Dwylkz's Algorithm Library

GuangZhou University
Team Kimady
Dwylkz(Wūdōng Yáng)

November 8, 2013

Contents

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
set langmenu=en_US.UTF-8
source $VIMRUNTIME/delmenu.vim
source $VIMRUNTIME/menu.vim
language messages en_US.UTF-8
syntax on
filetype on
filetype plugin indent on
set smartindent
set autochdir
set autoindent
set smartindent
set backspace=2
set columns=120
set foldmethod=syntax
set nohlsearch
set incsearch
set lines=40
set nocompatible
set noswapfile
set number
set shiftwidth=2
set tabstop=2
set expandtab

func Compile()
        exec ”w”
        exec ”make”
endfunc
func Debug()
        exec ”w”
        exec ”make␣debug”
endfunc
func Run()
        exec ”w”
        exec ”make␣run”
endfunc

map <F4> :tabp<CR>
map <F5> :tabn<CR>
map <C-a> ggVG
map <C-p> ”+p
map␣<F8>␣<ESC>:clist <CR>
map␣<C-F8>␣<ESC>:call␣Debug()<CR>
map␣<F9>␣<ESC>:call␣Compile()<CR>
map␣<C-F9>␣<ESC>:call␣Run()<CR>
map␣<silent ><F6>␣:s#^#//␣#g<CR>
map␣<silent ><F7>␣:s#^//␣##g<CR>
vmap␣<C-y>␣”+y
```

3 main

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {
#if 1
  freopen(”input.in”, ”r”, stdin);
#endif
  return 0;
}
```

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

```
/* High_Precision_Integer
 * */
struct int_t {
  string d;
  int_t(string _d = ”0”): d(_d) {}
  int_t(int _d) {
    static char buff[20];
    sprintf(buff, ”%d”, _d);
    d = buff;
  }
  static void trans(string &s) {
    for (int i = 0; i < s.length(); i++) s[i] += ’0’;
  }
  friend int_t &operator + (const int_t &lhs, const int_t &rhs) {
    static int_t result;
    const string &a = lhs.d, &b = rhs.d;
    string &c = result.d;
    int maxlen = max(a.length(), b.length())+1;
    c.resize(maxlen);
    fill(c.begin(), c.end(), 0);
    for (int i = 0; i < maxlen-1; i++) {
      int x = a.length() <= i? 0: a[a.length()-1-i]-’0’,
          y = b.length() <= i? 0: b[b.length()-1-i]-’0’;
      c[i] += x+y;
      c[i+1] += c[i]/10;
      c[i] %= 10;
    }
    if (!c[maxlen-1]) c.resize(maxlen-1);
    reverse(c.begin(), c.end());
    trans(c);
    return result;
  }
  friend int_t &operator += (const int_t &lhs, const int_t &rhs) {
    return lhs+rhs;
  }
  friend int_t &operator * (const int_t &lhs, const int_t &rhs) {
    static int_t result;
    const string &a = lhs.d, &b = rhs.d;
    string &c = result.d;
    int maxlen = a.length()+b.length();
    c.resize(maxlen);
    fill(c.begin(), c.end(), 0);
    for (int i = 0; i < a.length(); i++) {
      int x = a[a.length()-1-i]-'0';
      for (int j = 0; j < b.length(); j++) {
        int y = b[b.length()-1-j]-'0';
        c[i+j] += x*y;
        c[i+j+1] += c[i+j]/10;
        c[i+j] %= 10;
      }
    }
    for ( ; maxlen > 1 && !c[maxlen-1]; maxlen--) {}
    c.resize(maxlen);
    reverse(c.begin(), c.end());
    trans(c);
    return result;
  }
  friend int_t &operator *= (const int_t &lhs, const int_t &rhs) {
    return lhs*rhs;
  }
  const char *show() {
    return d.data();
  }
};
```

5.2 Minimum Prime Factor Sieve

```
/* Minimum_Prime_Factor_Sieve
 * N  : upper bound
 * p[]: primes
 * n  : primes number
 * e[]: eular funtion
 * d[]: divisors number
 * f[]: minimum prime factor
 * c[]: minimum prime factor's power
 * m[]: mobius function
 * */
template<int N> struct sieve_t {
  int b[N], p[N], n, e[N], d[N], f[N], c[N], m[N];
  sieve_t() {
    memset(this, 0, sizeof(sieve_t));
    d[1] = m[1] = 1;
    for (int i = 2; i < N; i++) {
      if (!b[i]) {
        e[i] = i-1;
        c[i] = 1;
        d[i] = 2;
        f[i] = i;
        m[i] = -1;
        p[n++] = i;
      }
      for (int j = 0; j < n && 1ll*i*p[j] < N; j++) {
        int k = i*p[j];
        b[k] = 1;
        f[k] = p[j];
        if (i%p[j]) {
          e[k] = e[i]*(p[j]-1);
          c[k] = 1;
          d[k] = d[i]*(c[k]+1);
          m[k] = m[i]*m[p[j]];
        } else {
          e[k] = e[i]*p[j];
          c[k] = c[i]+1;
          d[k] = d[i]/(c[i]+1)*(c[k]+1);
          m[k] = 0;
          break;
        }
      }
    }
  }
};
```

5.3 Contor coding.

```
/* Contor_coding.
 * Notice that x in [1, l!] in array->integer mapping
 * while x in [0, l!) in integer->array mapping. */
template<int N> struct contor_t {
  int f[N];
  contor_t() {
    for (int i = f[0]= 1; i < N; i++)
      f[i] = f[i - 1]*i;
  }
  void operator () (int l, int x, int *t) {
    int id = 0, h[100] = {0};
    x--;
    for (int i = l-1; 0 <= i; i--) {
      int rm = x/f[i], rank = 0;
      for (int j = 1; j <= l; j++) {
        rank += !h[j];
        if (rank == rm+1) {
          t[id++] = j;
          h[j] = 1;
          break;
        }
      }
      x %= f[i];
    }
  }
  int operator () (int l, int *t) {
    int rv = 0;
    for (int i = 0; i < l; i++) {
      int cnt = 0;
      for (int j = i+1; j < l; j++)
        if (t[j] < t[i]) cnt++;
      rv += cnt*f[l-i-1];
    }
    return rv;
  }
};
```

5.4 Chinese Remind Theory

```
/* Chinese_Remind_Theory
 * */
template<int N> struct crt_t {
  vector<int> a, b;
  int gcd(int a, int b, int &x, int &y) {
    int d, tx, ty;
    if (b == 0) {
      x = 1;
      y = 0;
      return a;
    }
    d = gcd(b, a%b, tx, ty);
    x = ty;
    y = tx-(a/b)*ty;
    return d;
  }
  int mle(int a, int b, int n) {
    int d, x, y;
    d = gcd(a, n, x, y);
    if (b%d == 0) {
      x = 1ll*x*b/d%n;
      return x;
    }
    return 0;
  }
  int init() {
    a.clear();
    b.clear();
  }
  int operator () () {
    int x = 0, n = 1, i, bi;
    for (i = 0; i < b.size(); i++) n *= b[i];
    for (i = 0; i < a.size(); i++) {
      bi = mle(n/b[i], 1, b[i]);
      x = (x+1ll*a[i]*bi*(n/b[i]))%n;
    }
    return x;
  }
};
```

5.5 Base 2 Fast Fourier Transfrom

```
/* Base_2_Fast_Fourier_Transfrom
 * (): transfrom
 * []: inversion */
struct b2_fft_t {
  typedef complex<double> cd_t;
  typedef vector<cd_t> vcd_t;
  vcd_t c;
  void brc(vcd_t &x) {
    int l;
    for (l = 1; l < x.size(); l <<= 1) {}
    c.resize(l);
    for (int i = 0; i < c.size(); i++) {
      int to = 0;
      for (int o = l>>1, t = i; o; o >>= 1, t >>= 1)
        if (t&1) to += o;
      c[to] = i < x.size()? x[i]: cd_t(0., 0.);
    }
  }
  void fft(int on) {
    double dpi = acos(-1.)*on;
    for (int m = 1; m < c.size(); m <<= 1) {
      cd_t wn(cos(dpi/m), sin(dpi/m));
      for (int j = 0; j < c.size(); j += m<<1) {
        cd_t w = 1.;
        for (int k = j; k < j+m; k++, w *= wn) {
          cd_t u = c[k], t = w*c[k+m];
          c[k] = u+t, c[k+m] = u-t;
        }
      }
    }
    if (~on) return ;
    for (int i = 0; i < c.size(); i++)
      c[i] /= c.size()*1.;
  }
  void operator () (vcd_t &x) {
    brc(x), fft(1), x = c;
  }
  void operator [] (vcd_t &x) {
    brc(x), fft(-1), x = c;
  }
};
```

5.6 Triangle Diagonal Matrix Algorithm

```
/* Triangle_Diagonal_Matrix_Algorithm
 * */
template<class T> struct tdma_t {
  void operator () (int n, T *a, T *b, T *c, T *d, T *x) {
    for (int i = 0; i < n; i++) {
      T tp = a[i]/b[i-1];
      b[i] -= tp*c[i-1];
      d[i] -= tp*d[i-1];
    }
    x[n-1] = d[n-1]/b[n-1];
    for (int i = n-2; ~i; i--) x[i] = (d[i]-c[i]*x[i+1])/b[i];
  }
};
```

6 pattern

6.1 KMP

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

```
      for ( ; ~j && p[i] != p[j]; ) j = f[j];
  }
  void operator () (T *p, int pl, int *f) {
    int i = 0, j = f[0] = -1;
    for ( ; i < pl; i++, j++, f[i] = p[i] == p[j]? f[j]: j)
      for ( ; ~j && p[i] != p[j]; ) j = f[j];
  }
  int operator () (T *s, int sl, T *p, int pl, int *f) {
    int i = 0, j = 0;
    for ( ; i < sl && j < pl; i++, j++)
      for ( ; ~j && s[i] != p[j]; ) j = f[j];
    return j;
  }
};
```

6.2 Extend KMP

```
/* Extend_KMP
 * */
template<class T> struct exkmp_t {
  void operator () (T *p, int pl, int *g) {
    g[g[1] = 0] = pl;
    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]++)
        if (i+g[i] >= pl || p[i+g[i]] != p[g[i]]) break;
  }
  void operator () (T *s, int sl, int *f, T *p, int pl, int *g) {
    for (int i = f[0] = 0, k = 0; i < sl; (k+f[k] < i+f[i]? k = i: 0), i++)
      for (f[i] = min(g[i-k], max(k+f[k]-i, 0)); ; f[i]++)
        if (i+f[i] >= sl || f[i] >= pl || s[i+f[i]] != p[f[i]]) break;
  }
};
```

6.3 Manacher

```
/* Manacher
 * */
template<class T> struct mana_t {
  void operator () (T *s, int &n, int *p) {
    for (int i = n<<1; i >= 0; i--) s[i] = i&1? s[i>>1]: -1;
    p[s[n = n<<1|1] = 0] = 1;
    for (int i = p[1] = 2, k = 1; i < n; i++) {
      p[i] = min(p[2*k-i], max(k+p[k]-i, 1));
      for (; p[i] <= i && i+p[i] < n && s[i-p[i]] == s[i+p[i]]; ) p[i]++;
      if (k+p[k] < i+p[i]) k = i;
    }
  }
};
```

6.4 Minimum Notation

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

```
      if (s[(i+k)%n] == s[(j+k)%n]) k++;
      else if (C()(s[(i+k)%n], s[(j+k)%n])) j += k+1, k = 0;
      else i += k+1, j = i+1, k = 0;
    return i;
  }
};
```

6.5 AC automaton

```
/* AC_automaton
 * */
template<class T, int n, int m> struct aca_t {
  struct node {
    node *s[m], *p;
    int ac;
  } s[n], *top, *rt, *q[n];
  void init() {
    memset(top = s, 0, sizeof(s));
    rt = top++;
  }
  void put(T *k, int l, int ac) {
    node *x = rt;
    for (int i = 0; i < l; i++) {
      if (!x->s[k[i]]) x->s[k[i]] = top++;
      x = x->s[k[i]];
    }
    x->ac = ac;
  }
  void link() {
    int l = 0;
    for (int i = 0; i < m; i++)
      if (rt->s[i]) (q[l++] = rt->s[i])->p = rt;
      else rt->s[i] = rt;
    for (int h = 0; h < l; h++)
      for (int i = 0; i < m; i++)
        if (q[h]->s[i]) {
          (q[l++] = q[h]->s[i])->p = q[h]->p->s[i];
          q[h]->s[i]->ac |= q[h]->s[i]->p->ac;
        } else q[h]->s[i] = q[h]->p->s[i];
  }
  void tom(int mt[][n]) {
    for (node *x = s; x < top; x++)
      for (int i = 0; i < m; i++)
        if (!x->s[i]->ac) mt[x-s][x->s[i]-s] = 1;
  }
};
```

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, l) (s[a] == s[b] && s[a+l] == s[b+l])
#define da_R(x, y, z) da_F(i, 0, m) ws[i] = 0; da_F(i, 0, n) ws[x]++;\
    da_F(i, 1, m) ws[i] += ws[i-1]; da_F(i, 0, n) sa[--ws[y]] = z;
    int *x = wa, *y = wb;
    da_R(x[i] = s[i], x[n-i-1], n-i-1);
    for(int j = 1,  p = 1; p < n; j *= 2, m = p) {
      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;
      da_F(i, 0, n) wv[i] = x[y[i]];
      da_R(wv[i], wv[n-i-1], y[n-i-1]);
      da_F(i, (swap(x, y), x[sa[0]] = 0, p = 1), n)
        x[sa[i]] = da_C(y, sa[i-1], sa[i], j)? p-1: p++;
    }
  }
  int dc3_c12(int k, int *r, int a, int b, int *wv) {
    if (k != 2) return r[a]<r[b] || r[a]==r[b] && wv[a+1]<wv[b+1];
    return r[a]<r[b] || r[a]==r[b] && dc3_c12(1, r, a+1, b+1, wv);
  }
  void dc3(int *s, int n, int *sa, int m) {
#define dc3_H(x) ((x)/3+((x)%3 == 1? 0: tb))
#define dc3_G(x) ((x) < tb? (x)*3+1: ((x)-tb)*3+2)
#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++)
#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]]++;\
    dc3_F(i, 1, m) ws[i] += ws[i-1];\
    dc3_F(i, 0, n) (b)[--ws[wv[n-i-1]]] = a[n-i-1];
    int i, j, *rn = s+n, *san = sa+n, ta = 0, tb = (n+1)/3, tbc = 0, p;
    dc3_F(i, s[n] = s[n+1] = 0, n) if(i%3) wa[tbc++] = i;
    dc3_sort(s+2, wa, wb, tbc, m);
    dc3_sort(s+1, wb, wa, tbc, m);
    dc3_sort(s, wa, wb, tbc, m);
    dc3_F(i, (rn[dc3_H(wb[0])] = 0, p = 1), tbc)
      rn[dc3_H(wb[i])] = dc3_c0(s, wb[i-1], wb[i])? p-1: p++;
    if(p < tbc) dc3(rn, tbc, san, p);
    else dc3_F(i, 0, tbc) san[rn[i]] = i;
    dc3_F(i, 0, tbc) if(san[i] < tb) wb[ta++] = san[i]*3;
    if(n%3 == 1) wb[ta++] = n-1;
    dc3_sort(s, wb, wa, ta, m);
    dc3_F(i, 0, tbc) wv[wb[i] = dc3_G(san[i])] = i;
    for(i = j = p = 0; i < ta && j < tbc; p++)
      sa[p] = dc3_c12(wb[j]%3, s, wa[i], wb[j], wv)? wa[i++]:wb[j++];
    for( ; i < ta; p++) sa[p] = wa[i++];
    for( ; j < tbc; p++) sa[p] = wb[j++];
  }
  void ch(int *s, int n, int *sa, int *h) {
    for (int i = 1; i < n; i++) r[sa[i]] = i;
    for (int i = 0, j, k = 0; i < n-1; h[r[i++]] = k)
      for (k? k--: 0, j = sa[r[i]-1]; s[i+k] == s[j+k]; k++);
  }
  void icats(int *b, int *l, char *s) {
    static int delim = 'z'+1;
    for (*l += strlen(s)+1; *s; s++) *b++ = *s;
    *b++ = delim++;
  }
};
```

6.7 Suffix Automaton

```
/* Suffix_Automaton
 * */
```

```
template<int N, int M> struct sam_t {
  static const int n = N*3;
  struct node {
    node *s[M], *p;
    int l;
    int range() {
      return l-(p? l-p->l: 0);
    }
  } s[n], *top, *back;
  node *make(int l) {
    memset(top, 0, sizeof(node));
    top->l = l;
    return top++;
  }
  void init() {
    top = s;
    back = make(0);
  }
  void put(int k) {
    node *x = make(back->l+1), *y = back;
    for ( ; y && !y->s[k]; y = y->p) y->s[k] = x;
    if (!y) x->p = s;
    else {
      node *w = y->s[k];
      if (w->l == y->l+1) x->p = w;
      else {
        node *z = make(0);
        *z = *w;
        z->l = y->l+1;
        x->p = w->p = z;
        for ( ; y && y->s[k] == w; y = y->p) y->s[k] = z;
      }
    }
  }
};
```

7 data

7.1 RMQ Sparse Table

```
/* RMQ_Sparse_Table
 * */
template<int N> struct rmq_t {
  int s[20][N], *k;
  void operator () (int l, int *_k) {
    k = _k;
    for (int i = 0; i < l; i++) s[0][i] = i;
    for (int i = 1; i < 20; i++)
      if ((1<<i) <= l) for (int j = 0; j < l; j++)
        if (k[s[i-1][j]] < k[s[i-1][j+(1<<(i-1))]]) s[i][j] = s[i-1][j];
        else s[i][j] = s[i-1][j+(1<<(i-1))];
  }
  int operator () (int l, int r) {
    if (l > r) swap(l ,r);
    int i = r-l+1, o = 1, j = 0;
    for (int o = 1 ; o <= i; o <<= 1) j++;
    j--, r = r-(1<<j)+1;
    return k[s[j][l]] < k[s[j][r]]? s[j][l]: s[j][r];
  }
};
```

7.2 Splay

```
/* Splay
 * */
struct splay_t {
  struct node {
    node *p, *s[2];
    int size, key;
    int sum, lsum, rsum, msum;
    int cover_tag, reverse_tag;
    int side() {
      return p->s[1] == this;
    }
    int rank() {
      return s[0]? 1+s[0]->size: 1;
    }
    node *set_cover(int _key) {
      cover_tag = 1;
      key = _key;
      sum = size*key;
      lsum = rsum = msum = max(sum, key);
      return this;
    }
    node *set_reverse() {
      reverse_tag ^= 1;
      swap(s[0], s[1]);
      swap(lsum, rsum);
      return this;
    }
    node *push() {
      for (int i = 0; i < 2; i++) {
        if (!s[i]) continue;
        if (cover_tag) s[i]->set_cover(key);
        if (reverse_tag) s[i]->set_reverse();
      }
      cover_tag = reverse_tag = 0;
      return this;
    }
    node *merge_sum(node *x, node *y) {
      if (!x || !y) return &(*this = x? *x: *y);
      sum = x->sum+y->sum;
      lsum = max(x->lsum, x->sum+y->lsum);
      rsum = max(y->rsum, x->rsum+y->sum);
      msum = max(x->msum, y->msum);
      msum = max(msum, max(lsum, rsum));
      msum = max(msum, x->rsum+y->lsum);
      return this;
    }
    node *pull() {
      size = 1;
      sum = lsum = rsum = msum = key;
      for (int i = 0; i < 2; i++) {
        if (!s[i]) continue;
        size += s[i]->size;
      }
      return merge_sum(node(*this).merge_sum(s[0], this), s[1]);
    }
    node *set(int b, node *x) {
      if (push()->s[b] = x) x->p = this;
      return pull();
    }
    node *get(int b) {
      return push()->s[b];
    }
    node *cut(int b, node *&x) {
      if (x = push()->s[b]) s[b]->p = 0;
      s[b] = 0;
      return pull();
    }
    node *spin() {
      node *y = p->push();
      int b = push()->side();
      if (p = y->p) p->s[y->side()] = this;
      if (y->s[b] = s[!b]) s[!b]->p = y;
      return (s[!b] = y)->pull()->p = this;
    }
    node *fine(node *x = 0) {
      for ( ; p != x; spin())
        if (p->p != x)
          if (side() == p->side()) p->spin();
          else spin();
      return pull();
    }
    node *pick(int k, node *y = 0) {
      node *x = this;
      for ( ; x->rank() != k; ) {
        int b = x->rank() < k;
        k -= b*x->rank();
        x = x->get(b);
      }
      return x->fine(y);
    }
  };
  node *give(node *t = 0) {
    static node *top = 0;
    static int size = 1;
    if (t) t->s[1] = top, top = t;
    else {
      if (!top) {
        top = new node[size <<=1];
        for (int i = 0; i < size-1; i++)
          top[i].s[1] = top+i+1;
        top[size-1].s[1] = 0;
      }
      t = top, top = top->s[1];
    }
    return t;
  }
  node *make(int key) {
    node t = {0, {0}, 1, key, key, key, key, key, 0, 0};
    return &(*give() = t);
  }
  void drop(node *&t) {
    if (!t) return ;
    drop(t->s[0]), drop(t->s[1]);
    give(t), t = 0;
  }
  void show(node *t) {
    if (!t) return ;
    t->push();
    show(t->s[0]);
    printf(" %d", t->key);
    show(t->s[1]);
    t->pull();
  }
};
```

7.3 Treap

```
/* Treap
 * */
struct treap_t {
  struct node {
    node *s[2];
    int size, weight;
    int key, cover_tag, sum,
        max_sum, lsum, rsum, reverse_tag;
    int rank() {
      return s[0]? s[0]->size+1: 1;
    }
    node *set_cover(int _key) {
      key = _key;
      cover_tag = 1;
      sum = size*key;
      max_sum = lsum = rsum = max(key, sum);
      return this;
    }
    node *set_reverse() {
      reverse_tag ^= 1;
      swap(s[0], s[1]);
      swap(lsum, rsum);
      return this;
    }
    node *push() {
      for (int i = 0; i < 2; i++) {
        if (!s[i]) continue;
        if (cover_tag) s[i]->set_cover(key);
        if (reverse_tag) s[i]->set_reverse();
      }
      cover_tag = reverse_tag = 0;
      return this;
    }
    node *merge_sum(node *x, node *y) {
      if (!x || !y) {
        x? *this = *x: *this = *y;
        return this;
      }
      sum = x->sum+y->sum;
      lsum = max(x->lsum, x->sum+y->lsum);
      rsum = max(y->rsum, x->rsum+y->sum);
      max_sum = max(x->max_sum, y->max_sum);
      max_sum = max(max_sum, max(lsum, rsum));
      max_sum = max(max_sum, x->rsum+y->lsum);
      return this;
    }
    node *pull() {
```

```
    size = 1;
    max_sum = sum = lsum = rsum = key;
    for (int i = 0; i < 2; i++) {
      if (!s[i]) continue;
      size += s[i]->size;
    }
    return merge_sum(node(*this).merge_sum(s[0], this), s[1]);
  }
};
node *give(node *t = 0) {
  static node *top = 0;
  static int size = 1;
  if (t) t->s[1] = top, top = t;
  else {
    if (!top) {
      top = new node[size<<=1];
      for (int i = 0; i < size-1; i++)
        top[i].s[1] = top+i+1;
      top[size-1].s[1] = 0;
    }
    t = top, top = top->s[1];
  }
  return t;
}
node *make(int key) {
  node t = {{0}, 1, rand()*rand(), key, 0, key, key, key, key, 0};
  return &(*give() = t);
}
void drop(node *&t) {
  if (!t) return ;
  drop(t->s[0]), drop(t->s[1]);
  give(t), t = 0;
}
void merge(node *x, node *y, node *&t) {
  if (!x || !y) t = x? x: y;
  else if (x->weight < y->weight)
    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();
}
void split(node *t, int k, node *&x, node *&y) {
  if (!k) x = 0, y = t;
  else if (t->size == k) x = t, y = 0;
  else if (k < t->rank())
    y = t->push(), split(t->s[0], k, x, y->s[0]), y->pull();
  else x = t->push(), split(t->s[1], k-t->rank(), x->s[1], y), x->pull();
}
void slice(node *&t, int l = -1, int r = -1) {
  static node *a, *b;
  if (~l) split(t, l-1, a, b), split(b, r-l+1, t, b);
  else merge(t, b, b), merge(a, b, t);
}
void show(node *t) {
  if (!t) return ;
  t->push();
  show(t->s[0]);
  printf("␣%2d", t->key);
  show(t->s[1]);
  t->pull();
}
int ask_sum(node *t) {
    return t? t->sum: 0;
  }
  int ask_max_sum(node *t) {
    return t? t->max_sum: 0;
  }
};
```

7.4 Link-Cut Tree

```
/* Link-Cut_Tree
 * */
template<int N> struct lct_t {
  struct node {
    node *s[2], *p;
    int sz, rev, w, mx, at;
    node *sets(int b, node *x) {
      if (s[b] = x) x->p = this;
      return this;
    }
    bool root() {
      return !p || !(p->s[0] == this || p->s[1] == this);
    }
    bool which() {
      return p->s[1] == this;
    }
    node *set() {
      swap(s[0], s[1]);
      rev ^= 1;
      return this;
    }
    node *cover(int d) {
      w += d;
      mx += d;
      at += d;
      return this;
    }
    node *push() {
      if (at) {
        for (int i = 0; i < 2; i++)
          if (s[i]) s[i]->cover(at);
        at = 0;
      }
      if (rev) {
        for (int i = 0; i < 2; i++)
          if (s[i]) s[i]->set();
        rev = 0;
      }
      return this;
    }
    node *pull() {
      sz = 1;
      mx = w;
      for (int i = 0; i < 2; i++)
        if (s[i]) {
          sz += s[i]->sz;
          mx = max(mx, s[i]->mx);
        }
      return this;
    }
    node *spin() {
      node *y = p->push();
      int b = push()->which();
      y->sets(b, s[!b])->pull();
      if (y->root()) p = y->p;
      else y->p->sets(y->which(), this);
      return sets(!b, y);
    }
    node *splay() {
      for ( ; !root(); )
        if (p->root()) spin();
        else {
          if (which() == p->which()) p->spin();
          else spin();
          spin();
        }
      return pull();
    }
    node *end(int b) {
      node *x = this;
      for ( ; x->push()->s[b]; x = x->s[b]) ;
      return x;
    }
  } lct[N], *top;
  void init() {
    top = lct;
  }
  node *make(int w) {
    *top = (node){{0, 0}, 0, 1, 0, w, w};
    return top++;
  }
  node *access(node *x, int o = 0, int d = 0) {
    static node rv;
    for (node *y = x, *z = 0; y; z = y, y = y->p) {
      y->splay()->push();
      if (!y->p) {
        if (o == 1) {
          y->w += d;
          if (y->s[1]) y->s[1]->cover(d);
          if (z) z->cover(d);
        } else if (o == 2) {
          int mx = y->w;
          if (y->s[1]) mx = max(mx, y->s[1]->mx);
          if (z) mx = max(mx, z->mx);
          rv.mx = mx;
          return &rv;
        }
      }
      y->sets(1, z)->pull();
    }
    return x->splay();
  }
  node *join(node *x, node *y) {
    return x->p = y;
  }
  node *cut(node *x) {
    if (access(x)->s[0]) x->s[0]->p = 0;
    x->s[0] = 0;
    return x;
  }
```

```
  node *find(node *x) {
    return access(x)->end(0);
  }
  node *rooting(node *x) {
    return access(x)->set();
  }
  node *cover(node *x, node *y, int w) {
    access(x);
    access(y, 1, w);
    return x;
  }
  int ask(node *x, node *y) {
    access(x);
    return access(y, 2)->mx;
  }
};
```

7.5 Functional Segment

```
/* Functional_Segment
 * */
template<int N> struct fs_t {
  struct node {
    int l, r, sm;
    node *ls, *rs;
    int m() {
      return l+r>>1;
    }
  } s[N*20], *top;
  void init() {
    top = s;
  }
  node *phi(int l, int r) {
    node *x = top++, t = {l, r, 0};
    *x = t;
    if (l < r) {
      x->ls = phi(l, x->m());
      x->rs = phi(x->m()+1, r);
    }
    return x;
  }
  node *put(int k, node *y) {
    node *x = top++;
    *x = *y;
    x->sm++;
    if (x->l < y->r) {
      if (k <= x->m()) x->ls = put(k, y->ls);
      else x->rs = put(k, y->rs);
    }
    return x;
  }
  int ask(int l, int r, node *x, node *y) {
    int rv = 0;
    if (l <= x->l && x->r <= r) rv = x->sm-y->sm;
    else {
      if (l <= x->m()) rv += ask(l, r, x->ls, y->ls);
      if (x->m() < r) rv += ask(l, r, x->rs, y->rs);
    }
    return rv;
  }
};
```

7.6 Functional Trie

```
/* Functional_Trie
 * */
template<int N, int D> struct ftrie_t {
  struct node {
    node *s[2];
    int c[2];
  } s[D*N+D], *top, *phi;
  void init() {
    top = s;
    phi = top++;
    phi->c[0] = phi->c[1] = 0;
    phi->s[0] = phi->s[1] = phi;
  }
  node *put(int k, node *y, int d = D) {
    if (!d) return 0;
    node *x = top++;
    *x = *y;
    int i = k>>(d-1)&1;
    x->c[i]++;
    x->s[i] = put(k, y->s[i], d-1);
    return x;
  }
  int ask(int k, node *x, node *y, int d = D) {
    if (!d) return 0;
    int i = k>>(d-1)&1;
    if (x->c[!i]-y->c[!i])
      return (1<<d-1)+ask(k, x->s[!i], y->s[!i], d-1);
    return ask(k, x->s[i], y->s[i], d-1);
  }
};
```

7.7 Lefist Tree

```
/* Lefist_Tree
*/
template<int N> struct lefist_t {
  struct node {
    node *l, *r;
    int k, d;
  } s[N], *top;
  void init() {
    top = s;
  }
  node *make(int k) {
    node *x = top++, t = {0, 0, k, 0};
    *x = t;
    return x;
  }
  node *merge(node *x, node *y) {
    if (!x) return y;
    if (!y) return x;
    if (x->k < y->k) swap(x, y);
    x->r = merge(x->r, y);
    if (!x->l || x->r && x->l->d < x->r->d) swap(x->l, x->r);
    if (x->r) x->d = x->r->d+1;
    return x;
  }
  node *drop(node *x) {
    return merge(x->l, x->r);
  }
};
```

8 geometry

8.1 Float Compare Functions

```
/* Float_Compare_Functions
 * */
struct fc_t {
  double eps;
  fc_t() {
    eps = 1e-8;
  }
  bool e(double lhs, double rhs) {
    return abs(lhs-rhs) < eps;
  }
  bool l(double lhs, double rhs) {
    return lhs+eps < rhs;
  }
  bool g(double lhs, double rhs) {
    return lhs-eps > rhs;
  }
} fc;
```

8.2 2D point

```
/* 2D_point
 * */
struct pt_t {
  double x, y;
  pt_t(double _x = 0, double _y = 0) {
    x = _x, y = _y;
  }
  double operator [] (int b) {
    return b? b < 2? abs(x)+abs(y): x*x+y*y: sqrt(x*x+y*y);
  }
  friend pt_t operator + (const pt_t &lhs, const pt_t &rhs) {
    return pt_t(lhs.x+rhs.x, lhs.y+rhs.y);
  }
  friend pt_t operator - (const pt_t &lhs, const pt_t &rhs) {
    return pt_t(lhs.x-rhs.x, lhs.y-rhs.y);
  }
  friend double operator * (const pt_t &lhs, const pt_t &rhs) {
    return lhs.x*rhs.x+lhs.y*rhs.y;
  }
  friend double operator % (const pt_t &lhs, const pt_t &rhs) {
    return lhs.x*rhs.y-lhs.y*rhs.x;
  }
  pt_t &input() {
    scanf("%lf%lf", &x, &y);
    return *this;
  }
};
```

8.3 Angle Sort

```
/* Angle_Sort
 * */
struct asort_t {
  bool cmpl(pt_t lhs, pt_t rhs) {
    return fc.l(lhs.y, rhs.y) || (fc.e(lhs.y, rhs.y) && fc.l(lhs.x, rhs.x));
  }
  static pt_t o;
  static bool cmp(pt_t lhs, pt_t rhs) {
    double c = (lhs-o)%(rhs-o);
    if (!fc.e(c, 0.0)) return fc.g(c, 0.0);
    return fc.g((lhs-o)[1], (rhs-o)[1]);
  }
  void operator () (vector<pt_t> &p) {
    int mn = 0;
    for (int i = 0; i < p.size(); i++)
      if (cmpl(p[i], p[mn])) mn = i;
    swap(p[0], p[mn]);
    o = p[0];
    sort(p.begin()+1, p.end(), cmp);
  }
} asort;
pt_t asort_t::o;
```

8.4 Graham Scan

```
/* Graham_Scan
 * */
struct graham_t {
  vector<pt_t> p;
  double l;
  graham_t(vector<pt_t> &ps) {
    asort(p = ps);
    vector<pt_t> s(p.begin(), p.begin()+2);
    ps.clear();
    for (int i = 2; i < p.size(); i++) {
      for ( ; fc.g((s[s.size()-2]-s.back())%(p[i]-s.back()), 0.0); )
        ps.push_back(s.back()), s.pop_back();
      s.push_back(p[i]);
    }
    p = s;
    for (int i = l = 0; i < p.size(); i++)
      l += (p[(i+1)%p.size()]-p[i])[0];
  }
};
```

9 graph

9.1 Graph

```
/* Graph
 * */
```

```
template<int N> struct graph_t {
  struct edge_t {
    int v, to;
  };
  vector<edge_t> E;
  int L[N];
  void init() {
    E.clear();
    memset(L, -1, sizeof(L));
  }
  void add(int u, int v) {
    edge_t t = {v, L[u]};
    L[u] = E.size();
    E.push_back(t);
  }
};
```

9.2 Shortest Path Algorithm

```
/* Shortest_Path_Algorithm
 * */
template<class edge_t, int N> struct spfa_t {
  int d[N], b[N], c[N], s[N], mx[N];
  int operator () (vector<edge_t> &E, int *L, int n, int u) {
    memset(d, 0x7f, sizeof(d));
    memset(b, 0, sizeof(b));
    memset(c, 0, sizeof(c));
    d[s[s[0] = 1] = u] = 0;
    b[u] = c[u] = 1;
    for ( ; s[0]; ) {
      b[u = s[s[0]--]] = 0;
      for (int e = L[u]; ~e; e = E[e].to) {
        int v = E[e].v, w = E[e].w;
        if (d[v]-w > d[u]) {
          d[v] = d[u]+w;
          if (!b[v]) {
            if ((c[v] += b[v] = 1) > n) return 0;
            s[++s[0]] = v;
          }
        }
      }
    }
    return 1;
  }
  struct node {
    int u, w;
    node (int _u = 0, int _w = 0): u(_u), w(_w) {}
    friend bool operator < (const node &lhs, const node &rhs) {
      return lhs.w > rhs.w;
    }
  };
  void operator () (vector<edge_t> &E, int *L, int u) {
    memset(d, 0x7f, sizeof(d));
    memset(b, 0, sizeof(b));
    priority_queue<node> q;
    for (q.push(node(u, d[u] = 0)); q.size(); ) {
      u = q.top().u, q.pop();
      if (b[u]++) continue;
      for (int e = L[u]; ~e; e = E[e].to) {
        int v = E[e].v, w = E[e].w;
        if (b[u] && d[v]-w > d[u])
          q.push(node(v, d[v] = d[u]+w));
      }
    }
  }
};
```

9.3 Bipartite Graph match

```
/* Bipartite_Graph_match
 * */
template<class edge_t, int N> struct bgm_t {
  int vis[N], pre[N], lma[N], rma[N];
  bool bfs(vector<edge_t> &E, int *L, int u) {
    vector<int> q(1, u);
    memset(vis, 0, sizeof(vis));
    memset(pre, -1, sizeof(pre));
    for (int h = 0; h < q.size(); h++) {
      u = q[h];
      for (int e = L[u]; ~e; e = E[e].to) {
        int v = E[e].v;
        if (!vis[v]) {
          vis[v] = 1;
          if (rma[v] == -1) {
            for ( ; ~u; ) {
              rma[v] = u;
              swap(v, lma[u]);
              u = pre[u];
            }
            return 1;
          } else {
            pre[rma[v]] = u;
            q.push_back(rma[v]);
          }
        }
      }
    }
    return 0;
  }
  int operator () (vector<edge_t> &E, int *L, int V) {
    int mmat = 0;
    memset(lma, -1, sizeof(lma));
    memset(rma, -1, sizeof(rma));
    for (int u = 0; u < V; u++)
      mmat += bfs(E, L, u);
    return mmat;
  }
};
```

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];
```

```
int match[N],pre[N],base[N];
int findancestor(int u,int v){
  bool inpath[N]={false};
  while(1){
    u=base[u];
    inpath[u]=true;
    if(match[u]==-1)break;
    u=pre[match[u]];
  }
  while(1){
    v=base[v];
    if(inpath[v])return v;
    v=pre[match[v]];
  }
}
void reset(int u,int anc){
  while(u!=anc){
    int v=match[u];
    inblossom[base[u]]=1;
    inblossom[base[v]]=1;
    v=pre[v];
    if(base[v]!=anc)pre[v]=match[u];
    u=v;
  }
}
void contract(int u,int v,int n){
  int anc=findancestor(u,v);
  //SET(inblossom,0);
  memset(inblossom,0,sizeof(inblossom));
  reset(u,anc);reset(v,anc);
  if(base[u]!=anc)pre[u]=v;
  if(base[v]!=anc)pre[v]=u;
  for(int i=1;i<=n;i++)
    if(inblossom[base[i]]){
      base[i]=anc;
      if(!inque[i]){
        Q.push_back(i);
        inque[i]=1;
      }
    }
}
bool dfs(int S,int n){
  for(int i=0;i<=n;i++)pre[i]=-1,inque[i]=0,base[i]=i;
  Q.clear();Q.push_back(S);inque[S]=1;
  while(!Q.empty()){
    int u=Q.front();Q.pop_front();
    for(int v=1;v<=n;v++){
      if(g[u][v]&&base[v]!=base[u]&&match[u]!=v){
        if(v==S||(match[v]!=-1&&pre[match[v]]!=-1))contract(u,v,n);
        else if(pre[v]==-1){
          pre[v]=u;
          if(match[v]!=-1)Q.push_back(match[v]),inque[match[v]]=1;
          else{
            u=v;
            while(u!=-1){
              v=pre[u];
              int w=match[v];
              match[u]=v;
              match[v]=u;
              u=w;
              }
              return true;
            }
          }
        }
      }
    }
    return false;
  }
  void init(int n) {
    this->n = n;memset(match,-1,sizeof(match));
    memset(g,0,sizeof(g));
  }
  void addEdge(int a, int b) {
    ++a;
    ++b;
    g[a][b] = g[b][a] = 1;
  }
  int gao() {
    int ans = 0;
    for (int i = 1; i <= n; ++i) {
      if (match[i] == -1 && dfs(i, n)) {
        ++ans;
      }
    }
    return ans;
  }
};
```

9.5 Dancing Link

```
/* Dancing_Link
 * */
template<int N, int M> struct dancing {
#define dfor(c, a, b) for (int c = a[b]; c != b; c = a[c])
  static const int row_size = N, column_size = M,
               total_size = row_size * column_size;
  typedef int row[row_size],
          column[column_size],
          total[total_size];
  total l, r, u, d, in_column;
  column s;
  int index, current_row, row_head;
  void init(int n)
  {
    index = ++n;
    for (int i = 0; i < n; i++) {
      l[i] = (i - 1 + n) % n;
      r[i] = (i + 1) % n;
      u[i] = d[i] = i;
    }
    current_row = 0;
    memset(s, 0, sizeof(s));
  }
  void push(int i, int j)
  {
    i++; j++;
    if (current_row < i) {
      row_head = l[index] = r[index] = index;
      current_row = i;
    }
    l[index] = l[row_head]; r[index] = row_head;
    r[l[row_head]] = index; l[row_head] = index;
    u[index] = u[j]; d[index] = j;
    d[u[j]] = index; u[j] = index;
    s[j]++;
    in_column[index++] = j;
  }
  void exactly_remove(int c)
  {
    l[r[c]] = l[c];
    r[l[c]] = r[c];
    dfor(i, d, c) {
      dfor (j, r, i) {
        u[d[j]] = u[j];
        d[u[j]] = d[j];
        s[in_column[j]]--;
      }
    }
  }
  void exactly_resume(int c)
  {
    dfor(i, u, c) {
      dfor(j, l, i) {
        s[in_column[j]]++;
        d[u[j]] = u[d[j]] = j;
      }
    }
    r[l[c]] = l[r[c]] = c;
  }
  bool exactly_dance(int step = 0)
  {
    if (!r[0]) {
      return 1;
    }
    int x = r[0];
    dfor(i, r, 0) {
      if (s[i] < s[x]) {
        x = i;
      }
    }
    exactly_remove(x);
    dfor(i, d, x) {
      dfor(j, r, i) {
        exactly_remove(in_column[j]);
      }
      if (exactly_dance(step + 1)) {
        return 1;
      }
      dfor(j, l, i) {
        exactly_resume(in_column[j]);
      }
    }
    exactly_resume(x);
    return 0;
  }
  int limit;
  void remove(int c)
  {
```

```
  dfor(i, d, c) {
    l[r[i]] = l[i];
    r[l[i]] = r[i];
  }
}
void resume(int c)
{
  dfor(i, u, c) {
    r[l[i]] = l[r[i]] = i;
  }
}
bool dance(int step = 0)
{
  if (limit <= step + heuristic()) {
    return 0;
  }
  if (!r[0]) {
    limit = min(limit, step);
    return 1;
  }
  int x = r[0];
  dfor(i, r, 0) {
    if (s[i] < s[x]) {
      x = i;
    }
  }
  dfor(i, d, x) {
    remove(i);
    dfor(j, r, i) {
      remove(j);
    }
    if (dance(step + 1)) {
      return 1;
    }
    dfor(j, l, i) {
      resume(j);
    }
    resume(i);
  }
  return 0;
}
int heuristic()
{
  int rv = 0;
  column visit = {0};
  dfor(c, r, 0) {
    if (!visit[c]) {
      rv++;
      visit[c] = 1;
      dfor(i, d, c) {
        dfor(j, r, i) {
          visit[in_column[j]] = 1;
        }
      }
    }
  }
  return rv;
}
int dfs()
{
    for (limit = heuristic(); !dance(); limit++) {}
    return limit;
  }
#undef dfor
};
```

9.6 Directed Minimum Spanning Tree

```
/* Directed_Minimum_Spanning_Tree
 * */
template<int N> struct dmst_t {
  struct edge_t {
    int u, v, w;
  };
  vector<edge_t> E;
  static const int inf = 0x7f7f7f7f;
  int n, ine[N], pre[N], id[N], vis[N];
  void init(int _n) {
    n = _n;
    E.clear();
  }
  void add(int u, int v, int w) {
    edge_t t = {u, v, w};
    E.push_back(t);
  }
  int operator () (int rt) {
    int i, u, v, w, tn = n+1, index, rv = 0;
    for ( ; ; ) {
      fill(ine, ine+tn, inf);
      for (i = 0; i < E.size(); i++) {
        u = E[i].u; v = E[i].v; w = E[i].w;
        if (u != v && w < ine[v]) {
          pre[v] = u;
          ine[v] = w;
        }
      }
      for (u = 0; u < tn; u++) {
        if (u == rt) continue;
        if (ine[u] == inf)
          return -1;
      }
      index = 0;
      fill(id, id + tn, -1);
      fill(vis, vis + tn, -1);
      ine[rt] = 0;
      for (u = 0; u < tn; u++) {
        rv += ine[v = u];
        for ( ; v != rt && vis[v] != u && id[v] == -1; ) {
          vis[v] = u;
          v = pre[v];
        }
        if (v != rt && id[v] == -1) {
          for (i = pre[v]; i != v; i = pre[i]) id[i] = index;
          id[v] = index++;
        }
      }
      if (index == 0) break;
      for (u = 0; u < tn; u++)
        if (id[u] == -1) id[u] = index++;
      for (i = 0; i < E.size(); i++) {
        v = E[i].v;
        E[i].u = id[E[i].u];
        E[i].v = id[E[i].v];
        if (E[i].u != E[i].v) E[i].w -= ine[v];
      }
      tn = index;
      rt = id[rt];
    }
    return rv;
  }
};
```

9.7 Spfa Cost Stream

```
/* Spfa_Cost_Stream
 * */
template<class edge_t, int N> struct ek_t {
  vector<edge_t> E;
  static const int inf = 0x7f7f7f7f;
  int n, *L, src, snk, dis[N], ra[N], inq[N];
  int spfa(int u) {
    vector<int> q(1, u);
    memset(dis, 0x3f, sizeof(int)*n);
    memset(ra, -1, sizeof(int)*n);
    memset(inq, 0, sizeof(int)*n);
    dis[u] = 0;
    inq[u] = 1;
    for (int h = 0; h < q.size(); h++) {
      u = q[h], inq[u] = 0;
      for (int e = L[u]; ~e; e = E[e].to) {
        int v = E[e].v, w = E[e].w, c = E[e].c;
        if (w && dis[v] > dis[u]+c) {
          dis[v] = dis[u]+c;
          ra[v] = e^1;
          if (inq[v]) continue;
          inq[v] = 1;
          q.push_back(v);
        }
      }
    }
    return ra[snk] != -1;
  }
  int operator () (vector<edge_t> _E, int *_L, int _n, int _src, int _snk) {
    E = _E, L = _L, n = _n;
    src = _src, snk = _snk;
    int mmf = 0;
    for ( ; spfa(src); ) {
      int mf = inf;
      for (int e = ra[snk]; ~e; e = ra[E[e].v])
        mf = min(mf, E[e^1].w);
      for (int e = ra[snk]; ~e; e = ra[E[e].v])
        E[e].w += mf, E[e^1].w -= mf;
      mmf += dis[snk]*mf;
    }
    return mmf;
  }
};
```

9.8 KM Maximum perfect match

```
/* KM_Maximum_perfect_match
 * Notice that we could use this, when left side has the same amount
 * as right side. (perfect match)
 * If the situation above doesn't be hold, Cost-Flow algorithm is recommanded.
 * */
template<class edge_t, int N> struct km_t {
  vector<edge_t> E;
  static const int inf = 0x7f7f7f7f;
  typedef int kmia_t[N];
  kmia_t mat, lta, rta, sla, lvi, rvi;
  int n, *L;
  int dfs(int u) {
    lvi[u] = 1;
    for (int e = L[u]; ~e; e = E[e].to) {
      int v = E[e].v, w = E[e].w;
      if (!rvi[v]) {
        int t = lta[u]+rta[v]-w;
        if (!t) {
          rvi[v] = 1;
          if (mat[v] == -1 || dfs(mat[v])) {
            mat[v] = u;
            return 1;
          }
        } else if (t < sla[v]) sla[v] = t;
      }
    }
    return 0;
  }
  int operator () (vector<edge_t> &_E, int _L[N], int _n) {
    E = _E, L = _L, n = _n;
    memset(lta, 0, sizeof(lta));
    memset(rta, 0, sizeof(rta));
    memset(mat, -1, sizeof(mat));
    for (int u = 0; u < n; u++)
      for (int e = L[u]; ~e; e = E[e].to)
        if (lta[u] < E[e].w) lta[u] =  E[e].w;
    for (int u = 0; u < n; u++) {
      for (int e = L[u]; ~e; e = E[e].to) sla[E[e].v] = inf;
      for ( ; ; ) {
        memset(lvi, 0, sizeof(lvi));
        memset(rvi, 0, sizeof(rvi));
        if (dfs(u)) break;
        int mn = inf;
        for (int v = 0; v < n; v++)
          if (!rvi[v]) mn = min(mn, sla[v]);
        for (int v = 0; v < n; v++) {
          if (lvi[v]) lta[v] -= mn;
          if (rvi[v]) rta[v] += mn;
          else sla[v] -= mn;
        }
      }
    }
    int rv = 0;
    for (int v = 0; v < n; v++) if (~mat[v])
      for (int e = L[mat[v]]; ~e; e = E[e].to)
        if (E[e].v == v) {
          rv += E[e].w;
          break;
        }
    return rv;
  }
};
```

9.9 Doubling LCA

```
/* Doubling_LCA
 * */
template<class edge_t, int N> struct lca_t {
  static const int M = 16;
  int d[N], a[N][M], p[1<<M];
  void operator () (vector<edge_t> E, int *L, int u) {
    vector<int> q(1, u);
    memset(a, -1, sizeof(a));
    for (int h = d[u] = 0; h < q.size(); h++) {
      u = q[h];
      for (int i = 1; i < M; i++)
        if (~a[u][i-1]) a[u][i] = a[a[u][i-1]][i-1];
      for (int e = L[u]; ~e; e = E[e].to) {
        int v = E[e].v;
        if (v == a[u][0]) continue;
        d[v] = d[u]+1;
        a[v][0] = u;
        q.push_back(v);
      }
    }
    for (int i = 0; i < M; i++) p[1<<i] = i;
  }
  int skip(int u, int x) {
    for ( ; x; x -= -x&x) u = a[u][p[-x&x]];
    return u;
  }
  int operator () (int u, int v) {
    if (d[u] < d[v]) swap(u, v);
    u = skip(u, d[u]-d[v]);
    if (u == v) return u;
    for (int i = M-1; ~i && a[u][0] != a[v][0]; i--)
      if (~a[u][i] && a[u][i] != a[v][i])
        u = a[u][i], v = a[v][i];
    return a[u][0];
  }
};
```

9.10 Shortest Augment Path

```
/* Shortest_Augment_Path
 * */
template<class edge_t, int N> struct sap_t {
  int dis[N], gap[N], _L[N], se[N];
  int operator () (vector<edge_t> &E, int *L, int V, int src, int snk) {
    int mxf = 0, te = 0;
    memcpy(_L, L, sizeof(L));
    memset(dis, -1, sizeof(dis));
    memset(gap, 0, sizeof(gap));
    gap[dis[snk] = 0] = 1;
    vector<int> q(1, snk);
    for (int h = 0; h < q.size(); h++)
```

```
      for (int i = L[q[h]]; i != -1; i = E[i].to)
        if (E[i].w && dis[E[i].v] < 0) {
          gap[dis[E[i].v] = dis[q[h]]+1]++;
          q.push_back(E[i].v);
        }
    for (int u = src; dis[src] < V; ) {
      for (int &i = _L[u]; i != -1; i = E[i].to)
        if (E[i].w && dis[u] == dis[E[i].v] + 1) break;
      if (_L[u] != -1) {
        u = E[se[te++] = _L[u]].v;
        if (u == snk) {
          int _i = 0, mf = 0x7fffffff;
          for (int i = 0; i < te; i++)
            if (E[se[i]].w < mf) {
              mf = E[se[i]].w;
              _i = i;
            }
          for (int i = 0; i < te; i++) {
            E[se[i]].w -= mf;
            E[se[i]^1].w += mf;
          }
          mxf += mf;
          u = E[se[te = _i]^1].v;
        }
        continue;
      }
      int md = V;
      _L[u] = -1;
      for (int i = L[u]; i != -1; i = E[i].to)
        if (E[i].w && dis[E[i].v] < md) {
          md = dis[E[i].v];
          _L[u] = i;
        }
      if (!--gap[dis[u]]) break;
      gap[dis[u] = md+1]++;
      if (u != src) u = E[se[te---1]^1].v;
    }
    return mxf;
  }
};
```

9.11 ZKW Cost Stream

```
/* ZKW_Cost_Stream
 * */
template<class edge_t, int N> struct zkw_t {
  vector<edge_t> E;
  static const int inf = 0x7f7f7f7f;
  int n, src, snk, mc, mf, dis, vis[N], *L;
  int ap(int u, int f) {
    if (u == snk) {
      mc += dis*f;
      mf += f;
      return f;
    }
    vis[u] = 1;
    int rf = f;
    for (int e = L[u]; e > -1; e = E[e].to)
      if (!vis[E[e].v] && E[e].w && !E[e].c) {
        int df = ap(E[e].v, min(rf, E[e].w));
        E[e].w -= df;
        E[e^1].w += df;
        rf -= df;
        if (!rf) return f;
      }
    return f-rf;
  }
  int ml() {
    int md = inf;
    for (int u = 0; u < n; u++) if (vis[u])
      for (int e = L[u]; ~e; e = E[e].to)
        if (!vis[E[e].v] && E[e].w)
          md = min(md, E[e].c);
    if (md == inf) return 0;
    for (int u = 0; u < n; u++) if (vis[u])
      for (int e = L[u]; ~e; e = E[e].to) {
        E[e].c -= md;
        E[e^1].c += md;
      }
    dis += md;
    return 1;
  }
  int operator () (vector<edge_t> &_E, int *_L, int _n, int _src, int _snk) {
    E = _E, L = _L, n = _n;
    src = _src, snk = _snk;
    mf = mc = dis = 0;
    for ( ; ; ) {
      for ( ; ; ) {
        memset(vis, 0, sizeof vis);
        if (!ap(src, inf)) break;
      }
      if (!ml()) break;
    }
    return mc;
  }
};
```

10 graph test

10.1 Graph

```
/* Graph
 * */
struct graph_t {
  struct edge_t {
    int v, to;
  };
  vector<edge_t> e;
  vector<int> h;
  edge_t &operator [] (int x) {
    return e[x];
  }
  int &operator () (int x) {
    return h[x];
  }
  int size() {
    return h.size();
  }
  void init(int n) {
    e.clear(), h.resize(n);
    fill(h.begin(), h.end(), -1);
  }
  void add(int u, int v) {
    edge_t t = {v, h[u]};
    h[u] = e.size();
    e.push_back(t);
  }
  void badd(int u, int v) {
    add(u, v), add(v, u);
  }
};
```

10.2 Shortest Augment Path

```
/* Shortest_Augment_Path
 * */
template<class graph_t> struct sap_t {
  vector<int> dis, gap;
  int dfs(graph_t &g, int src, int snk, int u, int f = ~1u>>1) {
    if (u == snk) return f;
    int rf = f, md = g.size()-1;
    for (int e = g(u); ~e; e = g[e].to) {
      int v = g[e].v, w = g[e].w;
      if (!w) continue;
      md = min(md, dis[v]);
      if (dis[u] != dis[v]+1) continue;
      int df = dfs(g, src, snk, v, min(w, f));
      g[e].w -= df, g[e^1].w += df;
      if (gap[src] == g.size() || !(rf -= df)) return f;
    }
    if (!--gap[dis[u]]) gap[src] = g.size();
    else gap[dis[u] = md+1]++;
    return f-rf;
  }
  int operator () (graph_t &g, int src, int snk) {
    dis.clear(), gap.clear();
    for (int i = g.size()<<1; i; i--)
      dis.push_back(-1), gap.push_back(0);
    vector<int> q(gap[dis[snk] = 0] = 1, snk);
    for (int h = 0; h < q.size(); h++)
      for (int e = g(q[h]); ~e; e = g[e].to)
        if (g[e^1].w && !~dis[g[e].v])
          gap[dis[g[e].v] = dis[q[h]]+1]++, q.push_back(g[e].v);
    for (int i = 0; i < g.size(); i++)
      if (!~dis[i]) gap[dis[i] = 0]++;
    int result = 0;
    for ( ; gap[src] < g.size(); ) result += dfs(g, src, snk, src);
    return result;
  }
};
```

10.3 Strong Connected Component

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

```cpp
  int time, cc;
  vector<int> dfn, low, in, pushed, st;
  void dfs(graph_t &g, int u) {
    st.push_back(u), pushed[u] = 1;
    dfn[u] = low[u] = time++;
    for (int e = g(u); ~e; e = g[e].to) {
      int v = g[e].v;
      if (!~dfn[v]) dfs(g, v), low[u] = min(low[u], low[v]);
      else if (pushed[v]) low[u] = min(low[u], dfn[v]);
    }
    if (dfn[u] == low[u]) {
      for ( ; ; ) {
        in[u = st.back()] = cc;
        st.pop_back(), pushed[u] = 0;
        if (dfn[u] == low[u]) break;
      }
      cc++;
    }
  }
  void operator () (graph_t &g) {
    dfn.clear(), low.clear(), in.clear(), pushed.clear(), st.clear();
    for (int i = 0; i < g.size(); i++)
      dfn.push_back(-1), low.push_back(-1), in.push_back(-1), pushed.push_back(0);
    for (int u = time = cc = 0; u < g.size(); u++)
      if (!~dfn[u]) dfs(g, u);
  }
};
```

10.4 Heavy Light Division

```cpp
/* Heavy_Light_Division
 * */
template<class graph_t, int N> struct hld_t {
  typedef int ai_t[N];
  ai_t d, sz, hb, fa, cl, in, id;
  void link(int h) {
    cl[h] = 1, in[h] = h, id[h] = 0;
    for (int v = h; ~hb[v]; )
      in[v = hb[v]] = h, id[v] = cl[h]++;
  }
  void go(graph_t &g, int u, int p = -1, int l = 0) {
    d[u] = l, sz[u] = 1, hb[u] = -1, fa[u] = p;
    for (int e = g(u); ~e; e = g[e].to) {
      int v = g[e].v;
      if (v == p) continue;
      go(g, v, u, l+1);
      sz[u] += sz[v];
      if (!~hb[u] || sz[hb[u]] < sz[v]) hb[u] = v;
    }
    for (int e = g(u); ~e; e = g[e].to)
      if (g[e].v != p && g[e].v != hb[u]) link(g[e].v);
    if (!~p) link(u);
  }
  void make(int *w, int n) {
  }
  int ask(int u, int v) {
    int result;
    for ( ; in[u]^in[v]; u = fa[in[u]]) {
      if (d[in[u]] < d[in[v]]) swap(u, v);
    }
    if (id[u] > id[v]) swap(u, v);
    return result;
  }
};
```

10.5 Biconnected Component

```
/* Biconnected_Component
 * */
struct bcc_t {
  int time, cc, cut[N], dfn[N], low[N];
  vector<int> in, st;
  void dfs(graph_t &g, int u, int p = -1) {
    int branch = 0;
    dfn[u] = low[u] = time++;
    for (int e = g(u); ~e; e = g[e].to) {
      int v = g[e].v;
      if (e == p || dfn[v] >= dfn[u]) continue;
      st.push_back(e);
      if (~dfn[v]) low[u] = min(low[u], dfn[v]);
      else {
        branch++;
        dfs(g, v, e^1);
        low[u] = min(low[u], low[v]);
        if (dfn[u] > low[v]) continue;
        for (cut[u] = 1; ; ) {
          int t = st.back();
          st.pop_back();
          in[t] = in[t^1] = cc;
          if (t == e) break;
        }
        cc++;
      }
    }
    if (!~p && cut[u] && branch < 2) cut[u] = 0;
  }
  void operator () (graph_t &g) {
    in.resize(g.e.size());
    for (int u = 0; u < g.size(); u++)
      dfn[u] = low[u] = -1, cut[u] = 0;
    st.clear();
    for (int u = time = cc = 0; u < g.size(); u++)
      if (!~dfn[u]) dfs(g, u);
  }
};
```

10.6 Static Lowest Ancestor

```
/* Static_Lowest_Ancestor
 * */
struct slca_t {
  graph_t q;
  vector<int> r, f, c, d;
  void init(int n) {
    q.init(n), f.resize(n), c.resize(n);
    for (int i = 0; i < n; i++)
      f[i] = i, c[i] = 0;
  }
  void add(int u, int v) {
    r.push_back(-1);
    q.badd(u, v);
  }
  int find(int x) {
    if (x != f[x]) f[x] = find(f[x]);
    return f[x];
  }
  void go(graph_t &g, int u, int p = -1) {
    for (int e = g(u); ~e; e = g[e].to) {
      if (e == p) continue;
      go(g, g[e].v, e^1);
      f[find(g[e].v)] = u;
    }
    c[u] = 1;
    for (int e = q(u); ~e; e = q[e].to) {
      if (!c[q[e].v]) continue;
      r[e>>1] = find(q[e].v);
    }
  }
};
```