

Contents

1	Basic	
1.1	default code	
1.2	.vimrc	
2	math	
2.1	ext gcd	
2.2	FFT	
2.3	NTT	
2.4	MillerRabin other	
2.5	Guass	
2.6	xorFFT	
2.7	orFFT	
2.8	andFFT	
2.9	Them.	
3	flow	
3.1	dinic	
3.2	min-cost-max-flow	
4	string	
4.1	KMP	
4.2	Z-value	
4.3	Z-value-palindrome	
4.4	Suffix Array($O(N\log N)$)	
4.5	Suffix Array(SAIS, twt)	
4.6	Aho-Corasic-2016ioicamp	
4.7	Palindrome Automaton	
4.8	Suffix Automaton(bcw)	
5	graph	
5.1	Bipartite matching($O(N^3)$)	
5.2	KM($O(N^4)$)	
5.3	general graph matching(bcw)	
5.4	Max clique(bcw)	
5.5	EdgeBCC	
5.6	VertexBCC	
5.7	Dominating Tree	
5.8	Min Cut	
5.9	DMST with sol(bcw)	
5.10	Them.	
6	data structure	
6.1	Treap	
6.2	copy on write treap	
6.3	copy on write segment tree	
6.4	Treap+(HOJ 92)	
6.5	Leftist Tree	
6.6	Link Cut Tree	
6.7	Heavy Light Decomposition	
6.8	Disjoint Sets + offline skill	
6.9	2D Segment Tree	
7	geometry	
7.1	Basic	
7.2	CircleCover	
7.3	ConvexHull	
7.4	HalfPlaneNSquare	
7.5	LineIntersection	
7.6	OldCircleInter	
7.7	PolyCircleIntersect	
7.8	SegmentIntersection	
7.9	Triangulation	
7.10	Smallest circle problem	
8	Others	
8.1	Random	
8.2	Fraction	

Basic

default code

```

1 #include <bits/stdc++.h>
2 #define PB push_back
3 #define MP make_pair
4 #define F first
5 #define S second
6 #define SZ(x) ((int)(x).size())
7 #define ALL(x) (x).begin(),(x).end()
8 #ifdef _DEBUG_
9     #define debug(...) printf(__VA_ARGS__)
10 #else
11     #define debug(...) (void)0
12 #endif
13 using namespace std;
14 typedef long long ll;
15 typedef pair<int,int> PII;
16 typedef vector<int> VI;
17
18 int main() {
19     return 0;
20 }
```

.vimrc

```

1 color torte
2 syn on
3 set guifont=Consolas:h16: nu sc ai si ts=4
  sm sts=4 sw=4
4
5 map <F9> <ESC>:w<CR>:!g++ % -o %< -O2 -Wall
  -Wno-unused-result -std=c++0x<CR>
6 map <S-F9> <ESC>:w<CR>:!g++ % -o %< -O2 -
  Wall -Wno-unused-result -D_DEBUG_ -std=c
  ++0x<CR>
7 map <F5> <ESC>:!./%<<CR>
8 map <F6> <ESC>:w<CR>ggVG"+y
9 map <S-F5> <ESC>:!./%< < %<.in<CR>
10 imap <Home> <ESC>^i
11 com INPUT sp %<.in
```

math

ext gcd

```

1 // find one solution (x,y) of ax+by=gcd(
  a,b)
2 void ext_gcd(int a,int b,int &g,int &x,int
  &y)
3 {
4     if(!b){ g=a; x=1; y=0; }
5     else{ ext_gcd(b, a%b, g, y, x); y -= x*(a
      /b); }
6 }
```

FFT

```

1 typedef complex<double> CD;
2
3 const double PI=acos(-1.0);
4 inline CD ang(double t) { return CD(cos(t),
    sin(t)); }
5
6 int rev_int(int x,int lgn) {
7     int re=0;
8     for(int i=0;i<lgn;i++) {
9         re=(re<<1)+(x&1);
10        x>>=1;
11    }
12    return re;
13 }
14 void fft(CD* A, int lgn, bool inv=false) {
15     int n=1<<lgn;
16     for(int i=0;i<n;i++)
17         if(i<rev_int(i, lgn)) swap(A[i], A[
            rev_int(i, lgn)]);
18     for(int i=1;i<n;i*=2) {
19         CD W(1.0, 0.0), Wn;
20         if(inv) Wn=ang(-PI/i);
21         else Wn=ang(PI/i);
22         for(int j=0;j<n;j++) {
23             if(j&i) {
24                 W=CD(1.0, 0.0);
25                 continue;
26             }
27             CD x=A[j], y=A[j+i]*W;
28             A[j]=x+y;
29             A[j+i]=x-y;
30             W*=Wn;
31         }
32     }
33     if(inv)
34         for(int i=0;i<n;i++)
35             A[i]/=n;
36 }

```

NTT

```

1 //      MOD      Wn_      LGN
2 //      5767169      177147 19
3 //      7340033      2187 20
4 //      2013265921 440564289 27
5 const int MOD=786433;
6 const int Wn_=5; // 25 625
7 const int LGN=18; // 17 16
8 inline int add(int x,int y) { return (x+y)%
    MOD; }
9 inline int mul(int x,int y) { return 111*x*
    y%MOD; }
10 inline int sub(int x,int y) { return (x-y+
    MOD)%MOD; }
11
12 int pW[MOD]; // power of Wn
13 int divN;
14 int inv(int a) {
15     int re=1, k=MOD-2, t=a;
16     while(k) {
17         if(k%2) re=mul(re, t);

```

```

18         k/=2;
19         t=mul(t, t);
20     }
21     return re;
22 }
23 void NTTinit(int lgn) { // call every time
    using new lgn !
24     int Wn=Wn_;
25     for(int i=lgn;i<LGN;i++) Wn=mul(Wn,Wn);
26     divN=inv(1<<lgn);
27     pW[0]=1;
28     for(int i=1;i<LGN;i++) {
29         pW[i]=mul(pW[i-1], Wn);
30         if(pW[i]==1) break;
31     }
32 }
33
34 int rev_int(int x,int lgn) {
35     int re=0;
36     for(int i=0;i<lgn;i++) {
37         re=(re<<1)+(x&1);
38         x>>=1;
39     }
40     return re;
41 }
42 void ntt(int *A,int lgn,bool inv=false) {
43     int n=1<<lgn;
44     for(int i=0;i<n;i++)
45         if(i<rev_int(i,lgn))
46             swap(A[i], A[rev_int(i,lgn)]);
47     for(int i=1;i<n;i*=2) {
48         int W=1, Wn;
49         if(inv) Wn=pW[n-(n/2/i)];
50         else Wn=pW[n/2/i];
51         for(int j=0;j<n;j++) {
52             if(j&i) {
53                 W=1;
54                 continue;
55             }
56             int x=A[j], y=mul(A[j+i],W);
57             A[j]=add(x,y);
58             A[j+i]=sub(x,y);
59             W=mul(W,Wn);
60         }
61     }
62     if(inv)
63         for(int i=0;i<n;i++)
64             A[i]=mul(A[i],divN);
65 }

```

MillerRabin other

```

1 /* Miller Rabin code from ioicamp */
2 ll mul(ll a, ll b, ll n) {
3     ll r = 0;
4     a %= n, b %= n;
5     while(b) {
6         if(b&1) r = (a+r>=n ? a+r-n : a+r);
7         a = (a+a>=n ? a+a-n : a+a);
8         b >>= 1;
9     }
10    return r;
11 }

```

```

12
13 ll bigmod(ll a, ll d, ll n) {
14     if(d==0) return 1LL;
15     if(d==1) return a % n;
16     return mul(bigmod(mul(a, a, n), d/2, n),
17                 d%2?a:1, n);
18 }
19 const bool PRIME = 1, COMPOSITE = 0;
20 bool miller_rabin(ll n, ll a) {
21     if(__gcd(a, n) == n) return PRIME;
22     if(__gcd(a, n) != 1) return COMPOSITE;
23     ll d = n-1, r = 0, res;
24     while(d%2==0) { ++r; d/=2; }
25     res = bigmod(a, d, n);
26     if(res == 1 || res == n-1) return PRIME;
27     while(r--) {
28         res = mul(res, res, n);
29         if(res == n-1) return PRIME;
30     }
31     return COMPOSITE;
32 }
33
34 bool isprime(ll n) {
35     if(n==1)
36         return COMPOSITE;
37     ll as[7] = {2, 325, 9375, 28178, 450775,
38                 9780504, 1795265022};
39     for(int i=0; i<7; i++)
40         if(miller_rabin(n, as[i]) == COMPOSITE)
41             return COMPOSITE;
42     return PRIME;
43 }

```

Guass

```

1 // be care of the magic number 7 & 8
2 void guass() {
3     for(int i = 0; i < 7; i++) {
4         Frac tmp = mat[i][i]; // Frac -> the
5                                 // type of data
6         for(int j = 0; j < 8; j++)
7             mat[i][j] = mat[i][j] / tmp;
8         for(int j = 0; j < 7; j++) {
9             if(i == j)
10                 continue;
11             Frac ratio = mat[j][i]; // Frac ->
12                                     // the type of data
13             for(int k = 0; k < 8; k++)
14                 mat[j][k] = mat[j][k] - ratio * mat
15                 [i][k];
16         }
17     }
18 }

```

xorFFT

```

1 //      1  1
2 // H = 1 -1
3 //      /sqrt(2)
4 vector<ll> FWHT(vector<ll> P, bool inverse) {
5     {

```

```

5     for(int len = 1; 2 * len <= SZ(P); len
6         <= 1) {
7         for(int i = 0; i < SZ(P); i += 2 * len)
8             {
9                 for(int j = 0; j < len; j++) {
10                     ll u = P[i + j];
11                     ll v = P[i + len + j];
12                     P[i + j] = u + v;
13                     P[i + len + j] = u - v;
14                 }
15             }
16         if (inverse) {
17             for (int i = 0; i < SZ(P); i++)
18                 P[i] = P[i] / SZ(P);
19         }
20     }
21 }

```

orFFT

```

1 //      1  1
2 // T = 1  0
3 //      0  1
4 //T-1= 1 -1
5 vector<ll> transform(vector<ll> P, bool
6     inverse) {
7     for(int len = 1; 2 * len <= SZ(P); len
8         <= 1) {
9         for(int i = 0; i < SZ(P); i += 2 * len)
10             {
11                 for(int j = 0; j < len; j++) {
12                     ll u = P[i + j];
13                     ll v = P[i + len + j];
14                     if (!inverse) {
15                         P[i + j] = u + v;
16                         P[i + len + j] = u;
17                     } else {
18                         P[i + j] = v;
19                         P[i + len + j] = u - v;
20                     }
21                 }
22             }
23         }
24     }
25     return P;
26 }

```

andFFT

```

1 //      0  1
2 // T = 1  1
3 //      -1  1
4 //T-1= 1  0
5 vector<ll> transform(vector<ll> P, bool
6     inverse) {
7     for(int len = 1; 2 * len <= SZ(P); len
8         <= 1) {
9         for(int i = 0; i < SZ(P); i += 2 * len)
10             {
11                 for(int j = 0; j < len; j++) {
12                     ll u = P[i + j];

```

```

10     ll v = P[i + len + j];
11     if (!inverse) {
12         P[i + j] = v;
13         P[i + len + j] = u + v;
14     } else {
15         P[i + j] = -u + v;
16         P[i + len + j] = u;
17     }
18 }
19 }
20 }
21 return P;
22 }

```

Them.

Catalan number: $C_0 = 1$, $C_{n+1} = \frac{2(2n+1)}{n+2} C_n$

flow

dinic

```

1 const int MAXV=300;
2 const int MAXE=10000;
3 const int INF=(int)1e9+10;
4 // ^ config those things
5
6 struct E {
7     int to,co;//capacity
8     E(int t=0,int c=0):to(t),co(c) {}
9 }eg[2*MAXE];
10
11 // source:0 sink:n-1
12 struct Flow {
13     VI e[MAXV];
14     int ei,v;
15     void init(int n) {
16         v=n;
17         ei=0;
18         for(int i=0;i<n;i++)
19             e[i]=VI();
20     }
21     void add(int a,int b,int c) { //a to b ,
22         maxflow=c
23         eg[ei]=E(b,c);
24         e[a].PB(ei);
25         ei++;
26         eg[ei]=E(a,0);
27         e[b].PB(ei);
28         ei++;
29     }
30     int d[MAXV],qu[MAXV],ql,qr;
31     bool BFS() {
32         memset(d,-1,v*sizeof(int));
33         ql=qr=0;
34         qu[qr++]=0;
35         d[0]=0;
36         while(ql<qr && d[v-1]==-1) {
37             int n=qu[ql++];
38             VI &v=e[n];

```

```

39         for(int i=SZ(v)-1;i>=0;i--) {
40             int u=v[i];
41             if(d[eg[u].to]==-1 && eg[u].co>0) {
42                 d[eg[u].to]=d[n]+1;
43                 qu[qr++]=eg[u].to;
44             }
45         }
46     }
47     return d[v-1]!=-1;
48 }
49 int ptr[MAXV];
50 int go(int n,int p) {
51     if(n==v-1)
52         return p;
53     VI &u=e[n];
54     int temp;
55     for(int i=ptr[n];i<SZ(u);i++) {
56         if(d[n]+1!=d[eg[u[i]].to] || eg[u[i]]
57             ].co==0)
58             continue;
59         if((temp=go(eg[u[i]].to,min(p,eg[u[i]]
60             ]).co))==0)
61             continue;
62         eg[u[i]].co-=temp;
63         eg[u[i]^1].co+=temp;
64         ptr[n]=i;
65         return temp;
66     }
67     ptr[n]=SZ(u);
68     return 0;
69 }
70 int max_flow() {
71     int ans=0,temp;
72     while(BFS()) {
73         for(int i=0;i<v;i++)
74             ptr[i]=0;
75         while((temp=go(0,INF))>0)
76             ans+=temp;
77     }
78 }flow;

```

min-cost-max-flow

```

1 typedef pair<int,ll> PIL;
2 const int MAXV=60;
3 const int MAXE=6000;
4 const int INF=(int)1e9+10;
5 const ll cINF=(ll)1e18+10;
6 // ^ config those things
7
8 struct E {
9     int to,ca,cost;//capacity, cost
10     E(int t=0,int c=0,int co=0):to(t),ca(c),
11         cost(co) {}
12 }eg[2*MAXE];
13 // source:0 sink:n-1
14 struct Flow {
15     VI e[MAXV];
16     int ei,n;
17     void init(int n_) {

```

```

18     n=n_;
19     ei=0;
20     for(int i=0;i<n;i++)
21         e[i]=VI();
22 }
23 void add(int a,int b,int c,int d) {
24     //a to b ,maxflow=c, cost=d
25     eg[ei]=E(b,c,d);
26     e[a].PB(ei);
27     ei++;
28     eg[ei]=E(a,0,-d);
29     e[b].PB(ei);
30     ei++;
31 }
32
33 PII d[MAXV]={};
34 bool inq[MAXV]={};
35 queue<int> que;
36 VI pe;
37 bool SPFA() {
38     fill(d, d+n, MP(INF,INF));
39     d[0]=MP(0,0);
40     que.push(0);
41     inq[0]=1;
42     while(!que.empty()) {
43         int v=que.front(); que.pop();
44         inq[v]=0;
45         for(int id:e[v]) {
46             if(eg[id].ca>0 && MP(d[v].F+eg[id].
47                 cost,d[v].S+1)<d[eg[id].to]) {
48                 d[eg[id].to]=MP(d[v].F+eg[id].
49                     cost,d[v].S+1);
50                 if(!inq[eg[id].to]) {
51                     que.push(eg[id].to);
52                     inq[eg[id].to]=1;
53                 }
54             }
55         }
56         return d[n-1].F<INF;
57     }
58     PIL go(ll cb=cINF) {
59         // cost_bound
60         if(!SPFA()) return MP(0,0);
61         pe.clear();
62         int fl=INF;
63         for(int v=n-1;v!=0;) {
64             for(int id:e[v]) {
65                 int u=eg[id].to;
66                 const E& t=eg[id^1];
67                 if(t.ca>0 && MP(d[u].F+t.cost,d[u].
68                     S+1)==d[v]) {
69                     fl=min(fl, t.ca);
70                     v=u;
71                     pe.PB(id^1);
72                     break;
73                 }
74             }
75         }
76         if(d[n-1].F>0) fl=min(1ll*fl, cb/d[n
77             -1].F);
78         for(int id:pe) {
79             eg[id].ca-=fl;
80             eg[id^1].ca+=fl;
81         }

```

```

79     return MP(fl, 1ll*fl*d[n-1].F);
80 }
81 PIL max_flow() {
82     PIL ans=MP(0,0),temp;
83     while((temp=go()).F>0) {
84         ans.F+=temp.F;
85         ans.S+=temp.S;
86     }
87     return ans;
88 }
89 } flow;

```

string

KMP

```

1 void KMP_build(const char *S,int *F) {
2     int p=F[0]=-1;
3     for(int i=1;S[i];i++) {
4         while(p!=-1 && S[p+1]!=S[i])
5             p=F[p];
6         if(S[p+1]==S[i])
7             p++;
8         F[i]=p;
9     }
10 }
11
12 VI KMP_match(const char *S,const int *F,
13     const char *T) {
14     VI ans;
15     int p=-1;
16     for(int i=0;T[i];i++) {
17         while(p!=-1 && S[p+1]!=T[i])
18             p=F[p];
19         if(S[p+1]==T[i])
20             p++;
21         if(!S[p+1]) {
22             ans.PB(i-p);
23             p=F[p];
24         }
25     }
26     return ans;
27 }

```

Z-value

```

1 void Z_build(const char *S,int *Z) {
2     Z[0]=0;
3     int bst=0;
4     for(int i=1;S[i];i++) {
5         if(Z[bst]+bst<i) Z[i]=0;
6         else Z[i]=min(Z[bst]+bst-i,Z[i-bst]);
7         while(S[Z[i]]==S[i+Z[i]]) Z[i]++;
8         if(Z[i]+i>Z[bst]+bst) bst=i;
9     }
10 }

```

Z-value-palindrome

```

1 // AC code of NTUJ1871
2 char in[100100];
3 char s[200100];
4 int z[200100];
5
6 int main() {
7     while(gets(in)) {
8         int len=1;
9         for(int i=0;in[i];i++) {
10             s[len++]='*';
11             s[len++]=in[i];
12         }
13         s[len]=0;
14         z[0]=0;
15         z[1]=0;
16         int bst=1;
17         for(int i=1;i<len;i++) {
18             z[i]=min(bst+z[bst]-i,z[bst+bst-i]);
19             while(s[i+z[i]+1]==s[i-z[i]-1])
20                 z[i]++;
21             if(z[i]+i>bst+z[bst])
22                 bst=i;
23         }
24         bool yes=0;
25         for(int i=3;i<len;i+=2)
26             if(z[(i+1)/2]==i/2 && z[(i+len)/2]==(
27                 len-i-1)/2)
28                 yes=1;
29         if(yes)
30             puts("www");
31         else
32             puts("vvvvvv");
33     }
34 }

```

Suffix Array($O(N\log N)$)

```

1 const int SASIZE=100020; // >= (max length
   of string + 20)
2 struct SA{
3     char S[SASIZE]; // put target string into
   S[0:(len-1)]
4     // you can change the type of S into int
   if required
5     // if the string is in int, please avoid
   number < 0
6     int R[SASIZE*2],SA[SASIZE];
7     int tR[SASIZE*2],tSA[SASIZE];
8     int cnt[SASIZE],len; // set len
   before calling build()
9     int H[SASIZE];
10
11 void build_SA() {
12     int maxR=0;
13     for(int i=0;i<len;i++)
14         R[i]=S[i];
15     for(int i=0;i<len;i++)
16         R[len+i]=-1;
17     memset(cnt,0,sizeof(cnt));
18     for(int i=0;i<len;i++)
19         maxR=max(maxR,R[i]);
20     for(int i=0;i<len;i++)

```

```

21     cnt[R[i]+1]++;
22     for(int i=1;i<=maxR;i++)
23         cnt[i]+=cnt[i-1];
24     for(int i=0;i<len;i++)
25         SA[cnt[R[i]]++]=i;
26     for(int i=1;i<len;i*=2)
27     {
28         memset(cnt,0,sizeof(int)*(maxR+10));
29         memcpy(tSA,SA,sizeof(int)*(len+10));
30         memcpy(tR,R,sizeof(int)*(len+i+10));
31         for(int j=0;j<len;j++)
32             cnt[R[j]+1]++;
33         for(int j=1;j<=maxR;j++)
34             cnt[j]+=cnt[j-1];
35         for(int j=len-i;j<len;j++)
36             SA[cnt[R[j]]++]=j;
37         for(int j=0;j<len;j++)
38         {
39             int k=tSA[j]-i;
40             if(k<0)
41                 continue;
42             SA[cnt[R[k]]++]=k;
43         }
44         int num=0;
45         maxR=0;
46         R[SA[0]]=num;
47         for(int j=1;j<len;j++)
48         {
49             if(tR[SA[j-1]]<tR[SA[j]] || tR[SA[j]
50                 -1]+i<tR[SA[j]+i])
51                 num++;
52             R[SA[j]]=num;
53             maxR=max(maxR,R[SA[j]]);
54         }
55     }
56     if (len == 1)
57         SA[0] = R[0] = 0;
58 }
59 void build_H() {
60     memset(H,0,sizeof(int)*(len+10));
61     for(int i=0;i<len;i++)
62     {
63         if(R[i]==0)
64             continue;
65         int &t=H[R[i]];
66         if(i>0)
67             t=max(0,H[R[i-1]]-1);
68         while(S[i+t]==S[SA[R[i]-1]+t]) t++;
69     }
70 }sa;

```

Suffix Array(SAIS, twt)

```

1 struct SA{
2     #define REP(i,n) for ( int i=0; i<int(n); i
   ++ )
3     #define REP1(i,a,b) for ( int i=(a); i<=int
   (b); i++ )
4     static const int MXN = 300010;
5     bool _t[MXN*2];
6     int _s[MXN*2], _sa[MXN*2], _c[MXN*2], x[
   MXN], _p[MXN], _q[MXN*2], hei[MXN], r[

```

```

    MXN];
7  int operator [] (int i){ return _sa[i]; }
8  void build(int *s, int n, int m){
9      memcpy(_s, s, sizeof(int) * n);
10     sais(_s, _sa, _p, _q, _t, _c, n, m);
11     mkhei(n);
12 }
13 void mkhei(int n){
14     REP(i,n) r[_sa[i]] = i;
15     hei[0] = 0;
16     REP(i,n) if(r[i]) {
17         int ans = i>0 ? max(hei[r[i-1]] - 1,
18             0) : 0;
19         while(_s[i+ans] == _s[_sa[r[i]-1]+ans]
20             ) ans++;
21         hei[r[i]] = ans;
22     }
23 void sais(int *s, int *sa, int *p, int *q
24     , bool *t, int *c, int n, int z){
25     bool uniq = t[n-1] = true, neq;
26     int nn = 0, nmzx = -1, *nsa = sa + n, *
27         ns = s + n, lst = -1;
28 #define MS0(x,n) memset((x),0,n*sizeof(*(x)
29     ))
30 #define MAGIC(XD) MS0(sa, n); \
31     memcpy(x, c, sizeof(int) * z); \
32     XD; \
33     memcpy(x + 1, c, sizeof(int) * (z - 1))
34     ; \
35     REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[
36         s[sa[i]-1]]++] = sa[i]-1; \
37     memcpy(x, c, sizeof(int) * z); \
38     for(int i = n - 1; i >= 0; i--) if(sa[i
39         ] && t[sa[i]-1]) sa[--x[s[sa[i]-1]]]
40         = sa[i]-1;
41     MS0(c, z);
42     REP(i,n) uniq &= ++c[s[i]] < 2;
43     REP(i,z-1) c[i+1] += c[i];
44     if (uniq) { REP(i,n) sa[--c[s[i]]] = i;
45         return; }
46     for(int i = n - 2; i >= 0; i--) t[i] =
47         (s[i]==s[i+1] ? t[i+1] : s[i]<s[i
48             +1]);
49     MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1])
50         sa[--x[s[i]]]=p[q[i]=nn++]=i);
51     REP(i, n) if (sa[i] && t[sa[i]] && !t[
52         sa[i]-1]) {
53         neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[
54             sa[i]]+1]-sa[i])*sizeof(int));
55         ns[q[lst=sa[i]]]=nmzx+=neq;
56     }
57     sais(ns, nsa, p + nn, q + n, t + n, c +
58         z, nn, nmzx + 1);
59     MAGIC(for(int i = nn - 1; i >= 0; i--)
60         sa[--x[s[p[nsa[i]]]]] = p[nsa[i]]);
61 }
62 }sa;
63
64 void suffix_array(int* ip, int len) {
65     // should padding a zero in the back
66     // s is int array, n is array length
67     // s[0..n-1] != 0, and s[n] = 0
68     // resulting SA will be length n+1
69     ip[len++] = 0;

```

```

54 sa.build(ip, len, 128);
55 // original 1-base
56 for (int i=0; i<l; i++) {
57     hei[i] = sa.hei[i + 1];
58     sa[i] = sa._sa[i + 1];
59 }
60 }

```

Aho-Corasick-2016ioicamp

```

1 // AC code of 2016ioicamp 54
2 const int MAXNM=100010;
3 int pp[MAXNM];
4
5 const int sizz=100010;
6 int nx[sizz][26],spt;
7 int fl[sizz],efl[sizz],ed[sizz];
8 int len[sizz];
9 int newnode(int len_=0) {
10     for(int i=0;i<26;i++)nx[spt][i]=0;
11     ed[spt]=0;
12     len[spt]=len_;
13     return spt++;
14 }
15 int add(char *s,int p) {
16     int l=1;
17     for(int i=0;s[i];i++) {
18         int a=s[i]-'a';
19         if(nx[p][a]==0) nx[p][a]=newnode(1);
20         p=nx[p][a];
21         l++;
22     }
23     ed[p]=1;
24     return p;
25 }
26 int q[sizz],qs,qe;
27 void make_fl(int root) {
28     fl[root]=efl[root]=0;
29     qs=qe=0;
30     q[qe++]=root;
31     for(;qs!=qe;) {
32         int p=q[qs++];
33         for(int i=0;i<26;i++) {
34             int t=nx[p][i];
35             if(t==0) continue;
36             int tmp=fl[p];
37             for(;tmp&&nx[tmp][i]==0;) tmp=fl[tmp]
38                 ];
39             fl[t]=tmp?nx[tmp][i]:root;
40             efl[t]=ed[fl[t]]?fl[t]:efl[fl[t]];
41             q[qe++]=t;
42         }
43     }
44     char s[MAXNM];
45     char a[MAXNM];
46
47     int dp[MAXNM][4];
48
49     void mmax(int &a,int b) {
50         a=max(a,b);
51     }
52

```



```

53 void match(int root) {
54     int p=root;
55     for(int i=1;s[i];i++) {
56         int a=s[i]-'a';
57         for(;p&nx[p][a]==0;p=f1[p]);
58         p=p?nx[p][a]:root;
59         for(int j=1;j<=3;j++)
60             dp[i][j]=dp[i-1][j];
61         for(int t=p;t=t=efl[t]) {
62             if(!ed[t])
63                 continue;
64             for(int j=1;j<=3;j++)
65                 mmax(dp[i][j],dp[i-len[t]][j-1]+(pp
66                     [i]-pp[i-len[t]]));
67         }
68     }
69 }
70 int main() {
71     int T;
72     scanf("%d",&T);
73     while(T--) {
74         int n,m;
75         scanf("%d%d",&n,&m);
76         scanf("%s",s+1);
77         for(int i=1;i<=n;i++)
78             scanf("%d",pp+i);
79         for(int i=1;i<=n;i++)
80             pp[i]+=pp[i-1];
81         spt=1;
82         int root=newnode();
83         for(int i=0;i<m;i++) {
84             scanf("%s",a);
85             add(a,root);
86         }
87         make_f1(root);
88         for(int i=1;i<=n;i++)
89             dp[i][1]=dp[i][2]=dp[i][3]=0;
90         match(root);
91         printf("%d\n",dp[n][3]);
92     }
93     return 0;
94 }

```

Palindrome Automaton

```

1  const int MAXN=100050;
2  char s[MAXN];
3  int n; // n: string length
4
5  typedef pair<PII,int> PD;
6  vector<PD> pal;
7
8  int ch[MAXN][26], fail[MAXN], len[MAXN],
9      cnt[MAXN];
10 int edp[MAXN];
11 int nid=1;
12 int new_node(int len_) {
13     len[nid]=len_;
14     return nid++;
15 }
16 void build_pa() {

```

```

17 int odd_root=new_node(-1);
18 int even_root=new_node(0);
19 fail[even_root]=odd_root;
20 int cur=even_root;
21 for(int i=1;i<=n;i++) {
22     while(1) {
23         if(s[i-len[cur]-1] == s[i]) break;
24         cur=fail[cur];
25     }
26     if(ch[cur][s[i]-'a']==0) {
27         int nt=ch[cur][s[i]-'a']=new_node(len
28             [cur]+2);
29         int tmp=fail[cur];
30         while(tmp && s[i-len[tmp]-1]!=s[i])
31             tmp=fail[tmp];
32         if(tmp==0) fail[nt]=even_root;
33         else {
34             assert(ch[tmp][s[i]-'a']);
35             fail[nt]=ch[tmp][s[i]-'a'];
36             edp[nt]=i;
37         }
38         cur=ch[cur][s[i]-'a'];
39         cnt[cur]++;
40     }
41     for(int i=nid-1;i>even_root;i--) {
42         cnt[fail[i]]+=cnt[i];
43         pal.PB( MP( MP(edp[i]-len[i]+1, len[i])
44             , cnt[i]) ));
45     }
46 }

```

Suffix Automaton(bcw)

```

1 // par : fail link
2 // val : a topological order ( useful for
3 // go[x] : automata edge ( x is integer in
4 // [0,26) )
5 struct SAM{
6     struct State{
7         int par, go[26], val;
8         State () : par(0), val(0){ FZ(go); }
9         State (int _val) : par(0), val(_val){
10             FZ(go); }
11 };
12 vector<State> vec;
13 int root, tail;
14 void init(int arr[], int len){
15     vec.resize(2);
16     vec[0] = vec[1] = State(0);
17     root = tail = 1;
18     for (int i=0; i<len; i++)
19         extend(arr[i]);
20 }
21 void extend(int w){
22     int p = tail, np = vec.size();
23     vec.PB(State(vec[p].val+1));
24     for ( ; p && vec[p].go[w]==0; p=vec[p].
25         par)
26         vec[p].go[w] = np;

```



```

26 if (p == 0){
27     vec[np].par = root;
28 } else {
29     if (vec[vec[p].go[w]].val == vec[p].
30         val+1){
31         vec[np].par = vec[p].go[w];
32     } else {
33         int q = vec[p].go[w], r = vec.size
34             ();
35         vec.PB(vec[q]);
36         vec[r].val = vec[p].val+1;
37         vec[q].par = vec[np].par = r;
38         for ( ; p && vec[p].go[w] == q; p=
39             vec[p].par)
40             vec[p].go[w] = r;
41     }
42 }
43 tail = np;
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }

```

graph

Bipartite matching($O(N^3)$)

```

1 // NTUJ1263
2 bool is(ll x) {
3     ll l=1,r=2000000,m;
4     while(l<=r) {
5         m=(l+r)/2;
6         if(m*m==x)
7             return 1;
8         if(m*m<x)
9             l=m+1;
10        else
11            r=m-1;
12    }
13    return 0;
14 }
15
16 VI odd,even;
17 int in[300];
18 VI e[300];
19 int match[300];
20 bool vis[300];
21
22 bool DFS(int x) {
23     vis[x]=1;
24     for(int u:e[x]) {
25         if(match[u]==-1 || (!vis[match[u]]&&DFS
26             (match[u]))) {
27             match[u]=x;
28             match[x]=u;
29             return 1;
30         }
31     }
32     return 0;
33 }
34
35 int main() {
36     int N;
37     while(scanf("%d",&N)==1) {

```

```

37     odd.clear();
38     even.clear();
39     for(int i=0;i<N;i++)
40         e[i].clear();
41     for(int i=0;i<N;i++) {
42         scanf("%d",in+i);
43         if(in[i]%2==0)
44             even.pb(i);
45         else
46             odd.pb(i);
47     }
48     for(int i:even)
49         for(int j:odd)
50             if(is(1ll*in[i]*in[i]+1ll*in[j]*in[
51                 j]) && __gcd(in[i],in[j])==1)
52                 e[i].pb(j), e[j].pb(i);
53     int ans=0;
54     fill(match,match+N,-1);
55     for(int i=0;i<N;i++)
56         if(match[i]==-1) {
57             fill(vis,vis+N,0);
58             if(DFS(i))
59                 ans++;
60         }
61     printf("%d\n",ans);
62 }
63 }

```

KM($O(N^4)$)

```

1 const int INF=1016; //> max(a[i][j])
2 const int MAXN=650;
3 int a[MAXN][MAXN]; // weight [x][y] , two
4 // set of vertex
5 int N; // two set: each set have exactly N
6 // vertex
7
8 bool DFS(int x) {
9     vis[x]=1;
10    for(int i=0;i<N;i++) {
11        if(weight[x]+weight[N+i]!=a[x][i])
12            continue;
13        vis[N+i]=1;
14        if(match[N+i]==-1 || (!vis[match[N+i]]
15            &&DFS(match[N+i]))) {
16            match[N+i]=x;
17            match[x]=N+i;
18            return 1;
19        }
20    }
21    return 0;
22 }
23
24 int KM() {
25     fill(weight, weight+N*N, 0);
26     for(int i=0;i<N;i++) {
27         for(int j=0;j<N;j++)
28             weight[i]=max(weight[i], a[i][j]);
29     }
30     fill(match, match+N*N, -1);

```

```

29 for(int u=0;u<N;u++) {
30     fill(vis, vis+N+N, 0);
31     while(!DFS(u)) {
32         int d=INF;
33         for(int i=0;i<N;i++) {
34             if(!vis[i]) continue;
35             for(int j=0;j<N;j++)
36                 if(!vis[N+j])
37                     d=min(d, weight[i]+weight[N+j]-
38                         a[i][j]);
39         }
40         for(int i=0;i<N;i++)
41             if(vis[i])
42                 weight[i]-=d;
43         for(int i=N;i<N+N;i++)
44             if(vis[i])
45                 weight[i]+=d;
46         fill(vis, vis+N+N, 0);
47     }
48     int ans=0;
49     for(int i=0;i<N+N;i++) ans+=weight[i];
50     return ans;
51 }

```

general graph matching(bcw)

```

1 #define FZ(x) memset(x,0,sizeof(x))
2 struct GenMatch { // 1-base
3     static const int MAXN = 250;
4     int V;
5     bool el[MAXN][MAXN];
6     int pr[MAXN];
7     bool inq[MAXN],inp[MAXN],inb[MAXN];
8     queue<int> qe;
9     int st,ed;
10    int nb;
11    int bk[MAXN],djs[MAXN];
12    int ans;
13    void init(int _V) {
14        V = _V;
15        FZ(el); FZ(pr);
16        FZ(inq); FZ(inp); FZ(inb);
17        FZ(bk); FZ(djs);
18        ans = 0;
19    }
20    void add_edge(int u, int v) {
21        el[u][v] = el[v][u] = 1;
22    }
23    int lca(int u,int v) {
24        memset(inp,0,sizeof(inp));
25        while(1) {
26            u = djs[u];
27            inp[u] = true;
28            if(u == st) break;
29            u = bk[pr[u]];
30        }
31        while(1) {
32            v = djs[v];
33            if(inp[v]) return v;
34            v = bk[pr[v]];
35        }
36        return v;

```

```

37    }
38    void upd(int u) {
39        int v;
40        while(djs[u] != nb) {
41            v = pr[u];
42            inb[djs[u]] = inb[djs[v]] = true;
43            u = bk[v];
44            if(djs[u] != nb) bk[u] = v;
45        }
46    }
47    void blo(int u,int v) {
48        nb = lca(u,v);
49        memset(inb,0,sizeof(inb));
50        upd(u); upd(v);
51        if(djs[u] != nb) bk[u] = v;
52        if(djs[v] != nb) bk[v] = u;
53        for(int tu = 1; tu <= V; tu++)
54            if(inb[djs[tu]]) {
55                djs[tu] = nb;
56                if(!inq[tu]){
57                    qe.push(tu);
58                    inq[tu] = 1;
59                }
60            }
61    }
62    void flow() {
63        memset(inq,false,sizeof(inq));
64        memset(bk,0,sizeof(bk));
65        for(int i = 1; i <= V;i++)
66            djs[i] = i;
67
68        while(qe.size()) qe.pop();
69        qe.push(st);
70        inq[st] = 1;
71        ed = 0;
72        while(qe.size()) {
73            int u = qe.front(); qe.pop();
74            for(int v = 1; v <= V; v++)
75                if(el[u][v] && (djs[u] != djs[v])
76                    && (pr[u] != v)) {
77                    if((v == st) || ((pr[v] > 0) &&
78                        bk[pr[v]] > 0))
79                        blo(u,v);
80                    else if(bk[v] == 0) {
81                        bk[v] = u;
82                        if(pr[v] > 0) {
83                            if(!inq[pr[v]]) qe.push(pr[v]);
84                        } else {
85                            ed = v;
86                            return;
87                        }
88                    }
89                }
90    }
91    void aug() {
92        int u,v,w;
93        u = ed;
94        while(u > 0) {
95            v = bk[u];
96            w = pr[v];
97            pr[v] = u;
98            pr[u] = v;
99            u = w;

```

```

99     }
100 }
101 int solve() {
102     memset(pr,0,sizeof(pr));
103     for(int u = 1; u <= V; u++)
104         if(pr[u] == 0) {
105             st = u;
106             flow();
107             if(ed > 0) {
108                 aug();
109                 ans ++;
110             }
111         }
112     return ans;
113 }
114 } gm;

```

Max clique(bcw)

```

1 class MaxClique {
2 public:
3     static const int MV = 210;
4
5     int V;
6     int el[MV][MV/30+1];
7     int dp[MV];
8     int ans;
9     int s[MV][MV/30+1];
10    vector<int> sol;
11
12    void init(int v) {
13        V = v; ans = 0;
14        FZ(el); FZ(dp);
15    }
16
17    /* Zero Base */
18    void addEdge(int u, int v) {
19        if(u > v) swap(u, v);
20        if(u == v) return;
21        el[u][v/32] |= (1<<(v%32));
22    }
23
24    bool dfs(int v, int k) {
25        int c = 0, d = 0;
26        for(int i=0; i<(V+31)/32; i++) {
27            s[k][i] = el[v][i];
28            if(k != 1) s[k][i] &= s[k-1][i];
29            c += __builtin_popcount(s[k][i]);
30        }
31        if(c == 0) {
32            if(k > ans) {
33                ans = k;
34                sol.clear();
35                sol.push_back(v);
36                return 1;
37            }
38            return 0;
39        }
40        for(int i=0; i<(V+31)/32; i++) {
41            for(int a = s[k][i]; a ; d++) {
42                if(k + (c-d) <= ans) return 0;
43                int lb = a&(-a), lg = 0;
44                a ^= lb;

```

```

45        while(lb!=1) {
46            lb = (unsigned int)(lb) >> 1;
47            lg ++;
48        }
49        int u = i*32 + lg;
50        if(k + dp[u] <= ans) return 0;
51        if(dfs(u, k+1)) {
52            sol.push_back(v);
53            return 1;
54        }
55    }
56 }
57 return 0;
58 }
59
60 int solve() {
61     for(int i=V-1; i>=0; i--) {
62         dfs(i, 1);
63         dp[i] = ans;
64     }
65     return ans;
66 }
67 };

```

EdgeBCC

```

1 const int MAXN=1010;
2 const int MAXM=5010;
3 VI e[MAXN];
4 int low[MAXN],lv1[MAXN],bel[MAXN];
5 bool vis[MAXN];
6 int cnt;
7 VI st;
8 void DFS(int x,int l,int p) {
9     st.PB(x);
10    vis[x]=1;
11    low[x]=lv1[x]=1;
12    bool top=0;
13    for(int u:e[x]) {
14        if(u==p && !top) {
15            top=1;
16            continue;
17        }
18        if(!vis[u]) {
19            DFS(u,l+1,x);
20        }
21        low[x]=min(low[x],low[u]);
22    }
23    if(x==1 || low[x]==1) {
24        while(st.back()!=x) {
25            bel[st.back()]=cnt;
26            st.pop_back();
27        }
28        bel[st.back()]=cnt;
29        st.pop_back();
30        cnt++;
31    }
32 }
33 int main() {
34     int T;
35     scanf("%d",&T);
36     while(T--) {
37         int N,M,a,b;

```

```

38     scanf("%d%d",&N,&M);
39     fill(vis,vis+N+1,0);
40     for(int i=1;i<=N;i++)
41         e[i].clear();
42     while(M--) {
43         scanf("%d%d",&a,&b);
44         e[a].PB(b);
45         e[b].PB(a);
46     }
47     cnt=0;
48     DFS(1,0,-1);
49     /******/
50 }
51 return 0;
52 }

```

VerticeBCC

```

1  const int MAXN=10000;
2  const int MAXE=100000;
3
4  VI e[MAXN+10];
5  vector<PII> BCC[MAXE];
6  int bccnt;
7  vector<PII> st;
8  bool vis[MAXN+10];
9  int low[MAXN+10],level[MAXN+10];
10
11 void DFS(int x,int p,int l) {
12     vis[x]=1;
13     level[x]=low[x]=1;
14     for(int u:e[x]) {
15         if(u==p)
16             continue;
17         if(vis[u]) {
18             if(level[u]<l) {
19                 st.PB(MP(x,u));
20                 low[x]=min(low[x],level[u]);
21             }
22         }
23         else {
24             st.PB(MP(x,u));
25             DFS(u,x,l+1);
26             if(low[u]>=1) {
27                 PII t=st.back();
28                 st.pop_back();
29                 while(t!=MP(x,u)) {
30                     BCC[bccnt].PB(t);
31                     t=st.back();
32                     st.pop_back();
33                 }
34                 BCC[bccnt].PB(t);
35                 bccnt++;
36             }
37             low[x]=min(low[x],low[u]);
38         }
39     }
40 }
41
42 int main() {
43     int T,N,M;
44     scanf("%d",&T);
45     while(T--){

```

```

46     scanf("%d%d",&N,&M);
47     for(int i=0;i<N;i++)
48         e[i].clear();
49     int cnt=0;
50     while(1) {
51         int x,y;
52         scanf("%d%d",&x,&y);
53         if(x==-1 && y==-1)
54             break;
55         cnt++;
56         e[x].PB(y);
57         e[y].PB(x);
58     }
59     for(int i=0;i<N;i++) { // no multi-edge
60         sort(ALL(e[i]));
61         e[i].erase(unique(ALL(e[i])),e[i].end
62             ());
63     }
64     fill(vis,vis+N,0);
65     while(bccnt)
66         BCC[--bccnt].clear();
67     DFS(0,-1,0);
68     /****/
69 }
70 return 0;
71 }

```

Dominating Tree

```

1  const int MAXN = 200000 + 10;
2
3  VI e[MAXN], re[MAXN];
4  int par[MAXN], num[MAXN], t, rn[MAXN];
5  int sd[MAXN], id[MAXN];
6  PII p[MAXN];
7  VI sdom_at[MAXN];
8
9  void dfs(int u) {
10     num[u] = ++t;
11     rn[t] = u;
12     for(int v : e[u]) {
13         if(num[v]) continue;
14         par[v] = u;
15         dfs(v);
16     }
17 }
18
19 void LINK(int x, int y) {
20     p[x].F = y;
21     if(sd[y] < sd[p[x].S]) p[x].S = y;
22 }
23
24 int EVAL(int x) {
25     if(p[p[x].F].F != p[x].F) {
26         int w = EVAL(p[x].F);
27         if(sd[w] < sd[p[x].S]) p[x].S = w;
28         p[x].F = p[p[x].F].F;
29     }
30     return p[x].S;
31 }
32
33 void DominatingTree(int n) {
34     // 1-indexed

```

```

35 par[1] = 1;
36 fill(num, num+n+1, 0);
37 fill(rn, rn+n+1, 0);
38 t = 0;
39 dfs(1);
40
41 for(int i=1; i<=n; i++) {
42     p[i] = MP(i, i);
43 }
44 for(int i=1; i<=n; i++) {
45     sd[i] = (num[i] ? num[i] : MAXN+10);
46     id[i] = i;
47 }
48 for(int i=n; i>1; i--) {
49     int v = rn[i];
50     if(!v) continue;
51     for(int u : re[v]) {
52         int w = EVAL(u);
53         sd[v] = min(sd[v], sd[w]);
54     }
55     sdom_at[rn[sd[v]]].PB(v);
56     LINK(v, par[v]);
57
58     for(int w : sdom_at[par[v]]) {
59         int u = EVAL(w);
60         id[w] = (sd[u]<sd[w] ? u : par[v]);
61     }
62     sdom_at[par[v]].clear();
63 }
64
65 for(int i=2; i<=n; i++) {
66     int v = rn[i];
67     if(!v) break;
68     if(id[v] != rn[sd[v]]) id[v] = id[id[v]
69     ];
70 }

```

Min Cut

```

1 const int MAXN = 500 + 50;
2 int w[MAXN][MAXN];
3 ll cost[MAXN];
4 bool out[MAXN];
5
6 int N, M;
7 ll go(int V) {
8     fill(cost, cost + N, 0);
9     cost[0] = -1;
10    int nxt = 0, lst = 0;
11    for(int i = 0; i < N; i++) {
12        if(out[i]) continue;
13        cost[i] += w[0][i];
14        if(cost[i] > cost[nxt]) nxt = i;
15    }
16    while(V > 2) {
17        int u = nxt;
18        if(cost[u] <= 0) return 0;
19        cost[u] = -1;
20        for(int i = 0; i < N; i++) {
21            if(cost[i] == -1 || out[i]) continue;
22            cost[i] += w[u][i];
23            if(cost[i] > cost[nxt]) nxt = i;

```

```

24    }
25    V--;
26    lst = u;
27    }
28    if(cost[nxt] <= 0) return 0;
29    out[nxt] = true;
30    for(int i = 0; i < N; i++)
31        w[i][lst] = w[lst][i] = (w[lst][i] + w[
32            nxt][i]);
33    return cost[nxt];
34 }
35
36 ll min_cut() {
37     fill(out, out + N, false);
38     ll res = go(N);
39     for(int v = N - 1; v > 1; v--) {
40         res = min(go(v), res);
41         if(res == 0) return res;
42     }
43     return res;
44 }

```

DMST with sol(bcw)

```

1 const int INF = 1029384756;
2
3 struct edge_t{
4     int u,v,w;
5     set< pair<int,int> > add, sub;
6     edge_t() : u(-1), v(-1), w(0) {}
7     edge_t(int _u, int _v, int _w) {
8         u = _u; v = _v; w = _w;
9         add.insert({u, v});
10    }
11    edge_t& operator += (const edge_t& obj) {
12        w += obj.w;
13        FOR (it, obj.add) {
14            if (!sub.count(*it)) add.insert(*it);
15            else sub.erase(*it);
16        }
17        FOR (it, obj.sub) {
18            if (!add.count(*it)) sub.insert(*it);
19            else add.erase(*it);
20        }
21        return *this;
22    }
23    edge_t& operator -= (const edge_t& obj) {
24        w -= obj.w;
25        FOR (it, obj.sub) {
26            if (!sub.count(*it)) add.insert(*it);
27            else sub.erase(*it);
28        }
29        for (auto it : obj.add) {
30            if (!add.count(it)) sub.insert(it);
31            else add.erase(it);
32        }
33        return *this;
34    }
35 }eg[MAXN*MAXN],prv[MAXN],EDGE_INF(-1,-1,INF);
36 int N,M;
37 int cid,incyc[MAXN],contracted[MAXN];
38 vector<int> E[MAXN];
39

```

```

40 edge_t dmst(int rt){
41     edge_t cost;
42     for (int i=0; i<N; i++){
43         contracted[i] = incyc[i] = 0;
44         prv[i] = EDGE_INF;
45     }
46     cid = 0;
47     int u,v;
48     while (true){
49         for (v=0; v<N; v++){
50             if (v != rt && !contracted[v] && prv[
51                 v].w == INF) break;
52             if (v >= N) break; // end
53             for (int i=0; i<M; i++){
54                 if (eg[i].v == v && eg[i].w < prv[v].
55                     w)
56                     prv[v] = eg[i];
57             }
58             if (prv[v].w == INF) // not connected
59                 return EDGE_INF;
60             cost += prv[v];
61             for (u=prv[v].u; u!=v && u!=-1; u=prv[u
62                 ].u);
63             if (u == -1) continue;
64             incyc[v] = ++cid;
65             for (u=prv[v].u; u!=v; u=prv[u].u){
66                 contracted[u] = 1;
67                 incyc[u] = cid;
68             }
69             for (int i=0; i<M; i++){
70                 if (incyc[eg[i].u] != cid && incyc[eg
71                     [i].v] == cid){
72                     eg[i] -= prv[eg[i].v];
73                 }
74             }
75             for (int i=0; i<M; i++){
76                 if (incyc[eg[i].u] == cid) eg[i].u =
77                     v;
78                 if (incyc[eg[i].v] == cid) eg[i].v =
79                     v;
80                 if (eg[i].u == eg[i].v) eg[i--] = eg
81                     [--M];
82             }
83             for (int i=0; i<N; i++){
84                 if (contracted[i]) continue;
85                 if (prv[i].u>=0 && incyc[prv[i].u] ==
86                     cid)
87                     prv[i].u = v;
88             }
89             prv[v] = EDGE_INF;
90         }
91         return cost;
92     }
93 }

94 void solve(){
95     edge_t cost = dmst(0);
96     for (auto it : cost.add){ // find a
97         solution
98         E[it.F].PB(it.S);
99         prv[it.S] = edge_t(it.F,it.S,0);
100     }
101 }

```

Them.

- 1 1. Max (vertex) independent set = Max clique on Complement graph
- 2 2. Min vertex cover = $|V|$ - Max independent set
- 3 3. On bipartite: Min vertex cover = Max Matching(edge independent)
- 4 4. Any graph with no isolated vertices: Min edge cover + Max Matching = $|V|$

data structure

Treap

```

1 const int N = ;
2 struct Treap {
3     static Treap mem[N], *pmem;
4     int sz, pri;
5     ll val, sum, add;
6     Treap *l, *r;
7     Treap() {}
8     Treap(ll _val):
9         l(NULL), r(NULL), sz(1), pri(rand()),
10         val(_val), sum(_val), add(0) {}
11 } Treap::mem[N], *Treap::pmem = Treap::mem;
12 Treap* make(ll val) {
13     return new (Treap::pmem++) Treap(val);
14 }
15 inline int sz(Treap *t) {
16     return t ? t->sz : 0;
17 }
18 inline ll sum(Treap *t) {
19     return t ? t->sum + t->add * sz(t) : 0;
20 }
21 inline void add(Treap *t, ll x) {
22     t->add += x;
23 }
24 void push(Treap *t) {
25     t->val += t->add;
26     if(t->l) t->l->add += t->add;
27     if(t->r) t->r->add += t->add;
28     t->add = 0;
29 }
30 void pull(Treap *t) {
31     t->sum = sum(t->l) + sum(t->r) + t->val;
32     t->sz = sz(t->l) + sz(t->r) + 1;
33 }
34 Treap* merge(Treap *a, Treap *b) {
35     if(!a || !b) return a ? a : b;
36     else if(a->pri > b->pri) {
37         push(a);
38         a->r = merge(a->r, b);
39         pull(a);
40         return a;
41     }
42     else {
43         push(b);
44         b->l = merge(a, b->l);
45         pull(b);
46         return b;
47     }
48 }

```

```

46 }
47 }
48 void split(Treap* t, int k, Treap *&a,
    Treap *&b) {
49     if(!t) a = b = NULL;
50     else if(sz(t->l) < k) {
51         a = t;
52         push(a);
53         split(t->r, k - sz(t->l) - 1, a->r, b);
54         pull(a);
55     }
56     else {
57         b = t;
58         push(b);
59         split(t->l, k, a, b->l);
60         pull(b);
61     }
62 }

```

copy on write treap

```

1 #include <cstdlib>
2 #include <cstdio>
3 #include <algorithm>
4 #include <climits>
5 #include <cstring>
6
7 using namespace std;
8
9 const int N = 1000000 + 10;
10
11 struct Treap {
12     char val;
13     int sz, refs;
14     Treap *l, *r;
15
16     Treap() {}
17     Treap(char _val):
18         val(_val), sz(1), refs(0), l(NULL),
19         r(NULL) {}
20 };
21 Treap* make(Treap* t) {
22     return new Treap(*t);
23 }
24
25 Treap* make(char _val) {
26     return new Treap(_val);
27 }
28
29 void print_ref(Treap* t) {
30     if(!t) return;
31     print_ref(t->l);
32     printf("%d ", t->refs);
33     print_ref(t->r);
34 }
35
36 void print(Treap* t) {
37     if(!t) return;
38     print(t->l);
39     putchar(t->val);
40     print(t->r);
41 }

```

```

42
43 void takeRef(Treap* t) {
44     if(t) t->refs++;
45 }
46
47 void dropRef(Treap* t) {
48     if(t) {
49         char c = t->val;
50         t->refs--;
51         if(t->refs <= 0) {
52             dropRef(t->l);
53             dropRef(t->r);
54             delete t;
55         }
56     }
57 }
58
59 int sz(Treap* t) {
60     return t ? t->sz : 0;
61 }
62
63 int rnd(int m) {
64     static int x = 851025;
65     return (x = (x*0xdefaced+1) & INT_MAX)
66         % m;
67 }
68
69 void pull(Treap* t) {
70     t->sz = sz(t->l) + sz(t->r) + 1;
71 }
72
73 Treap* merge(Treap* a, Treap* b) {
74     if(!a || !b) {
75         Treap* t = a ? make(a) : make(b);
76         t->refs = 0;
77         takeRef(t->l);
78         takeRef(t->r);
79         return t;
80     }
81
82     Treap* t;
83     if(rnd(a->sz+b->sz) < a->sz) {
84         t = make(a);
85         t->refs = 0;
86         t->r = merge(a->r, b);
87         takeRef(t->l);
88         takeRef(t->r);
89     }
90     else {
91         t = make(b);
92         t->refs = 0;
93         t->l = merge(a, b->l);
94         takeRef(t->l);
95         takeRef(t->r);
96     }
97
98     pull(t);
99     return t;
100 }
101
102 void split(Treap* t, int k, Treap* &a,
    Treap* &b) {
103     if(!t) a = b = NULL;
104     else if(sz(t->l) < k) {
105         a = make(t);

```



```

105     a->refs = 0;
106     split(a->r, k-sz(t->l)-1, a->r, b);
107     takeRef(a->l);
108     takeRef(a->r);
109     pull(a);
110 }
111 else {
112     b = make(t);
113     b->refs = 0;
114     split(b->l, k, a, b->l);
115     takeRef(b->l);
116     takeRef(b->r);
117     pull(b);
118 }
119 }
120
121 void print_inorder(Treap* t) {
122     if(!t) return;
123     putchar(t->val);
124     print_inorder(t->l);
125     print_inorder(t->r);
126 }
127
128 char s[N];
129
130 int main() {
131     int m;
132     scanf("%d", &m);
133     scanf("%s", s);
134     int n = strlen(s);
135     int q;
136     scanf("%d", &q);
137
138     Treap* t = NULL;
139     for(int i = 0; i < n; i++) {
140         Treap *a = t, *b = make(s[i]);
141         t = merge(a, b);
142         dropRef(a);
143         dropRef(b);
144     }
145
146     while(q--) {
147         int l, r, x;
148         scanf("%d%d%d", &l, &r, &x);
149         r++;
150
151         Treap *a, *b, *c, *d;
152         a = b = c = d = NULL;
153         split(t, l, a, b); dropRef(a);
154         split(b, r-l, c, d); dropRef(b);
155         dropRef(d);
156         split(t, x, a, b); dropRef(t);
157
158         Treap* t2 = merge(c, b); dropRef(b);
159         dropRef(c);
160
161         t = merge(a, t2); dropRef(a); dropRef(t2);
162         if(t->sz > m) {
163             Treap* t2 = NULL;
164             split(t, m, t2, a); dropRef(a);
165             dropRef(t);
166             t = t2;
167         }
168     }
169 }

```

```

print(t);
putchar('\n');

return 0;
}

```

copy on write segment tree

```

1 const int N = ;
2 const int Q = ;
3 struct Seg {
4     static Seg mem[N*80], *pmem;
5     int val;
6     Seg *tl, *tr;
7     Seg() :
8         tl(NULL), tr(NULL), val(0) {}
9     Seg* init(int l, int r) {
10         Seg* t = new (pmem++) Seg();
11         if(l != r) {
12             int m = (l+r)/2;
13             t->tl = init(l, m);
14             t->tr = init(m+1, r);
15         }
16         return t;
17     }
18     Seg* add(int k, int l, int r) {
19         Seg* _t = new (pmem++) Seg(*this);
20         if(l==r) {
21             _t->val++;
22             return _t;
23         }
24         int m = (l+r)/2;
25         if(k <= m) _t->tl = tl->add(k, l, m);
26         else _t->tr = tr->add(k, m+1, r);
27         _t->val = _t->tl->val + _t->tr->val;
28         return _t;
29     }
30 } Seg::mem[N*80], *Seg::pmem = mem;
31
32 int query(Seg* ta, Seg* tb, int k, int l,
33     int r) {
34     if(l == r) return l;
35     int m = (l+r)/2;
36     int a = ta->tl->val;
37     int b = tb->tl->val;
38     if(b-a >= k) return query(ta->tl, tb->tl,
39         k, l, m);
40     else return query(ta->tr, tb->tr, k
41         -(b-a), m+1, r);
42 }

```

Treap+(HOJ 92)

```

1 const int INF = 103456789;
2 struct Treap {
3     int pri, sz, val, chg, rev, sum, lsum,
4         rsum, mx_sum;
5     Treap *l, *r;
6     Treap() {}

```

```

7   Treap(int _val) :
8       pri(rand()), sz(1), val(_val), chg(
        INF), rev(0), sum(_val), lsum(
        _val), rsum(_val), mx_sum(_val),
        l(NULL), r(NULL) {}
9   };
10  int sz(Treap* t) {return t ? t->sz : 0;}
11  int sum(Treap* t) {
12      if(!t) return 0;
13      if(t->chg == INF) return t->sum;
14      else return t->chg*t->sz;
15  }
16  int lsum(Treap* t) {
17      if(!t) return -INF;
18      if(t->chg != INF) return max(t->chg,
        (t->chg)*(t->sz));
19      if(t->rev) return t->rsum;
20      return t->lsum;
21  }
22  int rsum(Treap* t) {
23      if(!t) return -INF;
24      if(t->chg != INF) return max(t->chg,
        (t->chg)*(t->sz));
25      if(t->rev) return t->lsum;
26      return t->rsum;
27  }
28  int mx_sum(Treap* t) {
29      if(!t) return -INF;
30      if(t->chg != INF) return max(t->chg,
        (t->chg)*(t->sz));
31      return t->mx_sum;
32  }
33  void push(Treap* t) {
34      if(t->chg != INF) {
35          t->val = t->chg;
36          t->sum = (t->sz) * (t->chg);
37          t->lsum = t->rsum = t->mx_sum = max
            (t->sum, t->val);
38          if(t->l) t->l->chg = t->chg;
39          if(t->r) t->r->chg = t->chg;
40          t->chg = INF;
41      }
42      if(t->rev) {
43          swap(t->l, t->r);
44          if(t->l) t->l->rev ^= 1;
45          if(t->r) t->r->rev ^= 1;
46          t->rev = 0;
47      }
48  }
49  void pull(Treap* t) {
50      t->sz = sz(t->l)+sz(t->r)+1;
51      t->sum = sum(t->l)+sum(t->r)+t->val;
52      t->lsum = max(lsum(t->l), sum(t->l)+max
        (0, lsum(t->r))+t->val);
53      t->rsum = max(rsum(t->r), sum(t->r)+max
        (0, rsum(t->l))+t->val);
54      t->mx_sum = max(max(mx_sum(t->l),
        mx_sum(t->r)), max(0, rsum(t->l))+
        max(0, lsum(t->r))+t->val);
55  }
56  Treap* merge(Treap* a, Treap* b) {
57      if(!a || !b) return a ? a : b;
58      if(a->pri > b->pri) {
59          push(a);
60          a->r = merge(a->r, b);
61          pull(a);
62          return a;
63      }
64      else {
65          push(b);
66          b->l = merge(a, b->l);
67          pull(b);
68          return b;
69      }
70  }
71  void split(Treap* t, int k, Treap* &a,
        Treap* &b) {
72      if(!t) {
73          a = b = NULL;
74          return ;
75      }
76      push(t);
77      if(sz(t->l) < k) {
78          a = t;
79          push(a);
80          split(t->r, k-sz(t->l)-1, a->r, b);
81          pull(a);
82      }
83      else {
84          b = t;
85          push(b);
86          split(t->l, k, a, b->l);
87          pull(b);
88      }
89  }
90  void del(Treap* t) {
91      if(!t) return;
92      del(t->l);
93      del(t->r);
94      delete t;
95  }

```

Leftist Tree

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct Left {
5      Left *l,*r;
6      int v,h;
7      Left(int v_) : v(v_), h(1), l(0), r(0) {}
8  };
9
10 int height(Left *p) { return p ? p->h : 0
    ; }
11
12 Left* combine(Left *a,Left *b) {
13     if(!a || !b) return a ? a : b ;
14     Left *p ;
15     if( a->v > b->v) {
16         p = a;
17         p->r = combine( p->r , b );
18     }
19     else {
20         p = b;
21         p->r = combine( p->r , a );
22     }
23     if( height( p->l ) < height( p->r ) )

```

```

24     swap( p->l , p->r );
25     p->h = min( height( p->l ) , height( p->r
        ) ) + 1;
26     return p;
27 }
28 Left *root;
29
30 void push(int v) {
31     Left *p = new Left(v);
32     root = combine( root , p );
33 }
34 int top() { return root? root->v : -1; }
35 void pop() {
36     if(!root) return;
37     Left *a = root->l , *b = root->r ;
38     delete root;
39     root = combine( a , b );
40 }
41 void clear(Left* &p) {
42     if(!p)
43         return;
44     if(p->l) clear(p->l);
45     if(p->r) clear(p->r);
46     delete p;
47     p = 0 ;
48 }
49
50 int main() {
51     while (T--) {
52         clear(root);
53         /** do something **/
54     }
55     return 0;
56 }

```

Link Cut Tree

```

1  const int MAXN = ;
2  struct SplayTree {
3      int val, mx, ch[2], pa;
4      bool rev;
5      void init() {
6          val = mx = -1;
7          rev = false;
8          pa = ch[0] = ch[1] = 0;
9      }
10 } node[MAXN*2];
11 inline bool isroot(int x) {
12     return node[node[x].pa].ch[0]!=x && node[
        node[x].pa].ch[1]!=x;
13 }
14 inline void pull(int x) {
15     node[x].mx = max(node[x].val, max(node[
        node[x].ch[0]].mx, node[node[x].ch
        [1]].mx));
16 }
17 inline void push(int x) {
18     if(node[x].rev) {
19         node[node[x].ch[0]].rev ^= 1;
20         node[node[x].ch[1]].rev ^= 1;
21         swap(node[x].ch[0], node[x].ch[1]);
22         node[x].rev ^= 1;
23     }
24 }
25 void push_all(int x) {
26     if(!isroot(x)) push_all(node[x].pa);
27     push(x);
28 }
29 inline void rotate(int x) {
30     int y = node[x].pa, z = node[y].pa, d =
        node[y].ch[1]==x;
31     node[x].pa = z;
32     if(!isroot(y)) node[z].ch[node[z].ch
        [1]==y] = x;
33     node[y].ch[d] = node[x].ch[d^1];
34     node[node[x].ch[d^1]].pa = y;
35     node[x].ch[!d] = y;
36     node[y].pa = x;
37     pull(y);
38     pull(x);
39 }
40 void splay(int x) {
41     push_all(x);
42     while(!isroot(x)) {
43         int y = node[x].pa;
44         if(!isroot(y)) {
45             int z = node[y].pa;
46             if((node[z].ch[1]==y) ^ (node[y].ch
                [1]==x)) rotate(y);
47             else rotate(x);
48         }
49         rotate(x);
50     }
51 }
52 inline int access(int x) {
53     int last = 0;
54     while(x) {
55         splay(x);
56         node[x].ch[1] = last;
57         pull(x);
58         last = x;
59         x = node[x].pa;
60     }
61     return last;
62 }
63 inline void make_root(int x) {
64     node[access(x)].rev ^= 1;
65     splay(x);
66 }
67 inline void link(int x, int y) {
68     make_root(x);
69     node[x].pa = y;
70 }
71 inline void cut(int x, int y) {
72     make_root(x);
73     access(y);
74     splay(y);
75     node[y].ch[0] = 0;
76     node[x].pa = 0;
77 }
78 inline void cut_parent(int x) {
79     x = access(x);
80     splay(x);
81     node[node[x].ch[0]].pa = 0;
82     node[x].ch[0] = 0;
83     pull(x);
84 }
85 inline int find_root(int x) {

```

```

86 x = access(x);
87 while(node[x].ch[0]) x = node[x].ch[0];
88 splay(x);
89 return x;
90 }
91 int find_mx(int x) {
92     if(node[x].val == node[x].mx) return x;
93     return node[node[x].ch[0]].mx == node[x].mx
        ? find_mx(node[x].ch[0]) : find_mx(
            node[x].ch[1]);
94 }
95 inline void change(int x,int b){
96     splay(x);
97     node[x].data=b;
98     up(x);
99 }
100 inline int query_lca(int u,int v){
101     /*retrun: sum of weight of vertices on the
        chain (u->v)
102     sum: total weight of the subtree
103     data: weight of the vertex */
104     access(u);
105     int lca=access(v);
106     splay(u);
107     if(u==lca){
108         return node[lca].data+node[node[lca].ch
            [1]].sum;
109     }else{
110         return node[lca].data+node[node[lca].ch
            [1]].sum+node[u].sum;
111     }
112 }

```

Heavy Light Decomposition

```

1 const int MAXN = 10000 + 10;
2 vector<PII> e[MAXN];
3 int val[MAXN];
4 int sz[MAXN], max_son[MAXN], p[MAXN], dep[
    MAXN];
5 int link[MAXN], link_top[MAXN], cnt;
6 void find_max_son(int u) {
7     sz[u] = 1;
8     max_son[u] = -1;
9     for(int i=0; i<SZ(e[u]); i++) {
10         PII tmp = e[u][i];
11         int v = tmp.F;
12         if(v == p[u]) continue;
13
14         p[v] = u;
15         dep[v] = dep[u]+1;
16         val[v] = tmp.S;
17         find_max_son(v);
18         if(max_son[u]<0 || sz[v]>sz[ max_son[u]
            ]) max_son[u] = v;
19         sz[u] += sz[v];
20     }
21 }
22 void build_link(int u, int top) {
23     link[u] = ++cnt;
24     link_top[u] = top;
25     if(max_son[u] > 0) build_link(max_son[u]
        ], top);

```

```

26 for(int i=0; i<SZ(e[u]); i++) {
27     PII tmp = e[u][i];
28     int v = tmp.F;
29     if(v==p[u] || v==max_son[u]) continue;
30     build_link(v, v);
31 }
32 }
33 int query(int a, int b) {
34     int res = -1;
35     int ta = link_top[a], tb = link_top[b];
36     while(ta != tb) {
37         if(dep[ta] < dep[tb]) {
38             swap(a, b);
39             swap(ta, tb);
40         }
41         res = max(res, seg->qry(link[ta], link[
            a], 1, cnt));
42         ta = link_top[a=p[ta]];
43     }
44     if(a != b) {
45         if(dep[a] > dep[b]) swap(a, b);
46         a = max_son[a];
47         res = max(res, seg->qry(link[a], link[b
            ], 1, cnt));
48     }
49     return res;
50 }

```

Disjoint Sets + offline skill

```

1 const int MAXN = ;
2 bool q[MAXN];
3 struct DisJointSet {
4     int p[MAXN], sz[MAXN], gps;
5     vector<pair<int*, int> > h;
6     VI sf;
7     void init(int n) {
8         for(int i=1; i<=n; i++) {
9             p[i] = i;
10            sz[i] = 1;
11        }
12        gps = n;
13    }
14    void assign(int *k, int v) {
15        h.PB(MP(k, *k));
16        *k = v;
17    }
18    void save() {
19        sf.PB(SZ(h));
20    }
21    void load() {
22        int last = sf.back(); sf.pop_back();
23        while(SZ(h) != last) {
24            auto x = h.back(); h.pop_back();
25            *x.F = x.S;
26        }
27    }
28    int find(int x) {
29        return x==p[x] ? x : find(p[x]);
30    }
31    void uni(int x, int y) {
32        x = find(x), y = find(y);
33        if(x == y) return ;

```

```

34     if(sz[x] < sz[y]) swap(x, y);
35     assign(&sz[x], sz[x]+sz[y]);
36     assign(&p[y], x);
37     assign(&gps, gps-1);
38 }
39 } djs;
40 struct Seg {
41     vector<PII> es;
42     Seg *tl, *tr;
43     Seg() {}
44     Seg(int l, int r) {
45         if(l == r) tl = tr = NULL;
46         else {
47             int m = (l+r) / 2;
48             tl = new Seg(l, m);
49             tr = new Seg(m+1, r);
50         }
51     }
52     // add an edge e from time a to time b
53     void add(int a, int b, PII e, int l, int
54         r) {
55         if(a <= l && r <= b) es.PB(e);
56         else if(b < l || r < a) return ;
57         else {
58             int m = (l+r) / 2;
59             tl->add(a, b, e, l, m);
60             tr->add(a, b, e, m+1, r);
61         }
62     }
63     void solve(int l, int r) {
64         djs.save();
65         for(auto p : es) djs.uni(p.F, p.S);
66         if(l == r) {
67             if(q[l]); // answer the query here
68         }
69         else {
70             int m = (l+r) / 2;
71             tl->solve(l, m);
72             tr->solve(m+1, r);
73         }
74         djs.load();
75     }
};

```

2D Segment Tree

```

1 struct Seg1D {
2     Seg1D *tl, *tr;
3     ll val;
4     // ll tmp;
5     //int _x, _y;
6     Seg1D() :
7         tl(NULL), tr(NULL), val(0), tmp(-1), _x
8         (-1), _y(-1) {}
9     ll query1D(int x1, int x2, int y1, int y2
10         , int l, int r) {
11         /*
12         if no Brian improvement, dont need to
13         pass x1 and x2
14         if(tmp >= 0) {
15             if(x1<=_x&&_x<=x2 && y1<=_y&&_y<=y2)
16                 return tmp;
17             else return 0;
18         }
19         */
20     }
};

```

```

14     }
15     /*
16     if(y1 <= l && r <= y2) return val;
17     else if(r < y1 || y2 < l) return 0;
18     else {
19         int m = (l+r)/2;
20         ll a = tl ? tl->query1D(x1, x2, y1,
21             y2, l, m) : 0;
22         ll b = tr ? tr->query1D(x1, x2, y1,
23             y2, m+1, r) : 0;
24         return gcd(a, b);
25     }
26 }
27 void update1D(int x, int y, ll num, int l
28     , int r) {
29     if(l == r) {
30         val = num;
31         return ;
32     }
33     /*
34     if(tmp < 0 && !tl && !tr) {
35         tmp = val = num;
36         _x = x;
37         _y = y;
38         return ;
39     }
40     else if(tmp >= 0) {
41         int m = (l+r)/2;
42         if(_y <= m) {
43             if(!tl) tl = new Seg1D();
44             tl->update1D(_x, _y, tmp, l, m);
45         }
46         else {
47             if(!tr) tr = new Seg1D();
48             tr->update1D(_x, _y, tmp, m+1, r);
49         }
50         tmp = _x = _y = -1;
51     }*/
52     int m = (l+r)/2;
53     if(y <= m) {
54         if(!tl) tl = new Seg1D();
55         tl->update1D(x, y, num, l, m);
56     }
57     else {
58         if(!tr) tr = new Seg1D();
59         tr->update1D(x, y, num, m+1, r);
60     }
61     ll a = tl ? tl->val : 0;
62     ll b = tr ? tr->val : 0;
63     val = gcd(a, b);
64 }
65 struct Seg2D {
66     Seg2D *tl, *tr;
67     Seg1D *t2;
68     Seg2D() :
69         tl(NULL), tr(NULL), t2(NULL) {}
70     ll query2D(int x1, int x2, int y1, int y2
71         , int l, int r) {
72         if(x1 <= l && r <= x2) {
73             if(!t2) t2 = new Seg1D();
74             return t2->query1D(x1, x2, y1, y2, 0,
75                 C-1);
76         }
77         else if(x2 < l || r < x1) return 0;
78     }
};

```

```

74     else {
75         int m = (l+r)/2;
76         ll a = t1 ? t1->query2D(x1, x2, y1,
77             y2, l, m) : 0,
78         b = tr ? tr->query2D(x1, x2, y1,
79             y2, m+1, r) : 0;
80         return gcd(a, b);
81     }
82 }
83 void update2D(int x, int y, ll num, int l
84     , int r) {
85     int m = (l+r)/2;
86     if(l == r) {
87         if(!t2) t2 = new Seg1D();
88         t2->update1D(x, y, num, 0, C-1);
89         return ;
90     }
91     if(x <= m) {
92         if(!t1) t1 = new Seg2D();
93         t1->update2D(x, y, num, l, m);
94     }
95     else {
96         if(!tr) tr = new Seg2D();
97         tr->update2D(x, y, num, m+1, r);
98     }
99     if(!t1) t1 = new Seg2D();
100    if(!tr) tr = new Seg2D();
101    ll a = t1->t2 ? t1->t2->query1D(l, m, y
102        , y, 0, C-1) : 0,
103    b = tr->t2 ? tr->t2->query1D(m+1, r,
104        y, y, 0, C-1) : 0;
105    if(!t2) t2 = new Seg1D();
106    t2->update1D(x, y, gcd(a, b), 0, C-1);
107 }
108 };

```

geometry

Basic

```

1 typedef double tp;
2 typedef double db;
3
4 const db PI = acos(-1.0);
5 const tp INF = 1e18;
6 const tp EPS = 1e-9;
7
8 bool eq(tp a, tp b) { return a-b<=EPS && b-
9     a<=EPS; }
10 bool lt(tp a, tp b) { return a < b-EPS; }
11 bool le(tp a, tp b) { return !lt(b, a); }
12 bool gt(tp a, tp b) { return lt(b, a); }
13 bool ge(tp a, tp b) { return !lt(a, b); }
14
15 struct coor {
16     tp x, y, z;
17     coor(tp _x=0, tp _y=0, tp _z=0): x(_x), y
18         (_y), z(_z) {}
19     coor operator+(const coor p) const {
20         return coor(x+p.x, y+p.y, z+p.z); }
21     coor operator-(const coor p) const {
22         return coor(x-p.x, y-p.y, z-p.z); }

```

```

19     coor operator*(const tp a) const { return
20         coor(x*a, y*a, z*a); }
21     coor operator/(const tp a) const { return
22         coor(x/a, y/a, z/a); }
23     tp operator*(const coor p) const { return
24         x*p.x + y*p.y + z*p.z; }
25     db atan() const {
26         db ret = atan2(y, x);
27         if(ret<0) ret += 2*PI;
28         return ret;
29     }
30     bool operator==(const coor p) const {
31         return eq(x, p.x) && eq(y, p.y) && eq(
32             z, p.z); }
33     void input() { cin >> x >> y; }
34     // 2D only
35     tp operator%(const coor p) const { return
36         x*p.y - y*p.x; }
37     bool operator<(const coor p) const {
38         if(x != p.x) return x<p.x;
39         if(y != p.y) return y<p.y;
40         return z<p.z;
41     }
42 };
43 tp abs2(const coor a) { return a.x*a.x+a.y*
44     a.y+a.z*a.z; }
45 db abs(const coor a) { return sqrt(abs2(a))
46     ; }
47 coor perp(const coor p) { return coor(-p.y,
48     p.x); } // +0.5pi
49
50 bool polar(const coor a, const coor b) {
51     // integral
52     if(a.y*b.y<0) return a.y>0;
53     if(b.y==0 and b.x>0) return false;
54     if(a.y==0 and a.x>0) return true;
55     return a%b>0;
56     //floating
57     return a.atan() < b.atan();
58 }

```

CircleCover

```

1 #define N 1021
2
3 struct Circ {
4     coor O;
5     db R;
6     Circ(coor _o=0, db _r=0): O(_o), R(_r) {}
7 };
8
9 struct CircleCover{
10     int C; Circ c[ N ];
11     bool g[ N ][ N ], overlap[ N ][ N ];
12     // Area[i] : area covered by at least i
13     // circles
14     db Area[ N ];
15     void init( int _C ){ C = _C; }
16     bool CCinter( Circ& a , Circ& b , coor&
17         p1 , coor& p2 ){
18         coor o1 = a.O , o2 = b.O;
19         db r1 = a.R , r2 = b.R;
20         tp d2 = abs2(o1-o2);

```



```

19 db d = abs(o1-o2);
20 if( d > r1 + r2 ) return false;
21 if( d < max(r1, r2) - min(r1, r2) )
    return false;
22 //if( d > r1 + r2 ) return false;
23 coor u=(o1+o2)*0.5 + (o1-o2)*((r2*r2-r1
    *r1)/(2*d2));
24 db A=sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)
    *(-r1+r2+d));
25 coor v=coor( o1.y-o2.y , -o1.x + o2.x )
    * A / (2*d2);
26 p1 = u + v; p2 = u - v;
27 return true;
28 }
29 struct Teve {
30     coor p; db ang; int add;
31     Teve() {}
32     Teve(coor _a, db _b, int _c):p(_a), ang
        (_b), add(_c){}
33     bool operator<(const Teve &a)const
34     {return ang < a.ang;}
35 }eve[ N * 2 ];
36 int sign(db x) {
37     return x < 0 ? -1 : x > 0;
38 }
39 // strict: x = 0, otherwise x = -1
40 bool disjuct( Circ& a, Circ &b, int x )
41 {return sign( abs( a.O - b.O ) - a.R - b.
    R ) > x;}
42 bool contain( Circ& a, Circ &b, int x )
43 {return sign( a.R - b.R - abs( a.O - b.O
    ) ) > x;}
44 bool contain(int i, int j){
45     /* c[j] is non-strictly in c[i]. */
46     return (sign(c[i].R - c[j].R) > 0 ||
47         (sign(c[i].R - c[j].R) == 0 && i <
48             j) ) && contain(c[i], c[j], -1);
49 }
50 void solve(){
51     for( int i = 0 ; i <= C + 1 ; i ++ )
52         Area[ i ] = 0;
53     for( int i = 0 ; i < C ; i ++ )
54         for( int j = 0 ; j < C ; j ++ )
55             overlap[i][j] = contain(i, j);
56     for( int i = 0 ; i < C ; i ++ )
57         for( int j = 0 ; j < C ; j ++ )
58             g[i][j] = !(overlap[i][j] ||
59                 overlap[j][i] ||
60                 disjuct(c[i], c[j], -1));
61     for( int i = 0 ; i < C ; i ++ ){
62         int E = 0, cnt = 1;
63         for( int j = 0 ; j < C ; j ++ )
64             if( j != i && overlap[j][i] )
65                 cnt ++;
66         for( int j = 0 ; j < C ; j ++ )
67             if( i != j && g[i][j] ){
68                 coor aa, bb;
69                 CCinter(c[i], c[j], aa, bb);
70                 db A=atan2(aa.y - c[i].O.y, aa.x
71                     - c[i].O.x);
72                 db B=atan2(bb.y - c[i].O.y, bb.x
73                     - c[i].O.x);
74                 eve[E ++] = Teve(bb, B, 1);
75                 eve[E ++] = Teve(aa, A, -1);
76                 if(B > A) cnt ++;

```

```

77     }
78     if( E == 0 ) Area[ cnt ] += PI * c[i
79         ].R * c[i].R;
80     else{
81         sort( eve , eve + E );
82         eve[E] = eve[0];
83         for( int j = 0 ; j < E ; j ++ ){
84             cnt += eve[j].add;
85             Area[cnt] += (eve[j].p % eve[j +
86                 1].p) * .5;
87             db theta = eve[j + 1].ang - eve[j
88                 ].ang;
89             if (theta < 0) theta += 2. * PI;
90             Area[cnt] +=
91                 (theta - sin(theta)) * c[i].R*c
92                 [i].R * .5;
93         }
94     }
95 } oracle;

```

ConvexHull

```

1 typedef vector<coor> VP;
2
3 // Convex Hull
4 // keep redundant points or not
5 VP CH(VP arr, bool keep=false) {
6     sort(ALL(arr));
7     VP upper, lower;
8     for(int i=0; i<SZ(arr); i++) {
9         if(i>0 and arr[i] == arr[i-1])
10             continue;
11         coor c = arr[i];
12         while(SZ(upper)>=2) {
13             int last = SZ(upper)-1;
14             coor a = upper[last-1], b=upper[last
15                 ];
16             if(!t((c-a)%(b-a), 0) or (!keep and
17                 le((c-a)%(b-a), 0)))
18                 upper.pop_back();
19             else
20                 break;
21         }
22         upper.PB(c);
23         while(SZ(lower)>=2) {
24             int last = SZ(lower)-1;
25             coor a = lower[last-1], b=lower[last
26                 ];
27             if(gt((c-a)%(b-a), 0) or (!keep and
28                 ge((c-a)%(b-a), 0)))
29                 lower.pop_back();
30             else
31                 break;
32         }
33         lower.PB(c);
34     }
35     for(int i=SZ(upper)-2; i>0; i--)
36         lower.PB(upper[i]);
37     return lower;

```


HalfPlaneNSquare

```

1 typedef vector<coor> poly;
2 void cut(poly &tar, const coor vec, const
  db c) {
3   poly tmp;
4   coor st = tar[0];
5   tar.PB(st);
6   for(int k = 1; k < SZ(tar); k++) {
7     coor ed = tar[k];
8     db a = st*vec, b = ed*vec;
9     coor v2 = st * ((b-c)/(b-a)) + ed * ((c
      -a)/(b-a));
10
11     if(le(a, c))
12       tmp.PB(st);
13     if((lt(a, c) and gt(b, c)) || (gt(a, c)
      and lt(b, c)))
14       tmp.PB(v2);
15     st = ed;
16   }
17   tar.clear();
18   for(int i=0; i<SZ(tmp); i++)
19     tar.PB(tmp[i]);
20 }
21
22 void polyIntersect(poly &P, const poly &Q)
  {
23   for(int i=0; i<SZ(Q); i++) {
24     coor v = perp(Q[(i+1)%SZ(Q)]-Q[i])*(-1)
      ;
25     v = v/abs(v);
26     cut(P, v, v*Q[i]);
27   }
28 }

```

LineIntersection

```

1 coor line_inter(const coor p1, const coor
  v1, const coor p2, const coor v2) {
2   if(eq(v1%v2, 0.0))
3     return coor(INF, INF);
4   db k = ((p2-p1)%v2) / (v1%v2);
5   return p1 + v1*k;
6 }

```

OldCircleInter

```

1 void CircleInter(coor o1, db r1, coor o2,
  db r2) {
2   if(r2>r1)
3     swap(r1, r2), swap(o1, o2);
4   db d = (o2-o1).abs();
5   coor v = o2-o1;
6   v = v / v.abs();
7   coor t = coor(v.y, -v.x);
8
9   db area;
10  vector<coor> pts;
11  if(d > r1+r2+EPS)
12    area = 0;

```

```

13  else if(d < r1-r2)
14    area = r2*r2*PI;
15  else if(r2*r2+d*d > r1*r1){
16    db x = (r1*r1 - r2*r2 + d*d) / (2*d);
17    db th1 = 2*acos(x/r1), th2 = 2*acos((d-
      x)/r2);
18    area = (r1*r1*(th1 - sin(th1)) + r2*r2
      *(th2 - sin(th2))) / 2;
19    db y = sqrt(r1*r1 - x*x);
20    pts.PB(o1 + v*x + t*y), pts.PB(o1 + v*x
      - t*y);
21  } else {
22    db x = (r1*r1 - r2*r2 - d*d) / (2*d);
23    db th1 = acos((d+x)/r1), th2 = acos(x/
      r2);
24    area = r1*r1*th1 - r1*d*sin(th1) + r2*
      r2*(PI-th2);
25    db y = sqrt(r2*r2 - x*x);
26    pts.PB(o2 + v*x + t*y), pts.PB(o2 + v*x
      - t*y);
27  }
28  //Area: area
29  //Intersections: pts
30 }

```

PolyCircleIntersect

```

1 // Divides into multiple triangle, and sum
  up
2 // oriented area
3 int sign(db x) { return x < 0 ? -1 : x > 0;
  }
4 db area2(coor pa, coor pb, db r){
5   if( abs(pa) < abs(pb) ) swap(pa, pb);
6   if( abs(pb) < EPS ) return 0;
7   db S, h, theta;
8   db a = abs( pb ), b = abs( pa ), c = abs(
      pb - pa);
9   db cosB = (pb * (pb - pa)) / a / c, B =
      acos(cosB);
10  db cosC = (pa * pb) / a / b, C = acos(
      cosC);
11  if(a > r){
12    S = (C/2)*r*r;
13    h = a*b*sin(C)/c;
14    if (h < r && B < PI/2) S -= (acos(h/r)*
      r*r - h*sqrt(r*r-h*h));
15  }else if(b > r){
16    theta = PI - B - asin(sin(B)/r*a);
17    S = .5*a*r*sin(theta) + (C-theta)/2*r*r
      ;
18  }else S = .5*sin(C)*a*b;
19  return S;
20 }
21 db area(coor ori, db r, const poly &P) {
22   db S = 0;
23   int n = SZ(P);
24   for(int i = 0; i < n; ++i) {
25     coor v1=P[i]-ori, v2=P[(i+1)%n]-ori;
26     S += area2(v1, v2, r) * sign(v1%v2);
27   }
28   return fabs(S);
29 }

```

SegmentIntersection

```

1 int ori(const coor o, const coor a, const
  coor b) {
2   tp val = (a-o)%(b-o);
3   return gt(val, 0) - lt(val, 0);
4 }
5 bool SegmentIntersect(const coor p1, const
  coor p2, const coor q1, const coor q2) {
6   if( eq((p2-p1)%(q2-q1), 0) ) {
7     if( ori(p1, p2, q1) ) return false;
8     return le( ( p1 - q1 ) * ( p2 - q1 ),
9               0) ||
10            le( ( p1 - q2 ) * ( p2 - q2 ),
11              0) ||
12            le( ( q1 - p1 ) * ( q2 - p1 ), 0)
13            ||
14            le( ( q1 - p2 ) * ( q2 - p2 ), 0);
15 }
16 }
17 return (ori( p1, p2, q1 ) * ori( p1, p2,
18   q2 ) <= 0) &&
19   (ori( q1, q2, p1 ) * ori( q1, q2,
20   p2 ) <= 0);
21 }
22 }
23 }
24 }
25 }
26 }
27 }
28 }
29 }
30 }
31 }
32 }
33 }
34 }
35 }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }

```

Triangulation

```

1 typedef vector<coor> poly;
2 const int N = 105;
3
4 bool inside(const coor pnt, const poly &P)
5 {
6   int n = SZ(P);
7   for(int i=0; i<n; i++) {
8     coor p1=P[i], p2=P[(i+1)%n];
9     if(lt((p1-pnt) % (p2-pnt), 0))
10       return false;
11   }
12   return true;
13 }
14
15 int prv[N], nxt[N];
16 bool isear(int x, const poly &P) {
17   int n = SZ(P);
18   int x1 = nxt[x], x2=prv[x];
19   coor v=P[x], v1=P[x1], v2=P[x2];
20   if(le((v1-v)%(v2-v), 0)) return false;
21   poly cand;
22   cand.PB(v), cand.PB(v1), cand.PB(v2);
23   for(int j=0; j<n; j++) {
24     if(j==x or j==x1 or j==x2) continue;
25     if(inside(P[j], cand))
26       return false;
27   }
28   return true;
29 }
30
31 vector<poly> triangulation(const poly &P) {
32   bool used[N]={}, ear[N]={};
33   int n = SZ(P);
34   for(int i=0; i<n; i++) prv[i] = (i-1+n)%
35     n, nxt[i] = (i+1)%n;
36   queue<int> que;
37 }

```

Smallest circle problem

```

1 const int MAXN = ;
2 struct PT {
3   double x, y;
4   PT() {}
5   PT(double x, double y):
6     x(x), y(y) {}
7   PT operator+(const PT &b) const {
8     return (PT) {x+b.x, y+b.y};
9   }
10   PT operator-(const PT &b) const {
11     return (PT) {x-b.x, y-b.y};
12   }
13   PT operator*(const double b) const {
14     return (PT) {x*b, y*b};
15   }
16   PT operator/(const double b) const {
17     return (PT) {x/b, y/b};
18   }
19   double operator%(const PT &b) const {
20     return x*b.y - y*b.x;
21   }
22   double len() const {
23     return sqrt(x*x + y*y);
24   }
25   PT T() const {
26     return (PT) {-y, x};
27   }
28 } p[MAXN];
29 void update(PT a, PT b, PT c, PT &o, double
30   &r) {
31   if(c.x < 0.0) o = (a+b) / 2.0;
32   else {
33     PT p1 = (a+b)/2.0, p2 = p1 + (b-a).T();
34     PT p3 = (a+c)/2.0, p4 = p3 + (c-a).T();
35   }
36 }

```

```

34 double a123 = (p2-p1)%(p3-p1), a124 = (
    p2-p1)%(p4-p1);
35 if(a123 * a124 > 0.0) a123 = -a123;
36 else a123 = abs(a123), a124 = abs(a124
    );
37 o = (p4*a123 + p3*a124) / (a123 + a124)
    ;
38 }
39 r = (a-o).len();
40 }
41 void solve(PT &o, double &r) {
42     random_shuffle(p, p+n);
43     PT a = p[0], b = p[1], c(-1.0, -1.0);
44     o = (a+b) / 2.0;
45     double r = (a-o).len();
46     for(int i = 2; i < n; i++) {
47         if((p[i]-o).len() <= r) continue;
48         a = p[i], b = p[0], c = (PT) {-1.0,
            -1.0};
49         update(a, b, c, o, r);
50         for(int j = 1; j < i; j++) {
51             if((p[j]-o).len() <= r) continue;
52             b = p[j], c = (PT) {-1.0, -1.0};
53             update(a, b, c, o, r);
54             for(int k = 0; k < j; k++) {
55                 if((p[k]-o).len() <= r) continue;
56                 c = p[k];
57                 update(a, b, c, o, r);
58             }
59         }
60     }
61 }

```

Others

Random

```

1 const int seed=1;
2
3 mt19937 rng(seed);
4 int randint(int lb,int ub) { // [lb, ub]
5     return uniform_int_distribution<int>(lb,
        ub)(rng);
6 }

```

Fraction

```

1 struct Frac {
2     ll a,b; // a/b
3     void relax() {
4         ll g=__gcd(a,b);
5         if(g!=0 && g!=1)
6             a/=g, b/=g;
7         if(b<0)
8             a*=-1, b*=-1;
9     }
10    Frac(ll a_=0,ll b_=1): a(a_), b(b_) {
11        relax();
12    }
13    Frac operator + (Frac x) {

```

```

14    relax();
15    x.relax();
16    ll g=__gcd(b,x.b);
17    ll lcm=b/g*x.b;
18    return Frac(a*(lcm/b)+x.a*(lcm/x.b),lcm
        );
19    }
20    Frac operator - (Frac x) {
21        relax();
22        x.relax();
23        Frac t=x;
24        t.a*=-1;
25        return *this+t;
26    }
27    Frac operator * (Frac x) {
28        relax();
29        x.relax();
30        return Frac(a*x.a,b*x.b);
31    }
32    Frac operator / (Frac x) {
33        relax();
34        x.relax();
35        Frac t=Frac(x.b,x.a);
36        return (*this)*t;
37    }
38    bool operator < (Frac x) {
39        ll lcm=b/__gcd(b,x.b)*x.b;
40        return ( (lcm/b)*a < (lcm/x.b)*x.a );
41    }
42 };

```