National Chiao Tung University Team Reference Document

Contents

```
" system "
set nocompatible                    " safty option "
set printoptions=number:y           " print with line number "
set autoread                        " reload file when modified "

" tabs and indents "
set tabstop=4                       " number of visual spaces per TAB "
set softtabstop=4                   " number of spaces in tab when editing "
set shiftwidth=4                    " number of spaces of indent "
set smartindent                     " autoindent "
set smarttab                        " autoindent "
filetype plugin indent on           " load filetype-specific indent files "

" appearance "
syntax on                           " syntax hilight "
set ruler                           " show line and column number "
set number                          " line number "
set showcmd                         " show commands "
set showmatch                       " blink matched brackets "
set cursorline                      " hilight currsor line "
set cursorcolumn                    " hilight currsor column "
highlight CursorLine ctermbg=235
highlight CursorColumn ctermbg=235
set background=dark                 " background color "

" search "
set incsearch                       " search as charcaters entered "
set hlsearch                        " hilight matches "

" shortcut "
map Z gg=G
set backspace=indent,eol,start
set whichwrap=b,s,<,>,[,]
```

#pragma GCC optimize (2)

Topic I

Math

1 Pick's Theorem

給定座標皆爲整數點的簡單多邊形，設其面積爲 S，內部整數點個數爲 a，邊界上整數點個數爲 b，則他們滿足 $S = a + \frac{b}{2} - 1$

2 Grey Code

The ith grey code is $i \oplus (i >> 1)$

3 Extended Euclidean

Find x, y s.t. $ax + by = gcd(a, b)$

```
lld ext_gcd(lld a, lld b, lld& x, lld& y){
  if( b==0 ){
    x=1, y=0;
    return a;
  }
  lld g=ext_gcd(b, a%b, y, x);
  y-=x*(a/b);
  return g;
}
```

4 Chinese Remainder Theorem

Find x s.t. $x \equiv a_i \, mod \, m_i$

```
lld CRT(int n, lld a[], lld m[]){
  for(int i=1; i<n; i++){
    lld x, y, g=ext_gcd(m[i-1], m[i], x, y), r=a[i]-a[i-1];
    if( r%g!=0 ) return -1;
    lld t=m[i]/g;
    x=(r/g*x%t+t)%t;
    a[i]=a[i-1]+m[i-1]*x;
    m[i]=m[i-1]*t;
  }
  return a[n-1];
}
```

5 Mod Equation

Find x s.t. $ax \equiv b \, mod \, n$

```
vector<lld> mod_eq(lld a, lld b, lld n){
  lld x, y, g=ext_gcd(a, n, x, y);
  vector<lld> ans;
  if( b%g==0 ){
    x=((x%n)+n)%n;
    ans.push_back(x*(b/g)%(n/g));
    for(lld i=1; i<g; i++) ans.push_back((ans[0]+i*n/g)%n);
  }
  return ans;
}
```

6 Quadratic residue

Find x s.t. $x^2 \equiv a \, mod \, p$

```
lld solve(lld a, lld p){
  if( a%p==0 ) return 0;
  lld q=p-1, z=1, s=0;
  for(; q>0 && !(q&1); q>>=1) s++;
  for(; pmd(z, p>>1, p)!=p-1; z=rnd(1, p));//rnd(l, r): [l, r)
  lld c=pmd(z, q, p), t=pmd(a, q, p), r=pmd(a, (q+1)>>1, p);
  for(lld i=1; t!=1; i=1){
    for(lld tt=t*t%p; i<s && tt!=1; i++, tt=tt*tt%p);
    if( i==s ) return -1;
    lld b=pmd(c, pmd(2, s-i-1, p), p);
    s=i; c=b*b%p; t=t*c%p; r=r*b%p;
  }
  return r;
}
```

7 Rho Algorithm for Factorization

Find a non-trivial divisor of composite n in $O(\sqrt[4]{n})$

```
inline lld f(lld x, lld n) {
  return (x*x+1)%n;
}
lld rho_fact(lld n) {
  for(lld x=2, y=2; ; ){
    x=f(x, n), y=f(f(y, n), n);
    lld d=gcd(abs(x-y), n);
    if( d>1 ) return d==n ? -1 : d ;
  }
}
```

8 Discrete Logarithm

Let $gcd(a, n) = 1$, find $\log_a b \, mod \, n$ in $O(\sqrt{n} \log n)$

```
lld solve(lld a, lld b, lld p){
  int sp=ceil(sqrt(p));
  map<lld, int> M;
  lld tmp=1;
  for(int i=0; i<sp && M.find(tmp)==M.end(); i++)
    M[tmp]=i, tmp=tmp*a%p;
  lld x, y;
  ext_gcd(tmp, p, x, y);
  tmp=(x+p)%p;
  for(int i=0; i<sp; i++){
    auto res = M.find(b);
    if( res!=M.end() ) return i*(lld)sp+res->second;
    b=b*tmp%p;
  }
  return -1;
}
```

9 Möbius Function

$$\mu(n) = \begin{cases} 0 & \text{if } \exists p^2 \mid n \\ (-1)^{\text{number of prime factors of } n} & \text{otherwise} \end{cases}$$

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } n > 1 \end{cases}$$

For numerical functions $f(n)$ and $F(n)$, if

$$F(n) = \sum_{d|n} f(d)$$

then

$$f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$$

```
int mu[n]={0};
void get_mu(){
  mu[1]=-1;
  for(int i=1; i<n; i++){
    mu[i]=-mu[i];
    for(int j=i<<1; j<n; j+=i) mu[j]+=mu[i];
  }
}
```

10 Phi Function

$\phi(n)$ = number of integers less than and coprime to n (including 1)

```
int phi[n], minDiv[n];
void get_phi(){
  phi[1] = 1;
  for(int i=2; i<n; i++) minDiv[i]=i;
  for(lld i=2; i<n; i++) if( minDiv[i]==i )
    for(lld j=i*i; j<n; j+=i) minDiv[j]=i;
  for(int i=2; i<n; i++){
    phi[i]=phi[ i/minDiv[i] ];
    phi[i]*=minDiv[i]-(i/minDiv[i]%minDiv[i]==0 ? 0 : 1);
  }
}
```

11 Miller—Rabin Primality Test

```
bool test(lld n, lld a, lld d){
  if( n==a ) return true;
  while( !(d&1) ) d>>=1;
  lld t=pow_mod(a, d, n);//a^d(mod n)
  while( d!=n-1 && t!=1 && t!=n-1 )
    t=t*t%n, d<<=1;
  return t==n-1 || d&1;
}
bool is_prime(lld n){
  if( n<2 ) return false;
  if( n<4 ) return true;
  if( !(n&1) ) return false;
  //int a[3]={2, 7, 61};//for int n
  int a[12]={2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
  for(int i=0; i<12; i++)
    if( !test(n, a[i], n-1) ) return false;
  return true;
}
```

12 Simplex

0-based index
Find $\max\{cx\}$ subjected to $ax \leq b \wedge x \geq 0$
n: constraints, m: variables

```
const double eps = 1E-10;
double a[maxn][maxm], b[maxn], c[maxm], d[maxn][maxm], x[maxm];
int ix[maxn + maxm];
double simplex(int n, int m){
  ++m;
  int r = n, s = m - 1;
  memset(d, 0, sizeof(d));
  for (int i = 0; i < n + m; ++i) ix[i] = i;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
    d[i][m - 1] = 1; d[i][m] = b[i];
    if (d[r][m] > d[i][m]) r = i;
  }
  for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
  d[n + 1][m - 1] = -1;
  for (double dd;; ) {
    if (r < n) {
      int t = ix[s];
      ix[s] = ix[r + m]; ix[r + m] = t;
      d[r][s] = 1.0 / d[r][s];
      for (int j = 0; j <= m; ++j) if (j != s) d[r][j] *= -d[r][s];
      for (int i = 0; i <= n + 1; ++i) if (i != r) {
          for (int j = 0; j <= m; ++j)
            if (j != s) d[i][j] += d[r][j]*d[i][s];
          d[i][s] *= d[r][s];
        }
    }
    r = -1; s = -1;
    for (int j = 0; j < m; ++j) if (s < 0 || ix[s] > ix[j])
        if(d[n + 1][j] > eps || (d[n + 1][j] > -eps && d[n][j] > eps)) s = j;
    if (s < 0) break;
    for (int i=0; i<n; ++i) if (d[i][s] < -eps)
      if (r < 0 || (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s]) < -eps || (dd < eps && ix[r + m] > ix[i + m])) r = i;
    if (r < 0) return -1; // not bounded
  }
  if (d[n + 1][m] < -eps) return -1; // not executable
  double ans = 0;
  for (int i = 0; i < m; ++i) x[i] = 0;
  for (int i = m; i < n + m; ++i) { // the missing enumerated x[i] = 0
    if (ix[i] < m - 1) {
      ans += d[i - m][m] * c[ix[i]];
      x[ix[i]] = d[i-m][m];
    }
  }
  return ans;
}
```

13 Determinant

```
lld det(int n, lld m, vector<vector<lld>> a){
  lld div=1;
  for(int k=n-1; k>0; k--){
    if( a[k][k]==0 ) for(int i=0; i<k; i++) if( a[i][k]!=0 )
      swap(a[i], a[k]), div=m-div;
    for(int i=0; i<k; i++) for(int j=0; j<k; j++)
      a[i][j]=((a[i][j]*a[k][k]-a[i][k]*a[k][j])%m+m)%m;
    div=div*pmd(a[k][k], k-1, m)%m;
  }
  return a[0][0]*inv(div, m)%m;
}
```

14 Fast Fourier Transform

```
typedef complex<double> cplx;
void fft(cplx A[], int lgn, bool inv=false){
  int n=1<<lgn;
  for(int i=0, j=1; j<n-1; j++){
    for(int k=n>>1; k>(i^=k); k>>=1);
    if( j<i ) swap(A[i], A[j]);
  }
  for(int i=1; i<n; i<<=1){
    cplx W(1, 0), Wn(cos(PI/i), sin((inv ? -PI : PI)/i));
    for(int j=0; j<n; j++){
      if( j&i ){
        W=cplx(1, 0); continue;
      }
      cplx x=A[j], y=A[j+i]*W;
      A[j]=x+y; A[j+i]=x-y;
      W*=Wn;
    }
  }
  if( inv ) for(int i=0; i<n; i++) A[i]/=n;
}
```

15 Polynomial Solver

$p(x) = \sum_{i=0}^{n} coef_i \cdot x^i$

Solve $p(x) = 0$ recursively between successive mins and maxs

```
const double EPS=1e-12, INF=1e12;
int sign(double x){
  return x<-EPS ? -1 : x>EPS ;
}
double get(const vector<double>& coef, double x){
  double ans=0;
  for(int i=coef.size()-1; i>=0; i--)
    ans=ans*x+coef[i];
  return ans;
}
double find(const vector<double>& coef, int n, double lo, double hi){
  int sign_lo=sign(get(coef, lo));
  int sign_hi=sign(get(coef, hi));
  double m=INF;
  if( sign_lo==0 ) return lo;
  if( sign_hi==0 ) return hi;
  if( sign_lo==sign_hi ) return INF;
  for(int step=0; step<100 && hi-lo>EPS; step++){
    m=(hi+lo)*.5;
    int sign_m=sign(get(coef, m));
    if( sign_m==0 ) return m;
    (sign_m*sign_lo<0 ? hi : lo)=m;
  }
  return m;
}
vector<double> equation(vector<double>& coef, int n){
  vector<double> ans;
  if( n==1 )
    return sign(coef[1]) ? vector<double>(1, -coef[0]/coef[1]) : ans ;
  if( !sign(coef[n]) )
    return equation(coef, n-1);
  if( sign(coef[n])<0 )
    for(int i=0; i<=n; i++) coef[i]=-coef[i];
  vector<double> dcoef(n);
  for(int i=1; i<=n; i++) dcoef[i-1]=coef[i]*i;
  vector<double> droot=equation(dcoef, n-1);
  droot.insert(droot.begin(), n&1 ? -INF : INF);
  droot.insert(droot.end(), INF);
  for(int i=1; i<droot.size(); i++){
    double tmp=find(coef, n, droot[i-1], droot[i]);
    if( tmp<INF ) ans.push_back(tmp);
  }
  return ans;
}
```

16 2-SAT

n: number of vars

```
struct SAT2{
  int n;
  vector<vector<int>> edg;
  vector<int> dfn, low, com;
  stack<int> S;
  SAT2(int _n) : n(_n), edg(n<<1), dfn(n<<1, 0), low(n<<1), com(n<<1, 0){}
  //void tarjan(int u, d=0);
  int trans(int a){
    return abs(a)-1<<1|a<0;
  }
  void add_or(int a, int b){
    a=trans(a), b=trans(b);
    if( a>>1!=b>>1 ){
      edg[a^1].push_back(b); edg[b^1].push_back(a);
    }
    else if( a==b ) edg[a^1].push_back(a);
  }
  void add_xor(int a, int b){
    add_or(a, b); add_or(-a, -b);
  }
  bool solve(){
    for(int i=0; i<n<<1; i++) if( !dfn[i] ) tarjan_scc(i);
    for(int i=0; i<n<<1; i++) if( com[i]==com[i^1] ) return false;
    return true;
  }
};
```

17 SAT Solver

n: number of vars

```
struct SAT
{
  int n, m, rem, level;
  vector<vector<int>> C, CNT;//0: false, 1: true, 2: unassigned
  vector<int> UNIT, X, STAT;//0: unassigned, 1: decided, 2: implied
  vector<vector<vector<int>>> POS;//0: false, 1: true
  vector<vector<int>> I, ORDER;
  SAT(int _n, vector<vector<int>>& cnf) : n(_n), level(0), C(cnf), I(1){
    m=C.size(), rem=n;
    for(vector<int> &c : C) CNT.push_back({ 0, 0, (int)c.size() });
    X.resize(n + 1);
    STAT.assign(n + 1, 0);
    POS.assign(n + 1, { {}, {} });
    for(int i=0; i<m; i++) for(int x : C[i])
      POS[abs(x)][x < 0 ? 0 : 1].push_back(i);
    for(vector<int> &c : C) if( c.size()==1 )
      UNIT.push_back(c[0]);
    for(int i=1; i<=n; i++)
      ORDER.push_back({ (int)(POS[i][0].size() + POS[i][1].size()), i });
    sort(ORDER.rbegin(), ORDER.rend());
  }
  bool assign(int x, int val, int stat){//assumption: x is unassigned
    rem--, X[x]=val, STAT[x]=stat;
    bool res=true;
    for(int i=0; i<2; i++) for(int c : POS[x][i]){
      CNT[c][2]--, CNT[c][val==i]++;
      if( CNT[c][2]==1 && CNT[c][1]==0 )
        for(int y : C[c])
          if( STAT[abs(y)]==0 ){
            UNIT.push_back(y);
            break;
          }
      res&= !(CNT[c][1]==0 && CNT[c][2]==0);
    }
    return res;
  }
  void unassign(int x){//assumption: x is assigned
    rem++, STAT[x]=0;
    for(int i=0; i<2; i++)
      for(int c : POS[x][i])
        CNT[c][2]++, CNT[c][X[x]==i]--;
  }
  bool imply(){
    while( !UNIT.empty() ){
      int x=UNIT.back(), val=x>0;
      UNIT.pop_back();
      if( STAT[abs(x)]!=0 && X[abs(x)]==val ) continue;
      if( STAT[abs(x)]!=0 && X[abs(x)]!=val ) return false;
      I.back().push_back(abs(x));
      if (!assign(abs(x), val, 2)) return false;
    }
    return true;
  }
  void unimply(){
    for(int x : I.back()) unassign(x);
    I.back().clear();
  }
  bool sol(){
    if( !imply() ) return false;
    if( rem==0 ) return true;
    int x;
    for(int i=0; STAT[ x=ORDER[i][1] ]; i++);
    level++, I.push_back({});
    assign(x, rand()&1, 1);
    if( sol() ) return true;
    unassign(x), unimply(), UNIT.clear();
    assign(x, X[x]^1, 1);
    if( sol() ) return true;
    unassign(x), unimply();
    level--, I.pop_back();
    return false;
  }
};
```

18 Point

```
struct point{
  double x,y;
  point(double _x=0, double _y=0) : x(_x), y(_y){}
  point operator+(const point& p) const{//vector sum
    return point(x+p.x, y+p.y);
  }
  point operator-(const point& p) const{//vector difference
    return point(x-p.x, y-p.y);
  }
  point operator*(double s) const{//vector scaling
    return point(x*s, y*s);
  }
  point operator/(double f) const{//vector scaling
    return point(x/f, y/f);
  }
  double operator^(const point& P)const{//cross
    return x*P.y-y*P.x;
  }
  double operator&(const point& P)const{//dot
    return x*P.x+y*P.y;
  }
  double operator()() const{//square of length
    return x*x+y*y;
  }
  bool operator<(const point& P) const{
    return x==P.x ? y<P.y : x<P.x ;
  }
};
```

19 Convex Hull

```
vector<point> convex_hull(vector<point> p){
  int n=p.size(), k=0;
  vector<point> h(n<<1);
  sort(p.begin(), p.end());
  for (int i=0; i<n; i++){
    for(; k>=2 && (h[k-1]-h[k-2])^(p[i]-h[k-1])<=0; k--);
    h[k++]=p[i];
  }
  for (int i=n-2, t=k; i>=0; i--){
    for(; k>t && (h[k-1]-h[k-2])^(p[i]-h[k-1])<=0; k--);
    h[k++]=p[i];
  }
  h.resize(k);
  return h;
}
```

20 Minimum Enclosing Disc

```
struct circle{
  point c; double r;
  circle(const point& _c, double _r) : c(_c), r(_r){}
  circle(const point& p, const point& q) : c((p+q)*0.5), r((c-p)()*(c-p)()){}
  circle(const point& A, const point& B, const point& C){
    point a=B-A, b=C-A;
    double c1=a()*a()*0.5, c2=b()*b()*0.5, d=a^b;
    double x=A.x+(c1*b.y-c2*a.y)/d, y=A.y+(a.x*c2-b.x*c1)/d;
    c=point(x, y); r=(c-A)()*(c-A)();
  }
  bool in(const point& p) const{
    return (p-c)()*(p-c)()<=r+EPS;
  }
};
circle solve(vector<point> p){
  random_shuffle(p.begin(), p.end());
  circle ans(point(), 0);
  for(int n=p.size(), i=0; i<n; i++){
  if( ans.in(p[i]) ) continue;
    ans=circle(p[i], 0);
    for(int j=0; j<i; j++){
      if( ans.in(p[j]) ) continue;
      ans=circle(p[i], p[j]);
      for(int k=0; k<j; k++){
        if( ans.in(p[k]) ) continue;
        ans=circle(p[i], p[j], p[k]);
      }
    }
  }
  ans.r=sqrt(ans.r);
  return ans;
}
```

21 Roman Numerals

```
const int num[13]={1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
const string str[13]={"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX",
    "V", "IV", "I"};
string rome(int x){
    for(int i=0; i<13; i++) if( x>=num[i] ) return str[i]+rome(x-num[i]);
  return "";
}
int rome(const string& s){
  for(int i=0; i<13; i++)
    if( s.length()>=str[i].length() && s.substr(0, str[i].length())==str[i] )
      return num[i]+rome(s.substr(str[i].length()));
  return 0;
}
```

Topic II

String

1 KMP

```
void failure(char *s, int f[]){
  f[0]=-1;
  for(int k, m=strlen(s), i=1; i<=m; i++){
    for(k=f[i-1]; k>=0 && s[k]!=s[i-1]; k=f[k]);
    f[i]=k+1;
  }
}
vector<int> KMP(char* s, char* t){//search for s in t
  int m=strlen(s), n=strlen(t), f[m+1];
  failure(s, f);
  vector<int> result;
  for(int k=0, i=0; i<n; ){
    if( k==-1 ) i++, k=0;
    else if( t[i]==s[k] ){
      i++, k++;
      if(k==m){
        result.push_back(i-m);
        k=f[k];
      }
    }else k=f[k];
  }
  return result;
}
```

2 Z-algorithm

z_i is the length of the longest substring starting from s_i which is also a prefix of s.

```
void Z(const char* s, int z[]){
    int n=z[0]=strlen(s);
  for(int l=0, r=0, i=1; i<n; i++)
      if( r<i || r-i<z[i-l] ){
          if( r<i ) r=i;
          for(l=i; r<n && s[r]==s[r-l]; r++);
          z[i]=r---l;
      }else z[i]=z[i-l];
}
```

3 Longest Palindromic Substring

s should be preprocessed to the form $|a|b|c|c|b|a|$
p_i is the length of LPS with center i

```
void solve(const char* s){
  int l=strlen(s);
  vector<int> p(l+1, 1);
  for(int c=0, r=0, n=0, m=0, i=1; i<l; i++)
  {
    int j=(c<<1)-i;
    if( i>r )
      p[i]=0, n=i+1, m=i-1;
    else if( p[j]<r-i )
      p[i]=p[j], m=-1;
    else
      p[i]=r-i, n=r+1, m=(i<<1)-n;
    while( n<l && m>=0 && s[m]==s[n] )
      p[i]++, n++, m--;
    if( i+p[i]>r )
      c=i, r=i+p[i];
  }
}
```

4 Lexicographically Smallest Rotation

```
string lsr(string s){
  int n=s.length(), i=0, j=1;
  for(s+=s; i<n && j<n; j+=i==j){
    int k=0;
    while( k<n && s[i+k]==s[j+k] ) k++;
    (s[i+k]<=s[j+k] ? j : i)+=k+1;
  }
  return s.substr(i<n ? i : j, n);
}
```

5 AC Trie

```
struct actrie{
  struct node{
    node *fl, *nx[26], *dl;
    int cnt, d;
    node(){
      memset(this, 0, sizeof(node));
    }
  } *root;
  actrie(){
    root = new node();
  }
  void add(const char *p){
    node *now=root;
    for(int i=0; p[i]; i++){
      node*& t=now->nx[ p[i]-'a' ];
      if( !t ) t=new node();
      now=t;
    }
    now->cnt++;
  }
  void build(){
    queue<node*> Q;
    for(Q.push(root); !Q.empty(); Q.pop()){
      node* now=Q.front();
      for(int i=0; i<26; i++){
        node*& t=now->nx[i], *fn=now->fl;
        if( t ){
          while( fn && !fn->nx[i] ) fn=fn->fl;
          t->fl= fn ? fn->nx[i] : root ;
          t->dl= t->fl->cnt ? t->fl : t->fl->dl ;
          t->d=now->d+1; Q.push(t);
        }
      }
    }
  }
  void match(const char *p){
    node* now=root;
    for(int i=0; p[i]; i++){
      while( now && !now->nx[ p[i]-'a' ] ) now=now->fl;
      if( !now ) now=root;
      else{
        now=now->nx[ p[i]-'a' ];
        for(node *tmp=now; tmp; tmp=tmp->dl);
      }
    }
  }
};
```

6 Suffix Array

sar_i is the index of sorted suffices
rk_i is the rank of suffix starting from s_i

```
struct suffixarray{
  char s[N];
  int n, sa[N], r[N], lcp[N], sa2[N], r2[N], c[N], a;
  void init(const char* _s){
    memset(this, 0, sizeof(suffixarray));
    n=strlen(_s), a=128;
    memcpy(s, _s, sizeof(char)*n);
    for(int i=0; i<n; i++) c[ r[i]=s[i] ]++;
    for(int i=1; i<a; i++) c[i]+=c[i-1];
    for(int i=n-1; i>=0; i--) sa[ --c[ r[i] ] ]=i;
    for(int l=1; l<n; l<<=1){
      int p=0;
      for(int i=n-l; i<n; i++) sa2[p++]=i;
      for(int i=0; i<n; i++) if( sa[i]-l>=0 ) sa2[p++]=sa[i]-l;
      for(int i=0; i<a; i++) c[i]=0;
      for(int i=0; i<n; i++) c[ r[i] ]++;
      for(int i=1; i<a; i++) c[i]+=c[i-1];
      for(int i=n-1; i>=0; i--) sa[ --c[ r[ sa2[i] ] ] ]=sa2[i];
      r2[ sa[0] ]=0;
      for (int i=1; i<n; i++){
        r2[ sa[i] ]=r2[ sa[i-1] ]+1;
        if( r[ sa[i-1] ]==r[ sa[i] ] && sa[i-1]+l<n && r[ sa[i-1]+l ]==r[ sa[
    i]+l ] ) r2[sa[i]]--;
      }
      for(int i=0; i<n; i++) swap(r[i], r2[i]);
      a=r[ sa[n-1] ]+1;
      if( a==n ) break;
    }
    for(int i=0; i<n; i++) r[ sa[i] ]=i;
    for(int k=0, i=0; i<n; i++, k=max(0, k-1)){
      if( r[i]==n-1 ){
        lcp[ r[i] ]=k=0;
        continue;
      }
      for(int j=sa[ r[i]+1 ]; max(i, j)+k<n && s[i+k]==s[j+k]; k++);
      lcp[ r[i] ]=k;
    }
  }
} SA;
```

Topic III

Graph

1 Biconnected Connected Component

Tarjan's Algorithm in undirected graph

```
vector<vector<int>> edg;
vector<pair<int, int>> brg;
int cut[N]={0}, dfn[N]={0}, low[N];
void tarjan_bcc(int u, int p=-1, int d=0){
  dfn[u]=low[u]=++d;
  int cnt=0;
  for(int v : edg[u]){
    if( v!=p && dfn[v]>0 ) low[u]=min(low[u], dfn[v]);
    else if( dfn[v]==0 ){
      cnt++;
      tarjan_bcc(v, u, d);
      low[u]=min(low[u], low[v]);
      if( d<low[v] ) brg.push_back(pair<int, int>(u, v));
      if( (p<0 && cnt>1) || (p>=0 && d<=low[v]) ) cut[u]=true;
    }
  }
}
```

2 Strongly Connected Component

Tarjan's Algorithm in directed graph

```
vector<vector<int>> edg;
int dfn[N]={0}, low[N], com[N]={0}, d=0;
stack<int> S;
void tarjan_scc(int u){
  dfn[u]=low[u]=++d;
  S.push(u);
  for(int v : edg[u]){
    if( !dfn[v] ) tarjan_scc(v);
    if ( !com[v] ) low[u]=min(low[u], low[v]);
  }
  if( d==low[u] ){
    for(bool f=true; f; S.pop()){
      f= S.top()!=u;
      com[S.top()]=u+1;
    }
  }
}
```

3 Bellman-Ford Algorithm

Collection of $O(VE)$ weighted directed graph algorithms including SSSP, negative cycle detection, minimum mean cycle.

```
struct BF{
  struct edge{//directed edge u->v
    int u, v, w;
    edge(int _u, int _v, int _w) : u(_u), v(_v), w(_w){}
  };
  int n;
  vector<int> d;
  vector<edge> e;
  BF(int _n) : n(_n){}//zero-base vertices
  void add(int u, int v, int w){//add an edge
    e.push_back(edge(u, v, w));
  }
  bool relax(){//does relaxation with all edges once
    bool any=false;
    for(const edge& E : e)
      if( d[E.v]>d[E.u]+E.w ) d[E.v]=d[E.u]+E.w, any=true;
    return any;
  }
  void operator()(int s){//compute SSSP start with s
    d.assign(n, INF); d[s]=0;
    for(int i=1; i<n; i++) if( !relax() ) break;
  }
  bool neg_cycle(){//detect negative cycle
    d.assign(n, 0);
    for(int i=0; i<n; i++) if( !relax() ) return false;
    return relax();
  }
  double karp_mmc(){//calculate the min mean cycle ratio
    double ans=INF;
    vector<vector<int>> d(n+1, vector<int>(n, INF));
    d[0].assign(n, 0);
    for(int i=1; i<=n; i++) for(const edge& E : e)
      d[i][E.v]=min(d[i][E.v], d[i-1][E.u]+E.w);
    for(int i=0; i<n; i++){
      double tmp=-INF;
      if( d[n][i]>=INF ) continue;
      for(int j=0; j<n; j++)
        tmp=max(tmp, (d[n][i]-d[j][i])/(double)(n-j));
      ans=min(ans, tmp);
    }
    return ans;
  }
};
```

4 Maximum Flow

Runs in $O(V^2E)$ in general, and $O(min(V^{2/3}E, E^{3/2}))$ for unit network

```
struct Dinic{
  struct edge{
    int t, c, r;
    edge(int _t, int _c, int _r): t(_t), c(_c), r(_r){}
  };
  vector<int> l;
  vector<vector<edge>> e;
  Dinic(int n) : e(n+1){}
  void add(int u, int v, int w){//directed
    e[u].push_back(edge(v, w, e[v].size()));
    e[v].push_back(edge(u, 0, e[u].size()-1));
  }
  edge& rev(const edge& E){
    return e[E.t][E.r];
  }
  bool bfs(int s, int t){
    l.assign(e.size(), INF);
    l[s]=1;
    queue<int> Q;
    for(Q.push(s); !Q.empty(); Q.pop()){
      s=Q.front();
      for(const edge& E : e[s])
        if( E.c>0 && l[E.t]>l[s]+1 ){
          l[E.t]=l[s]+1;
          Q.push(E.t);
        }
    }
    return l[t]<INF;
  }
  int dfs(int s, int t, int num=INF){
    if( s==t || num==0 ) return num;
    int ans=0;
    for(edge& E : e[s])
      if( E.c>0 && l[s]+1==l[E.t] ){
        int tmp=dfs(E.t, t, min(num, E.c));
        rev(E).c+=tmp, ans+=tmp;
        E.c-=tmp, num-=tmp;
      }
    return ans>0 ? ans : l[s]=0;
  }
  int operator()(int s, int t){
    int ans=0, tmp=0;
    while( bfs(s, t) )
      while( (tmp=dfs(s, t)) )
        ans+=tmp;
    return ans;
  }
};
```

5 Minimum-Cost Maximum Flow

$O(maxf \cdot V^2)$

```
struct costflow{
  struct edge{
    int t, f, c, r;//_c, c
    edge(int _t, int _f, int _c, int _r) : t(_t), f(_f), c(_c), r(_r){}
  };
  int n;
  vector<int> prv, plv, dis;//dis
  vector<vector<edge>> e;
  costflow(int _n) : n(_n), prv(n+1), plv(n+1), e(n+1){}
  void add(int u, int v, int f, int c){//c
    e[u].push_back(edge(v, f,  c, e[v].size()));
    e[v].push_back(edge(u, 0, -c, e[u].size()-1));
  }
  edge& rev(const edge& E){
    return e[E.t][E.r];
  }
  bool bfs(int s, int t){
    vector<bool> inq(n+1, false);
    dis.assign(n+1, INF);//INF
    dis[s]=0;
    queue<int> Q;
    for(Q.push(s); !Q.empty(); Q.pop()){
      s=Q.front(), inq[s]=0;
      for(int i=e[s].size()-1; i>=0; i--){
        const edge& E=e[s][i];
        if( dis[E.t]>dis[s]+E.c && E.f>0 ){
          dis[E.t]=dis[s]+E.c;
          prv[E.t]=s, plv[E.t]=i;
          if( !inq[E.t] ) Q.push(E.t), inq[E.t]=true;
        }
      }
    }
    return dis[t]<INF;
  }
  pair<int, int> operator()(int s, int t){//second
    int fl=0, cs=0;//cs
    for(int tf=INF; bfs(s, t); tf=INF){
      for(int v=t, u, l; v!=s; v=u){
        u=prv[v], l=plv[v];
        tf=min(tf, e[u][l].f);
      }
      for(int v=t, u, l; v!=s; v=u){
        u=prv[v], l=plv[v];
        rev(e[u][l]).f+=tf;
        e[u][l].f-=tf;
      }
      cs+=tf*dis[t], fl+=tf;
    }
    return pair<int, int>(fl, cs);//second
  }
};
```

6 Maximum-Weight Bipartite Perfect Matching

$O(V^3)$

```
struct KM{
  static const int INF=2147483647;//long long
  int n;
  vector<int> match, vx, vy;
  vector<int> lx, ly, slack;//long long
  vector<vector<int>> edge;//long long
  KM(int _n) : n(_n), match(n, -1), lx(n, -INF), ly(n, 0), edge(n, vector<int
    >(n, 0)){}
  void add_edge(int x, int y, int w){//long long
    edge[x][y] = w;
  }
  bool dfs(int x){
    vx[x]=1;
    for(int y=0; y<n; y++){
      if( vy[y] ) continue;
      if( lx[x]+ly[y]>edge[x][y] )
        slack[y]=min(slack[y], lx[x]+ly[y]-edge[x][y]);
      else{
        vy[y]=1;
        if( match[y]==-1 || dfs(match[y]) ){
          match[y]=x; return true;
        }
      }
    }
    return false;
  }
  int operator()(){
    for(int i=0; i<n; i++) for(int j=0; j<n; j++)
      lx[i]=max(lx[i], edge[i][j]);
    for(int i=0; i<n; i++) for(slack.assign(n, INF); ; ){
      vx.assign(n, 0); vy.assign(n, 0);
      if( dfs(i) ) break;
      int d=INF;// long long
      for(int j=0; j<n; j++)
        if( !vy[j] ) d=min(d, slack[j]);
      for(int j=0; j<n; j++){
        if( vx[j] ) lx[j]-=d;
        if( vy[j] ) ly[j]+=d;
        else slack[j]-=d;
      }
    }
    int res=0;
    for(int i=0; i<n; i++) res+=edge[match[i]][i];
    return res;
  }
};
```

7 Maximum-Cardinality Bipartite Matching

$O(\sqrt{V}E)$

```
struct HK{
  int n, m;
  vector<int> d, p;
  vector<vector<int>> e;
  HK(int _n, int _m) : n(_n), m(_m), e(n+m+1){}
  void add(int u, int v){//one base index: [1, u]*[1, v]
    e[u].push_back(n+v);
    e[n+v].push_back(u);
  }
  bool bfs(){
    d.assign(n+m+1, INF);
    queue<int> Q;
    for(int i=1; i<=n; i++)
      if( p[i]==0 ){
        d[i]=0;
        Q.push(i);
      }
    for(; !Q.empty(); Q.pop()){
      int u=Q.front();
      if( d[u]>d[0] ) break;
      for(int v : e[u])
        if( d[ p[v] ]==INF ){
          d[ p[v] ]=d[u]+1;
          Q.push(p[v]);
        }
    }
    return d[0]<INF;
  }
  bool dfs(int u){
    if( u==0 ) return true;
    for(int v : e[u])
      if( d[ p[v] ]==d[u]+1 && dfs(p[v]) ){
        p[v]=u, p[u]=v;
        return true;
      }
    d[u]=INF;
    return false;
  }
  int operator()(){
    int ans=0;
    p.assign(n+m+1, 0);
    while( bfs() )
      for(int i=1; i<=n; i++)
        if( p[i]==0 && dfs(i) )
          ans++;
    return ans;
  }
};
```

8 Minimum-Weight General Perfect Matching

$O(V^2E)$

```
struct Graph
{
  int n;
  vector<vector<int>> edge;//0-base
  vector<int> match, dis, ons, stk;
  Graph(int _n) : n(_n), edge(n, vector<int>(n, 0)), match(n){}
  void add_edge(int u, int v, int w){
    edge[u][v]=edge[v][u]=w;
  }
  bool SPFA(int u){
    if( ons[u] ) return true;
    stk.push_back(u); ons[u]=1;
    for(int v=0; v<n; v++){
      if( u!=v && match[u]!=v && !ons[v] ){
        int m=match[v];
        if( dis[m]>dis[u]-edge[v][m]+edge[u][v] ){
          dis[m]=dis[u]-edge[v][m]+edge[u][v];
          stk.push_back(v); ons[v]=1;
          if( SPFA(m) ) return true;
          stk.pop_back(); ons[v]=0;
        }
      }
    }
    stk.pop_back(); ons[u]=0;
    return false;
  }
  int operator()(){
    for (int i=0; i<n; i+=2)
      match[i]=i+1, match[i+1]=i;
    for(bool found=true; found; ){
      found=false;
      dis.assign(n, 0);
      ons.assign(n, 0);
      for(int i=0; i<n; i++){
        stk.clear();
        if( !ons[i] && SPFA(i) )
          for(found=true; stk.size()>=2; ){
            int u=stk.back(); stk.pop_back();
            int v=stk.back(); stk.pop_back();
            match[u]=v; match[v]=u;
          }
      }
    }
    int ans=0;
    for(int i=0; i<n; i++) ans+=edge[i][ match[i] ];
    return ans>>1;
  }
};
```

9 Maximum-Cardinality General Matching

$O(\sqrt{V}E)$

```
struct Graph{
  int n, st, ed, nb, ans=0;
  vector<vector<int>> edg;//1-base
  vector<int> pr, bk, ds;
  vector<bool> inq, inp, inb;
  queue<int> Q;
  Graph(int _n) : n(_n), edg(n+1), pr(n+1, 0), ds(n+1){}
  void add_edge(int u, int v){
    edg[u].push_back(v); edg[v].push_back(u);
  }
  int lca(int u, int v){
    inp.assign(n+1, false);
    for(u=ds[u]; ; u=ds[ bk[ pr[u] ] ]){
      inp[u]=true;
      if( u==st ) break;
    }
    for(v=ds[v]; !inp[v]; v=ds[ bk[ pr[v] ] ]);
    return v;
  }
  void upd(int u){
    while( ds[u]!=nb ){
      int v=pr[u];
      inb[ ds[u] ]=inb[ ds[v] ]=true;
      u=bk[v];
      if( ds[u]!=nb ) bk[u]=v;
    }
  }
  void blo(int u, int v){
    nb=lca(u, v);
    inb.assign(n+1, false);
    upd(u); upd(v);
    if( ds[u]!=nb ) bk[u]=v;
    if( ds[v]!=nb ) bk[v]=u;
    for(int tu=1; tu<=n; tu++) if( inb[ ds[tu] ] ){
      ds[tu]=nb;
      if( !inq[tu] ){ Q.push(tu); inq[tu]=true; }
    }
  }
  void flo()
  {
    bk.assign(n+1, 0);
    inq.assign(n+1, false);
    inq[st]=true;
    for(int i=1; i<=n; i++) ds[i]=i;
    for(ed=0; !Q.empty() ; Q.pop());
    for(Q.push(st); !Q.empty(); Q.pop()){
      int u=Q.front();
      for(int v : edg[u]) if( ds[u]!=ds[v] && pr[u]!=v ){
        if( v==st || pr[v]>0 && bk[ pr[v] ]>0 ) blo(u, v);
        else if( bk[v]==0 ){
          bk[v]=u;
          if( pr[v]<=0 ){ ed=v; return ;
```

```
          } else if( pr[v]>0 && !inq[ pr[v] ] ) Q.push(pr[v]);
        }
      }
    }
  }
  void aug(){
    for(int w, v, u=ed; u>0; u=w){
      v=bk[u]; w=pr[v];
      pr[v]=u; pr[u]=v;
    }
  }
  int operator()(){
    for(int u=1; u<=n; u++) if( pr[u]==0 )
    {
      st=u; flo();
      if( ed>0 ){
        aug(); ans++;
      }
    }
    return ans;
  }
};
```

10 Bron-Kerbosch Algorithm

Find all maximal cliques and store them to c
e is the adjacency matrix
R and X are initially empty while P is full
Runs in $O(3^{V/3})$

```
typedef bitset<N> set;
vector<set> c, e;
void BronKerbosch(set R, set P, set X){
  if( P.none() && X.none() ){
    c.push_back(R); return;
  }
  int u=0;
  for(; u<n && !(P|X)[u]; u++);
  set T(1);
  for(int i=0; i<n; i++, T<<=1){
    if( (P&~e[u])[i] ){
      BronKerbosch(R|T, P&e[i], X&e[i]);
      P[i]=false, X[i]=true;
    }
  }
}
```

11 Least Common Ancestor

```
#define edge pair<int,int>
#define v first
#define w second
struct lca{
  const int H=20;
  int n;
  vector<int> h;
  vector<vector<int>> p, b;
  lca(const vector<vector<edge>>& e) : n(e.size()-1),//one-base vertex index
    h(n+1, -1), p(H, vector<int>(n+1, -1)), b(H, vector<int>(n+1)){
    p[0][1]=1; b[0][1]=0; dfs(e, 1, 0);
    for(int i=1; i<H; i++) for(int j=1; j<=n; j++){
      p[i][j]=p[i-1][ p[i-1][j] ];
      b[i][j]=max(b[i-1][ p[i-1][j] ], b[i-1][j]);
      //b is something you want to calculate on the path from root
    }
  }
  void dfs(const vector<vector<edge>>& e, int u, int d=0){
    h[u]=++d;
    for(const edge& E : e[u]){
      if( h[E.v]>=0 ) continue;
      p[0][E.v]=u; b[0][E.v]=E.w;
      dfs(e, E.v, d);
    }
  }
  int operator()(int u, int v) const{
    if( h[u]>h[v] ) swap(u, v);
    int ans=0;
    for(int i=0, d=h[v]-h[u]; d>0; d>>=1, i++)
      if( d&1 ) ans=max(ans, b[i][v]), v=p[i][v];
    for(int i=0; u!=v; i++){
      for(; i>0 && p[i][u]==p[i][v]; i--);
      ans=max(ans, max(b[i][u], b[i][v]));
      u=p[i][u], v=p[i][v];
    }
    return ans;
  }
};
```

12 Stable Matching

$O(nm \log n)$

```
struct stable{
  int n, m, ans=0;//n: left size, m: right size
  vector<vector<int>> a;//left preference
  vector<int> b, c;//b: left match, c: right capacity
  vector<map<int, int>> M, PQ;//M: right preferencr, PQ: right match
  int operator()(){
    queue<int> Q;
    for(int i=1; i<=n; i++) Q.push(i);
    for(; !Q.empty(); Q.pop()){
      for(int u=Q.front(); !b[u] && !a[u].empty(); a[u].pop_back()){
        int v=a[u].back();
        if( (int)PQ[v].size()<c[v] ){
          PQ[v][ M[v][u] ]=u;
          b[u]=v;
          ans++;
        }
        else if( PQ[v].begin()->first<M[v][u] ){
          Q.push(PQ[v].begin()->second);
          b[PQ[v].begin()->second]=0;
          PQ[v].erase(PQ[v].begin());
          PQ[v][ M[v][u] ]=u;
          b[u]=v;
        }
      }
    }
    return ans;
  }
};
```

Topic IV

Data Structure and Others

1 Treap

```
struct treap{
  struct node{
    node *l=NULL, *r=NULL, *p=NULL;
    int pri=rand(), siz=1, key, val;
  };
  static void pull(node* p){}
  static void push(node* p){
    if( p!=NULL ){
      set_parent(p->l, p);
      set_parent(p->r, p);
    }
  }
  static void set_parent(node* x, node* p=NULL){
    if( x!=NULL ) x->p=p;
  }
  static node* merge(node* l, node* r){
    if( l==NULL )
      return r;
    else if( r==NULL )
      return l;
    else if( l->pri>r->pri ){
      push(l); l->r=merge(l->r, r);
      pull(l); return l;
    }
    else{
      push(r); r->l=merge(l, r->l);
      pull(r); return r;
    }
  }
  static void split(node* p, int key, node*& l, node*& r){
    push(p);
    if( p==NULL ) l=r=NULL;
    else if( p->key<=key ){
      l=p;
      split(l->r, key, l->r, r);
      pull(l);
    }
    else{
      r=p;
      split(r->l, key, l, r->l);
      pull(r);
    }
  }
};
```

2 2D Binary Indexed Tree

```
struct bit{
  int w, h;//1<=x<=w, 1<=y<=h
  vector<vector<int>> a;
  bit(int w, int h=1) : w(w), h(h), a(w+1, vector<int>(h+1, 0)){}
  void add(int x, int y=1, int val=1){
    for(; x<=w; x+=x&-x)
      for(int yy=y; yy<=h; yy+=yy&-yy)
        a[x][yy]+=val;
  }
  int query(int x, int y=1){
    int res=0;
    for(; x>0; x-=x&-x)
      for(int yy=y; yy>0; yy-=yy&-yy)
        res+=a[x][yy];
    return res;
  }
};
```

3 Sparse Table

```
const int N=10;
struct ST
{
  int n, m, a[N][N][1<<N][1<<N];
  void init(int _n, int _m){
    n=_n, m=_m;
    memset(a, 0, sizeof(a));
    for(int i=0; i<n; i++) for (int j=0; j<m; j++)
      scanf("%d", &a[0][0][i][j]);
    for(int u=1; u<=lg(n); u++){
      int u2=1<<u, u3=u2>>1;
      for(int i=0; i+u2<=n; i++) for(int j=0; j<m; j++)
        a[u][0][i][j]=max(a[u-1][0][i][j], a[u-1][0][i+u3][j]);
    }
    for(int u=0; u<=lg(n); u++) for(int v=1; v<=lg(m); v++){
      int u2=1<<u, v2=1<<v, v3=v2>>1;
      for(int i=0; i+u2<=n; i++) for(int j=0; j+v2<=m; j++)
        a[u][v][i][j]=max(a[u][v-1][i][j], a[u][v-1][i][j + v3]);
    }
  }
  int query(int l, int r, int u, int d){//[l, r)*[u, d)
    int x=lg(d-u), x2=1<<x, y=lg(r-l), y2=1<<y;
    int maxv=max(a[x][y][u][l], a[x][y][d-x2][l]);
    maxv=max(maxv, a[x][y][u][r-y2]);
    return max(maxv, a[x][y][d-x2][r-y2]);
  }
  static int lg(int x){
    return log2((double)x)+EPS;//return 30-__builtin_clrsb(x);
  }
} st;
```

4 Subset Sum

$O((\sum a_i)^{1.5})$

```
const int N=1<<20;//sum of all mumbers
struct SSS{
  int tot;
  bitset<N> ok;
  SSS(const vector<int>& a) : tot(0), ok(1){
    vector<int> c(N, 0);
    for(int x : a) tot+=x, c[x]++;
    for(int sum=0, i=1; i<=tot>>1; i++) if( c[i]>0 ){
      for(int m=min(tot>>1, sum+=i*c[i]), j=max(m-i+1, 1); j<=m; j++){
        int cnt=0;
        for(int k=0; k<c[i] && j>=i*k; k++) cnt+=ok[j-i*k];
        for(int k=j; k>0; k-=i){ cnt-=ok[k];
          if( k>=i*c[i] ) cnt+=ok[k-i*c[i]];
          if( cnt>0 ) ok[k]=true;
        }
      }
    }
  }
  bool operator[](int n) const{
    return n<0 || n>tot ? false : ok[min(n, tot-n)];
  }
};
```

5 Built-in Functions Provided by GCC

```
int __builtin_ffsll(lld x)
  //Returns one plus the index of the least significant 1-bit of x, or if x
    is zero, returns zero.

int __builtin_clzll(llu x)
  //Returns the number of leading 0-bits in x, starting at the most
    significant bit position. If x is 0, the result is undefined.

int __builtin_ctzll(llu x)
  //Returns the number of trailing 0-bits in x, starting at the least
    significant bit position. If x is 0, the result is undefined.

int __builtin_clrsbll(lld x)
  //Returns the number of leading redundant sign bits in x, i.e. the number
    of bits following the most significant bit that are identical to it.
    There are no special cases for 0 or other values.

int __builtin_popcountll(llu x)
  //Returns the number of 1-bits in x.
```

6 Link Cut Tree

```
struct Node{
  int sz, label; /* size, label */
  Node *p, *pp, *l, *r; /* parent, path-parent, left, right pointers */
  Node() { p = pp = l = r = 0; }
};
void update(Node *x){
  x->sz = 1;
  if (x->l) x->sz += x->l->sz;
  if (x->r) x->sz += x->r->sz;
}
void rotr(Node *x){
  Node *y, *z;
  y = x->p, z = y->p;
  if ((y->l = x->r)) y->l->p = y;
  x->r = y, y->p = x;
  if ((x->p = z)){
    if (y == z->l) z->l = x;
    else z->r = x;
  }
  x->pp = y->pp; y->pp = 0; update(y);
}
void rotl(Node *x){
  Node *y, *z;
  y = x->p, z = y->p;
  if ((y->r = x->l)) y->r->p = y;
  x->l = y, y->p = x;
  if ((x->p = z)){
    if (y == z->l) z->l = x;
    else z->r = x;
  }
  x->pp = y->pp; y->pp = 0; update(y);
}
void splay(Node *x){
  for(Node *y, *z; x->p; ){
    y = x->p;
    if (y->p == 0){
      if (x == y->l) rotr(x);
      else rotl(x);
    }
    else{
      z = y->p;
      if (y == z->l){
        if (x == y->l) rotr(y), rotr(x);
        else rotl(x), rotr(x);
      }
      else{
        if (x == y->r) rotl(y), rotl(x);
        else rotr(x), rotl(x);
      }
    }
  }
  update(x);
}
Node *access(Node *x){
  splay(x);
  if (x->r){
    x->r->pp = x; x->r->p = 0; x->r = 0; update(x);
  }
  Node *last = x;
  while (x->pp){
    Node *y = x->pp; last = y; splay(y);
    if (y->r){
      y->r->pp = y; y->r->p = 0;
    }
    y->r = x; x->p = y; x->pp = 0; update(y); splay(x);
  }
  return last;
}
Node *root(Node *x){
  access(x); while (x->l) x = x->l; splay(x); return x;
}
void cut(Node *x){
  access(x); x->l->p = 0; x->l = 0; update(x);
}
void link(Node *x, Node *y){
  access(x); access(y); x->l = y; y->p = x; update(x);
}
Node *lca(Node *x, Node *y){
  access(x); return access(y);
}
int depth(Node *x){
  access(x); return x->sz - 1;
}
struct LinkCut{
  Node *x;
  LinkCut(int n){
    x = new Node[n];
    for (int i = 0; i < n; i++){
      x[i].label = i;
      update(&x[i]);
    }
  }
  void link(int u, int v){
    ::link(&x[u], &x[v]);
  }
  void cut(int u){
    ::cut(&x[u]);
  }
  int root(int u){
    return ::root(&x[u])->label;
  }
  int depth(int u){
    return ::depth(&x[u]);
  }
  int lca(int u, int v){
    return ::lca(&x[u], &x[v])->label;
  }
};
```

```
const int MAXN = 30000;
```

```
template <typename T>
struct LinkCutTree {
  enum Relation {
    L = 0, R = 1
  };
  struct Node {
    Node *child[2], *parent, *pathParent;
    T value, sum, max;
    bool reversed;
    Node(const T &value) : reversed(false), value(value), sum(value), max(
     value), parent(NULL), pathParent(NULL) {
      child[L] = child[R] = NULL;
    }
    Relation relation() {
      return this == parent->child[L] ? L : R;
    }
    void pushDown() {
      if (reversed) {
        std::swap(child[L], child[R]);
        if (child[L]) child[L]->reversed ^= 1;
        if (child[R]) child[R]->reversed ^= 1;
        reversed = false;
      }
    }
    void maintain() {
      sum = value;
      if (child[L]) sum += child[L]->sum;
      if (child[R]) sum += child[R]->sum;

      max = value;
      if (child[L]) max = std::max(max, child[L]->max);
      if (child[R]) max = std::max(max, child[R]->max);
    }
    void rotate() {
      if (parent->parent) parent->parent->pushDown();
      parent->pushDown(), pushDown();
      std::swap(pathParent, parent->pathParent);

      Relation x = relation();
      Node *oldParent = parent;

      if (oldParent->parent) oldParent->parent->child[oldParent->relation()]
     = this;
      parent = oldParent->parent;

      oldParent->child[x] = child[x ^ 1];
      if (child[x ^ 1]) child[x ^ 1]->parent = oldParent;

      child[x ^ 1] = oldParent;
      oldParent->parent = this;

      oldParent->maintain(), maintain();
    }
    void splay() {
      while (parent) {
        if (!parent->parent) rotate();
        else {
          parent->parent->pushDown(), parent->pushDown();
          if (relation() == parent->relation()) parent->rotate(), rotate();
          else rotate(), rotate();
        }
      }
    }
    void evert() {
      access();
      splay();
      reversed ^= 1;
    }
    void expose() {
      splay();
      pushDown();
      if (child[R]) {
        child[R]->parent = NULL;
        child[R]->pathParent = this;
        child[R] = NULL;
        maintain();
      }
    }
    bool splice() {
      splay();
      if (!pathParent) return false;

      pathParent->expose();
      pathParent->child[R] = this;
      parent = pathParent;
      pathParent = NULL;
      parent->maintain();
      return true;
    }
    void access() {
      expose();
      while (splice());
    }
    const T &querySum() {
      access();
      splay();
      return sum;
    }
    const T &queryMax() {
      access();
      splay();
      return max;
    }
  };
  Node *nodes[MAXN];
  void makeTree(int u, const T &value) {
    nodes[u - 1] = new Node(value);
  }
  void link(int u, int v) {
    nodes[v - 1]->evert();
    nodes[v - 1]->pathParent = nodes[u - 1];
  }
```

```
  void cut(int u, int v) {
    nodes[u - 1]->evert();
    nodes[v - 1]->access();
    nodes[v - 1]->splay();
    nodes[v - 1]->pushDown();
    nodes[v - 1]->child[L]->parent = NULL;
    nodes[v - 1]->child[L] = NULL;
    nodes[v - 1]->maintain();
  }
  const T &querySum(int u, int v) {
    nodes[u - 1]->evert();
    return nodes[v - 1]->querySum();
  }
  const T &queryMax(int u, int v) {
    nodes[u - 1]->evert();
    return nodes[v - 1]->queryMax();
  }
  void update(int u, const T &value) {
    nodes[u - 1]->splay();
    nodes[u - 1]->value = value;
    nodes[u - 1]->maintain();
  }
};
```