs(int s, int t) {
20         level.assign(n, -1);
21         queue<int> q;
22         level[s] = 0, q.push(s);
23         while (not q.empty()) {
24             const int v = q.front();
25             q.pop();
26             if (v == t) return;
27             for (const auto& e : G[v]) {
28                 if (e.cap > 0 and level[e.to] < 0) {
29                     level[e.to] = level[v] + 1;
30                     q.push(e.to);
31                 }
32             }
33         }
34     }
35     int dfs(int v, int t, int f) {
36         if (v == t) return f;
37         for (int& i = iter[v]; i < (int) G[v].size(); ++i) {
38             edge& e = G[v][i];
39             if (e.cap > 0 and level[v] < level[e.to]) {
40                 const int d = dfs(e.to, t, min(f, e.cap));
41                 if (d > 0) {
42                     e.cap -= d, G[e.to][e.rev].cap += d;
43                     return d;
44                 }
45             }
46         }
47         return 0;
48     }
49 public:
50     max_flow(graph& G) : n(G.size()), G(G) {}
51     int calc(int s, int t) {
52         int ret = 0, d;
53         while (bfs(s, t), level[t] >= 0) {
54             iter.assign(n, 0);
55             while ((d = dfs(s, t, inf)) > 0) ret += d;
56         }
57         return ret;
58     }
59 };

```

3.2.2 二部マッチング

$O(EV)$

```

1  int V;
2  vector<int> G[MAX_V];
3  int match[MAX_V];
4  bool used[MAX_V];
5
6  void add_edge(int u, int v){
7      G[u].push_back(v);

```

```

8     G[v].push_back(u);
9 }
10
11 bool dfs(int v){
12     used[v] = 1;
13     rep(i, G[v].size()){
14         int u = G[v][i], w = match[u];
15         if(w < 0 || !used[w] && dfs(w)){
16             match[v] = u;
17             match[u] = v;
18             return 1;
19         }
20     }
21     return 0;
22 }
23
24 int bi_matching(){
25     int res = 0;
26     memset(match, -1, sizeof(match));
27     rep(v, V) if(match[v] < 0){
28         memset(used, 0, sizeof(used));
29         if(dfs(v)) res++;
30     }
31     return res;
32 }

```

3.2.3 最小費用流

$O(FE \log V)$

```

1  #include "macro.cpp"
2
3  const int inf = 1e9;
4  struct edge {
5      int to, cap, cost, rev;
6      edge(int to, int cap, int cost, int rev) : to(to), cap(cap), cost(cost), rev(rev) {}
7  };
8  typedef vector<vector<edge>> graph;
9
10 void add_edge(graph& G, int from, int to, int cap, int cost) {
11     G[from].eb(to, cap, cost, G[to].size());
12     G[to].eb(from, 0, -cost, G[from].size() - 1);
13 }
14
15 int min_cost_flow(graph& G, int s, int t, int f) {
16     const int n = G.size();
17     struct state {
18         int v, d;
19         state(int v, int d) : v(v), d(d) {}
20         bool operator <(const state& t) const { return d > t.d; }
21     };
22
23     int ret = 0;
24     vector<int> h(n, 0), dist, prev(n), prev_e(n);
25     while (f > 0) {
26         dist.assign(n, inf);
27         priority_queue<state> q;
28         dist[s] = 0, q.emplace(s, 0);
29         while (not q.empty()) {
30             const int v = q.top().v;
31             const int d = q.top().d;
32             q.pop();
33             if (dist[v] <= d) continue;
34             rep(i, G[v].size()) {
35                 const edge& e = G[v][i];

```

```

36                 if (e.cap > 0 and dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]) {
37                     dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
38                     prev[e.to] = v, prev_e[e.to] = i;
39                     q.emplace(e.to, dist[e.to]);
40                 }
41             }
42         }
43         if (dist[t] == inf) return -1;
44         rep(i, n) h[i] += dist[i];
45
46         int d = f;
47         for (int v = t; v != s; v = prev[v]) {
48             d = min(d, G[prev[v]][prev_e[v]].cap);
49         }
50         f -= d, ret += d * h[t];
51         for (int v = t; v != s; v = prev[v]) {
52             edge& e = G[prev[v]][prev_e[v]];
53             e.cap -= d, G[v][e.rev].cap += d;
54         }
55     }
56     return ret;
57 }

```

3.3 木

3.3.1 木の直径

ある点 (どこでもよい) から一番遠い点 a を求める. 点 a から一番遠い点までの距離がその木の直径になる.

3.3.2 最小全域木

```

1  #include "macro.cpp"
2  #include "disjoint_set.cpp"
3  #include "graph.cpp"
4
5  struct mst_edge {
6      int u, v; long w;
7      mst_edge(int u, int v, long w) : u(u), v(v), w(w) {}
8      bool operator <(const mst_edge& t) const { return w < t.w; }
9      bool operator >(const mst_edge& t) const { return w > t.w; }
10 };
11
12 graph kruskal(const graph& G) {
13     const int n = G.size();
14     vector<mst_edge> E;
15     rep(i, n) for (const auto& e : G[i]) {
16         if (i < e.to) E.eb(i, e.to, e.w);
17     }
18     sort(all(E));
19
20     graph T(n);
21     disjoint_set uf(n);
22     for (const auto& e : E) {
23         if (not uf.same(e.u, e.v)) {
24             T[e.u].eb(e.v, e.w);
25             T[e.v].eb(e.u, e.w);
26             uf.merge(e.u, e.v);
27         }
28     }
29     return T;
30 }

```

```

31 graph prim(const vector<vector<long> >& A, int s = 0) {
32     const int n = A.size();
33     graph T(n);
34     vector<int> done(n);
35     priority_queue<mst_edge, vector<mst_edge>, greater<mst_edge> > q;
36     q.emplace(-1, s, 0);
37     while (not q.empty()) {
38         const auto e = q.top();
39         q.pop();
40         if (done[e.v]) continue;
41         done[e.v] = 1;
42         if (e.u >= 0) {
43             T[e.u].eb(e.v, e.w);
44             T[e.v].eb(e.u, e.w);
45         }
46         rep(i, n) if (not done[i]) {
47             q.emplace(e.v, i, A[e.v][i]);
48         }
49     }
50     return T;
51 }
52

```

3.3.3 最小シュタイナー木

$O(4^{|T|}V)$

g は無向グラフの隣接行列. T は使いたい頂点の集合.

```

1 int minimum_steiner_tree(vi &T, vvi &g){
2     int n = g.size(), t = T.size();
3     if(t <= 1) return 0;
4     vvi d(g); // all-pair shortest
5     rep(k,n)rep(i,n)rep(j,n) //Warshall Floyd
6         d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
7
8     int opt[1 << t][n];
9     rep(S,1<<t) rep(x,n)
10         opt[S][x] = INF;
11
12     rep(p,t) rep(q,n) // trivial case
13         opt[1 << p][q] = d[T[p]][q];
14
15     repi(S,1,1<<t){ // DP step
16         if(!(S & (S-1))) continue;
17         rep(p,n) rep(E,S)
18             if((E | S) == S)
19                 opt[S][p] = min(opt[S][p], opt[E][p] + opt[S-E][p]);
20         rep(p,n) rep(q,n)
21             opt[S][p] = min(opt[S][p], opt[S][q] + d[p][q]);
22     }
23
24     int ans = INF;
25     rep(S,1<<t) rep(q,n)
26         ans = min(ans, opt[S][q] + opt[((1<<t)-1)-S][q]);
27     return ans;
28 }

```

3.4 包除原理

3.4.1 彩色数

$O(2^V V)$

$N[i] := i$ と隣接する頂点の集合 (i も含む)

```

1 const int MAX_V=16;
2 const int mod = 10009;
3 int N[MAX_V], I[1<<MAX_V], V;
4 inline int mpow(int a, int k){ return k==0? 1: k%2? a*mpow(a,k-1)%mod: mpow(a*a%mod,k/2);}
5
6 bool can(int k){
7     int res = 0;
8     rep(S, 1<<V){
9         if(__builtin_popcountll(S)%2) res -= mpow(I[S], k);
10        else res += mpow(I[S],k);
11    }
12    return (res%mod+mod)%mod;
13 }
14
15 int color_number(){
16     memset(I, 0, sizeof(I));
17     I[0] = 1;
18     repi(S,1,1<<V){
19         int v = 0;
20         while(!(S&(1<<v))) v++;
21         I[S] = I[S-(1<<v)] + I[S&(~N[v])];
22     }
23     int lb = 0, ub = V, mid;
24     while(ub-lb>1){
25         mid = (lb+ub)/2;
26         if(can(mid)) ub = mid;
27         else lb = mid;
28     }
29     return ub;
30 }

```

4 数学

4.1 整数

4.1.1 剰余

```

1 #include "macro.cpp"
2
3 // (x, y) s.t. a x + b y = gcd(a, b)
4 long extgcd(long a, long b, long& x, long& y) {
5     long g = a; x = 1, y = 0;
6     if (b != 0) g = extgcd(b, a % b, y, x), y -= (a / b) * x;
7     return g;
8 }
9
10 // repi(i, 2, n) mod_inv[i] = mod_inv[m % i] * (m - m / i) % m
11 long mod_inv(long a, long m) {
12     long x, y;
13     if (extgcd(a, m, x, y) != 1) return 0;
14     return (x % m + m) % m;
15 }
16
17 // a mod p where n! = a p^e in O(log_p n)
18 long mod_fact(long n, long p, long& e) {
19     const int P = 1000010;
20     static long fact[P] = {1};
21     static bool done = false;
22     if (not done) {
23         repi(i, 1, P) fact[i] = fact[i - 1] * i % p;
24     }
25 }

```

```

24     done = true;
25 }
26 e = 0;
27 if (n == 0) return 1;
28 long ret = mod_fact(n / p, p, e);
29 e += n / p;
30 if (n / p % 2) return ret * (p - fact[n % p]) % p;
31 return ret * fact[n % p] % p;
32 }
33
34 // nCk mod p
35 long mod_binom(long n, long k, long p) {
36     if (k < 0 or n < k) return 0;
37     long e1, e2, e3;
38     long a1 = mod_fact(n, p, e1);
39     long a2 = mod_fact(k, p, e2);
40     long a3 = mod_fact(n - k, p, e3);
41     if (e1 > e2 + e3) return 0;
42     return a1 * mod_inv(a2 * a3 % p, p) % p;
43 }
44
45 // a^b mod m
46 long mod_pow(long a, long b, long m) {
47     long ret = 1;
48     do {
49         if (b & 1) ret = ret * a % m;
50         a = a * a % m;
51     } while (b >>= 1);
52     return ret;
53 }

```

4.1.2 カタラン数

$n \leq 16$ 程度が限度. $n \geq 1$ について以下が成り立つ.

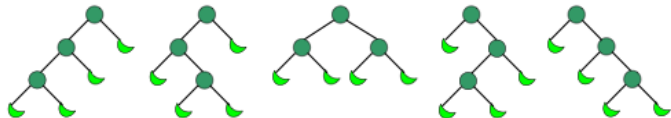
$$\begin{aligned}
 C_n &= \frac{1}{n+1} \binom{2n}{n} \\
 &= \binom{2n}{n} - \binom{2n}{n-1}
 \end{aligned}$$

n が十分大きいとき, カタラン数は以下に近似できる.

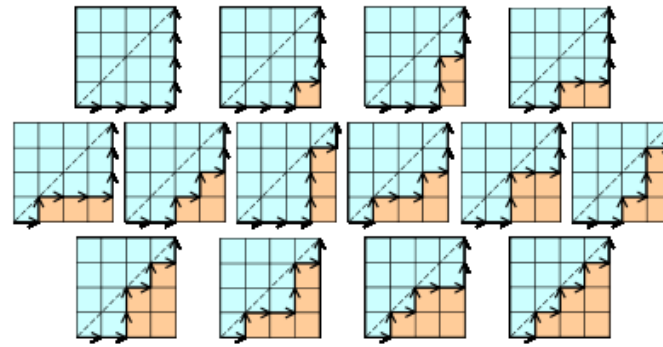
$$C_n = \frac{4^n}{n^{3/2} \sqrt{\pi}}$$

() を正しく並べる方法, 二分木, 格子状の経路の数え上げ, 平面グラフの交差などに使われる.

$C_3 = 5$



$C_4 = 14$



4.2 多項式

FFT は基本定数重めなので TLE に注意する.

4.2.1 FFT(complex)

$O(N \log N)$

複素数を用いた FFT. 変換する vector のサイズは 2 の冪乗にすること.

```

1  #include "macro.cpp"
2
3  typedef complex<double> cd;
4  vector<cd> fft(vector<cd> f, bool inv){
5      int n, N = f.size();
6      for(n=0;n++; if(N == (1<<n)) break;
7      rep(m,N){
8          int m2 = 0;
9          rep(i,n) if(m&(1<<i)) m2 |= (1<<(n-1-i));
10         if(m < m2) swap(f[m], f[m2]);
11     }
12
13     for(int t=1;t<N;t*=2){
14         double theta = acos(-1.0) / t;
15         cd w(cos(theta), sin(theta));
16         if(inv) w = cd(cos(theta), -sin(theta));
17         for(int i=0;i<N;i+=2*t){
18             cd power(1.0, 0.0);
19             rep(j,t){
20                 cd tmp1 = f[i+j] + f[i+t+j] * power;
21                 cd tmp2 = f[i+j] - f[i+t+j] * power;
22                 f[i+j] = tmp1;
23                 f[i+t+j] = tmp2;
24                 power = power * w;
25             }
26         }
27     }
28     if(inv) rep(i,N) f[i] /= N;
29     return f;
30 }

```

4.2.2 FFT(modulo)

$O(N \log N)$

剰余環を用いた FFT(FMT). 変換する vector のサイズは 2 の冪乗にすること. mod は $a * 2^e + 1$ の形.

```
1 #include "macro.cpp"
2 #include "number_theory.cpp"
3
4 const int mod = 7*17*(1<<23)+1;
5 vector<int> fmt(vector<int> f, bool inv){
6     int e, N = f.size();
7     // assert((N&(N-1))==0 and "f.size() must be power of 2");
8     for(e=0;;e++) if(N == (1<<e)) break;
9     rep(m,N){
10         int m2 = 0;
11         rep(i,e) if(m&(1<<i)) m2 |= (1<<(e-1-i));
12         if(m < m2) swap(f[m], f[m2]);
13     }
14     for(int t=1; t<N; t*=2){
15         int r = pow_mod(3,(mod-1)/(t*2),mod);
16         if(inv) r = mod_inverse(r,mod);
17         for(int i=0; i<N; i+=2*t){
18             int power = 1;
19             rep(j,t){
20                 int x = f[i+j], y = 1LL*f[i+t+j]*power%mod;
21                 f[i+j] = (x+y)%mod;
22                 f[i+t+j] = (x-y+mod)%mod;
23                 power = 1LL*power*r%mod;
24             }
25         }
26     }
27     if(inv) for(int i=0,ni=mod_inv(N,mod);i<N;i++) f[i] = 1LL*f[i]*ni%mod;
28     return f;
29 }
```

4.2.3 積 (FMT)

$O(N \log N)$

poly_mul() が必要.

```
1 #include "fmt.cpp"
2 vector<int> poly_mul(vector<int> f, vector<int> g){
3     int N = max(f.size(),g.size())*2;
4     f.resize(N); g.resize(N);
5     f = fmt(f,0); g = fmt(g,0);
6     rep(i,N) f[i] = 1LL*f[i]*g[i]%mod;
7     f = fmt(f,1);
8     return f;
9 }
```

4.2.4 逆元 (FMT)

$O(N \log N)$

extgcd(), mod_inverse(), poly_mul(), fmt() が必要.

```
1 #include "poly_mul.cpp"
2 vector<int> poly_inv(const vector<int> &f){
3     int N = f.size();
4     vector<int> r(1,mod_inv(f[0],mod));
5     for(int k = 2; k <= N; k <= 1){
6         vector<int> nr = poly_mul(poly_mul(r,r), vector<int>(f.begin(),f.begin()+k));
7         nr.resize(k);
8     }
```

```
8         rep(i,k/2) {
9             nr[i] = (2*r[i]-nr[i]+mod)%mod;
10            nr[i+k/2] = (mod-nr[i+k/2])%mod;
11        }
12        r = nr;
13    }
14    return r;
15 }
```

4.2.5 平方根 (FMT)

$O(N \log N)$

extgcd(), mod_inverse(), poly_inv(), poly_mul(), fmt() が必要.

```
1 #include "poly_inv.cpp"
2 const int inv2 = (mod+1)/2;
3 vector<int> poly_sqrt(const vector<int> &f) {
4     int N = f.size();
5     vector<int> s(1,1); // s[0] = sqrt(f[0])
6     for(int k = 2; k <= N; k <= 1) {
7         s.resize(k);
8         vector<int> ns = poly_mul(poly_inv(s), vector<int>(f.begin(),f.begin()+k));
9         ns.resize(k);
10        rep(i,k) s[i] = 1LL*(s[i]+ns[i])*inv2%mod;
11    }
12    return s;
13 }
```

4.3 行列

C++11 だと array という名前では衝突するので arr にしている.

```
1 #include "macro.cpp"
2
3 typedef double number;
4 typedef vector<number> vec;
5 typedef vector<vec> mat;
6
7 vec mul(const mat& A, const vec& x) {
8     const int n = A.size();
9     vec b(n);
10    rep(i, n) rep(j, A[0].size()) {
11        b[i] = A[i][j] * x[j];
12    }
13    return b;
14 }
15
16 mat mul(const mat& A, const mat& B) {
17     const int n = A.size();
18     const int o = A[0].size();
19     const int m = B[0].size();
20     mat C(n, vec(m));
21     rep(i, n) rep(k, o) rep(j, m) {
22         C[i][j] += A[i][k] * B[k][j];
23     }
24     return C;
25 }
26
27 mat pow(mat A, long m) {
28     const int n = A.size();
29     mat B(n, vec(n));
```



```

30     rep(i, n) B[i][i] = 1;
31     do {
32         if (m & 1) B = mul(B, A);
33         A = mul(A, A);
34     } while (m >= 1);
35     return B;
36 }
37
38 const number eps = 1e-4;
39
40 // determinant; O(n^3)
41 number det(mat A) {
42     int n = A.size();
43     number D = 1;
44     rep(i,n){
45         int pivot = i;
46         repi(j,i+1,n)
47             if (abs(A[j][i]) > abs(A[pivot][i])) pivot = j;
48         swap(A[pivot], A[i]);
49         D *= A[i][i] * (i != pivot ? -1 : 1);
50         if (abs(A[i][i]) < eps) break;
51         repi(j,i+1,n)
52             for(int k=n-1;k>=i;--k)
53                 A[j][k] -= A[i][k] * A[j][i] / A[i][i];
54     }
55     return D;
56 }
57
58 // rank; O(n^3)
59 int rank(mat A) {
60     int n = A.size(), m = A[0].size(), r = 0;
61     for(int i = 0; i < m and r < n; i++){
62         int pivot = r;
63         repi(j,r+1,n)
64             if (abs(A[j][i]) > abs(A[pivot][i])) pivot = j;
65         swap(A[pivot], A[r]);
66         if (abs(A[r][i]) < eps) continue;
67         for(int k=m-1;k>=i;--k)
68             A[r][k] /= A[r][i];
69         repi(j,r+1,n) repi(k,i,m)
70             A[j][k] -= A[r][k] * A[j][i];
71         ++r;
72     }
73     return r;
74 }

```

4.3.1 線形方程式の解 (Givens 消去法)

$O(N^3)$

```

1  #include "macro.cpp"
2
3  // Givens elimination; O(n^3)
4
5  typedef double number;
6  typedef vector<vector<number>> matrix;
7
8  inline double my_hypot(double x, double y) { return sqrt(x * x + y * y); }
9  inline void givens_rotate(number& x, number& y, number c, number s) {
10     number u = c * x + s * y, v = -s * x + c * y;
11     x = u, y = v;
12 }
13 vector<number> givens(matrix A, vector<number> b) {
14     const int n = b.size();
15     rep(i, n) repi(j, i + 1, n) {

```

```

16         const number r = my_hypot(A[i][i], A[j][i]);
17         const number c = A[i][i] / r, s = A[j][i] / r;
18         givens_rotate(b[i], b[j], c, s);
19         repi(k, i + 1, n) givens_rotate(A[i][k], A[j][k], c, s);
20     }
21     for (int i = n - 1; i >= 0; --i) {
22         repi(j, i + 1, n) b[i] -= A[i][j] * b[j];
23         b[i] /= A[i][i];
24     }
25     return b;
26 }

```

5 幾何

```

1  #include "macro.cpp"
2
3  // constants and eps-considered operators
4
5  const double eps = 1e-8; // choose carefully!
6  const double pi = acos(-1.0);
7
8  inline bool lt(double a, double b) { return a < b - eps; }
9  inline bool gt(double a, double b) { return lt(b, a); }
10 inline bool le(double a, double b) { return !lt(b, a); }
11 inline bool ge(double a, double b) { return !lt(a, b); }
12 inline bool ne(double a, double b) { return lt(a, b) or lt(b, a); }
13 inline bool eq(double a, double b) { return !ne(a, b); }
14
15 // points and lines
16
17 typedef complex<double> point;
18
19 inline double dot (point a, point b) { return real(conj(a) * b); }
20 inline double cross(point a, point b) { return imag(conj(a) * b); }
21
22 struct line {
23     point a, b;
24     line(point a, point b) : a(a), b(b) {}
25 };
26
27 /*
28  * Here is what ccw(a, b, c) returns:
29  *
30  *      1
31  *  -----
32  *      2 |a  0  b| -2
33  *  -----
34  *      -1
35  *
36  * Note: we can implement intersectPS(p, s) as !ccw(s.a, s.b, p).
37  */
38 int ccw(point a, point b, point c) {
39     b -= a, c -= a;
40     if (cross(b, c) > eps) return +1;
41     if (cross(b, c) < eps) return -1;
42     if (dot(b, c) < eps) return +2; // c -- a -- b
43     if (lt(norm(b), norm(c))) return -2; // a -- b -- c
44     return 0;
45 }
46 bool intersectLS(const line& l, const line& s) {
47     return ccw(l.a, l.b, s.a) * ccw(l.a, l.b, s.b) <= 0;
48 }
49 bool intersectSS(const line& s, const line& t) {

```

```

50     return intersectLS(s, t) and intersectLS(t, s);
51 }
52 bool intersectLL(const line& l, const line& m) {
53     return ne(cross(l.b - l.a, m.b - m.a), 0.0) // not parallel
54         or eq(cross(l.b - l.a, m.a - l.a), 0.0); // overlap
55 }
56 point crosspointLL(const line& l, const line& m) {
57     double A = cross(l.b - l.a, m.b - m.a);
58     double B = cross(l.b - l.a, m.a - l.a);
59     if (eq(A, 0.0) and eq(B, 0.0)) return m.a; // overlap
60     assert(ne(A, 0.0)); // not parallel
61     return m.a - B / A * (m.b - m.a);
62 }
63 point proj(const line& l, point p) {
64     double t = dot(l.b - l.a, p - l.a) / norm(l.b - l.a);
65     return l.a + t * (l.b - l.a);
66 }
67 point reflection(const line& l, point p) { return 2.0 * proj(l, p) - p; }
68
69 // distances (for shortest path)
70
71 double distanceLP(const line& l, point p) { return abs(proj(l, p) - p); }
72 double distanceLL(const line& l, const line& m) {
73     return intersectLL(l, m) ? 0.0 : distanceLP(l, m.a);
74 }
75 double distanceLS(const line& l, const line& s) {
76     return intersectLS(l, s) ? 0.0 : min(distanceLP(l, s.a), distanceLP(l, s.b));
77 }
78 double distancePS(point p, const line& s) {
79     point h = proj(s, p);
80     return ccw(s.a, s.b, h) ? min(abs(s.a - p), abs(s.b - p)) : abs(h - p);
81 }
82 double distanceSS(const line& s, const line& t) {
83     if (intersectSS(s, t)) return 0.0;
84     return min(min(distancePS(s.a, t), distancePS(s.b, t)),
85                min(distancePS(t.a, s), distancePS(t.b, s)));
86 }
87
88 // circles
89
90 struct circle {
91     point o; double r;
92     circle(point o, double r) : o(o), r(r) {}
93 };
94
95 bool intersectCL(const circle& c, const line& l) {
96     return le(norm(proj(l, c.o) - c.o), c.r * c.r);
97 }
98 int intersectCS(const circle& c, const line& s) {
99     if (not intersectCL(c, s)) return 0;
100     double a = abs(s.a - c.o);
101     double b = abs(s.b - c.o);
102     if (lt(a, c.r) and lt(b, c.r)) return 0;
103     if (lt(a, c.r) or lt(b, c.r)) return 1;
104     return ccw(s.a, s.b, proj(s, c.o)) ? 0 : 2;
105 }
106 bool intersectCC(const circle& c, const circle& d) {
107     double dist = abs(d.o - c.o);
108     return le(abs(c.r - d.r), dist) and le(dist, c.r + d.r);
109 }
110 line crosspointCL(const circle& c, const line& l) {
111     point h = proj(l, c.o);
112     double a = sqrt(c.r * c.r - norm(h - c.o));
113     point d = a * (l.b - l.a) / abs(l.b - l.a);
114     return line(h - d, h + d);
115 }
116 line crosspointCC(const circle& c, const circle& d) {

```

```

117     double dist = abs(d.o - c.o), th = arg(d.o - c.o);
118     double ph = acos((c.r * c.r + dist * dist - d.r * d.r) / (2.0 * c.r * dist));
119     return line(c.o + polar(c.r, th - ph), c.o + polar(c.r, th + ph));
120 }
121
122 line tangent(const circle& c, double th) {
123     point h = c.o + polar(c.r, th);
124     point d = polar(c.r, th) * point(0, 1);
125     return line(h - d, h + d);
126 }
127 vector<line> common_tangents(const circle& c, const circle& d) {
128     vector<line> ret;
129     double dist = abs(d.o - c.o), th = arg(d.o - c.o);
130     if (abs(c.r - d.r) < dist) { // outer
131         double ph = acos((c.r - d.r) / dist);
132         ret.pb(tangent(c, th - ph));
133         ret.pb(tangent(c, th + ph));
134     }
135     if (abs(c.r + d.r) < dist) { // inner
136         double ph = acos((c.r + d.r) / dist);
137         ret.pb(tangent(c, th - ph));
138         ret.pb(tangent(c, th + ph));
139     }
140     return ret;
141 }
142 pair<circle, circle> tangent_circles(const line& l, const line& m, double r) {
143     double th = arg(m.b - m.a) - arg(l.b - l.a);
144     double ph = (arg(m.b - m.a) + arg(l.b - l.a)) / 2.0;
145     point p = crosspointLL(l, m);
146     point d = polar(r / sin(th / 2.0), ph);
147     return mp(circle(p - d, r), circle(p + d, r));
148 }
149 line bisector(point a, point b);
150 circle circum_circle(point a, point b, point c) {
151     point o = crosspointLL(bisector(a, b), bisector(a, c));
152     return circle(o, abs(a - o));
153 }
154
155 // polygons
156
157 typedef vector<point> polygon;
158
159 double area(const polygon& g) {
160     double ret = 0.0;
161     int j = g.size() - 1;
162     rep(i, g.size()) {
163         ret += cross(g[j], g[i]), j = i;
164     }
165     return ret / 2.0;
166 }
167 point centroid(const polygon& g) {
168     if (g.size() == 1) return g[0];
169     if (g.size() == 2) return (g[0] + g[1]) / 2.0;
170     point ret = 0.0;
171     int j = g.size() - 1;
172     rep(i, g.size()) {
173         ret += cross(g[j], g[i]) * (g[j] + g[i]), j = i;
174     }
175     return ret / area(g) / 6.0;
176 }
177 line bisector(point a, point b) {
178     point m = (a + b) / 2.0;
179     return line(m, m + (b - a) * point(0, 1));
180 }
181 polygon convex_cut(const polygon& g, const line& l) {
182     polygon ret;
183     int j = g.size() - 1;

```

```

184     rep(i, g.size()) {
185         if (ccw(l.a, l.b, g[j]) != -1) ret.pb(g[j]);
186         if (intersectLS(l, line(g[j], g[i]))) ret.pb(crosspointLL(l, line(g[j], g[i])));
187         j = i;
188     }
189     return ret;
190 }
191 polygon voronoi_cell(polygon g, const vector<point>& v, int k) {
192     rep(i, v.size()) if (i != k) {
193         g = convex_cut(g, bisector(v[i], v[k]));
194     }
195     return g;
196 }

```

```

1  #include "macro.cpp"
2  #include "geometry.cpp"
3
4  namespace std {
5      bool operator <(const point& a, const point& b) {
6          return ne(real(a), real(b)) ? lt(real(a), real(b)) : lt(imag(a), imag(b));
7      }
8  }
9
10 polygon convex_hull(vector<point> v) {
11     const int n = v.size();
12     sort(all(v));
13     polygon ret(2 * n);
14     int k = 0;
15     for (int i = 0; i < n; ret[k++] = v[i++]) {
16         while (k >= 2 and ccw(ret[k - 2], ret[k - 1], v[i]) <= 0) --k;
17     }
18     for (int i = n - 2, t = k + 1; i >= 0; ret[k++] = v[i--]) {
19         while (k >= t and ccw(ret[k - 2], ret[k - 1], v[i]) <= 0) --k;
20     }
21     ret.resize(k - 1);
22     return ret;
23 }

```

6 データ構造

6.1 Union-Find 木

```

1  #include "macro.cpp"
2
3  class disjoint_set {
4      vector<int> p;
5      int root(int i) { return p[i] >= 0 ? p[i] = root(p[i]) : i; }
6  public:
7      disjoint_set(int n) : p(n, -1) {}
8      bool same(int i, int j) { return root(i) == root(j); }
9      int size(int i) { return -p[root(i)]; }
10     void merge(int i, int j) {
11         i = root(i), j = root(j);
12         if (i == j) return;
13         if (p[i] > p[j]) swap(i, j);
14         p[i] += p[j], p[j] = i;
15     }
16 };

```

6.2 赤黒木

```

1  template<class T> class rbtree
2  {
3  public:
4      enum COL { BLACK, RED,};
5      struct node {
6          T val;
7          int color;
8          int rnk, size;
9          node *left, *right;
10     };
11
12     node() {}
13     node(T v) : val(v), color(BLACK), rnk(0), size(1) {
14         left = right = NULL;
15     }
16     node(node *l, node *r, int c) : color(c) {
17         left = l;
18         right = r;
19         update();
20     }
21     void update() {
22         rnk = max((left? left->rnk+(left->color==BLACK): 0),
23                 (right? right->rnk+(right->color==BLACK): 0));
24         size = (left? left->size: 0)+(right? right->size: 0)+(!left and !right);
25     }
26
27     node *root;
28
29     rbtree() { root = NULL;}
30     rbtree(T val) { root = new_node(val);}
31
32     node *new_node(T v) { return new node(v);}
33     node *new_node(node *l, node *r, int c) { return new node(l,r,c);}
34
35     node *right_rotate(node *v) {
36         node *w = v->left;
37         v->left = w->right;
38         w->right = v;
39         v->left->update();
40         v->update();
41         w->right->update();
42         v->color = RED;
43         w->color = BLACK;
44         return w;
45     }
46
47     node *left_rotate(node *v) {
48         node *w = v->right;
49         v->right = w->left;
50         w->left = v;
51         v->right->update();
52         v->update();
53         w->left->update();
54         v->color = RED;
55         w->color = BLACK;
56         return w;
57     }
58
59     node *merge_sub(node *u, node *v) {
60         if (u->rnk < v->rnk) {
61             node *w = merge_sub(u,v->left);
62             v->left = w;
63             v->update();
64             if (v->color == BLACK and w->color == RED and w->left->color == RED) {

```

```

65         if(v->right->color == BLACK) return right_rotate(v);
66     else {
67         v->color = RED;
68         v->right->color = BLACK;
69         w->color = BLACK;
70         return v;
71     }
72 }
73 else return v;
74 }
75 else if(u->rnk > v->rnk) {
76     node *w = merge_sub(u->right, v);
77     u->right = w;
78     u->update();
79     if(u->color == BLACK and w->color == RED and w->right->color == RED) {
80         if(u->left->color == BLACK) return left_rotate(u);
81     else {
82         u->color = RED;
83         u->left->color = BLACK;
84         w->color = BLACK;
85         return u;
86     }
87 }
88 else return u;
89 }
90 else return new_node(u, v, RED);
91 }
92
93 node *merge(node *u, node *v) {
94     if(!u) return v;
95     if(!v) return u;
96     u = merge_sub(u, v);
97     u->color = BLACK;
98     return u;
99 }
100
101 pair<node*, node*> split(node *v, int k) {
102     if(!k) return pair<node*, node*>(NULL, v);
103     if(k == v->size) return pair<node*, node*>(v, NULL);
104     if(k < v->left->size) {
105         auto p = split(v->left, k);
106         return pair<node*, node*>(p.first, merge(p.second, v->right));
107     }
108     else if(k > v->left->size) {
109         auto p = split(v->right, k - v->left->size);
110         return pair<node*, node*>(merge(v->left, p.first), p.second);
111     }
112     else return pair<node*, node*>(v->left, v->right);
113 }
114
115 // insert val at k
116 node *insert(T val, int k) { return insert(new_node(val), k); }
117 // insert tree v at k
118 node *insert(node *v, int k) {
119     auto p = split(root, k);
120     return root = merge(merge(p.first, v), p.second);
121 }
122
123 // delete at k
124 node *erase(int k) {
125     auto p = split(root, k + 1);
126     return root = merge(split(p.first, k).first, p.second);
127 }
128
129 node *build(const vector<T> &vs) {
130     if(!vs.size()) return NULL;
131     if((int)vs.size() == 1) return new_node(vs[0]);

```

```

132     int m = vs.size()/2;
133     return merge(build(vector<T>(begin(vs), begin(vs)+m)),
134                 build(vector<T>(begin(vs)+m, end(vs))));
135 }
136
137 int size() { return root->size; }
138
139 void get(vector<T> &vs) { get(root, vs); }
140 void get(node *v, vector<T> &vs) {
141     if(!v->left and !v->right) vs.push_back(v->val);
142     else {
143         if(v->left) get(v->left, vs);
144         if(v->right) get(v->right, vs);
145     }
146 }
147
148 node *push_back(T val) {
149     node *v = new_node(val);
150     return root = merge(root, v);
151 }
152 };

```

6.3 永続赤黒木

```

1 //const int MAX = 15000000, BOUND = 14000000;
2 template<class T> class prbtree {
3 public:
4     enum COL { BLACK, RED, };
5     struct node {
6         T val;
7         int color;
8         int rnk, size;
9         node *left, *right;
10
11         node() {}
12         node(T v) : val(v), color(BLACK), rnk(0), size(1) {
13             left = right = NULL;
14         }
15         node(node *l, node *r, int c) : color(c) {
16             left = l;
17             right = r;
18             rnk = max((l? l->rnk + (l->color==BLACK): 0),
19                     (r? r->rnk + (r->color==BLACK): 0));
20             size = !l and !r? 1: !l? r->size: !r? r->size: l->size + r->size;
21         }
22     };
23
24     node *root;
25     // node nodes[MAX];
26     // int called;
27
28     prbtree() {
29         root = NULL;
30         // called = 0;
31     }
32
33     prbtree(T val) {
34         root = new_node(val);
35         // called = 0;
36     }
37
38     // node *new_node(T v) { return &(nodes[called++] = node(v)); }
39     // node *new_node(node *l, node *r, int c) { return &(nodes[called++] = node(l, r, c)); }

```

```

40 node *new_node(T v) { return new node(v);}
41 node *new_node(node *l, node *r, int c) { return new node(l,r,c);}
42
43 node *merge_sub(node *u, node *v) {
44     if(u->rnk < v->rnk) {
45         node *w = merge_sub(u,v->left);
46         if(v->color == BLACK and w->color == RED and w->left->color == RED){
47             if(v->right->color == BLACK) return new_node(w->left,new_node(w->right,
48                 v->right,RED),BLACK);
49             else return new_node(new_node(w->left,w->right,BLACK),new_node(v->right
50                 ->left,v->right->right,BLACK),RED);
51         }
52         else return new_node(w,v->right,v->color);
53     }
54     else if(u->rnk > v->rnk) {
55         node *w = merge_sub(u->right,v);
56         if(u->color == BLACK and w->color == RED and w->right->color == RED){
57             if(u->left->color == BLACK) return new_node(new_node(u->left,w->left,
58                 RED),w->right,BLACK);
59             else return new_node(new_node(u->left->left,u->left->right,BLACK),
60                 new_node(w->left,w->right,BLACK),RED);
61         }
62         else return new_node(u->left,w,u->color);
63     }
64     else return new_node(u,v,RED);
65 }
66
67 node *merge(node *u, node *v) {
68     if(!u) return v;
69     if(!v) return u;
70     u = merge_sub(u,v);
71     if(u->color == RED) return new_node(u->left,u->right,BLACK);
72     return u;
73 }
74
75 pair<node*,node*> split(node *v, int k) {
76     if(!k) return pair<node*,node*>(NULL,v);
77     if(k == v->size) return pair<node*,node*>(v,NULL);
78     if(k < v->left->size) {
79         auto p = split(v->left,k);
80         return pair<node*,node*>(p.first,merge(p.second,v->right));
81     }
82     else if(k > v->left->size) {
83         auto p = split(v->right,k-v->left->size);
84         return pair<node*,node*>(merge(v->left,p.first),p.second);
85     }
86     else return pair<node*,node*>(v->left,v->right);
87 }
88
89 node *build(const vector<T> &vs) {
90     if(!vs.size()) return NULL;
91     if((int)vs.size() == 1) return new_node(vs[0]);
92     int m = vs.size()/2;
93     return merge(build(vector<T>(begin(vs),begin(vs)+m)), build(vector<T>(begin(vs)+
94         m,end(vs))));
95 }
96
97 int size() { return root->size;}
98
99 void get(vector<T> &vs) { get(root,vs);}
100 void get(node *v, vector<T> &vs) {
101     if(!v->left and !v->right) vs.push_back(v->val);
102     else {
103         if(v->left) get(v->left,vs);
104         if(v->right) get(v->right,vs);
105     }
106 }

```

```

102
103 node *push_back(T val) {
104     node *v = new_node(val);
105     return root = merge(root,v);
106 }
107
108 // insert leaf at k
109 node *insert(int k, T val) {
110     return insert(new_node(val), k);
111 }
112
113 // insert tree v at k
114 node *insert(node *v, int k) {
115     auto p = split(root,k);
116     return root = merge(merge(p.first,v),p.second);
117 }
118
119 // copy [l,r)
120 node *copy(int l, int r) {
121     return split(split(root, l).second, r-l).first;
122 }
123
124 // copy and insert [l,r) at k
125 node *copy_paste(int l, int r, int k) {
126     return insert(copy(l,r),k);
127 }
128
129 };

```