

National Chiao Tung University

Team Reference Document

Contents

I Math

- 1 Pick's Theorem
- 2 Grey Code
- 3 Extended Euclidean
- 4 Chinese Remainder Theorem
- 5 Mod Equation
- 6 Quadratic residue
- 7 Rho Algorithm for Factorization
- 8 Discrete Logarithm
- 9 Möbius Function
- 10 Phi Function
- 11 Miller-Rabin Primality Test
- 12 Simplex
- 13 Determinant
- 14 Fast Fourier Transform
- 15 Polynomial Solver
- 16 2-SAT
- 17 SAT Solver
- 18 Point
- 19 Convex Hull
- 20 Minimum Enclosing Disc
- 21 Roman Numerals
- 22 Geometry from NTU

II String

- 1 KMP
- 2 Z-algorithm

- 3 Longest Palindromic Substring
- 4 Lexicographically Smallest Rotation
- 5 AC Trie
- 6 Suffix Array

1 III Graph

- 1 1 Biconnected Connected Component
- 1 2 Strongly Connected Component
- 1 3 Bellman-Ford Algorithm
- 1 4 Maximum Flow
- 1 5 Minimum-Cost Maximum Flow
- 2 6 Maximum-Weight Bipartite Perfect Matching
- 2 7 Maximum-Cardinality Bipartite Matching
- 2 8 Minimum-Weight General Perfect Matching
- 2 9 Maximum-Cardinality General Matching
- 2 10 Bron-Kerbosch Algorithm
- 2 11 Least Common Ancestor
- 3 12 Stable Matching

3 IV Data Structure and Others

- 4 1 Treap
- 4 2 2D Binary Indexed Tree
- 4 3 Sparse Table
- 5 4 Subset Sum
- 5 5 Built-in Functions Provided by GCC
- 5 6 Link Cut Tree

5 V Theorems

```

6   1 " system "
7   2 set nocompatible           " safty option "
8   3 set printoptions=number:y  " print with line number "
9   4 set autoread              " reload file when modified "
10  5
11  6 " tabs and indents "
12  7 set tabstop=4              " number of visual spaces per TAB "
13  8 set softtabstop=4          " number of spaces in tab when editing "
14  9 set shiftwidth=4           " number of spaces of indent "
15 10 set smartindent            " autoindent "

```

```

11 set smarttab           " autoindent "
12 filetype plugin indent on
13
14 " appearance "
15 syntax on              " syntax hilight "
16 set ruler               " show line and column number "
17 set number              " line number "
18 set showcmd             " show commands "
19 set showmatch            " blink matched brackets "
20 set cursorline          " hilight cursor line "
21 set cursorcolumn         " hilight cursor column "
22 highlight CursorLine ctermfg=235
23 highlight CursorColumn ctermfg=235
24 set background=dark
25
26 " search "
27 set incsearch            " search as charaters entered "
28 set hlsearch             " hilight matches "
29
30 " shortcut "
31 map Z gg=G
32 set backspace=indent,eol,start
33 set whichwrap=b,s,<,>,[,]

```

#pragma GCC optimize (2)

Topic I Math

1 Pick's Theorem

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

2 Grey Code

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

3 Extended Euclidean

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

```

1 lld ext_gcd(lld a, lld b, lld& x, lld& y){
2     if( b==0 ){
3         x=1, y=0;
4         return a;
5     }
6     lld g=ext_gcd(b, a%b, y, x);
7     y-=x*(a/b);
8     return g;
9 }

```

4 Chinese Remainder Theorem

Find x s.t. $x \equiv a_i \pmod{m_i}$

```

1 lld CRT(int n, lld a[], lld m[]){
2     for(int i=1; i<n; i++){
3         lld x, y, g=ext_gcd(m[i-1], m[i], x, y), r=a[i]-a[i-1];
4         if( r%g!=0 ) return -1;
5         lld t=m[i]/g;
6         x=(r/g*x%t+t)%t;
7         a[i]=a[i-1]+[i-1]*x;
8         m[i]=m[i-1]*t;
9     }
10    return a[n-1];
11 }

```

5 Mod Equation

Find x s.t. $ax \equiv b \pmod{n}$

```

1 vector<lld> mod_eq(lld a, lld b, lld n){
2     lld x, y, g=ext_gcd(a, n, x, y);
3     vector<lld> ans;
4     if( b%g==0 ){
5         x=((x%n)+n)%n;
6         ans.push_back(x*(b/g)%(n/g));
7         for(lld i=1; i<g; i++) ans.push_back((ans[0]+i*n/g)%n);
8     }
9     return ans;
10 }

```

6 Quadratic residue

Find x s.t. $x^2 \equiv a \pmod{p}$

```

1 lld solve(lld a, lld p){
2     if( a%p==0 ) return 0;
3     lld q=p-1, z=1, s=0;
4     for(; q>0 && !(q&1); q>>=1) s++;
5     for(;; pmd(z, p>>1, p); z=rnd(1, p));//rnd(1, r): [l, r]
6     lld c=pmd(z, q, p), t=pmd(a, q, p), r=pmd(a, (q+1)>>1, p);
7     for(lld i=1; t!=1; i=1){
8         for(lld tt=t*t%p; i<s && tt!=1; i++, tt=tt*tt%p);
9         if( i==s ) return -1;
10        lld b=pmd(c, pmd(2, s-i-1, p), p);
11        s=i; c=b*b%p; t=t*c%p; r=r*b%p;
12    }
13    return r;
14 }
```

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

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

then

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

```

1 int mu[n]={0};
2 void get_mu(){
3     mu[1]=-1;
4     for(int i=1; i<n; i++){
5         mu[i]=-mu[i];
6         for(int j=i<<1; j<n; j+=i) mu[j]+=mu[i];
7     }
8 }
```

7 Rho Algorithm for Factorization

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

```

1 inline lld f(lld x, lld n) {
2     return (x*x+1)%n;
3 }
4 lld rho_fact(lld n) {
5     for(lld x=2, y=2; ; ){
6         x=f(x, n), y=f(f(y, n), n);
7         lld d=gcd(abs(x-y), n);
8         if( d>1 ) return d==n ? -1 : d ;
9     }
10 }
```

8 Discrete Logarithm

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

```

1 lld solve(lld a, lld b, lld p){
2     int sp=ceil(sqrt(p));
3     map<lld, int> M;
4     lld tmp=1;
5     for(int i=0; i<sp && M.find(tmp)==M.end(); i++)
6         M[tmp]=i, tmp=tmp*a%p;
7     lld x, y;
8     ext_gcd(tmp, p, x, y);
9     tmp=(x+p)%p;
10    for(int i=0; i<sp; i++){
11        auto res = M.find(b);
12        if( res!=M.end() ) return i*(lld)sp+res->second;
13        b=b*tmp%p;
14    }
15    return -1;
16 }
```

10 Phi Function

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

```

1 int phi[n], minDiv[n];
2 void get_phi(){
3     phi[1] = 1;
4     for(int i=2; i<n; i++) minDiv[i]=i;
5     for(lld i=2; i<n; i++) if( minDiv[i]==i )
6         for(lld j=i*i; j<n; j+=i) minDiv[j]=i;
7     for(int i=2; i<n; i++){
8         phi[i]=phi[ i/minDiv[i] ];
9         phi[i]*=minDiv[i]-(i/minDiv[i])%minDiv[i]==0 ? 0 : 1;
10    }
11 }
```

9 Möbius Function

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

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

11 Miller-Rabin Primality Test

```

1 bool test(lld n, lld a, lld d){
2     if( n==a ) return true;
3     while( !(d&1) ) d>>=1;
4     lld t=power_mod(a, d, n); //a^d(mod n)
5     while( d!=n-1 && t!=1 && t!=n-1 )
6         t=t*t%n, d<<=1;
7     return t==n-1 || d&1;
8 }
9 bool is_prime(lld n){
10    if( n<2 ) return false;
11    if( n<4 ) return true;
12    if( !(n&1) ) return false;
13    //int a[3]={2, 7, 61}; //for int n
14    int a[12]={2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
15    for(int i=0; i<12; i++)
16        if( !test(n, a[i], n-1) ) return false;
17    return true;
18 }
```

12 Simplex

0-based index

Find $\max\{cx\}$ subjected to $ax \leq b \wedge x \geq 0$

n : constraints, m : variables

```

1 const double eps = 1E-10;
2 double a[maxn][maxm], b[maxn], c[maxm], d[maxn][maxm], x[maxn];
3 int ix[maxn + maxm];
4 double simplex(int n, int m) {
5     ++m;
6     int r = n, s = m - 1;
7     memset(d, 0, sizeof(d));
8     for (int i = 0; i < n + m; ++i) ix[i] = i;
9     for (int i = 0; i < n; ++i) {
10        for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
11        d[i][m - 1] = 1; d[i][m] = b[i];
12        if (d[r][m] > d[i][m]) r = i;
13    }
14    for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
15    d[n + 1][m - 1] = -1;
16    for (double dd;; ) {
17        if (r < n) {
18            int t = ix[s];
19            ix[s] = ix[r + m]; ix[r + m] = t;
20            d[r][s] = 1.0 / d[r][s];
21            for (int j = 0; j <= m; ++j) if (j != s) d[r][j] *= -d[r][s];
22            for (int i = 0; i <= n + 1; ++i) if (i != r) {
23                for (int j = 0; j <= m; ++j)
24                    if (j != s) d[i][j] += d[r][j]*d[i][s];
25                d[i][s] *= d[r][s];
26            }
27        }
28        r = -1; s = -1;
29        for (int j = 0; j < m; ++j) if (s < 0 || ix[s] > ix[j])
30            if(d[n + 1][j] > eps || (d[n + 1][j] > -eps && d[n][j] > eps)) s = j;
31        if (s < 0) break;
32        for (int i=0; i<n; ++i) if (d[i][s] < -eps)
33            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;
34        if (r < 0) return -1; // not bounded
35    }
36    if (d[n + 1][m] < -eps) return -1; // not executable
37    double ans = 0;
38    for (int i = 0; i < m; ++i) x[i] = 0;
39    for (int i = m; i < n + m; ++i) { // the missing enumerated x[i] = 0
40        if (ix[i] < m - 1) {
41            ans += d[i - m][m] * c[ix[i]];
42            x[ix[i]] = d[i - m][m];
43        }
44    }
45    return ans;
46 }
```

13 Determinant

```

1 lld det(int n, lld m, vector<vector<lld>> a){
2     lld div=1;
3     for(int k=n-1; k>0; k--){
4         if( a[k][k]==0 ) for(int i=0; i<k; i++) if( a[i][k]!=0 )
5             swap(a[i], a[k]), div=m-div;
6         for(int i=0; i<k; i++) for(int j=0; j<k; j++)
7             a[i][j]=((a[i][j]*a[k][k]-a[i][k]*a[k][j])%m+m)%m;
8         div=div*pmd(a[k][k], k-1, m)%m;
9     }
10    return a[0][0]*inv(div, m)%m;
11 }
```

14 Fast Fourier Transform

```

1 typedef complex<double> cplx;
2 void fft(cplx A[], int lgn, bool inv=false){
3     int n=1<<lgn;
4     for(int i=0, j=1; j<n-1; j++){
5         for(int k=n>>1; k>(i^k); k>>=1);
6         if( j<i ) swap(A[i], A[j]);
7     }
8     for(int i=1; i<n; i<<=1){
9         cplx W(1, 0), Wn(cos(PI/i), sin((inv ? -PI : PI)/i));
10        for(int j=0; j<n; j++){
11            if( j&i ){
12                W=cplx(1, 0); continue;
13            }
14            cplx x=A[j], y=A[j+i]*W;
15            A[j]=x+y; A[j+i]=x-y;
16            W*=Wn;
17        }
18    }
19    if( inv ) for(int i=0; i<n; i++) A[i]/=n;
20 }
```

15 Polynomial Solver

$$p(x) = \sum_{i=0}^n \text{coef}_i \cdot x^i$$

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

```

1 const double EPS=1e-12, INF=1e12;
2 int sign(double x){
3     return x<-EPS ? -1 : x>EPS ;
4 }
5 double get(const vector<double>& coef, double x){
6     double ans=0;
7     for(int i=coef.size()-1; i>=0; i--)
8         ans=ans*x+coef[i];
9     return ans;
10}
11 double find(const vector<double>& coef, int n, double lo, double hi){
12     int sign_lo=sign(get(coef, lo));
13     int sign_hi=sign(get(coef, hi));
14     double m=INF;
15     if( sign_lo==0 ) return lo;
16     if( sign_hi==0 ) return hi;
17     if( sign_lo==sign_hi ) return INF;
18     for(int step=0; step<100 && hi-lo>EPS; step++){
19         m=(hi+lo)*.5;
20         int sign_m=sign(get(coef, m));
21         if( sign_m==0 ) return m;
22         (sign_m*sign_lo<0 ? hi : lo)=m;
23     }
24     return m;
25 }
26 vector<double> equation(vector<double>& coef, int n){
27     vector<double> ans;
28     if( n==1 )
29         return sign(coef[1]) ? vector<double>(1, -coef[0]/coef[1]) : ans ;
30     if( !sign(coef[n]) )
31         return equation(coef, n-1);
32     if( sign(coef[n])<0 )
33         for(int i=0; i<n; i++) coef[i]=-coef[i];
34     vector<double> dcoef(n);
35     for(int i=1; i<n; i++) dcoef[i-1]=coef[i]*i;
36     vector<double> droot=equation(dcoef, n-1);
37     droot.insert(droot.begin(), n&1 ? -INF : INF);
38     droot.insert(droot.end(), INF);
39     for(int i=1; i<droot.size(); i++){
40         double tmp=find(coef, n, droot[i-1], droot[i]);
41         if( tmp<INF ) ans.push_back(tmp);
42     }
43     return ans;
44 }
```

16 2-SAT

n : number of vars

```

1 struct SAT2{
2     int n; SCC S;//0-base
3     vector<int> x;
4     SAT2(int _n) : n(_n), S(n<1), x(n+1, 0){}
5     int trans(int a){ return abs(a)-1<1?a<0: 1 }
6     int rev(int a){ return (a>>1)+1; }
7     void add_or(int a, int b){
8         a=trans(a), b=trans(b);
9         if( a>>1!=b>>1 ){
10             S.add(a^1, b); S.add(b^1, a);
11         }
12         else if( a==b ) S.add(a^1, a);
13     }
14     void add_xor(int a, int b){
15         add_or(a, b); add_or(-a, -b);
16     }
17     bool operator()(){
18         S();
19         for(int i=0; i<n<1; i+=2) if( S.com[i]==S.com[i^1] ) return false;
20         vector<vector<int>> edg(n<1), var(n<1); //generate one solution
21         vector<bool> flg(n<1, true);
22         vector<int> deg(n<1, 0);
23         queue<int> Q;
24         for(int u=0; u<n<1; u++){
25             var[S.com[u]-1].push_back(u);
26             for(int v : S.edg[u]) if( S.com[u]!=S.com[v] ){
27                 edg[S.com[v]-1].push_back(S.com[u]-1);
28                 deg[S.com[u]-1]++;
29             }
30         }
31         for(int i=0; i<n<1; i++) if( deg[i]==0 ) Q.push(i);
32         for(;; !Q.empty()) Q.pop(){
33             int u=Q.front();
34             for(int v : var[u]) if( (v&1 ? 1 : -1)*x[rev(v)]>0 ) flg[u]=false;
35             for(int v : var[u]) x[rev(v)]=(v^flg[u])&1 ? 1 : -1 ;
36             for(int v : edg[u]){
37                 flg[v]=flg[v]&&flg[u]; if( --deg[v]==0 ) Q.push(v);
38             }
39         }
40         return true;
41     }
42 }
```

17 SAT Solver

n : number of vars

```

1 struct SAT
2 {
3     int n, m, rem, level;
4     vector<vector<int>> C, CNT;//0: false, 1: true, 2: unassigned
5     vector<int> UNIT, X, STAT;//0: unassigned, 1: decided, 2: implied
6     vector<vector<vector<int>>> POS;//0: false, 1: true
7     vector<vector<int>> I, ORDER;
8     SAT(int _n, vector<vector<int>>& cnf) : n(_n), level(0), C(cnf), I(1){
9         m=C.size(), rem=n;
10        for(vector<int> &c : C) CNT.push_back({ 0, 0, (int)c.size() });
11        X.resize(n + 1);
12        STAT.assign(n + 1, { {}, {} });
13        POS.assign(n + 1, { {}, {} });
14        for(int i=0; i<n; i++) for(int x : C[i])
15            POS[abs(x)][x < 0 ? 0 : 1].push_back(i);
16        for(vector<int> &c : C) if( c.size()==1 )
17            UNIT.push_back(c[0]);
18        for(int i=1; i<n; i++)
19            ORDER.push_back({ (int)(POS[i][0].size() + POS[i][1].size()), i });
20        sort(ORDER.rbegin(), ORDER rend());
21    }
```

```

22  bool assign(int x, int val, int stat){//assumption: x is unassigned
23    rem--, X[x]=val, STAT[x]=stat;
24    bool res=true;
25    for(int i=0; i<2; i++) for(int c : POS[x][i]){
26      CNT[c][2]--, CNT[c][val==i]++;
27      if( CNT[c][2]==1 && CNT[c][1]==0 )
28        for(int y : C[c])
29          if( STAT[abs(y)]==0 ){
30            UNIT.push_back(y);
31            break;
32          }
33      res&= !(CNT[c][1]==0 && CNT[c][2]==0);
34    }
35    return res;
36  }
37 void unassign(int x){//assumption: x is assigned
38    rem++, STAT[x]=0;
39    for(int i=0; i<2; i++)
40      for(int c : POS[x][i])
41        CNT[c][2]++, CNT[c][X[x]==i]--;
42  }
43 bool imply(){
44  while( !UNIT.empty() ){
45    int x=UNIT.back(), val=x>0;
46    UNIT.pop_back();
47    if( STAT[abs(x)]!=0 && X[abs(x)]==val ) continue;
48    if( STAT[abs(x)]!=0 && X[abs(x)]!=val ) return false;
49    I.back().push_back(abs(x));
50    if( !assign(abs(x), val, 2) ) return false;
51  }
52  return true;
53 }
54 void unimply(){
55  for(int x : I.back()) unassign(x);
56  I.back().clear();
57 }
58 bool sol(){
59  if( !imply() ) return false;
60  if( rem==0 ) return true;
61  int x;
62  for(int i=0; STAT[ x=ORDER[i][1] ]; i++);
63  level++, I.push_back({});
64  assign(x, rand()&1, 1);
65  if( sol() ) return true;
66  unassign(x), unimply(), UNIT.clear();
67  assign(x, X[x]^1, 1);
68  if( sol() ) return true;
69  unassign(x), unimply();
70  level--, I.pop_back();
71  return false;
72 }
73 }

```

18 Point

```

1 struct point{
2   double x,y;
3   point(double _x=0, double _y=0) : x(_x), y(_y){}
4   point operator+(const point& p) const{//vector sum
5     return point(x+p.x, y+p.y);
6   }
7   point operator-(const point& p) const{//vector difference
8     return point(x-p.x, y-p.y);
9   }
10  point operator*(double s) const{//vector scaling
11    return point(x*s, y*s);
12  }
13  point operator/(double f) const{//vector scaling
14    return point(x/f, y/f);
15  }
16  double operator^(const point& P) const{//cross
17    return x*P.y-y*P.x;
18  }

```

```

19  double operator^(const point& P) const{//dot
20    return x*P.x+y*P.y;
21  }
22  double operator()() const{//square of length
23    return x*x+y*y;
24  }
25  bool operator<(const point& P) const{
26    return x==P.x ? y<P.y : x<P.x ;
27  }
28 }

```

19 Convex Hull

```

1 vector<point> convex_hull(vector<point> p){
2   int n=p.size(), k=0;
3   vector<point> h(n<1);
4   sort(p.begin(), p.end());
5   for (int i=0; i<n; i++){
6     for( k=2 && (h[k-1]-h[k-2])^(p[i]-h[k-1])<=0; k--);
7     h[k++]=p[i];
8   }
9   for (int i=n-2, t=k; i>=0; i--){
10    for( k>t && (h[k-1]-h[k-2])^(p[i]-h[k-1])<=0; k--);
11    h[k++]=p[i];
12  }
13  h.resize(k);
14  return h;
15 }

```

20 Minimum Enclosing Disc

```

1 struct circle{
2   point c; double r;
3   circle(const point& _c, double _r) : c(_c), r(_r){}
4   circle(const point& p, const point& q) : c((p+q)*0.5), r((c-p)*(c-p)){}
5   circle(const point& A, const point& B, const point& C){
6     point a=B-A, b=C-A;
7     double c1=a()*0.5, c2=b()*b()*0.5, d=a^b;
8     double x=A.x+(c1*b.y-c2*a.y)/d, y=A.y+(a.x*c2-b.x*c1)/d;
9     c=point(x, y); r=(c-A)*(c-A)();
10  }
11  bool in(const point& p) const{
12    return (p-c)*(p-c())<=r+EPS;
13  }
14 };
15 circle solve(vector<point> p){
16  random_shuffle(p.begin(), p.end());
17  circle ans(point(), 0);
18  for(int n=p.size(), i=0; i<n; i++){
19    if( ans.in(p[i]) ) continue;
20    ans=circle(p[i], 0);
21    for(int j=0; j<i; j++){
22      if( ans.in(p[j]) ) continue;
23      ans=circle(p[i], p[j]);
24      for(int k=0; k<j; k++){
25        if( ans.in(p[k]) ) continue;
26        ans=circle(p[i], p[j], p[k]);
27      }
28    }
29  }
30  ans.r=sqrt(ans.r);
31  return ans;
32 }

```

21 Roman Numerals

```

1 const int num[13]={1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
2 const string str[13]={"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"};
3 string rome(int x){
4     for(int i=0; i<13; i++) if( x>=num[i] ) return str[i]+rome(x-num[i]);
5     return "";
6 }
7 int rome(const string& s){
8     for(int i=0; i<13; i++)
9         if( s.length()>=str[i].length() && s.substr(0, str[i].length())==str[i].substr(0, str[i].length()) )
10            return num[i]+rome(s.substr(str[i].length()));
11    return 0;
12 }
```

22 Geometry from NTU

```

1 #define x first
2 #define y second
3 #define cpdd const pdd
4 struct pdd : pair<double, double>::pair {
5     using pair<double, double>::pair;
6     pdd operator + (cpdd &p) const {
7         return {x+p.x, y+p.y};
8     }
9     pdd operator - () const {
10        return {-x, -y};
11    }
12    pdd operator - (cpdd &p) const {
13        return (*this) + (-p);
14    }
15    pdd operator * (double f) const {
16        return {f*x, f*y};
17    }
18    double operator * (cpdd &p) const {
19        return x*p.x + y*p.y;
20    }
21 };
22 double abs(cpdd &p) { return hypot(p.x, p.y); }
23 double arg(cpdd &p) { return atan2(p.y, p.x); }
24 double cross(cpdd &p, cpdd &q) { return p.x*q.y - p.y*q.x; }
25 double cross(cpdd &p, cpdd &q, cpdd &o) { return cross(p-o, q-o); }
26 pdd operator * (double f, cpdd &p) { return p*f; } // !! Not f*p !!!
```

```

1 using ld = double;
2 vector<pdd> interCircle(pdd o1, double r1, pdd o2, double r2) {
3     ld d2 = (o1 - o2) * (o1 - o2);
4     ld d = sqrt(d2);
5     if (d < abs(r1-r2)) return {};
6     if (d > r1+r2) return {};
7     pdd u = 0.5*(o1+o2) + ((r2*r2-r1*r1)/(2*d2))*(o1-o2);
8     double A = sqrt((r1+r2+d) * (r1-r2-d) * (r1+r2-d) * (-r1+r2+d));
9     pdd v = A / (2*d2) * pdd(o1.S-o2.S, -o1.F+o2.F);
10    return {u+v, u-v};
11 }
12 //
13 pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2, bool &res){
14     double f1 = cross(p2, q1, p1);
15     double f2 = -cross(p2, q2, p1);
16     double f = (f1 + f2);
17     if(fabs(f) < EPS) {
18         res = false; return {};
19     }
20     res = true;
21     return (f2 / f) * q1 + (f1 / f) * q2;
22 }
```

```

1 const double EPS = 1e-9;
2 pdd interPnt(Line l1, Line l2, bool &res){
3     pdd p1, p2, q1, q2;
4     tie(p1, p2) = l1;
5     tie(q1, q2) = l2;
```

```

6     double f1 = cross(p2, q1, p1), f2 = -cross(p2, q2, p1), f = (f1 + f2);
7     if(fabs(f) < EPS) {
8         res=false; return {0, 0};
9     }
10    res = true;
11    return (f2 / f) * q1 + (f1 / f) * q2;
12 }
13 bool isin(Line l0, Line l1, Line l2) {// Check inter(l1, l2) in l0
14     bool res;
15     pdd p = interPnt(l1, l2, res);
16     return cross(l0.S, p, l0.F) > EPS;
17 }
18 /* If no solution, check: 1. ret.size() < 3
19  * Or more precisely, 2. interPnt(ret[0], ret[1])
20  * in all the lines. (use (l.S - l.F).cross(p - l.F) > 0 */
21 vector<Line> halfPlaneInter(vector<Line> lines) {
22     int sz = lines.size();
23     vector<double> ata(sz), ord(sz);
24     for (int i=0; i<sz; i++) {
25         ord[i] = i;
26         pdd d = lines[i].S - lines[i].F;
27         ata[i] = atan2(d.y, d.x);
28     }
29     sort(ALL(ord), [&](int i, int j) {
30         return (abs(ata[i] - ata[j]) < EPS) ? cross(lines[i].S, lines[j].S, lines[i].F) < 0 : ata[i] < ata[j];
31     });
32     vector<Line> fin;
33     for (int i=0; i<sz; i++) {
34         if (!i or fabs(ata[ord[i]] - ata[ord[i-1]]) > EPS)
35             fin.PB(lines[ord[i]]);
36     deque<Line> dq;
37     for (int i=0; i<SZ(fin); i++) {
38         while(SZ(dq) >= 2 and not isin(fin[i], dq[SZ(dq)-2], dq[SZ(dq)-1])) dq.pop_back();
39         while(SZ(dq) >= 2 and not isin(fin[i], dq[0], dq[1])) dq.pop_front();
40         dq.push_back(fin[i]);
41     }
42     while (SZ(dq) >= 3 and not isin(dq[0], dq[SZ(dq)-2], dq[SZ(dq)-1])) dq.pop_back();
43     while (SZ(dq) >= 3 and not isin(dq[SZ(dq)-1], dq[0], dq[1])) dq.pop_front();
44     return vector<Line>(ALL(dq));
45 }
```

```

1 /* Delaunay Triangulation:
2  Given a sets of points on 2D plane, find a
3  triangulation such that no points will strictly
4  inside circumcircle of any triangle.
5  find : return a triangle contain given point
6  add_point : add a point into triangulation
7  A Triangle is in triangulation iff. its has_chd is 0.
8  Region of triangle u: iterate each u.edge[i].tri,
9  each points are u.p[(i+1)%3], u.p[(i+2)%3]
10 calculation involves O(|V|6) */
11 const int N = 100000 + 5;
12 const type inf = 2e3;
13 type eps = 1e-6; // 0 when integer
14 type sqr(type x) { return x*x; }
15 // return p4 is in circumcircle of tri(p1,p2,p3)
16 bool in_cc(const Pt& p1, const Pt& p2, const Pt& p3, const Pt& p4){
17     type u11 = p1.X - p4.X; type u12 = p1.Y - p4.Y;
18     type u21 = p2.X - p4.X; type u22 = p2.Y - p4.Y;
19     type u31 = p3.X - p4.X; type u32 = p3.Y - p4.Y;
20     type u13 = sqr(p1.X)-sqr(p4.X)+sqr(p1.Y)-sqr(p4.Y);
21     type u23 = sqr(p2.X)-sqr(p4.X)+sqr(p2.Y)-sqr(p4.Y);
22     type u33 = sqr(p3.X)-sqr(p4.X)+sqr(p3.Y)-sqr(p4.Y);
23     type det = -u13*u22*u31 + u12*u23*u31 + u13*u21*u32
24     -u11*u23*u32 - u12*u21*u33 + u11*u22*u33;
25     return det > eps;
26 }
27 type side(const Pt& a, const Pt& b, const Pt& p)
28 { return (b - a) ^ (p - a); }
29 typedef int SdRef;
30 struct Tri;
31 typedef Tri* TriRef;
32 struct Edge {
33     TriRef tri; SdRef side;
34     Edge():tri(0), side(0){}
```

```

35   Edge(TriRef _tri, SdRef _side):tri(_tri), side(_side){}
36 };
37 struct Tri {
38   Pt p[3]; Edge edge[3]; TriRef chd[3]; Tri() {}
39   Tri(const Pt& p0, const Pt& p1, const Pt& p2) {
40     p[0] = p0; p[1] = p1; p[2] = p2;
41     chd[0] = chd[1] = chd[2] = 0;
42   }
43   bool has_chd() const { return chd[0] != 0; }
44   int num_chd() const { return chd[0] == 0 ? 0 : chd[1] == 0 ? 1 : chd[2] == 0 ? 2 : 3; }
45   bool contains(Pt const& q) const {
46     for( int i = 0 ; i < 3 ; i ++ )
47       if( side(p[i], p[(i + 1) % 3] , q) < -eps ) return false;
48   }
49   return true;
50 } pool[ N * 10 ], *tris;
51 void edge( Edge a, Edge b ){
52   if(a.tri) a.tri->edge[a.side] = b;
53   if(b.tri) b.tri->edge[b.side] = a;
54 }
55 struct Trig { // Triangulation
56   Trig() { // Tri should at least contain all points
57     the_root = new(tris++) Tri(Pt(-inf,-inf),Pt(+inf+inf,-inf),Pt(-inf,+inf+inf));
58   }
59   TriRef find(Pt p) const{ return find(the_root,p); }
60   void add_point(const Pt& p){ add_point(find(the_root,p),p); }
61   TriRef the_root;
62   static TriRef find(TriRef root, const Pt& p) {
63     while( true ){
64       if( !root->has_chd() ) return root;
65       for( int i = 0; i < 3 && root->chd[i] ; ++i )
66         if( root->chd[i]->contains(p) ) {
67           root = root->chd[i]; break;
68         }
69     }
70     assert( false ); // "point not found"
71   }
72   void add_point(TriRef root, Pt const& p) {
73     TriRef tab,tbc,tca;
74     /* split it into three triangles */
75     tab=new(tris++) Tri(root->p[0],root->p[1],p);
76     tbc=new(tris++) Tri(root->p[1],root->p[2],p);
77     tca=new(tris++) Tri(root->p[2],root->p[0],p);
78     edge(Edge(tab,0), Edge(tbc,1));
79     edge(Edge(tbc,0), Edge(tca,1));
80     edge(Edge(tca,0), Edge(tab,1));
81     edge(Edge(tab,2), root->edge[2]);
82     edge(Edge(tbc,2), root->edge[0]);
83     edge(Edge(tca,2), root->edge[1]);
84     root->chd[0] = tab; root->chd[1] = tbc; root->chd[2] = tca;
85     flip(tab,2); flip(tbc,2); flip(tca,2);
86   }
87   void flip(TriRef tri, SdRef pi) {
88     TriRef trj = tri->edge[pi].tri;
89     int pj = tri->edge[pi].side;
90     if( !trj ) return;
91     if( !in_cc(tri->p[0],tri->p[1],tri->p[2],trj->p[pj])) return;
92     /* flip edge between tri,trj */
93     TriRef trk = new(tris++) Tri(tri->p[(pi+1)%3], trj->p[pj], tri->p[pi]);
94     TriRef trl = new(tris++) Tri(trj->p[(pj+1)%3], tri->p[pi], trj->p[pj]);
95     edge(Edge(trk,0), Edge(trl,0));
96     edge(Edge(trk,1), tri->edge[(pi+2)%3]);
97     edge(Edge(trk,2), trj->edge[(pj+1)%3]);
98     edge(Edge(trl,1), trj->edge[(pj+2)%3]);
99     edge(Edge(trl,2), tri->edge[(pi+1)%3]);
100    tri->chd[0]=trk; tri->chd[1]=trl; tri->chd[2]=0;
101    trj->chd[0]=trk; trj->chd[1]=trl; trj->chd[2]=0;
102    flip(trk,1); flip(trk,2);
103    flip(trl,1); flip(trl,2);
104  }
105 };
106 vector<TriRef> triang;
107 set<TriRef> vst;
108 void go( TriRef now ) {
109   if( vst.find( now ) != vst.end() ) return;
110   vst.insert( now );

```

```

111   if( !now->has_chd() ){
112     triang.push_back( now );
113     return;
114   }
115   for( int i = 0 ; i < now->num_chd() ; i ++ ) go( now->chd[ i ] );
116 }
117 void build( int n , Pt* ps ){
118   tris = pool;
119   random_shuffle(ps, ps + n);
120   Trig tri;
121   for( int i = 0 ; i < n ; ++ i ) tri.add_point(ps[i]);
122   go( tri.the_root );
123 }

// Pt : { x , y , z }
#define N 202020
int n, nouter; Pt pt[ N ], outer[4], res;
double radius,tmp;
void ball(){
  Pt q[3]; double m[3][3], sol[3], L[3], det;
  int i,j; res.x = res.y = res.z = radius = 0;
  switch ( nouter ) {
  case 1: res=outer[0]; break;
  case 2: res=(outer[0]+outer[1])/2; radius=norm2(res, outer[0]); break;
  case 3:
    for (i=0; i<2; ++i) q[i]=outer[i+1]-outer[0];
    for (i=0; i<2; ++i) for(j=0; j<2; ++j) m[i][j]=(q[i] * q[j])*2;
    for (i=0; i<2; ++i) sol[i]=(q[i] * q[i]);
    if( fabs(det=m[0][0]*m[1][1]-m[0][1]*m[1][0])<eps ) return;
    L[0]=(sol[0]*m[1][1]-sol[1]*m[0][1])/det;
    L[1]=(sol[1]*m[0][0]-sol[0]*m[1][0])/det;
    res=outer[0]+q[0]*L[0]+q[1]*L[1];
    radius=norm2(res, outer[0]);
    break;
  case 4:
    for (i=0; i<3; ++i) q[i]=outer[i+1]-outer[0], sol[i]=(q[i] * q[i]);
    for (i=0; i<3; ++i) for(j=0; j<3; ++j) m[i][j]=(q[i] * q[j])*2;
    det= m[0][0]*m[1][1]*m[2][2]
      + m[0][1]*m[1][2]*m[2][0]
      + m[0][2]*m[2][1]*m[1][0]
      - m[0][2]*m[1][1]*m[2][0]
      - m[0][1]*m[1][0]*m[2][2]
      - m[0][0]*m[1][2]*m[2][1];
    if( ( fabs(det)<eps ) ) return;
    for (j=0; j<3; ++j) {
      for (i=0; i<3; ++i) m[i][j]=sol[i];
      L[j]=( m[0][0]*m[1][1]*m[2][2]
        + m[0][1]*m[1][2]*m[2][0]
        + m[0][2]*m[2][1]*m[1][0]
        - m[0][2]*m[1][1]*m[2][0]
        - m[0][1]*m[1][0]*m[2][2]
        - m[0][0]*m[1][2]*m[2][1] ) / det;
      for (i=0; i<3; ++i) m[i][j]=(q[i] * q[j])*2;
    }
    res=outer[0];
    for (i=0; i<3; ++i) res = res + q[i] * L[i];
    radius=norm2(res, outer[0]);
  }
  void minball(int n){ ball();
    if( nouter < 4 ) for( int i = 0 ; i < n ; i ++ )
      if( norm2(res, pt[i]) - radius > eps ){
        outer[ nouter ++ ] = pt[ i ]; minball(i); --nouter;
        if(i>0){ Pt Tt = pt[i];
          memmove(&pt[1], &pt[0], sizeof(Pt)*i); pt[0]=Tt;
        }
      }
  void solve{
    // n points in pt
    random_shuffle(pt, pt+n); radius=-1;
    for(int i=0;i<n;i++) if(norm2(res,pt[i])-radius>eps)
      nouter=1, outer[0]=pt[i], minball(i);
    printf("%.5f\n",sqrt(radius));
  }
}

const int MXN = 100005;

```

```

2 struct KDTree {
3     struct Node {
4         int x,y,x1,y1,x2,y2,id,f;
5         Node *L, *R;
6     }tree[MXN];
7     int n;
8     Node *root;
9     LL dis2(int x1, int y1, int x2, int y2) {
10    LL dx = x1-x2, dy = y1-y2;
11    return dx*dx+dy*dy;
12 }
13 static bool cmpx(Node& a, Node& b){ return a.x<b.x; }
14 static bool cmpy(Node& a, Node& b){ return a.y<b.y; }
15 void init(vector<pair<int,int>> ip) {
16     n = ip.size();
17     for (int i=0; i<n; i++) {
18         tree[i].id = i;
19         tree[i].x = ip[i].first; tree[i].y = ip[i].second;
20     }
21     root = build_tree(0, n-1, 0);
22 }
23 Node* build_tree(int L, int R, int dep) {
24     if (L>R) return nullptr;
25     int M = (L+R)/2;
26     tree[M].f = dep%2;
27     nth_element(tree+L, tree+M, tree+R+1, tree[M].f ? cmpy : cmpx);
28     tree[M].x1 = tree[M].x2 = tree[M].x;
29     tree[M].y1 = tree[M].y2 = tree[M].y;
30     tree[M].L = build_tree(L, M-1, dep+1);
31     if (tree[M].L) {
32         tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
33         tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
34         tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
35         tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
36     }
37     tree[M].R = build_tree(M+1, R, dep+1);
38     if (tree[M].R) {
39         tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
40         tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
41         tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
42         tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
43     }
44     return tree+M;
45 }
46 int touch(Node* r, int x, int y, LL d2){
47     LL dis = sqrt(d2)+1;
48     return !(x<r->x1-dis || x>r->x2+dis || y<r->y1-dis || y>r->y2+dis);
49 }
50 void nearest(Node* r, int x, int y, int &mID, LL &md2){
51     if (!r || !touch(r, x, y, md2)) return;
52     LL d2 = dis2(r->x, r->y, x, y);
53     if (d2 < md2 || (d2 == md2 && mID < r->id)) {
54         mID = r->id; md2 = d2;
55     }
56 // search order depends on split dim
57     if ((r->f == 0 && x < r->x) ||
58         (r->f == 1 && y < r->y)) {
59         nearest(r->L, x, y, mID, md2);
60         nearest(r->R, x, y, mID, md2);
61     } else {
62         nearest(r->R, x, y, mID, md2);
63         nearest(r->L, x, y, mID, md2);
64     }
65 }
66 int query(int x, int y) {
67     int id = 1029384756;
68     LL d2 = 102938475612345678LL;
69     nearest(root, x, y, id, d2);
70     return id;
71 }
72 }tree;

```

```

1 #include<bits/stdc++.h>
2 #define REP(i,n) for(int i=0;i<n;i++)
3 using namespace std;
4 typedef long long LL;

```

```

5     const int N=200100;
6     int n,m;
7     struct PT {int x,y,z,w,id;}p[N];
8     inline int dis(const PT &a,const PT &b){return abs(a.x-b.x)+abs(a.y-b.y);}
9     inline bool cpx(const PT &a,const PT &b){return a.x!=b.x? a.x>b.x:a.y>b.y;}
10    inline bool cpz(const PT &a,const PT &b){return a.z>b.z;}
11    struct E{int a,b,c;}e[8*N];
12    bool operator<(const E&a,const E&b){return a.c<b.c;}
13    struct Node{
14        int L,R,key;
15    }node[4*N];
16    int s[N];
17    int F(int x){return s[x]==x?s[x]=F(s[x]):x;}
18    void U(int a,int b){s[F(b)]=F(a);}
19    void init(int id,int L,int R) {
20        node[id]=(Node){L,R,-1};
21        if (L==R) return;
22        init(id*2,L,(L+R)/2); init(id*2+1,(L+R)/2+1,R);
23    }
24    void ins(int id,int x) {
25        if (node[id].key==-1 || p[node[id].key].w>p[x].w) node[id].key=x;
26        if (node[id].L==node[id].R) return;
27        if (p[x].z<=(node[id].L+node[id].R)/2)ins(id*2,x);
28        else ins(id*2+1,x);
29    }
30    int Q(int id,int L,int R){
31        if (R<node[id].L || L>node[id].R) return -1;
32        if (L<=node[id].L && node[id].R<=R) return node[id].key;
33        int a=Q(id*2,L,R),b=Q(id*2+1,L,R);
34        if (b===-1 || (a!=-1 && p[a].w<p[b].w)) return a;
35        else return b;
36    }
37    void calc() {
38        REP(i,n) {
39            p[i].z=p[i].y-p[i].x; p[i].w=p[i].x+p[i].y;
40        }
41        sort(p,p+n,cpz);
42        int cnt=0,j,k;
43        for(int i=0;i<n;i=j){
44            for(j=i+1;p[j].z==p[i].z && j<n;j++){
45                for(k=i,cnt++;k<j;k++)p[k].z=cnt;
46            }
47            init(1,i,cnt);
48            sort(p,p+n,cpx);
49            REP(i,n) {
50                j=Q(1,p[i].z,cnt);
51                if (j!=i) e[m++]=E{p[i].id,p[j].id,dis(p[i],p[j])};
52                ins(1,i);
53            }
54        }
55        LL MST() {
56            LL r=0;
57            sort(e,e+m);
58            REP(i,m) {
59                if (F(e[i].a)==F(e[i].b)) continue;
60                U(e[i].a,e[i].b);r+=e[i].c;
61            }
62            return r;
63        }
64        int main(){
65            m = 0;
66            scanf("%d",&n);
67            REP(i,n) {
68                scanf("%d%d",&p[i].x,&p[i].y);
69                p[i].id=s[i]=i;
70            }
71            calc();REP(i,n)p[i].y=-p[i].y;
72            calc();REP(i,n)swap(p[i].x,p[i].y);
73            calc();REP(i,n)p[i].x=-p[i].x;
74            calc();printf("%lld\n",MST()*2);
75        }

```

Topic II

String

1 KMP

```

1 void failure(char *s, int f[]){
2     f[0]=-1;
3     for(int k, m=strlen(s), i=1; i<=m; i++){
4         for(k=f[i-1]; k>=0 && s[k]!=s[i-1]; k=f[k]);
5         f[i]=k+1;
6     }
7 }
8 vector<int> KMP(char* s, char* t){//search for s in t
9     int m=strlen(s), n=strlen(t), f[m+1];
10    failure(s, f);
11    vector<int> result;
12    for(int k=0, i=0; i<n; ){
13        if( k==i-1 ) i++, k=0;
14        else if( t[i]==s[k] ){
15            i++;
16            if(k==m){
17                result.push_back(i-m);
18                k=f[k];
19            }
20        }else k=f[k];
21    }
22    return result;
23 }

```

2 Z-algorithm

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

```

1 void Z(const char* s, int z[]){
2     int n=z[0]=strlen(s);
3     for(int l=0, r=0, i=1; i<n; i++){
4         if( r>i || r-i<z[i-1] ){
5             if( r<i ) r=i;
6             for(l=i; r<n && s[r]==s[r-l]; r++);
7             z[i]=r-l;
8         }else z[i]=z[i-1];
9     }

```

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

```

1 void solve(const char* s){
2     int l=strlen(s);
3     vector<int> p(l+1, 1);
4     for(int c=0, r=0, n=0, m=0, i=1; i<l; i++){
5         {
6             int j=(c<<1)-i;
7             if( i>r )
8                 p[i]=0, n=i+1, m=i-1;
9             else if( p[j]<r-i )
10                 p[i]=p[j], m=-1;
11             else
12                 p[i]=r-i, n=r+1, m=(i<<1)-n;
13             while( n<l && m>=0 && s[m]==s[n] )
14                 p[i]++, n++, m--;

```

```

15     if( i+p[i]>r )
16         c=i, r=i+p[i];
17     }
18 }
```

4 Lexicographically Smallest Rotation

```

1 string lsr(string s){
2     int n=s.length(), i=0, j=1;
3     for(s+=s; i<n && j<n; j+=i==j){
4         int k=0;
5         while( k<n && s[i+k]==s[j+k] ) k++;
6         if( s[i+k]<=s[j+k] ? j : i )+=k+1;
7     }
8     return s.substr(i<n ? i : j, n);
9 }
```

5 AC Trie

```

1 struct actrie{
2     struct node{
3         node *fl, *nx[26], *dl;
4         int cnt, d;
5         node(){}
6         memset(this, 0, sizeof(node));
7     }
8     *root;
9     actrie(){}
10    root = new node();
11 }
12 void add(const char *p){
13     node *now=root;
14     for(int i=0; p[i]; i++){
15         node*& t=now->nx[ p[i]-'a' ];
16         if( !t ) t=new node();
17         now=t;
18     }
19     now->cnt++;
20 }
21 void build(){
22     queue<node*> Q;
23     for(Q.push(root); !Q.empty(); Q.pop()){
24         node* now=Q.front();
25         for(int i=0; i<26; i++){
26             node*& t=now->nx[i], *fn=now->fl;
27             if( t ){
28                 while( fn && !fn->nx[i] ) fn=fn->fl;
29                 t->fl= fn ? fn->nx[i] : root ;
30                 t->dl= t->fl->cnt ? t->fl : t->fl->dl ;
31                 t->d=now->d+1; Q.push(t);
32             }
33         }
34     }
35 }
36 void match(const char *p){
37     node* now=root;
38     for(int i=0; p[i]; i++){
39         while( now && !now->nx[ p[i]-'a' ] ) now=now->fl;
40         if( !now ) now=root;
41         else{
42             now=now->nx[ p[i]-'a' ];
43             for(node *tmp=now; tmp; tmp=tmp->dl);
44         }
45     }
46 }
47 };
```

6 Suffix Array

sar_i is the index of sorted suffixes

rk_i is the rank of suffix starting from s_i

```

1 struct suffixarray{
2     char s[N];
3     int n, sa[N], r[N], lcp[N], sa2[N], r2[N], c[N], a;
4     void init(const char* _s){
5         memset(this, 0, sizeof(suffixarray));
6         n=strlen(_s), a=128;
7         memcpy(s, _s, sizeof(char)*n);
8         for(int i=0; i<n; i++) c[ r[i]==s[i] ]++;
9         for(int i=1; i<a; i++) c[i]+=c[i-1];
10        for(int i=n-1; i>=0; i--) sa[ --c[ r[i] ] ]=i;
11        for(int l=1; l<n; l<=l){
12            int p=0;
13            for(int i=n-l; i<n; i++) sa2[p++]=i;
14            for(int i=0; i<n; i++) if( sa[i]-l>=0 ) sa2[p++]=sa[i]-l;
15            for(int i=0; i<a; i++) c[i]=0;
16            for(int i=0; i<n; i++) c[ r[i] ]++;
17            for(int i=1; i<a; i++) c[i]+=c[i-1];
18            for(int i=n-1; i>=0; i--) sa[ --c[ r[ sa2[i] ] ] ]=sa2[i];
19            r2[ sa[0] ]=0;
20            for( int i=1; i<n; i++){
21                r2[ sa[i] ]=r2[ sa[i-1] ]+1;
22                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]]--;
23            }
24            for(int i=0; i<n; i++) swap(r[i], r2[i]);
25            a=r[ sa[n-1] ]+1;
26            if( a==n ) break;
27        }
28        for(int i=0; i<n; i++) r[ sa[i] ]=i;
29        for(int k=0, i=0; i<n; i++, k=max(0, k-1)){
30            if( r[i]==n-1 ){
31                lcp[ r[i] ]=k=0;
32                continue;
33            }
34            for(int j=sa[ r[i]+1 ]; max(i, j)+k<n && s[i+k]==s[j+k]; k++);
35            lcp[ r[i] ]=k;
36        }
37    } SA;
38 }
```

Topic III Graph

1 Biconnected Connected Component

Tarjan's Algorithm in undirected graph

```

1 struct BCC{
2     vector<vector<int>> edg, brg;
3     vector<int> cut, dfn, low;
4     BCC(int n) : edg(n+1), cut(n+1, 0), dfn(n+1, 0), low(n+1){}
5     void add(int u, int v){
6         edg[u].push_back(v); edg[v].push_back(u);
7     }
8     void dfs(int u, int p=-1, int d=0){
9         dfn[u]=low[u]=++d;
10        int cnt=0;
11        for(int v : edg[u]){
12            if( v!=p && dfn[v]>0 ) low[u]=min(low[u], dfn[v]);
13            else if( dfn[v]==0 ){
14                cnt++;
15                dfs(v, u, d);
16                low[u]=min(low[u], low[v]);
17                if( d<low[v] ) brg.push_back({u, v});
18            }
19        }
20    }
21 }
```

2 Strongly Connected Component

Tarjan's Algorithm in directed graph

```

1 struct SCC{
2     int n, d=0, cnt=0;
3     vector<vector<int>> edg;
4     vector<int> com, dfn, low;
5     stack<int> S;
6     SCC(int _n) : n(_n), edg(n+1), com(n+1, 0), dfn(n+1, 0), low(n+1){}
7     void add(int u, int v){
8         edg[u].push_back(v);
9     }
10    void dfs(int u){
11        dfn[u]=low[u]=++d;
12        S.push(u);
13        for(int v : edg[u]){
14            if( !dfn[v] ) dfs(v);
15            if( !com[v] ) low[u]=min(low[u], low[v]);
16        }
17        if( dfn[u]==low[u] ){
18            cnt++;
19            for(int v; v=S.top(), S.pop(), com[v]=cnt, v!=u; );
20        }
21    }
22    void operator()(){//0-base or 1-base
23        for(int i=0; i<n; i++) if( !dfn[i] ) dfs(i);
24    }
25 }
```

3 Bellman-Ford Algorithm

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

```

1 struct BF{
2     struct edge{//directed edge u->v
3         int u, v, w;
4         edge(int _u, int _v, int _w) : u(_u), v(_v), w(_w){}
5     };
6     int n;
7     vector<int> d;
8     vector<edge> e;
9     BF(int _n) : n(_n){//zero-base vertices
10    void add(int u, int v, int w){//add an edge
11        e.push_back(edge(u, v, w));
12    }
13    bool relax()//does relaxation with all edges once
14        bool any=false;
15        for(const edge& E : e)
16            if( d[E.v]>d[E.u]+E.w ) d[E.v]=d[E.u]+E.w, any=true;
17        return any;
18    }
19    void operator()(int s){//compute SSSP start with s
20        d.assign(n, INF); d[s]=0;
21        for(int i=1; i<n; i++) if( !relax() ) break;
22    }
23    bool neg_cycle{//detect negative cycle
24        d.assign(n, 0);
25        for(int i=0; i<n; i++) if( !relax() ) return false;
26        return relax();
27    }
28    double karp_mmc(){//calculate the min mean cycle ratio
29        double ans=INF;
30        vector<vector<int>> d(n+1, vector<int>(n, INF));
31        d[0].assign(n, 0);
32        for(int i=1; i<=n; i++) for(const edge& E : e)
33            d[i][E.v]=min(d[i][E.v], d[i-1][E.u]+E.w);
34        for(int i=0; i<n; i++){
35            double tmp=-INF;
36            if( d[n][i]>=INF ) continue;
37            for(int j=0; j<n; j++)
38                tmp=max(tmp, (d[n][i]-d[j][i])/((double)(n-j)));
39            ans=min(ans, tmp);
40        }
41        return ans;
42    }
43 };

```

4 Maximum Flow

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

```

1 struct Dinic{
2     struct edge{
3         int t, c, r;
4         edge(int _t, int _c, int _r) : t(_t), c(_c), r(_r){}
5     };
6     vector<int> l;
7     vector<vector<edge>> e;
8     Dinic(int n) : e(n+1){}
9     void add(int u, int v, int w){//directed
10        e[u].push_back(edge(v, w, e[v].size()));
11        e[v].push_back(edge(u, 0, e[u].size()-1));
12    }
13    edge& rev(const edge& E){
14        return e[E.t][E.r];
15    }
16    bool bfs(int s, int t){
17        l.assign(e.size(), INF);
18        l[s]=1;
19        queue<int> Q;
20        for(Q.push(s); !Q.empty(); Q.pop()){
21            int s=Q.front();
22            for(const edge& E : e[s])
23                if( E.c>0 && l[E.t]>l[s]+1 ){
24                    l[E.t]=l[s]+1;
25                    Q.push(E.t);
26                }
27        }
28        return l[t]<INF;
29    }
30    int dfs(int s, int t, int num=INF){
31        if( s==t || num==0 ) return num;
32        int ans=0;
33        for(edge& E : e[s])
34            if( E.c>0 && l[s]+1==l[E.t] ){
35                int tmp=dfs(E.t, t, min(num, E.c));
36                rev(E).c+=tmp, ans+=tmp;
37                E.c-=tmp, num-=tmp;
38            }
39        return ans>0 ? ans : l[s]=0;
40    }
41    int operator()(int s, int t){
42        int ans=0, tmp=0;
43        while( (tmp=dfs(s, t)) )
44            ans+=tmp;
45        return ans;
46    }
47 }
48 };

```

5 Minimum-Cost Maximum Flow

$O(\max f \cdot V^2)$

```

1 struct costflow{
2     struct edge{
3         int t, f, c, r;//c, c
4         edge(int _t, int _f, int _c, int _r) : t(_t), f(_f), c(_c), r(_r){}
5     };
6     int n;
7     vector<int> prv, plv, dis;//dis
8     vector<vector<edge>> e;
9     costflow(int _n) : n(_n), prv(n+1), plv(n+1), e(n+1){}
10    void add(int u, int v, int f, int c){//c
11        e[u].push_back(edge(v, f, c, e[v].size()));
12        e[v].push_back(edge(u, 0, -c, e[u].size()-1));
13    }
14    edge& rev(const edge& E){
15        return e[E.t][E.r];
16    }
17 };
18 
```

```

16 }
17 bool bfs(int s, int t){
18     vector<bool> inq(n+1, false);
19     dis.assign(n+1, INF); //INF
20     dis[s]=0;
21     queue<int> Q;
22     for(Q.push(s); !Q.empty(); Q.pop()){
23         s=Q.front(), inq[s]=0;
24         for(int i=e[s].size()-1; i>=0; i--){
25             const edge& E=e[s][i];
26             if( dis[E.t]>dis[s]+E.c && E.f>0 ){
27                 dis[E.t]=dis[s]+E.c;
28                 pcv[E.t]=s, plv[E.t]=i;
29                 if( !inq[E.t] ) Q.push(E.t), inq[E.t]=true;
30             }
31         }
32     }
33     return dis[t]<INF;
34 }
35 pair<int, int> operator()(int s, int t){//second
36     int fl=0, cs=0;//cs
37     for(int tf=INF; bfs(s, t); tf=INF){
38         for(int v=t, u, l; v!=s; v=u){
39             u=pcv[v], l=plv[v];
40             tf=min(tf, e[u][l].f);
41         }
42         for(int v=t, u, l; v!=s; v=u){
43             u=pcv[v], l=plv[v];
44             rev(e[u][l]).f+=tf;
45             e[u][l].f-=tf;
46         }
47         cs+=tf*dis[t], fl+=tf;
48     }
49     return pair<int, int>(fl, cs); //second
50 }
51 };

```

6 Maximum-Weight Bipartite Perfect Matching

$O(V^3)$

```

1 struct KM{
2     static const int INF=2147483647; //long long
3     int n;
4     vector<int> match, vx, vy;
5     vector<int> lx, ly, slack; //long long
6     vector<vector<int>> edge; //long long
7     KM(int _n) : n(_n), match(n, -1), lx(n, -INF), ly(n, 0), edge(n, vector<int>(n, 0)) {}
8     void add_edge(int x, int y, int w){ //long long
9         edge[x][y] = w;
10    }
11    bool dfs(int x){
12        vx[x]=1;
13        for(int y=0; y<n; y++){
14            if( vy[y] ) continue;
15            if( lx[x]+ly[y]>edge[x][y] )
16                slack[y]=min(slack[y], lx[x]+ly[y]-edge[x][y]);
17            else{
18                vy[y]=1;
19                if( match[y]==-1 || dfs(match[y]) ){
20                    match[y]=x; return true;
21                }
22            }
23        }
24        return false;
25    }
26    int operator()(){
27        for(int i=0; i<n; i++) for(int j=0; j<n; j++)
28            lx[i]=max(lx[i], edge[i][j]);
29        for(int i=0; i<n; i++) for(slack.assign(n, INF); ; ){
30            vx.assign(n, 0); vy.assign(n, 0);
31            if( dfs(i) ) break;
32            int d=INF; // long long

```

```

33             for(int j=0; j<n; j++)
34                 if( !vy[j] ) d=min(d, slack[j]);
35             for(int j=0; j<n; j++){
36                 if( vx[j] ) lx[j]-=d;
37                 if( vy[j] ) ly[j]+=d;
38                 else slack[j]-=d;
39             }
40         }
41         int res=0;
42         for(int i=0; i<n; i++) res+=edge[match[i]][i];
43         return res;
44     }
45 }

```

7 Maximum-Cardinality Bipartite Matching

$O(\sqrt{V}E)$

```

1 struct HK{
2     int n, m;
3     vector<int> d, p;
4     vector<vector<int>> e;
5     HK(int _n, int _m) : n(_n), m(_m), e(n+m+1) {}
6     void add(int u, int v){ //one base index: [1, u]*[1, v]
7         e[u].push_back(n+v);
8         e[n+v].push_back(u);
9     }
10    bool bfs(){
11        d.assign(n+m+1, INF);
12        queue<int> Q;
13        for(int i=1; i<=n; i++)
14            if( p[i]==0 ){
15                d[i]=0;
16                Q.push(i);
17            }
18        for(;; !Q.empty(); Q.pop()){
19            int u=Q.front();
20            if( d[u]>d[0] ) break;
21            for(int v : e[u])
22                if( d[p[v]]==INF ){
23                    d[p[v]]=d[u]+1;
24                    Q.push(p[v]);
25                }
26        }
27        return d[0]<INF;
28    }
29    bool dfs(int u){
30        if( u==0 ) return true;
31        for(int v : e[u])
32            if( d[p[v]]==d[u]+1 && dfs(p[v]) ){
33                p[v]=u, p[u]=v;
34                return true;
35            }
36        d[u]=INF;
37        return false;
38    }
39    int operator()(){
40        int ans=0;
41        p.assign(n+m+1, 0);
42        while( bfs() )
43            for(int i=1; i<=n; i++)
44                if( p[i]==0 && dfs(i) )
45                    ans++;
46        return ans;
47    }
48 }

```

8 Minimum-Weight General Perfect Matching

$O(V^2E)$

```

1 struct Graph
2 {
3     int n;
4     vector<vector<int>> edge; //0-base
5     vector<int> match, dis, ons, stk;
6     Graph(int _n) : n(_n), edge(n, vector<int>(n, 0)), match(n){}
7     void add_edge(int u, int v, int w){
8         edge[u][v]=edge[v][u]=w;
9     }
10    bool SPFA(int u){
11        if( ons[u] ) return true;
12        stk.push_back(u); ons[u]=1;
13        for(int v=0; v<n; v++){
14            if( u==v && match[u]!=v && !ons[v] ){
15                int m=match[v];
16                if( dis[m]>dis[u]-edge[v][m]+edge[u][v] ){
17                    dis[m]=dis[u]-edge[v][m]+edge[u][v];
18                    stk.push_back(v); ons[v]=1;
19                    if( SPFA(m) ) return true;
20                    stk.pop_back(); ons[v]=0;
21                }
22            }
23        }
24        stk.pop_back(); ons[u]=0;
25        return false;
26    }
27    int operator()(){
28        for (int i=0; i<n; i+=2)
29            match[i]=i+1, match[i+1]=i;
30        for(bool found=true; found; ){
31            found=false;
32            dis.assign(n, 0);
33            ons.assign(n, 0);
34            for(int i=0; i<n; i++){
35                stk.clear();
36                if( !ons[i] && SPFA(i) )
37                    for(found=true; stk.size()>=2; ){
38                        int u=stk.back(); stk.pop_back();
39                        int v=stk.back(); stk.pop_back();
40                        match[u]=v; match[v]=u;
41                    }
42            }
43            int ans=0;
44            for(int i=0; i<n; i++) ans+=edge[i][ match[i] ];
45            return ans>>1;
46        }
47    }
48 };

```

9 Maximum-Cardinality General Matching

$O(\sqrt{V}E)$

```

1 struct Graph{
2     int n, st, ed, nb, ans=0;
3     vector<vector<int>> edg; //1-base
4     vector<int> pr, bk, ds;
5     vector<bool> inq, inp, inb;
6     queue<int> Q;
7     Graph(int _n) : n(_n), edg(n+1), pr(n+1, 0), ds(n+1){}
8     void add_edge(int u, int v){
9         edg[u].push_back(v); edg[v].push_back(u);
10    }
11    int lca(int u, int v){
12        inq.assign(n+1, false);
13        for(u=ds[u]; ; u=ds[ bk[ pr[u] ] ]){
14            inq[u]=true;
15            if( u==st ) break;
16        }
17        for(v=ds[v]; !inq[v]; v=ds[ bk[ pr[v] ] ]){
18            inq[v]=true;
19        }
20        void upd(int u){
21            while( ds[u]!=nb ){
22                int v=pr[u];
23                inb[ ds[u] ]=inb[ ds[v] ]=true;
24                u=bk[v];
25                if( ds[u]!=nb ) bk[u]=v;
26            }
27        }
28        void blo(int u, int v){
29            nb=lca(u, v);
30            inb.assign(n+1, false);
31            upd(u); upd(v);
32            if( ds[u]==nb ) bk[u]=v;
33            if( ds[v]==nb ) bk[v]=u;
34            for(int tu=1; tu<=n; tu++){
35                ds[tu]=nb;
36                if( !inq[tu] ) Q.push(tu); inq[tu]=true;
37            }
38        }
39        void flo(){
40        }
41        bk.assign(n+1, 0);
42        inq.assign(n+1, false);
43        inq[st]=true;
44        for(int i=1; i<=n; i++) ds[i]=i;
45        for(ed=0; !Q.empty(); Q.pop()){
46            for(Q.push(st); !Q.empty(); Q.pop()){
47                int u=Q.front();
48                for(int v : edg[u]) if( ds[u]!=ds[v] && pr[u]!=v ){
49                    if( v==st || pr[v]>0 && bk[ pr[v] ]>0 ) blo(u, v);
50                    else if( bk[v]==0 ){
51                        bk[v]=u;
52                        if( pr[v]<=0 ) { ed=v; return ; }
53                        else if( pr[v]>0 && !inq[ pr[v] ] ) Q.push(pr[v]);
54                    }
55                }
56            }
57        }
58        void aug(){
59            for(int w, v, u=ed; u>0; u=w){
60                v=bk[u]; w=pr[v];
61                pr[v]=u; pr[u]=v;
62            }
63        }
64        int operator()(){
65            for(int u=1; u<=n; u++)
66            {
67                st=u; flo();
68                if( ed>0 ){
69                    aug(); ans++;
70                }
71            }
72        }
73    }
74 };

```

```

72     return ans;
73 }
74 };

```

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 $O(3^{V/3})$

```

1 typedef bitset<N> set;
2 vector<set> c, e;
3 void BronKerbosch(set R, set P, set X){
4     if( P.none() && X.none() ){
5         c.push_back(R);
6         return;
7     }
8     int u=0;
9     for(; u<n && !(P|X)[u]; u++);
10    set T(1);
11    for(int i=0; i<n; i++, T<<=1){
12        if( (P&~e[u])[i] ){
13            BronKerbosch(R|T, P&e[i], X&e[i]);
14            P[i]=false, X[i]=true;
15        }
16    }

```

11 Least Common Ancestor

```

1 #define edge pair<int,int>
2 #define v first
3 #define w second
4 struct lca{
5     const int H=20;
6     int n;
7     vector<int> h;
8     vector<vector<int>> p, b;
9     lca(const vector<vector<edge>>& e) : n(e.size()-1),//one-base vertex index
10    h(n+1, -1), p(H, vector<int>(n+1, -1)), b(H, vector<int>(n+1)){
11        p[0][1]=1; b[0][1]=0; dfs(e, 1, 0);
12        for(int i=1; i<H; i++) for(int j=1; j<=n; j++){
13            p[i][j]=p[i-1][ p[i-1][j] ];
14            b[i][j]=max(b[i-1][ p[i-1][j] ], b[i-1][j]);
15            //b is something you want to calculate on the path from root
16        }
17    }
18    void dfs(const vector<vector<edge>>& e, int u, int d=0){
19        h[u]=++d;
20        for(const edge& E : e[u]){
21            if( h[E.v]>=0 ) continue;
22            p[0][E.v]=u; b[0][E.v]=E.w;
23            dfs(e, E.v, d);
24        }
25    }
26    int operator()(int u, int v) const{
27        if( h[u]>h[v] ) swap(u, v);
28        int ans=0;
29        for(int i=0, d=h[v]-h[u]; d>0; d>>=1, i++){
30            if( d&1 ) ans=max(ans, b[i][v]), v=p[i][v];
31            for(int i=0; u!=v; i++){
32                for(; i>0 && p[i][u]==p[i][v]; i--);
33                ans=max(ans, max(b[i][u], b[i][v]));
34                u=p[i][u], v=p[i][v];
35            }
36        }
37        return ans;
38    };

```

12 Stable Matching

$O(nm \log n)$

```

1 struct stable{
2     int n, m, ans=0; //n: left size, m: right size
3     vector<vector<int>> a; //left preference
4     vector<int> b, c; //b: left match, c: right capacity
5     vector<map<int, int>> M, PQ; //M: right preference, PQ: right match
6     int operator()(){
7         queue<int> Q;
8         for(int i=1; i<=n; i++) Q.push(i);
9         for(; !Q.empty(); Q.pop()){
10            for(int u=Q.front(); !b[u] && !a[u].empty(); a[u].pop_back()){
11                int v=a[u].back();
12                if( (int)PQ[v].size()<c[v] ){
13                    PQ[v][ M[v][u] ]=u;
14                    b[u]=v;
15                    ans++;
16                }
17                else if( PQ[v].begin()->first< M[v][u] ){
18                    Q.push(PQ[v].begin()->second);
19                    b[PQ[v].begin()->second]=0;
20                    PQ[v].erase(PQ[v].begin());
21                    PQ[v][ M[v][u] ]=u;
22                    b[u]=v;
23                }
24            }
25        }
26        return ans;
27    }
28 };

```

Topic IV

Data Structure and Others

1 Treap

```

1 struct treap{
2     struct node{
3         node *l=NULL, *r=NULL, *p=NULL;
4         int pri=rand(), siz=1, key, val;
5     };
6     static void pull(node* p){}
7     static void push(node* p){
8         if( p!=NULL ){
9             set_parent(p->l, p);
10            set_parent(p->r, p);
11        }
12    }
13    static void set_parent(node* x, node* p=NULL){
14        if( x!=NULL ) x->p=p;
15    }
16    static node* merge(node* l, node* r){
17        if( l==NULL )
18            return r;
19        else if( r==NULL )
20            return l;
21        else if( l->pri>r->pri ){
22            push(l); l->r=merge(l->r, r);
23            pull(l); return l;
24        }
25        else{
26            push(r); r->l=merge(l, r->l);
27            pull(r); return r;
28        }
29    }
30    static void split(node* p, int key, node*& l, node*& r){
31        push(p);
32        if( p==NULL ) l=r=NULL;
33        else if( p->key<=key ){
34            l=p;
35            split(l->r, key, l->r, r);
36            pull(l);
37        }
38        else{
39            r=p;
40            split(r->l, key, l, r->l);
41            pull(r);
42        }
43    }
44 };

```

2 2D Binary Indexed Tree

```

1 struct bit{
2     int w, h; //1<=x<=w, 1<=y<=h
3     vector<vector<int>> a;
4     bit(int w, int h=1) : w(w), h(h), a(w+1, vector<int>(h+1, 0)){}
5     void add(int x, int y=1, int val=1){
6         for(); x<=w; x+=x&-x)
7             for(int yy=y; yy<=h; yy+=yy&-yy)
8                 a[x][yy]+=val;
9     }
10    int query(int x, int y=1){
11        int res=0;
12        for(); x>0; x-=x&-x)
13            for(int yy=y; yy>0; yy-=yy&-yy)
14                res+=a[x][yy];
15        return res;
16    }
17 };

```

```
16 }
17 }
```

3 Sparse Table

```
1 const int N=10;
2 struct ST
3 {
4     int n, m, a[N][N][1<N][1<N];
5     void init(int _n, int _m){
6         n=_n, m=_m;
7         memset(a, 0, sizeof(a));
8         for(int i=0; i<n; i++) for (int j=0; j<m; j++)
9             scanf("%d", &a[0][0][i][j]);
10        for(int u=1; u<=lg(n); u++){
11            int u2=1<<u, u3=u2>>1;
12            for(int i=0; i+u2<=n; i++) for(int j=0; j<m; j++)
13                a[u][0][i][j]=max(a[u-1][0][i][j], a[u-1][0][i+u3][j]);
14        }
15        for(int u=0; u<=lg(n); u++) for(int v=1; v<=lg(m); v++){
16            int u2=1<<u, v2=1<<v, v3=v2>>1;
17            for(int i=0; i+u2<=n; i++) for(int j=0; j+v2<=m; j++)
18                a[u][v][i][j]=max(a[u][v-1][i][j], a[u][v-1][i][j+v3]);
19        }
20    }
21    int query(int l, int r, int u, int d){//[l, r)*[u, d)
22        int x=lg(d-u), x2=1<<x, y=lg(r-l), y2=1<<y;
23        int maxv=max(a[x][y][u][l], a[x][y][d-x2][l]);
24        maxv=max(maxv, a[x][y][u][r-y2]);
25        return max(maxv, a[x][y][d-x2][r-y2]);
26    }
27    static int lg(int x){
28        return log2((double)x)+EPS; //return 30-__builtin_clrsb(x);
29    }
30 } st;
```

4 Subset Sum

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

```
1 const int N=1<<20;//sum of all numbers
2 struct SSS{
3     int tot;
4     bitset<N> ok;
5     SSS(const vector<int>& a) : tot(0), ok(1){
6         vector<int> c(N, 0);
7         for(int x : a) tot+=x, c[x]++;
8         for(int sum=0, i=1; i<=tot>>1; i++) if( c[i]>0 ){
9             for(int m=min(tot>>1, sum+=i*c[i]), j=max(m-i+1, 1); j<=m; j++){
10                 int cnt=0;
11                 for(int k=0; k<c[i] && j>=i*k; k++) cnt+=ok[j-i*k];
12                 for(int k=j; k>0; k-=i) cnt+=ok[k];
13                 if( k>=i*c[i] ) cnt+=ok[k-i*c[i]];
14                 if( cnt>0 ) ok[k]=true;
15             }
16         }
17     }
18     bool operator[](int n) const{
19         return n<0 || n>tot ? false : ok[min(n, tot-n)];
20     }
21 };
22 }
```

5 Built-in Functions Provided by GCC

```
1 int __builtin_ffsll(lld x)
2     //Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.
3
4 int __builtin_clzll(llu x)
5     //Returns the number of leading 0-bits in x, starting at the most significant bit position. If x is
6     //0, the result is undefined.
7
8 int __builtin_ctzll(llu x)
9     //Returns the number of trailing 0-bits in x, starting at the least significant bit position. If x is
10    //0, the result is undefined.
11
12 int __builtin_clrsbll(lld x)
13     //Returns the number of leading redundant sign bits in x, i.e. the number of bits following the most
14     //significant bit that are identical to it. There are no special cases for 0 or other values.
15
16 int __builtin_popcountll(llu x)
17     //Returns the number of 1-bits in x.
```

6 Link Cut Tree

```
1 struct Node{
2     int sz, label; /* size, label */
3     Node *p, *pp, *l, *r; /* parent, path-parent, left, right pointers */
4     Node() { p = pp = l = r = 0; }
5 }
6 void update(Node *x){
7     x->sz = 1;
8     if (x->l) x->sz += x->l->sz;
9     if (x->r) x->sz += x->r->sz;
10}
11 void rotr(Node *x){
12     Node *y, *z;
13     y = x->p, z = y->p;
14     if ((y->l = x->r)) y->l->p = y;
15     x->r = y, y->p = x;
16     if ((x->p = z)){
17         if (y == z->l) z->l = x;
18         else z->r = x;
19     }
20     x->pp = y->pp; y->pp = 0; update(y);
21 }
22 void rotl(Node *x){
23     Node *y, *z;
24     y = x->p, z = y->p;
25     if ((y->r = x->l)) y->r->p = y;
26     x->l = y, y->p = x;
27     if ((x->p = z)){
28         if (y == z->r) z->r = x;
29         else z->l = x;
30     }
31     x->pp = y->pp; y->pp = 0; update(y);
32 }
33 void splay(Node *x){
34     for(Node *y, *z; x->p; ){
35         y = x->p;
36         if (y->p == 0){
37             if (x == y->l) rotr(x);
38             else rotl(x);
39         }
40         else{
41             z = y->p;
42             if (y == z->l){
43                 if (x == y->l) rotr(y), rotr(x);
44                 else rotl(x), rotr(x);
45             }
46             else{
47                 if (x == y->r) rotl(y), rotl(x);
48                 else rotr(x), rotl(x);
49             }
50         }
51     }
52 }
update(x);
```

```

53 }
54 Node *access(Node *x){
55     splay(x);
56     if (x->r){
57         x->r->pp = x; x->r->p = 0; x->r = 0; update(x);
58     }
59     Node *last = x;
60     while (x->pp){
61         Node *y = x->pp; last = y; splay(y);
62         if (y->r){
63             y->r->pp = y; y->r->p = 0;
64         }
65         y->r = x; x->p = y; x->pp = 0; update(y); splay(x);
66     }
67     return last;
68 }
69 Node *root(Node *x){
70     access(x); while (x->l) x = x->l; splay(x); return x;
71 }
72 void cut(Node *x){
73     access(x); x->l->p = 0; x->l = 0; update(x);
74 }
75 void link(Node *x, Node *y){
76     access(x); access(y); x->l = y; y->p = x; update(x);
77 }
78 Node *lca(Node *x, Node *y){
79     access(x); return access(y);
80 }
81 int depth(Node *x){
82     access(x); return x->sz - 1;
83 }
84 struct LinkCut{
85     Node *x;
86     LinkCut(int n){
87         x = new Node[n];
88         for (int i = 0; i < n; i++){
89             x[i].label = i;
90             update(&x[i]);
91         }
92     }
93     void link(int u, int v){
94         ::link(&x[u], &x[v]);
95     }
96     void cut(int u){
97         ::cut(&x[u]);
98     }
99     int root(int u){
100        return ::root(&x[u])->label;
101    }
102    int depth(int u){
103        return ::depth(&x[u]);
104    }
105    int lca(int u, int v){
106        return ::lca(&x[u], &x[v])->label;
107    }
108 };

```

```

1 const int MAXN = 30000;
2 template <typename T>
3 struct LinkCutree {
4     enum Relation {
5         L = 0, R = 1
6     };
7     struct Node {
8         Node *child[2], *parent, *pathParent;
9         T value, sum, max;
10        bool reversed;
11        Node(const T &value) : reversed(false), value(value), sum(value), max(value), parent(NULL),
12                         pathParent(NULL) {
13             child[L] = child[R] = NULL;
14         }
15         Relation relation() {
16             return this == parent->child[L] ? L : R;
17         }
18         void pushDown() {
19             if (reversed) {

```

```

19                 std::swap(child[L], child[R]);
20                 if (child[L]) child[L]->reversed ^= 1;
21                 if (child[R]) child[R]->reversed ^= 1;
22                 reversed = false;
23             }
24         }
25         void maintain() {
26             sum = value;
27             if (child[L]) sum += child[L]->sum;
28             if (child[R]) sum += child[R]->sum;
29             max = value;
30             if (child[L]) max = std::max(max, child[L]->max);
31             if (child[R]) max = std::max(max, child[R]->max);
32         }
33         void rotate() {
34             if (parent->parent) parent->parent->pushDown();
35             parent->pushDown(), pushDown();
36             std::swap(pathParent, parent->pathParent);
37             Relation x = relation();
38             Node *oldParent = parent;
39             if (oldParent->parent) oldParent->parent->child[oldParent->relation()] = this;
40             parent = oldParent->parent;
41             oldParent->child[x] = child[x ^ 1];
42             if (child[x ^ 1]) child[x ^ 1]->parent = oldParent;
43             child[x ^ 1] = oldParent;
44             oldParent->parent = this;
45             oldParent->maintain(), maintain();
46         }
47         void splay() {
48             while (parent) {
49                 if (!parent->parent) rotate();
50                 else {
51                     parent->parent->pushDown(), parent->pushDown();
52                     if (relation() == parent->relation()) parent->rotate(), rotate();
53                     else rotate(), rotate();
54                 }
55             }
56             void evert() {
57                 access();
58                 splay();
59                 reversed ^= 1;
60             }
61             void expose() {
62                 splay();
63                 pushDown();
64                 if (child[R]) {
65                     child[R]->parent = NULL;
66                     child[R]->pathParent = this;
67                     child[R] = NULL;
68                     maintain();
69                 }
70             }
71             bool splice() {
72                 splay();
73                 if (!pathParent) return false;
74                 pathParent->expose();
75                 pathParent->child[R] = this;
76                 parent = pathParent;
77                 pathParent = NULL;
78                 parent->maintain();
79                 return true;
80             }
81             void access() {
82                 expose();
83                 while (splice());
84             }
85             const T &querySum() {
86                 access();
87             }
88         }
89     };
90 };
91 
```

```

95     splay();
96     return sum;
97 }
98 const T &queryMax() {
99     access();
100    splay();
101    return max;
102 }
103 };
104 Node *nodes[MAXN];
105 void makeTree(int u, const T &value) {
106     nodes[u - 1] = new Node(value);
107 }
108 void link(int u, int v) {
109     nodes[v - 1]->evert();
110     nodes[v - 1]->pathParent = nodes[u - 1];
111 }
112 void cut(int u, int v) {
113     nodes[u - 1]->evert();
114     nodes[v - 1]->access();
115     nodes[v - 1]->splay();
116     nodes[v - 1]->pushDown();
117     nodes[v - 1]->child[L]->parent = NULL;
118     nodes[v - 1]->child[L] = NULL;
119     nodes[v - 1]->maintain();
120 }
121 const T &querySum(int u, int v) {
122     nodes[u - 1]->evert();
123     return nodes[v - 1]->querySum();
124 }
125 const T &queryMax(int u, int v) {
126     nodes[u - 1]->evert();
127     return nodes[v - 1]->queryMax();
128 }
129 void update(int u, const T &value) {
130     nodes[u - 1]->splay();
131     nodes[u - 1]->value = value;
132     nodes[u - 1]->maintain();
133 }
134 };

```

A sequence of non-negative integers $d_1 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + \dots + d_n$ is even and $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$ holds for every k in $1 \leq k \leq n$

A sequence $((a_1, b_1), \dots, (a_n, b_n))$ of non-negative integer pairs with $a_1 \geq \dots \geq a_n$ is realizable on simple digraph if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$ holds for every k in $1 \leq k \leq n$

A pair of sequences of non-negative integers (a_1, \dots, a_n) and (b_1, \dots, b_n) with $a_1 \geq \dots \geq a_n$ is realizable on bipartite graph if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$ holds for every k such that $1 \leq k \leq n$

```

1 void gomory(vector<vector<int>>& f, vector<int>& p, int n){
2     f.assign(n+1, vector<int>(n+1, INF)); //all pair min cut
3     p.assign(n+1, 1); //tree edge
4     for(int s=2; s<=n; s++){
5         int t=p[s];
6         Dinic F=0;
7         int tmp=F(s, t);
8         for(int i=1; i<s; i++) f[s][i]=f[i][s]=min(tmp, f[t][i]);
9         for(int i=s+1; i<=n; i++) if( p[i]==t && F.l[i]<INF ) p[i]=s;
10    }
11 }

```

```

1 //O(nmk)
2 //n: number of vertices
3 //m: number of edges
4 //a: a[i][2]: edges
5 //co: output
6 //K, A, B, C, D, P: factors
7 int a[M][2], co[N], f[N], tf[N];
8 void ghash(){
9     for(int i=0; i<n; i++) {
10        {
11            for(int j=0; j<n; j++) tf[j]=f[j]=1;
12            for(int k=0; k<k; k++){
13                for(int j=0; j<n; j++) f[j]*=A;
14                for(int j=0; j<m; j++) {
15                    {
16                        f[ a[j][0] ]+=tf[ a[j][1] ]*B;
17                        f[ a[j][1] ]+=tf[ a[j][0] ]*C;
18                    }
19                    f[i]+=D;
20                    for(int j=0; j<n; j++) tf[j]=f[j]%=P;
21                }
22                co[i]=f[i];
23            }
24            sort(co, co+n);
25        }

```

Topic V

Theorems

$\chi(G)$: vertex chromatic number

$\chi'(G)$: edge chromatic number

$D(G)$: maximum degree

$\chi(G) \leq D(G)$ for connected simple graph, unless G is complete or an odd cycle

$\chi(G) \leq D(G) + 1$

$\chi'(G) = D(G)$ or $D(G) + 1$

$\chi'(G) = D(G)$ if G is bipartite

Every planar graph can be 4-colored

$G : \max IS + \min VC = V$

$G : |\max matching| + |\min EC| = |V|$

$B : |\max matching| = |\max flow| = |\min VC|$

$B : |\min EC| + |\min VC| = |V|$

For non-negative integer n, m and prime p , $\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p}$

where m_i is the i -th digit of m in base p .