Dwylkz's Algorithm Library

GuangZhou University Team Kimady Dwylkz(Wūdōng Yáng)

November 8, 2013

Contents

1	Perface	1
2	vimrc	1
3	main	2
4	makefile	2
5	numeric5.1 High Precision Integer5.2 Minimum Prime Factor Sieve5.3 Contor coding5.4 Chinese Remind Theory5.5 Base 2 Fast Fourier Transfrom5.6 Triangle Diagonal Matrix Algorithm	2 2 3 4 5 6
6	pattern 6.1 KMP 6.2 Extend KMP 6.2 Extend KMP 6.3 Manacher 6.4 Minimum Notation 6.5 AC automaton 6.5 AC automaton 6.6 Suffix Array 6.7 Suffix Automaton 6.7 Suffix Automaton	6 77 77 88 88 9
7	7.1 RMQ Sparse Table 7.2 Splay 7.3 Treap 7.4 Link-Cut Tree 7.5 Functional Segment 7.6 Functional Trie	10 10 11 13 15 17 18
8	8.1 Float Compare Functions 8.2 2D point 8.3 Angle Sort	19 19 20 20
9	9.1 Graph	20 21 22 22

9.6	Directed Minimum Spanning Tree	27
9.7	Spfa Cost Stream	28
9.8	KM Maximum perfect match	29
9.9	Doubling LCA	30
9.10	Shortest Augment Path	30
9.11	ZKW Cost Stream	31
10 grap	oh test	32
10 grap 10.1	oh test Graph	
10.1	Shortest Augment Path	32
10.1 10.2	Graph	$\frac{32}{33}$
10.1 10.2 10.3	Graph	$\frac{32}{33}$
10.1 10.2 10.3 10.4	Graph	32 33 33 34

1 Perface

The content enclosed by '<' and '>' beside template, is just a note which remind us to define something before we use this template.

2 vimrc

```
colorscheme desert
 2
    set langmenu=en_US.UTF-8
    source $VIMRUNTIME/delmenu.vim
 3
    source $VIMRUNTIME/menu.vim
    language messages en_US.UTF-8
    syntax on
 7
    filetype on
    filetype plugin indent on
    set smartindent
10
    set autochdir
11
    set autoindent
12
    set smartindent
13
    set backspace=2
    set columns=120
14
    set foldmethod=syntax
15
    set nohlsearch
16
17
    set incsearch
    set lines=40
18
    set nocompatible
19
20
    set noswapfile
21
    set number
22
    set shiftwidth=2
23
    set tabstop=2
24
    set expandtab
25
26
    func Compile()
               exec "w"
27
28
               exec "make"
29
    endfunc
30
    func Debug()
               exec "w"
31
               exec "make_debug"
32
33
    endfunc
34
    func Run()
               exec "w"
35
36
               exec "make_run"
37
    endfunc
38
39
    map < F4 > : tabp < CR >
40
    map < F5 > : tabn < CR >
    \mathrm{map}~<\!\!\mathrm{C}\!\!-\!\mathrm{a}\!\!>~\mathrm{ggVG}
41
    \mathrm{map}\ <\!\!\mathrm{C}\!\!-\!\!\mathrm{p}\!\!>\ \ddot{"}\!\!+\!\!\mathrm{p}
42
43
    map = \langle F8 \rangle = \langle ESC \rangle : c list \langle CR \rangle
    map = C-F8 = ESC > call = Debug() < CR >
    map = <F9 > = <ESC > : call = Compile() < <CR >
45
    map = C-F9 = ESC > call = Run() < CR >
46
    map \le silent > F6 = s\#^{\#}//_{\#}g < CR > s
47
    map \le silent > F7> : s\#^// \#g < CR>
48
49
    vmap = <C-y> = "+y
```

3 main

```
#include <cstdio>
1
   #include <cstdlib>
2
   #include <iostream>
   using namespace std;
6
  int main() {
   #if 1
7
     freopen("input.in", "r", stdin);
9
   #endif
10
     return 0;
11
```

4 makefile

```
main: main.cc
gcc main.cc -o main -lstdc++ -g
.PHONY: debug clean run
debug:
gdb main
run:
./main
```

5 numeric

5.1 High Precision Integer

```
1
   /* High_Precision_Integer
2
   * */
3
   struct int_t {
4
     string d;
     int_t(string_d = "0"): d(_d) {}
5
6
     int_t(int_d) {
       static char buff [20];
7
8
       sprintf(buff, "%d", _d);
9
       d = buff;
10
11
     static void trans(string &s) {
       for (int i = 0; i < s.length(); i++) s[i] += '0';
12
13
     friend int_t & operator + (const int_t & lhs, const int_t & rhs) {
14
15
       static int_t result;
       const string &a = lhs.d, &b = rhs.d;
16
17
       string &c = result.d;
18
       int maxlen = \max(a.length(), b.length())+1;
19
       c.resize(maxlen);
20
        fill(c.begin(), c.end(), 0);
       for (int i = 0; i < maxlen - 1; i++) {
21
         int x = a.length() \le i? 0: a[a.length()-1-i]-'0',
22
23
              y = b. length() \le i? 0: b[b. length()-1-i]-'0';
24
         c[i] += x+y;
         c[i+1] += c[i]/10;
25
26
         c[i] %= 10;
27
       if (!c[maxlen-1]) c.resize(maxlen-1);
28
```

```
29
        reverse(c.begin(), c.end());
30
        trans(c);
        return result;
31
32
33
      friend int_t & operator += (const int_t & lhs, const int_t & rhs) {
34
        return lhs+rhs;
35
36
      friend int_t & operator * (const int_t & lhs, const int_t & rhs) {
37
        static int_t result;
38
        const string &a = lhs.d, &b = rhs.d;
39
        string &c = result.d;
        int maxlen = a.length()+b.length();
40
        c.resize(maxlen);
41
42
        fill(c.begin(), c.end(), 0);
        for (int i = 0; i < a.length(); i++) {
43
44
          int x = a[a.length()-1-i]-'0';
          for (int j = 0; j < b.length(); j++) {
45
            int y = b[b.length()-1-j]-'0';
46
            c[i+j] += x*y;
47
48
            c[i+j+1] += c[i+j]/10;
            c[i+j] \% = 10;
49
50
          }
51
52
        for (; \max len > 1 \&\& !c[\max len - 1]; \max len - -) {}
53
        c.resize(maxlen);
54
        reverse (c.begin(), c.end());
55
        trans(c);
56
        return result;
57
58
      friend int_t & operator *= (const int_t & lhs, const int_t & rhs) {
59
        return lhs*rhs;
60
      const char *show() {
61
62
        return d.data();
63
64
   };
```

5.2 Minimum Prime Factor Sieve

```
1
   /* Minimum_Prime_Factor_Sieve
2
    * N : upper bound
3
    * p[]: primes
         : primes number
4
    * n
    * e[]: eular funtion
5
6
    * d[]: divisors number
7
    * f[]: minimum prime factor
8
    * c[]: minimum prime factor's power
    * m[]: mobius function
9
10
    * */
   template<int N> struct sieve_t {
11
12
     int b[N], p[N], n, e[N], d[N], f[N], c[N], m[N];
13
     sieve_t() {
       memset(this, 0, sizeof(sieve_t));
14
15
       d[1] = m[1] = 1;
       for (int i = 2; i < N; i++) {
16
          if (!b[i]) {
17
18
            e[i] = i-1;
19
            c[i] = 1;
20
            d[i] = 2;
```

```
21
             f[i] = i;
22
             m[i] = -1;
23
             p[n++] = i;
24
25
           for (int j = 0; j < n && 1 ll * i * p[j] < N; j++) {
26
             int k = i*p[j];
             b[k] = 1;
27
             f\,[\,k\,] \ = \ p\,[\,j\,\,]\,;
28
             if (i%p[j]) {
29
30
                e[k] = e[i]*(p[j]-1);
31
                c[k] = 1;
32
                d[k] = d[i]*(c[k]+1);
33
               m[k] = m[i]*m[p[j]];
34
             } else {
                e[k] = e[i] * p[j];
35
                c[k] = c[i] + 1;
36
37
                d[k] = d[i]/(c[i]+1)*(c[k]+1);
38
               m[k] = 0;
39
               break;
40
41
          }
42
        }
43
      }
44
    };
```

5.3 Contor coding.

```
/* Contor_coding.
1
2
    * Notice that x in [1, 1!] in array->integer mapping
3
    * while x in [0, 1!) in integer -> array mapping. */
4
   template<int N> struct contor_t {
5
      int f[N];
      contor_t() {
6
        for (int i = f[0] = 1; i < N; i++)
7
8
          f[i] = f[i - 1] * i;
9
10
      void operator () (int 1, int x, int *t) {
        int id = 0, h[100] = \{0\};
11
12
       x--;
13
        for (int i = l-1; 0 \le i; i--) {
          int rm = x/f[i], rank = 0;
14
          for (int j = 1; j \ll l; j ++) {
15
16
            rank += !h[j];
17
            if (rank = rm+1) {
18
              t [id++] = j;
19
              h[j] = 1;
20
              break;
21
            }
22
23
          x %= f[i];
        }
24
25
26
      int operator () (int 1, int *t) {
27
        int rv = 0;
        for (int i = 0; i < 1; i++) {
28
29
          int cnt = 0;
30
          for (int j = i+1; j < l; j++)
31
            if (t[j] < t[i]) cnt++;
32
          rv += cnt * f [1-i-1];
```

```
33 | }
34 | return rv;
35 | }
36 | };
```

5.4 Chinese Remind Theory

```
/* Chinese_Remind_Theory
1
2
    * */
3
   template<int N> struct crt_t {
      vector < int > a, b;
4
5
      int gcd(int a, int b, int &x, int &y) {
6
        int d, tx, ty;
7
        if (b = 0) {
8
          x = 1;
          y = 0;
9
10
          return a;
11
        d = gcd(b, a\%b, tx, ty);
12
13
        x = ty;
14
        y = tx - (a/b) * ty;
        return d;
15
16
      int mle(int a, int b, int n) {
17
18
        int d, x, y;
        d = \gcd(a, n, x, y);
19
20
        if (b\%d == 0) {
21
          x = 111*x*b/d\%n;
22
          return x;
23
        }
24
        return 0;
25
26
      int init() {
27
        a.clear();
28
        b.clear();
29
30
      int operator () () {
31
        int x = 0, n = 1, i, bi;
32
        for (i = 0; i < b.size(); i++) n *= b[i];
33
        for (i = 0; i < a.size(); i++) {
34
          bi = mle(n/b[i], 1, b[i]);
          x = (x+111*a[i]*bi*(n/b[i]))\%n;
35
36
37
        return x;
38
39
   };
```

5.5 Base 2 Fast Fourier Transfrom

```
/* Base_2_Fast_Fourier_Transfrom
1
2
   * (): transfrom
3
   * []: inversion */
  struct b2_fft_t {
4
5
    typedef complex<double> cd_t;
6
    typedef vector<cd_t> vcd_t;
7
    vcd_t c;
    void brc(vcd_t &x) {
9
       int 1;
```

```
10
        for (1 = 1; 1 < x. size(); 1 <<= 1) {}
11
        c.resize(1);
12
        for (int i = 0; i < c.size(); i++) {
13
          int to = 0;
14
           for (int o = 1 >> 1, t = i; o; o >>= 1, t >>= 1)
             if (t\&1) to += o;
15
          c[to] = i < x.size()? x[i]: cd_t(0., 0.);
16
17
18
19
      void fft(int on) {
20
        double dpi = acos(-1)*on;
21
        for (int m = 1; m < c.size(); m << 1) {
22
           cd_t wn(cos(dpi/m), sin(dpi/m));
23
           for (int j = 0; j < c.size(); j += m << 1) {
             cd_t w = 1.;
24
             \label{eq:formula} \mbox{for (int } k = j \; ; \; k < j + m; \; k + +, \; w \; * = \; wn) \; \; \{
25
26
               cd_{-}t u = c[k], t = w*c[k+m];
27
               c[k] = u+t, c[k+m] = u-t;
28
29
          }
30
        }
31
        if (~on) return ;
        for (int i = 0; i < c.size(); i++)
32
33
          c[i] /= c.size()*1.;
34
35
      void operator () (vcd_t &x) {
36
        brc(x), fft(1), x = c;
37
38
      void operator [] (vcd_t &x) {
39
        brc(x), fft(-1), x = c;
40
41
    };
```

5.6 Triangle Diagonal Matrix Algorithm

```
1
   /* Triangle_Diagonal_Matrix_Algorithm
2
    * */
3
   template < class T> struct tdma_t {
4
      void operator () (int n, T *a, T *b, T *c, T *d, T *x) {
5
        for (int i = 0; i < n; i++) {
          T tp = a[i]/b[i-1];
6
7
          b[i] = tp*c[i-1];
8
          d[i] = tp*d[i-1];
9
        x[n-1] = d[n-1]/b[n-1];
10
         \  \, \text{for (int $i=n-2$; $\tilde{\ }i$; $i--$) $x[i] = (d[i]-c[i]*x[i+1])/b[i]$;} \\
11
12
13
   };
```

6 pattern

6.1 KMP

```
1   /* KMP
2   * */
3   template < class T> struct kmp_t {
4     void get(T *p, int pl, int *f) {
5     for (int i = 0, j = f[0] = -1; i < pl; f[++i] = ++j)</pre>
```

```
for (; \tilde{j} \&\& p[i] != p[j]; ) j = f[j];
6
     void operator () (T *p, int pl, int *f) {
8
9
       int i = 0, j = f[0] = -1;
10
       for ( ; i < pl; i++, j++, f[i] = p[i] = p[j]? f[j]: j)
         for (; ~j && p[i] != p[j]; ) j = f[j];
11
12
13
     int operator () (T *s, int sl, T *p, int pl, int *f) {
14
       int i = 0, j = 0;
15
       16
         for ( ; \tilde{j} \&\& s[i] != p[j]; ) j = f[j];
17
       return j;
18
19
   };
```

6.2 Extend KMP

```
/* Extend_KMP
1
2
    * */
3
   template < class T> struct exkmp_t {
4
      void operator () (T *p, int pl, int *g) {
        g[g[1] = 0] = pl;
5
6
        for (int i = 1, k = 1; i < pl; (k+g[k] < i+g[i]? k = i : 0), i++)
          for (g[i] = min(g[i-k], max(k+g[k]-i, 0)); ; g[i]++)
7
             if (i+g[i] >= pl \mid | p[i+g[i]] != p[g[i]]) break;
8
9
      void operator () (T *s, int sl, int *f, T *p, int pl, int *g) {
10
         for \ (int \ i = f[0] = 0 \,, \ k = 0; \ i < sl \,; \ (k + f[k] < i + f[i]? \ k = i \colon 0) \,, \ i + +) 
11
12
          for (f[i] = min(g[i-k], max(k+f[k]-i, 0)); ; f[i]++)
13
            if (i+f[i] >= sl \mid | f[i] >= pl \mid | s[i+f[i]] != p[f[i]]) break;
14
      }
15
   };
```

6.3 Manacher

```
/* Manacher
1
2
    * */
   template < class T> struct mana_t {
     void operator () (T *s, int &n, int *p) {
4
5
        for (int i = n <<1; i >= 0; i --) s[i] = i &1? <math>s[i >>1]: -1;
6
        p[s[n = n << 1|1] = 0] = 1;
7
        for (int i = p[1] = 2, k = 1; i < n; i++) {
          p[i] = \min(p[2*k-i], \max(k+p[k]-i, 1));
8
9
          for (; p[i] \le i \&\& i+p[i] < n \&\& s[i-p[i]] == s[i+p[i]]; ) p[i]++;
10
          if (k+p[k] < i+p[i]) k = i;
11
        }
12
     }
13
   };
```

6.4 Minimum Notation

```
1  /* Minimum_Notation
2  **/
3  template < class T, class C> struct mnn_t {
4   int operator () (T *s, int n) {
5   int i = 0, j = 1;
6  for (int k = 0; k < n; )</pre>
```

6.5 AC automaton

```
1
   /* AC_automaton
2
    * */
3
   template < class T, int n, int m> struct aca_t {
4
     struct node {
        node *s[m], *p;
5
6
        int ac;
7
     s[n], *top, *rt, *q[n];
8
     void init() {
9
        memset(top = s, 0, sizeof(s));
10
        rt = top++;
11
     void put(T *k, int l, int ac) {
12
13
        node *x = rt;
14
        for (int i = 0; i < 1; i++) {
          if (!x->s[k[i]]) x->s[k[i]] = top++;
15
16
          x = x - s[k[i]];
17
18
       x\rightarrow ac = ac;
19
20
     void link() {
21
        int l = 0;
22
        for (int i = 0; i < m; i++)
23
          if (rt->s[i]) (q[1++] = rt->s[i])->p = rt;
          else rt \rightarrow s[i] = rt;
24
25
        for (int h = 0; h < 1; h++)
          for (int i = 0; i < m; i++)
26
27
            if (q[h]->s[i]) {
28
              (q[1++] = q[h]->s[i])->p = q[h]->p->s[i];
29
              q[h]->s[i]->ac |= q[h]->s[i]->p->ac;
30
            else q[h]->s[i] = q[h]->p->s[i];
31
32
     void tom(int mt[][n]) {
33
        for (node *x = s; x < top; x++)
34
          for (int i = 0; i < m; i++)
35
            if (!x->s[i]->ac) mt[x-s][x->s[i]-s] = 1;
36
37
   };
```

6.6 Suffix Array

```
/* Suffix_Array
   * Notice that the input array should end with 0 (s[s's length-1] = 0)
   * and then invoke dc3, remember to expand N to 3 times of it. */
template<int N> struct sa_t {
   int wa[N], wb[N], wv[N], ws[N], r[N];
   void da(int *s, int n, int *sa, int m) {
   #define da_F(c, a, b) for (int c = (a); i < (b); i++)
   #define da_C(s, a, b, 1) (s[a] == s[b] && s[a+1] == s[b+1])
   #define da_R(x, y, z) da_F(i, 0, m) ws[i] = 0; da_F(i, 0, n) ws[x]++;\</pre>
```

```
10
       da_F(i, 1, m) ws[i] += ws[i-1]; da_F(i, 0, n) sa[-ws[y]] = z;
11
       int *x = wa, *y = wb;
12
       da_R(x[i] = s[i], x[n-i-1], n-i-1);
13
        for (int j = 1, p = 1; p < n; j *= 2, m = p) {
14
          da_F(i, (p = 0, n-j), n) y[p++] = i;
          da_F(i, 0, n) if(sa[i] >= j) y[p++] = sa[i]-j;
15
         da_{F}(i, 0, n) wv[i] = x[y[i]];
16
          da_R(wv[i], wv[n-i-1], y[n-i-1]);
17
18
          da_{-}F(i, (swap(x, y), x[sa[0]] = 0, p = 1), n)
19
            x[sa[i]] = da_C(y, sa[i-1], sa[i], j)? p-1: p++;
20
       }
21
22
     int dc3_c12(int k, int *r, int a, int b, int *wv) {
23
        if (k != 2) return r[a] < r[b] || r[a] == r[b] && wv[a+1] < wv[b+1];
24
       return r[a] < r[b] || r[a] = = r[b] && dc3 = c12(1, r, a+1, b+1, wv);
25
26
     void dc3(int *s, int n, int *sa, int m) {
27
   #define dc3_H(x) ((x)/3+((x)\%3 = 1? 0: tb))
28
   #define dc3_G(x) ((x) < tb? (x)*3+1: ((x)-tb)*3+2)
29
   \#define dc3_c0(s, a, b) (s[a]==s[b] \&\& s[a+1]==s[b+1] \&\& s[a+2]==s[b+2])
   #define dc3-F(c, a, b) for (int c = (a); c < (b); c++)
31
   \#define dc3_sort(s, a, b, n, m) dc3_F(i, 0, n) wv[i] = (s)[(a)[i]];
       dc3_F(i, 0, m) ws[i] = 0; dc3_F(i, 0, n) ws[wv[i]]++;
32
       dc3_F(i, 1, m) ws[i] += ws[i-1];
33
34
       dc3_F(i, 0, n) (b)[--ws[wv[n-i-1]]] = a[n-i-1];
35
       int i, j, *rn = s+n, *san = sa+n, ta = 0, tb = (n+1)/3, tbc = 0, p;
36
       dc3_{F}(i, s[n] = s[n+1] = 0, n) if (i\%3) wa [tbc++] = i;
37
       dc3-sort(s+2, wa, wb, tbc, m);
38
       dc3_{sort}(s+1, wb, wa, tbc, m);
39
       dc3_sort(s, wa, wb, tbc, m);
       dc3_F(i, (rn[dc3_H(wb[0])] = 0, p = 1), tbc)
40
41
          rn[dc3_H(wb[i])] = dc3_c0(s, wb[i-1], wb[i])? p-1: p++;
       if(p < tbc) dc3(rn, tbc, san, p);
42
43
       else dc3_F(i, 0, tbc) san[rn[i]] = i;
44
       dc3_F(i, 0, tbc) if(san[i] < tb) wb[ta++] = san[i]*3;
45
       if (n\%3 == 1) wb[ta++] = n-1;
46
       dc3_{sort}(s, wb, wa, ta, m);
       dc3_{F}(i, 0, tbc) wv[wb[i] = dc3_{G}(san[i])] = i;
47
       for (i = j = p = 0; i < ta && j < tbc; p++)
48
49
         sa[p] = dc3_c12(wb[j]\%3, s, wa[i], wb[j], wv)? wa[i++]:wb[j++];
50
       for ( ; i < ta; p++) sa[p] = wa[i++];
51
       for ( ; j < tbc; p++) sa[p] = wb[j++];
52
     void ch(int *s, int n, int *sa, int *h) {
53
       for (int i = 1; i < n; i++) r[sa[i]] = i;
54
        for (int i = 0, j, k = 0; i < n-1; h[r[i++]] = k)
55
56
          for (k? k--: 0, j = sa[r[i]-1]; s[i+k] = s[j+k]; k++);
57
     void icats(int *b, int *l, char *s) {
58
       static int delim = 'z'+1;
59
60
       for (*1 += strlen(s)+1; *s; s++) *b++ = *s;
61
       *b++ = delim++;
62
63
   };
```

6.7 Suffix Automaton

```
1 /* Suffix_Automaton
2 * */
```

```
template<int N, int M> struct sam_t {
      static const int n = N*3;
 4
 5
      struct node {
 6
        node *s[M], *p;
 7
        int 1;
 8
        int range() {
9
          return 1-(p? 1-p->1: 0);
10
11
      } s[n], *top, *back;
12
      node *make(int 1) {
13
        memset(top, 0, sizeof(node));
14
        top \rightarrow l = l;
15
        return top++;
16
17
      void init() {
18
        top = s;
19
        back = make(0);
20
21
      void put(int k) {
22
        node *x = make(back -> l + 1), *y = back;
23
        for ( ; y \&\& !y -> s[k]; y = y -> p) y -> s[k] = x;
24
        if (!y) x - > p = s;
25
        else {
26
          node *w = y - > s[k];
27
          if (w->l == y->l+1) x->p = w;
28
          else {
29
             node *z = make(0);
30
             *z = *w;
             z->l = y->l+1;
31
32
             x->p = w->p = z;
33
             for ( ; y &  y - s[k] = w; y = y - p) y - s[k] = z;
34
35
36
      }
37
    };
```

7 data

7.1 RMQ Sparse Table

```
1
   /* RMQ_Sparse_Table
2
    * */
3
   template<int N> struct rmq_t {
4
     int s [20][N], *k;
5
     void operator () (int l, int *_k) {
6
       k = _k;
7
       for (int i = 0; i < 1; i++) s[0][i] = i;
       for (int i = 1; i < 20; i++)
8
9
          if ((1 << i) <= 1) for (int j = 0; j < 1; j++)
            if (k[s[i-1][j]] < k[s[i-1][j+(1<<(i-1))]]) s[i][j] = s[i-1][j];
10
11
            else s[i][j] = s[i-1][j+(1<<(i-1))];
12
13
     int operator () (int 1, int r) {
14
       if (1 > r) swap(1, r);
15
       int i = r-l+1, o = 1, j = 0;
16
       for (int o = 1; o \le i; o \le 1) j++;
17
       j - -, r = r - (1 < < j) + 1;
       return k[s[j][1]] < k[s[j][r]]? s[j][1]: s[j][r];
18
19
```

 $20 \mid \};$

7.2 Splay

```
/* Splay
1
2
    * */
3
   struct splay_t {
4
      struct node {
5
        node *p, *s[2];
6
        int size , key;
        int sum, lsum, rsum, msum;
7
8
        int cover_tag , reverse_tag ;
9
        int side() {
10
          return p\rightarrow s[1] = this;
11
12
        int rank() {
13
          return s[0]? 1+s[0] -> size: 1;
14
15
        node *set_cover(int _key) {
16
          cover\_tag = 1;
17
          key = _key;
18
          sum = size*key;
19
          lsum = rsum = msum = max(sum, key);
20
          return this;
21
22
        node *set_reverse() {
23
          reverse_tag = 1;
          swap(s[0], s[1]);
24
25
          swap(lsum, rsum);
26
          return this;
27
        }
28
        node *push() {
29
          for (int i = 0; i < 2; i++) {
30
            if (!s[i]) continue;
31
            if (cover_tag) s[i]->set_cover(key);
32
            if (reverse_tag) s[i]->set_reverse();
33
34
          cover_tag = reverse_tag = 0;
35
          return this;
36
37
        node *merge_sum(node *x, node *y) {
38
          if (!x | | !y) return \&(*this = x? *x: *y);
39
          sum = x->sum+y->sum;
40
          lsum = max(x->lsum, x->sum+y->lsum);
41
          rsum = max(y->rsum, x->rsum+y->sum);
42
          msum = max(x->msum, y->msum);
43
          msum = max(msum, max(lsum, rsum));
44
          msum = max(msum, x->rsum+y->lsum);
45
          return this;
46
47
        node *pull() {
48
          size = 1;
          sum = lsum = rsum = msum = key;
49
50
          for (int i = 0; i < 2; i++) {
51
            if (!s[i]) continue;
            size += s[i] -> size;
52
53
          return merge_sum(node(*this).merge_sum(s[0], this), s[1]);
54
55
```

```
node *set(int b, node *x) {
 56
 57
            if (push()->s[b] = x) x->p = this;
58
           return pull();
 59
 60
         node *get(int b) {
 61
           return push()->s[b];
 62
         node *cut(int b, node *&x) {
 63
 64
            if (x = push()->s[b]) s[b]->p = 0;
 65
           s[b] = 0;
 66
           return pull();
 67
         node *spin() {
 68
 69
           node *y = p->push();
 70
            int b = push() - side();
 71
            if (p = y->p) p->s[y->side()] = this;
 72
            if (y->s[b] = s[!b]) s[!b]->p = y;
 73
           return (s[!b] = y)->pull()->p = this;
 74
 75
         node *fine(node *x = 0) {
 76
            for ( ; p != x; spin())
 77
              if (p\rightarrow p != x)
 78
                if (side() = p->side()) p->spin();
 79
                else spin();
 80
           return pull();
 81
         node *pick(int k, node *y = 0) {
 82
 83
           node *x = this;
            for ( ; x->rank() != k; ) {
 84
              int b = x->rank() < k;
 85
 86
              k = b*x->rank();
 87
              x = x \rightarrow get(b);
 88
 89
           return x \rightarrow \sin (y);
 90
         }
 91
       };
92
       node *give(node *t = 0)  {
 93
         static node *top = 0;
 94
         static int size = 1;
95
         if (t) t \rightarrow s[1] = top, top = t;
96
         else {
97
            if (!top) {
 98
              top = new node [size <<=1];
              for (int i = 0; i < size -1; i++)
99
                top[i].s[1] = top+i+1;
100
101
              top[size -1].s[1] = 0;
102
            t = top, top = top \rightarrow s[1];
103
         }
104
105
         return t;
106
       node *make(int key) {
107
         node t = \{0, \{0\}, 1, \text{ key}, \text{ key}, \text{ key}, \text{ key}, \text{ key}, 0, 0\};
108
109
         return \&(*give() = t);
110
111
       void drop(node *&t) {
112
         if (!t) return;
113
         drop(t->s[0]), drop(t->s[1]);
114
         give(t), t = 0;
115
```

```
116
       void show(node *t) {
117
          if (!t) return;
118
          t->push();
119
          show (t->s[0]);
120
          printf(" -%d", t->key);
          \operatorname{show}(t->s[1]);
121
122
          t->pull();
123
124
     };
```

7.3 Treap

```
/* Treap
 1
 2
    * */
 3
    struct treap_t {
      struct node {
 4
 5
         node *s[2];
 6
         int size, weight;
 7
         int key, cover_tag, sum,
 8
             max_sum, lsum, rsum, reverse_tag;
 9
         int rank() {
           return s[0]? s[0] -> size +1: 1;
10
11
         node *set_cover(int _key) {
12
13
           key = _key;
           cover\_tag = 1;
14
15
           sum = size*key;
16
           \max_{\text{sum}} = \text{lsum} = \text{rsum} = \max(\text{key}, \text{sum});
17
           return this;
18
         }
19
         node *set_reverse() {
20
           reverse_tag ^= 1;
21
           swap(s[0], s[1]);
22
           swap(lsum, rsum);
23
           return this;
24
25
         node *push()  {
26
           for (int i = 0; i < 2; i++) {
             if (!s[i]) continue;
27
28
             if (cover_tag) s[i]->set_cover(key);
              if (reverse_tag) s[i]->set_reverse();
29
30
31
           cover_tag = reverse_tag = 0;
32
           return this;
33
34
         node *merge\_sum(node *x, node *y) {
35
           if (!x || !y) {
             x? *this = *x: *this = *y;
36
37
              return this;
38
           }
39
           sum = x->sum+y->sum;
40
           lsum = max(x->lsum, x->sum+y->lsum);
41
           rsum = max(y->rsum, x->rsum+y->sum);
42
           \max_{\text{sum}} = \max(x->\max_{\text{sum}}, y->\max_{\text{sum}});
43
           \max_{\text{sum}} = \max(\max_{\text{sum}}, \max(\text{lsum}, \text{rsum}));
44
           \max_{\text{sum}} = \max(\max_{\text{sum}}, x->rsum+y->lsum);
45
           return this;
46
47
         node *pull() {
```

```
48
            size = 1;
 49
           \max_{\text{sum}} = \text{sum} = \text{lsum} = \text{rsum} = \text{key};
50
           for (int i = 0; i < 2; i++) {
 51
              if (!s[i]) continue;
 52
              size += s[i] -> size;
 53
 54
           return merge_sum(node(*this).merge_sum(s[0], this), s[1]);
 55
         }
 56
       };
 57
       node *give(node *t = 0)  {
         static node *top = 0;
58
 59
         static int size = 1;
         if (t) t \rightarrow s[1] = top, top = t;
 60
 61
         else {
            if (!top) {
 62
 63
              top = new node[size <<=1];
 64
              for (int i = 0; i < size -1; i++)
 65
                top[i].s[1] = top+i+1;
 66
              top[size -1].s[1] = 0;
 67
 68
           t = top, top = top \rightarrow s[1];
 69
         }
 70
         return t;
 71
 72
       node *make(int key) {
 73
         node t = \{\{0\}, 1, rand()*rand(), key, 0, key, key, key, key, 0\};
         return \&(*give() = t);
 74
 75
 76
       void drop(node *&t) {
77
         if (!t) return;
 78
         drop(t->s[0]), drop(t->s[1]);
 79
         give(t), t = 0;
 80
 81
       void merge(node *x, node *y, node *&t) {
 82
         if (!x | | !y) t = x? x: y;
 83
         else if (x->weight < y->weight)
 84
           x->push(), merge(x->s[1], y, x->s[1]), t = x->pull();
         else y->push(), merge(x, y->s[0], y->s[0]), t = y->pull();
 85
 86
 87
       void split(node *t, int k, node *&x, node *&y) {
 88
         if (!k) x = 0, y = t;
 89
         else if (t\rightarrow size == k) x = t, y = 0;
 90
         else if (k < t->rank())
           y = t - push(), split(t - s[0], k, x, y - s[0]), y - pull();
91
         else x = t-push(), split(t-s[1], k-t-rank(), x-s[1], y), x-pull();
92
93
 94
       void slice (node *&t, int l = -1, int r = -1) {
95
         static node *a, *b;
96
         if (^{-}1) split (t, l-1, a, b), split (b, r-l+1, t, b);
 97
         else merge(t, b, b), merge(a, b, t);
98
       }
99
       void show(node *t) {
100
         if (!t) return;
101
         t \rightarrow push();
102
         show (t->s[0]);
         \label{eq:printf} {\tt printf("-\%2d"}\;,\;\; t\!-\!\!>\!\! key\;)\;;
103
104
         show (t->s[1]);
105
         t \rightarrow pull();
106
107
       int ask_sum(node *t) {
```

```
108 | return t? t->sum: 0;

109 | }

110 | int ask_max_sum(node *t) {

111 | return t? t->max_sum: 0;

112 | }

113 | };
```

7.4 Link-Cut Tree

```
1
   /* Link-Cut_Tree
 2
    * */
 3
    template<int N> struct lct_t {
      struct node {
 4
 5
        node *s[2], *p;
 6
        int sz, rev, w, mx, at;
 7
        node *sets(int b, node *x) {
 8
          if (s[b] = x) x \rightarrow p = this;
          return this;
9
10
11
        bool root() {
12
          return !p || !(p->s[0] = this || p->s[1] = this);
13
14
        bool which() {
          return p\rightarrow s[1] = this;
15
16
        node *set() {
17
18
          swap(s[0], s[1]);
          rev = 1;
19
20
          return this;
21
        }
22
        node *cover(int d) {
23
          w += d;
24
          mx += d;
25
          at += d;
26
          return this;
27
28
        node *push() {
29
          if (at) {
30
             for (int i = 0; i < 2; i++)
31
               if (s[i]) s[i] \rightarrow cover(at);
32
             at = 0;
33
          if (rev) {
34
             for (int i = 0; i < 2; i++)
35
               if (s[i]) s[i] \rightarrow set();
36
37
             rev = 0;
38
          }
39
          return this;
40
41
        node *pull() {
42
          sz = 1;
43
          mx = w;
          for (int i = 0; i < 2; i++)
44
45
             if (s[i]) {
               sz += s[i] -> sz;
46
               mx = max(mx, s[i]->mx);
47
48
49
          return this;
50
```

```
51
         node *spin() {
 52
           node *y = p - push();
 53
           int b = push() -> which();
 54
           y - sets(b, s[!b]) - spull();
 55
           if (y->root()) p = y->p;
 56
           else y\rightarrow p\rightarrow sets(y\rightarrow which(), this);
57
           return sets(!b, y);
 58
 59
         node *splay() {
 60
           for ( ; !root(); )
              if (p->root()) spin();
 61
 62
              else {
                if (which() = p->which()) p->spin();
 63
 64
                else spin();
 65
                spin();
 66
 67
           return pull();
 68
         node *end(int b) {
 69
 70
           node *x = this;
 71
           for ( ; x->push()->s[b]; x = x->s[b]) ;
 72
           return x;
 73
 74
       \} lct [N], *top;
 75
       void init() {
 76
         top = lct;
 77
       node *make(int w) {
 78
         *top = (node)\{\{0, 0\}, 0, 1, 0, w, w\};
 79
 80
         return top++;
 81
 82
       node *access(node *x, int o = 0, int d = 0) {
 83
         static node rv;
         for (node *y = x, *z = 0; y; z = y, y = y -> p) {
 84
 85
           y \rightarrow splay() \rightarrow push();
 86
           if (!y->p) {
 87
              if (o == 1) {
                y->w += d;
 88
                if (y->s[1]) y->s[1]->cover(d);
 89
 90
                if (z) z->cover(d);
91
              else if (o = 2) {
 92
                int mx = y->w;
 93
                if (y->s[1]) mx = max(mx, y->s[1]->mx);
                if (z) mx = max(mx, z->mx);
 94
 95
                rv.mx = mx;
 96
                return &rv;
 97
98
99
           y->sets(1, z)->pull();
100
101
         return x->splay();
102
103
       node *join(node *x, node *y) {
104
         return x \rightarrow p = y;
105
106
       node *cut(node *x) {
107
         if (access(x)->s[0]) x->s[0]->p = 0;
108
         x - > s[0] = 0;
109
         return x;
110
```

```
111
       node * find (node *x)  {
112
         return access(x) -> end(0);
113
114
       node *rooting(node *x) {
115
        return access(x) - set();
116
       node *cover(node *x, node *y, int w) {
117
118
         access(x);
119
         access(y, 1, w);
120
         return x;
121
       int ask(node *x, node *y) {
122
123
         access(x);
124
         return access (y, 2)->mx;
125
    };
126
```

7.5 Functional Segment

```
1
    /* Functional_Segment
 2
     * */
 3
    template<int N> struct fs_t {
 4
      struct node {
         \quad \text{int} \quad l \ , \quad r \ , \quad sm \ ;
 5
 6
         node *ls , *rs;
 7
         int m()  {
 8
           return l+r>>1;
9
10
      s[N*20], *top;
11
      void init() {
12
         top = s;
13
      node *phi(int l, int r) {
14
         node \ *x = top++, \ t = \{l \ , \ r \ , \ 0\};
15
16
         *x = t;
17
         if (1 < r) 
           x -> ls = phi(l, x->m());
18
19
           x -> rs = phi(x -> m() + 1, r);
         }
20
21
         return x;
22
23
      node *put(int k, node *y) {
24
         node *x = top++;
25
         *x = *y;
26
         x->sm++;
27
         if (x->1 < y->r) {
            \mbox{if} \ (k <= x->\!\! m()) \ x->\!\! ls \ = \ put(k\,, \ y-\!\!>\!\! ls\,); 
28
29
           else x\rightarrow rs = put(k, y\rightarrow rs);
         }
30
31
         return x;
32
33
      int ask(int 1, int r, node *x, node *y) {
34
         int rv = 0;
35
         if (1 \le x-)1 \&\& x-)r \le r) rv = x-)sm-y-)sm;
36
         else {
37
           if (1 \le x-m()) rv += ask(1, r, x->ls, y->ls);
           if (x->m() < r) rv += ask(l, r, x->rs, y->rs);
38
39
40
         return rv;
```

```
egin{array}{c|c} 41 & \\ 42 & \\ \end{array} \};
```

7.6 Functional Trie

```
1
   /* Functional_Trie
2
3
   template<int N, int D> struct ftrie_t {
4
      struct node {
        node *s[2];
5
        int c[2];
6
      \} \ \ s \, [D*N\!\!+\!\!D] \; , \ \ *top \; , \ \ *phi \; ;
7
      void init() {
8
9
        top = s;
10
        phi = top++;
11
        phi - c[0] = phi - c[1] = 0;
12
        phi->s[0] = phi->s[1] = phi;
13
      node *put(int k, node *y, int d = D) {
14
15
        if (!d) return 0;
16
        node *x = top++;
17
        *x = *y;
18
        int i = k > (d-1)\&1;
        x->c[i]++;
19
20
        x->s[i] = put(k, y->s[i], d-1);
21
        return x;
22
      int ask(int k, node *x, node *y, int d = D) {
23
24
        if (!d) return 0;
25
        int i = k > (d-1)\&1;
26
        if (x->c[!i]-y->c[!i])
27
          return (1 << d-1) + ask(k, x->s[!i], y->s[!i], d-1);
28
        return ask(k, x->s[i], y->s[i], d-1);
29
30
   };
```

7.7 Lefist Tree

```
1
   /* Lefist_Tree
2
   */
3
   template<int N> struct lefist_t {
4
     struct node {
        node *l, *r;
5
6
        int k, d;
7
     \} s[N], *top;
8
     void init() {
9
        top = s;
10
11
     node *make(int k) {
        node *x = top++, t = \{0, 0, k, 0\};
12
13
        *x = t;
14
        return x;
15
16
     node *merge(node *x, node *y) {
17
        if (!x) return y;
18
        if (!y) return x;
        if (x->k < y->k) swap(x, y);
19
20
       x->r = merge(x->r, y);
```

```
21 | if (!x->l || x->r && x->l->d < x->r->d) swap(x->l, x->r);
22 | if (x->r) x->d = x->r->d+1;
23 | return x;
24 | }
25 | node *drop(node *x) {
26 | return merge(x->l, x->r);
27 | }
28 | };
```

8 geometry

8.1 Float Compare Functions

```
/* Float_Compare_Functions
1
2
    * */
3
   struct fc_t {
      double eps;
4
5
      fc_t() {
6
        eps = 1e-8;
7
      bool e(double lhs, double rhs) {
8
9
        return abs(lhs-rhs) < eps;
10
      bool l(double lhs, double rhs) {
11
12
        return lhs+eps < rhs;</pre>
13
      bool g(double lhs, double rhs) {
14
        return lhs-eps > rhs;
15
16
17
     fc;
```

8.2 2D point

```
1
   /* 2D_point
2
    * */
3
   struct pt_t {
4
     double x, y;
5
     pt_t(double_x = 0, double_y = 0) {
6
       x = _{-}x , y = _{-}y ;
7
     double operator [] (int b) {
8
       return b? b < 2? abs(x) + abs(y): x*x+y*y: sqrt(x*x+y*y);
9
10
11
     friend pt_t operator + (const pt_t &lhs, const pt_t &rhs) {
12
        return pt_t(lhs.x+rhs.x, lhs.y+rhs.y);
13
      friend pt_t operator - (const pt_t &lhs, const pt_t &rhs) {
14
        return pt_t(lhs.x-rhs.x, lhs.y-rhs.y);
15
16
17
      friend double operator * (const pt_t &lhs, const pt_t &rhs) {
18
        return lhs.x*rhs.x+lhs.y*rhs.y;
19
      friend double operator % (const pt_t &lhs, const pt_t &rhs) {
20
21
        return lhs.x*rhs.y-lhs.y*rhs.x;
22
23
     pt_t &input() {
        scanf("%lf%lf", &x, &y);
24
25
        return *this;
```

```
\begin{bmatrix} 26 & \\ 27 & \\ \end{bmatrix};
```

8.3 Angle Sort

```
1
   /* Angle_Sort
2
    * */
3
   struct asort_t {
4
     bool cmpl(pt_t lhs, pt_t rhs) {
       return fc.1(lhs.y, rhs.y) || (fc.e(lhs.y, rhs.y) && fc.1(lhs.x, rhs.x));
5
6
7
     static pt_t o;
     static bool cmp(pt_t lhs, pt_t rhs) {
8
9
       double c = (lhs-o)\%(rhs-o);
10
       if (!fc.e(c, 0.0)) return fc.g(c, 0.0);
11
       return fc.g((lhs-o)[1], (rhs-o)[1]);
12
     void operator () (vector<pt_t> &p) {
13
14
       int mn = 0;
15
       for (int i = 0; i < p.size(); i++)
         if (cmpl(p[i], p[mn])) mn = i;
16
17
       swap(p[0], p[mn]);
18
       o = p[0];
19
       sort(p.begin()+1, p.end(), cmp);
20
21
   asort;
22
   pt_t = asort_t :: o;
```

8.4 Graham Scan

```
/* Graham_Scan
1
2
    * */
3
   struct graham_t {
4
     vector < pt_t > p;
5
     double 1;
6
     graham_t (vector < pt_t > &ps) {
7
        asort(p = ps);
        vector < pt_t > s(p.begin(), p.begin()+2);
8
9
        ps.clear();
10
        for (int i = 2; i < p. size(); i++) {
11
          for (; fc.g((s[s.size()-2]-s.back())\%(p[i]-s.back()), 0.0);)
12
            ps.push_back(s.back()), s.pop_back();
13
          s.push_back(p[i]);
        }
14
15
        p = s;
16
        for (int i = l = 0; i < p.size(); i++)
17
          1 += (p[(i+1)\%p.size()]-p[i])[0];
18
19
   };
```

9 graph

9.1 Graph

```
1 /* Graph
2 /* */
```

```
template<int N> struct graph_t {
 4
      struct edge_t {
 5
        int v, to;
 6
      };
 7
      vector < edge_t > E;
 8
      int L[N];
      void init() {
 9
        E. clear();
10
11
        memset(L, -1, sizeof(L));
12
13
      void add(int u, int v) {
14
        edge_t t = \{v, L[u]\};
15
        L[u] = E. \operatorname{size}();
16
        E.push_back(t);
17
18
    };
```

9.2 Shortest Path Algorithm

```
1
   /* Shortest_Path_Algorithm
2
    * */
3
   template < class edge_t, int N> struct spfa_t {
4
     int d[N], b[N], c[N], s[N], mx[N];
     int operator () (vector<edge_t> &E, int *L, int n, int u) {
5
6
        memset(d, 0x7f, sizeof(d));
        memset(b, 0, sizeof(b));
7
8
        memset(c, 0, sizeof(c));
9
        d[s[s[0] = 1] = u] = 0;
10
        b[u] = c[u] = 1;
11
        for (; s[0]; ) {
12
          b[u = s[s[0] - -]] = 0;
          for (int e = L[u]; e; e = E[e].to) {
13
            int v = E[e].v, w = E[e].w;
14
15
            if (d[v]-w > d[u]) {
16
              d[v] = d[u] + w;
17
              if (!b[v]) {
                if ((c[v] += b[v] = 1) > n) return 0;
18
19
                s[++s[0]] = v;
20
21
22
          }
23
24
        return 1;
25
26
     struct node {
27
        int u, w;
        node (int _{u} = 0, int _{w} = 0): u(_{u}), w(_{w}) {}
28
29
        friend bool operator < (const node &lhs, const node &rhs) {
30
          return lhs.w > rhs.w;
31
       }
32
     };
33
     void operator () (vector<edge_t> &E, int *L, int u) {
34
       memset(d, 0x7f, sizeof(d));
35
       memset(b, 0, sizeof(b));
36
        priority_queue < node > q;
        for (q.push(node(u, d[u] = 0)); q.size();) {
37
38
          u = q. top().u, q. pop();
39
          if (b[u]++) continue;
40
          for (int e = L[u]; e = E[e].to) {
```

9.3 Bipartite Graph match

```
/* Bipartite_Graph_match
1
2
    * */
3
   template < class edge_t , int N> struct bgm_t {
4
      int vis [N], pre [N], lma [N], rma [N];
5
      bool bfs (vector < edge_t > &E, int *L, int u) {
6
        vector < int > q(1, u);
7
        memset(vis, 0, sizeof(vis));
        memset(pre, -1, sizeof(pre));
8
        for (int h = 0; h < q.size(); h++) {
9
10
          u = q[h];
          for (int e = L[u]; e = E[e].to) {
11
            int v = E[e].v;
12
13
             if (! vis [v]) {
14
               vis[v] = 1;
15
               if (rma[v] = -1) {
                 for (; ~u; ) {
rma[v] = u;
16
17
                   swap(v, lma[u]);
18
19
                   u = pre[u];
20
                 }
21
                 return 1;
22
               } else {
23
                 pre[rma[v]] = u;
24
                 q.push_back(rma[v]);
25
26
            }
          }
27
        }
28
29
        return 0;
30
31
      int operator () (vector<edge_t> &E, int *L, int V) {
32
        int mmat = 0;
33
        memset(lma, -1, sizeof(lma));
34
        memset(rma, -1, sizeof(rma));
35
        for (int u = 0; u < V; u++)
36
          mmat += bfs(E, L, u);
37
        return mmat;
38
39
   };
```

9.4 General Graph match

```
/* General_Graph_match
    * */
template<int N> struct blossom_t {
    deque<int> Q;
    int n;
    bool g[N][N], inque[N], inblossom[N];
```

```
7
      int match [N], pre [N], base [N];
      int findancestor(int u, int v){
 9
         bool inpath [N]={ false };
10
         while (1) {
11
           u=base[u];
12
           inpath[u] = true;
           if (\text{match}[u]==-1)break;
13
14
           u=pre[match[u]];
15
16
         while (1) {
17
           v=base[v];
           if (inpath[v]) return v;
18
19
           v=pre[match[v]];
         }
20
21
22
      void reset(int u, int anc){
23
         while (u!=anc) {
24
           int v=match[u];
25
           inblossom[base[u]]=1;
26
           inblossom[base[v]]=1;
27
           v=pre[v];
28
           if (base [v]!=anc) pre [v]=match [u];
29
30
        }
31
32
      void contract(int u,int v,int n){
33
         int anc=findancestor(u,v);
34
         //SET(inblossom, 0);
35
         memset(inblossom, 0, size of(inblossom));
36
         reset (u, anc); reset (v, anc);
37
         if (base [u]!=anc) pre [u]=v;
38
         if (base [v]! = anc) pre [v] = u;
39
         for (int i=1; i \le n; i++)
           if (inblossom [base[i]]) {
40
41
              base [i] = anc;
42
              if (!inque[i]) {
43
                Q. push_back(i);
                inque [i]=1;
44
             }
45
           }
46
47
      bool dfs(int S, int n){
48
         for (int i=0; i \le n; i++) pre [i]=-1, inque [i]=0, base [i]=i;
49
        Q. clear(); Q. push_back(S); inque[S]=1;
50
51
         while (!Q. empty()) {
52
           int u=Q. front();Q.pop_front();
53
           for (int v=1; v <= n; v++)
              if (g[u][v]&&base[v]!=base[u]&&match[u]!=v){
54
55
                if (v = S \mid | (match [v]! = -1 \& pre [match [v]]! = -1)) contract (u, v, n);
56
                else if (pre[v]==-1){
57
                  pre[v]=u;
                  if (\text{match}[v]! = -1)Q. push_back (\text{match}[v]), inque [\text{match}[v]] = 1;
58
59
                  else {
60
61
                     while (u!=-1)
62
                       v=pre[u];
63
                       int w=match[v];
64
                       match[u]=v;
65
                       match[v]=u;
66
                       u=w;
```

```
67
68
                            return true;
69
70
                 }
71
              }
72
73
           }
74
           return false;
75
76
        void init(int n) {
           \label{eq:this} \begin{array}{ll} \texttt{this} \mathbin{-\!\!\!>} n \; = \; n \, ; \\ \texttt{memset} \, (\, \mathtt{match} \, , -1 \, , \, \mathtt{sizeof} \, (\, \mathtt{match} \, ) \, ) \, ; \end{array}
77
78
           memset(g, 0, sizeof(g));
79
80
        void addEdge(int a, int b) {
81
           ++a;
82
           ++b;
83
           g[a][b] = g[b][a] = 1;
84
85
        int gao() {
86
           int ans = 0;
87
           for (int i = 1; i \le n; ++i) {
88
               if (match[i] = -1 \&\& dfs(i, n)) {
89
                 ++ans;
90
91
92
           return ans;
93
94
     };
```

9.5 Dancing Link

```
1
   /* Dancing_Link
2
3
   template<int N, int M> struct dancing {
4
   #define dfor(c, a, b) for (int c = a[b]; c != b; c = a[c])
5
     static const int row_size = N, column_size = M,
6
                    total_size = row_size * column_size;
7
     typedef int row[row_size],
              column [column_size],
8
9
              total[total_size];
10
     total 1, r, u, d, in_column;
11
     column s;
12
     int index , current_row , row_head;
13
     void init(int n)
14
15
        index = ++n;
        for (int i = 0; i < n; i++) {
16
17
          l[i] = (i - 1 + n) \% n;
          r[i] = (i + 1) \% n;
18
19
          u[i] = d[i] = i;
20
21
        current_row = 0;
22
       memset(s, 0, sizeof(s));
23
24
     void push(int i, int j)
25
26
        i++; j++;
        if (current_row < i) {
27
28
          row_head = l[index] = r[index] = index;
```

```
29
           current_row = i;
30
31
         l[index] = l[row_head]; r[index] = row_head;
32
         r[l[row_head]] = index; l[row_head] = index;
33
         u[index] = u[j]; d[index] = j;
34
         d[u[j]] = index; u[j] = index;
35
         s[j]++;
36
         \operatorname{in\_column}[\operatorname{index}++] = j;
37
38
      void exactly_remove(int c)
39
         l[r[c]] = l[c];
40
41
         r[l[c]] = r[c];
42
         dfor(i, d, c) {
           dfor (j, r, i) {
  u[d[j]] = u[j];
43
44
45
              d[u[j]] = d[j];
              s\left[\,i\,n\,\text{\_}colum\,n\left[\,j\,\right]\right]--;
46
47
48
         }
49
      }
50
      void exactly_resume(int c)
51
         dfor(i, u, c) {
52
           dfor\left(\,j\;,\;\;l\;,\;\;i\,\right)\;\;\{
53
54
              s[in\_column[j]]++;
              d[u[j]] = u[d[j]] = j;
55
56
57
58
         r[1[c]] = 1[r[c]] = c;
59
60
      bool exactly_dance(int step = 0)
61
62
         if (!r[0]) {
63
           return 1;
64
65
         int x = r[0];
         dfor(i, r, 0) {
66
           if (s[i] < s[x]) {
67
68
             x = i;
69
70
         exactly_remove(x);
71
72
         dfor(i, d, x) {
73
           dfor(j, r, i) {
74
              exactly_remove(in_column[j]);
75
76
           if (exactly_dance(step + 1)) {
77
              return 1;
78
79
           dfor(j, l, i) {
              exactly_resume(in_column[j]);
80
81
82
83
         exactly_resume(x);
84
         return 0;
85
86
      int limit;
87
      void remove(int c)
88
```

```
89
         dfor(i, d, c) {
90
           l[r[i]] = l[i];
91
           r[l[i]] = r[i];
92
93
       }
       void resume(int c)
94
95
         dfor(i, u, c) {
96
97
           r[l[i]] = l[r[i]] = i;
98
99
100
       bool dance (int step = 0)
101
         if (limit <= step + heuristic()) {</pre>
102
103
           return 0;
104
         if (!r[0]) {
105
106
           limit = min(limit, step);
107
           return 1;
108
109
         int x = r[0];
         dfor(i, r, 0) {
110
           if(s[i] < s[x]) {
111
112
             x = i;
113
114
         dfor(i, d, x) {
115
116
           remove(i);
           dfor(j, r, i) {
117
118
             remove(j);
119
120
           if (dance(step + 1)) {
121
             return 1;
122
123
           dfor(j, l, i) {
124
             resume(j);
125
126
           resume(i);
127
128
         return 0;
129
130
       int heuristic()
131
132
         int rv = 0;
         column \ visit = \{0\};
133
134
         dfor(c, r, 0) {
           if (!visit[c]) {
135
136
             rv++;
137
              visit[c] = 1;
138
              dfor(i, d, c) {
139
                dfor(j, r, i) {
                  visit[in\_column[j]] = 1;
140
141
142
             }
           }
143
         }
144
145
         return rv;
146
147
       int dfs()
```

148

```
149 | for (limit = heuristic(); !dance(); limit++) {}
150 | return limit;
151 | }
152 | #undef dfor
153 | };
```

9.6 Directed Minimum Spanning Tree

```
1
   /* Directed_Minimum_Spanning_Tree
2
    * */
3
   template<int N> struct dmst_t {
4
      struct edge_t {
5
        int u, v, w;
6
      };
7
      vector < edge_t > E;
8
      static const int inf = 0 \times 7f7f7f7f7f;
9
      int \ n, \ ine\left[N\right], \ pre\left[N\right], \ id\left[N\right], \ vis\left[N\right];
10
      void init(int _n) {
11
        n = _n;
12
        E. clear ();
13
      void add(int u, int v, int w) {
14
15
        edge_t t = \{u, v, w\};
16
        E. push_back(t);
17
      int operator () (int rt) {
18
19
        20
        for (;;) {
21
          fill (ine, ine+tn, inf);
22
          for (i = 0; i < E. size(); i++) {
23
            u = E[i].u; v = E[i].v; w = E[i].w;
24
             if (u != v \&\& w < ine[v]) {
               pre[v] = u;
25
               ine[v] = w;
26
            }
27
28
29
          for (u = 0; u < tn; u++) {
30
            if (u == rt) continue;
             if (ine[u] = inf)
31
32
               return -1;
33
34
          index = 0;
35
          fill(id, id + tn, -1);
          \label{eq:fill_state} \mbox{fill (vis , vis + tn , } -1);
36
37
          ine[rt] = 0;
38
          for (u = 0; u < tn; u++) {
39
            rv += ine[v = u];
             for (; v != rt \&\& vis[v] != u \&\& id[v] == -1;) {
40
               vis[v] = u;
41
42
               v = pre[v];
43
             if (v != rt \&\& id [v] == -1) {
44
               for (i = pre[v]; i != v; i = pre[i]) id[i] = index;
45
46
               id[v] = index++;
47
48
          if (index = 0) break;
49
50
          for (u = 0; u < tn; u++)
51
             if (id[u] = -1) id[u] = index++;
```

```
52
          for (i = 0; i < E. size(); i++) {
53
            v = E[i].v;
54
            E[i].u = id[E[i].u];
            E[i].v = id[E[i].v];
55
56
            if (E[i].u != E[i].v) E[i].w -= ine[v];
57
58
          tn = index;
59
          rt = id[rt];
60
61
        return rv;
62
63
   };
```

9.7 Spfa Cost Stream

```
/* Spfa_Cost_Stream
 1
 2
    * */
 3
    template < class edge_t, int N> struct ek_t {
 4
      vector < edge_t > E;
 5
      static const int inf = 0x7f7f7f7f;
      int \ n, \ *L, \ src \ , \ snk \ , \ dis \left[ N \right], \ ra \left[ N \right], \ inq \left[ N \right];
 6
 7
      int spfa(int u) {
 8
         vector < int > q(1, u);
 9
         memset(dis, 0x3f, sizeof(int)*n);
         memset(ra, -1, sizeof(int)*n);
10
         memset(inq, 0, sizeof(int)*n);
11
12
         dis[u] = 0;
13
         inq[u] = 1;
         for (int h = 0; h < q.size(); h++) {
14
15
           \mathbf{u} = \mathbf{q}[\mathbf{h}], \quad \mathbf{inq}[\mathbf{u}] = 0;
16
           for (int e = L[u]; e; e = E[e].to) {
17
              int v = E[e].v, w = E[e].w, c = E[e].c;
              if (w \&\& dis[v] > dis[u]+c) {
18
                dis[v] = dis[u]+c;
19
20
                ra[v] = e^1;
21
                if (inq[v]) continue;
22
                inq[v] = 1;
23
                q.push_back(v);
24
25
           }
26
27
         return ra [\operatorname{snk}] != -1;
28
29
      int operator () (vector<edge_t> _E, int *_L, int _n, int _src, int _snk) {
30
        E = _{-}E, L = _{-}L, n = _{-}n;
31
         src = \_src, snk = \_snk;
32
         int mmf = 0;
33
         for ( ; spfa(src); ) {
34
           int mf = inf;
35
           for (int e = ra[snk]; e; e = ra[E[e].v])
36
             mf = min(mf, E[e^1].w);
37
           for (int e = ra[snk]; e; e = ra[E[e].v])
             E[e].w += mf, E[e^1].w -= mf;
38
39
           mmf += dis [snk] * mf;
40
41
         return mmf;
42
43
    };
```

9.8 KM Maximum perfect match

```
/* KM_Maximum_perfect_match
    * Notice that we could use this, when left side has the same amount
3
    * as right side. (perfect match)
4
    * If the situation above doesn't be hold, Cost-Flow algorithm is recommanded.
5
6
   template < class edge_t, int N> struct km_t {
7
     vector<edge_t> E;
     static const int inf = 0 \times 7f7f7f7f7f;
8
9
     typedef int kmia_t[N];
10
     kmia_t mat, lta, rta, sla, lvi, rvi;
11
     int n, *L;
     int dfs(int u) {
12
        lvi[u] = 1;
13
14
        for (int e = L[u]; e; e = E[e].to) {
15
          int v = E[e].v, w = E[e].w;
16
          if (!rvi[v]) {
17
            int t = lta[u] + rta[v] - w;
            if (!t) {
18
              rvi[v] = 1;
19
              if (mat[v] = -1 \mid | dfs(mat[v])) {
20
21
                mat[v] = u;
22
                return 1;
23
24
            else if (t < sla[v]) sla[v] = t;
25
26
27
        return 0;
28
29
     int operator () (vector<edge_t> &_E, int _L[N], int _n) {
30
       E = _E, L = _L, n = _n;
31
       memset(lta, 0, sizeof(lta));
32
        memset(rta, 0, sizeof(rta));
33
        memset(mat, -1, sizeof(mat));
        for (int u = 0; u < n; u++)
34
          for (int e = L[u]; e = E[e].to)
35
36
            if (lta[u] < E[e].w) lta[u] = E[e].w;
37
        for (int u = 0; u < n; u++) {
38
          for (int e = L[u]; e; e = E[e].to) sla[E[e].v] = inf;
          for ( ; ; ) {
39
            memset(lvi, 0, sizeof(lvi));
40
41
            memset(rvi, 0, sizeof(rvi));
            if (dfs(u)) break;
42
            int mn = inf;
43
44
            for (int v = 0; v < n; v++)
              if (!rvi[v]) mn = min(mn, sla[v]);
45
46
            for (int v = 0; v < n; v++) {
47
              if (lvi[v]) lta[v] = mn;
              if (rvi[v]) rta[v] += mn;
48
49
              else sla [v] -= mn;
50
          }
51
52
53
        int rv = 0;
        for (int v = 0; v < n; v++) if (mat[v])
54
          for (int e = L[mat[v]]; e; e = E[e].to)
55
            if (E[e].v = v) {
56
57
              rv += E[e].w;
58
              break;
```

```
59 | }
60 | return rv;
61 | }
62 | };
```

9.9 Doubling LCA

```
1
   /* Doubling_LCA
2
    * */
3
   template < class edge_t, int N> struct lca_t {
4
     static const int M = 16;
     int d[N], a[N][M], p[1 << M];
5
     void operator () (vector<edge_t> E, int *L, int u) {
6
7
       vector < int > q(1, u);
8
       memset(a, -1, sizeof(a));
       for (int h = d[u] = 0; h < q.size(); h++) {
9
10
         u = q[h];
          for (int i = 1; i < M; i++)
11
12
            if ( a[u][i-1]) a[u][i] = a[a[u][i-1]][i-1];
13
          for (int e = L[u]; e = E[e].to) {
            int v = E[e].v;
14
            if (v = a[u][0]) continue;
15
            d[v] = d[u] + 1;
16
            a[v][0] = u;
17
            q.push_back(v);
18
19
20
21
       for (int i = 0; i < M; i++) p[1 << i] = i;
22
23
     int skip(int u, int x) {
24
       for ( ; x; x = -x\&x) u = a[u][p[-x\&x]];
25
       return u;
26
27
     int operator () (int u, int v) {
28
       if (d[u] < d[v]) swap(u, v);
29
       u = skip(u, d[u]-d[v]);
30
       if (u = v) return u;
       for (int i = M-1; i && a[u][0] != a[v][0]; i--)
31
32
          if (~a[u][i] && a[u][i] != a[v][i])
33
            u = a[u][i], v = a[v][i];
34
       return a[u][0];
35
36
   };
```

9.10 Shortest Augment Path

```
1
    /* Shortest_Augment_Path
2
    template < class edge_t, int N> struct sap_t {
4
      int \operatorname{dis}[N], \operatorname{gap}[N], L[N], \operatorname{se}[N];
      int operator () (vector<edge_t> &E, int *L, int V, int src, int snk) {
5
        int mxf = 0, te = 0;
6
7
        memcpy(LL, L, sizeof(L));
        memset(dis, -1, sizeof(dis));
8
9
        memset(gap, 0, sizeof(gap));
10
        gap[dis[snk] = 0] = 1;
11
        vector < int > q(1, snk);
12
        for (int h = 0; h < q.size(); h++)
```

```
13
          for (int i = L[q[h]]; i != -1; i = E[i].to)
14
            if (E[i].w && dis[E[i].v] < 0) {
15
              gap[dis[E[i].v] = dis[q[h]]+1]++;
16
              q.push_back(E[i].v);
17
18
        for (int u = src; dis[src] < V; ) {
          for (int &i = L[u]; i != -1; i = E[i].to)
19
20
            if (E[i].w \&\& dis[u] = dis[E[i].v] + 1) break;
          if (_L[u] != −1) {
21
22
            u = E[se[te++] = _L[u]].v;
23
            if (u = snk) {
24
              int _i = 0, _i = 0 \times 7 fffffff;
25
              for (int i = 0; i < te; i++)
26
                if (E[se[i]].w < mf) {
27
                  mf = E[se[i]].w;
28
                   _{-}i = i;
29
30
              for (int i = 0; i < te; i++) {
31
                E[se[i]].w = mf;
32
                E[se[i]^1].w += mf;
33
              }
34
              mxf += mf;
              u = E[se[te = _i]^1].v;
35
36
37
            continue;
38
39
          int md = V;
          _{L}[u] = -1;
40
          for (int i = L[u]; i != -1; i = E[i].to)
41
42
            if (E[i].w && dis[E[i].v] < md) {
43
              md = dis[E[i].v];
44
              L[u] = i;
45
          if (!--gap[dis[u]]) break;
46
47
          gap[dis[u] = md+1]++;
48
          if (u != src) u = E[se[te--1]^1].v;
49
50
        return mxf;
51
52
   };
```

9.11 ZKW Cost Stream

```
1
   /* ZKW_Cost_Stream
2
   * */
3
   template < class edge_t, int N> struct zkw_t {
4
     vector<edge_t> E;
     5
6
     int n, src, snk, mc, mf, dis, vis[N], *L;
7
     int ap(int u, int f) {
8
       if (u = snk) {
9
        mc += dis * f;
10
        mf += f;
11
         return f;
12
       vis[u] = 1;
13
14
       int rf = f;
15
       for (int e = L[u]; e > -1; e = E[e].to)
16
         if (!vis[E[e].v] && E[e].w && !E[e].c) {
```

```
17
              int df = ap(E[e].v, min(rf, E[e].w));
18
             E[e].w -= df;
19
             E[e^1].w += df;
20
              rf = df;
21
              if (!rf) return f;
22
23
         \begin{array}{ll} \textbf{return} & f\!-\!r\,f\;; \end{array}
24
25
      int ml() {
26
         int md = inf;
27
         for (int u = 0; u < n; u++) if (vis[u])
28
           for (int e = L[u]; e = E[e]. to)
              if (!vis[E[e].v] && E[e].w)
29
                md = min(md, E[e].c);
30
31
         if (md = inf) return 0;
         for (int u = 0; u < n; u++) if (vis[u])
32
33
           for (int e = L[u]; e : e = E[e].to) {
34
             E[e].c = md;
35
             E[e^1].c += md;
36
37
         dis += md;
38
         return 1;
39
40
      int operator () (vector<edge_t> &_E, int *_L, int _n, int _src, int _snk) {
41
        E = _{-}E, L = _{-}L, n = _{-}n;
42
         \operatorname{src} = \operatorname{\_src}, \operatorname{snk} = \operatorname{\_snk};
43
         mf = mc = dis = 0;
44
         for ( ; ; ) {
           for (;;) {
45
              memset(vis, 0, sizeof vis);
46
              if (!ap(src, inf)) break;
47
48
49
           if (!ml()) break;
50
51
         return mc;
52
      }
53
    };
```

10 graph test

10.1 Graph

```
/* Graph
1
2
    * */
3
   struct graph_t {
4
      struct edge_t {
5
        int v, to;
6
      };
7
      vector<edge_t> e;
8
      vector < int > h;
      edge_t & operator [] (int x) {
9
10
        return e[x];
11
12
      int & operator () (int x) {
13
        return h[x];
14
15
      int size() {
16
        return h.size();
17
```

```
18
     void init(int n) {
19
        e.clear(), h.resize(n);
20
        fill(h.begin(), h.end(), -1);
21
22
     void add(int u, int v) {
23
        edge_t t = \{v, h[u]\};
24
        h[u] = e.size();
25
        e.push_back(t);
26
27
     void badd(int u, int v) {
28
        add(u, v), add(v, u);
29
30
   };
```

10.2 Shortest Augment Path

```
1
    /* Shortest_Augment_Path
2
3
   template < class graph_t > struct sap_t {
4
      vector < int > dis, gap;
      int dfs(graph_t &g, int src, int snk, int u, int f = ~1u>>1) {
5
        if (u = snk) return f;
6
7
        int rf = f, md = g.size()-1;
        for (int e = g(u); e = g[e].to) {
8
          int v = g[e].v, w = g[e].w;
9
          if (!w) continue;
10
11
          md = min(md, dis[v]);
12
          if (dis[u] != dis[v]+1) continue;
13
          int df = dfs(g, src, snk, v, min(w, f));
14
          g[e].w = df, g[e^1].w + df;
15
          if (gap[src] = g.size() \mid | !(rf -= df)) return f;
16
        if (!--gap[dis[u]]) gap[src] = g.size();
17
18
        else gap [dis [u] = md+1]++;
19
        return f-rf;
20
21
      int operator () (graph_t &g, int src, int snk) {
22
        dis.clear(), gap.clear();
23
        for (int i = g.size() << 1; i; i--)
24
          dis.push_back(-1), gap.push_back(0);
25
        vector < int > q(gap[dis[snk] = 0] = 1, snk);
26
        for (int h = 0; h < q.size(); h++)
27
          for (int e = g(q[h]); e; e = g[e].to)
28
             if (g[e^1].w && !~dis[g[e].v])
29
               \operatorname{gap}[\operatorname{dis}[g[e].v] = \operatorname{dis}[q[h]]+1]++, q.\operatorname{push\_back}(g[e].v);
30
        for (int i = 0; i < g.size(); i++)
31
          if (!^{\sim} dis[i]) gap[dis[i] = 0]++;
32
        int result = 0;
33
        for (; gap[src] < g.size();) result += dfs(g, src, snk, src);
34
        return result;
35
36
   };
```

10.3 Strong Connected Component

```
1  /* Strong_Connected_Component
2  **/
3  template < class graph_t > struct scc_t {
```

```
4
     int time, cc;
5
     vector <int> dfn, low, in, pushed, st;
6
     void dfs(graph_t &g, int u) {
7
        st.push_back(u), pushed[u] = 1;
8
        dfn[u] = low[u] = time++;
9
        for (int e = g(u); e; e = g[e].to) {
          int v = g[e].v;
10
11
          if (!^{\sim} dfn[v]) dfs(g, v), low[u] = min(low[u], low[v]);
12
          else if (pushed[v]) low[u] = min(low[u], dfn[v]);
13
14
        if (dfn[u] = low[u]) {
15
          for (;;) {
            in[u = st.back()] = cc;
16
17
            st.pop\_back(), pushed[u] = 0;
            if (dfn[u] = low[u]) break;
18
19
20
          cc++;
21
        }
22
23
     void operator () (graph_t &g) {
        dfn.clear(), low.clear(), in.clear(), pushed.clear(), st.clear();
24
25
        for (int i = 0; i < g.size(); i++)
26
          dfn.push\_back(-1), low.push\_back(-1), in.push\_back(-1), pushed.push\_back(0);
27
        for (int u = time = cc = 0; u < g.size(); u++)
28
          if (!^{\sim} dfn[u]) dfs(g, u);
29
30
   };
```

10.4 Heavy Light Division

```
1
   /* Heavy_Light_Division
2
    * */
3
   template < class graph_t, int N> struct hld_t {
4
     typedef int ai_t[N];
5
      ai_t d, sz, hb, fa, cl, in, id;
6
     void link(int h) {
7
        cl[h] = 1, in[h] = h, id[h] = 0;
8
        for (int v = h; hb[v];)
9
          in[v = hb[v]] = h, id[v] = cl[h] ++;
10
     void go(graph_t &g, int u, int p = -1, int l = 0) {
11
        d[u] = 1, sz[u] = 1, hb[u] = -1, fa[u] = p;
12
13
        for (int e = g(u); e = g[e].to) {
14
          int v = g[e].v;
15
          if (v == p) continue;
16
          go(g, v, u, l+1);
17
          sz[u] += sz[v];
          if (!^{\tilde{}} hb[u] || sz[hb[u]] < sz[v]) hb[u] = v;
18
19
20
        for (int e = g(u); e; e = g[e].to)
21
          if (g[e].v != p \&\& g[e].v != hb[u]) link(g[e].v);
22
        if (!^{\tilde{p}}) link(u);
23
24
     void make(int *w, int n) {
25
26
     int ask(int u, int v) {
27
        int result;
28
        for ( ; in [u]^in [v]; u = fa[in [u]]) {
29
          if (d[in[u]] < d[in[v]]) swap(u, v);
```

```
30 | }
31 | if (id[u] > id[v]) swap(u, v);
32 | return result;
33 | }
34 | };
```

10.5 Biconnected Component

```
1
   /* Biconnected_Component
2
    * */
3
   struct bcc_t {
4
      int time, cc, cut[N], dfn[N], low[N];
      vector < int > in, st;
5
6
      void dfs (graph_t &g, int u, int p = -1) {
7
        int branch = 0;
8
        dfn[u] = low[u] = time++;
9
        for (int e = g(u); e; e = g[e].to) {
10
          int v = g[e].v;
          if (e = p \mid | dfn[v] >= dfn[u]) continue;
11
12
          st.push_back(e);
          if (\tilde{dfn}[v]) low [u] = min(low [u], dfn [v]);
13
14
          else {
15
             branch++;
             dfs\,(\,g\,,\ v\,,\ e\,\hat{\,}\,1\,)\,;
16
             low[u] = min(low[u], low[v]);
17
             if (dfn[u] > low[v]) continue;
18
             for (cut[u] = 1; ; ) {
19
20
               int t = st.back();
21
               st.pop_back();
22
               in[t] = in[t^1] = cc;
23
               if (t == e) break;
24
25
             cc++;
          }
26
27
28
        if (!^p \&\& cut[u] \&\& branch < 2) cut[u] = 0;
29
30
      void operator () (graph_t &g) {
31
        in.resize(g.e.size());
32
        for (int u = 0; u < g.size(); u++)
33
          dfn[u] = low[u] = -1, cut[u] = 0;
        st.clear();
34
        for (int u = time = cc = 0; u < g.size(); u++)
35
36
          if (!^{\sim} dfn[u]) dfs(g, u);
37
38
   };
```

10.6 Static Lowest Ancestor

```
1
  /* Static_Lowest_Ancestor
2
   * */
3
  struct slca_t {
     graph_t q;
4
5
     vector < int > r, f, c, d;
     void init(int n) {
6
7
       q.init(n), f.resize(n), c.resize(n);
       for (int i = 0; i < n; i++)
9
         f[i] = i, c[i] = 0;
```

```
10
     void add(int u, int v) {
11
12
       r.push_back(-1);
13
       q.badd(u, v);
14
15
     int find(int x) {
       if (x != f[x]) f[x] = find(f[x]);
16
       return f[x];
17
18
19
     void go(graph_t \& g, int u, int p = -1) {
20
       for (int e = g(u); e = g[e].to) {
21
         if (e == p) continue;
22
         go(g, g[e].v, e^1);
23
         f[find(g[e].v)] = u;
24
       c[u] = 1;
25
26
       for (int e = q(u); e = q[e].to) {
27
         if (!c[q[e].v]) continue;
28
         r[e>>1] = find(q[e].v);
29
30
     }
31
   };
```