Makefile

```
FN=
FL=-lm -lcrypt -O2 -pipe
TP=../template.cpp

${FN}:
	g++ ${FL} -g -pedantic -Wall -Wshadow -DDD ${FN}.cpp -o $
{FN}

judge:
	g++ ${FL} ${FN}.cpp -o ${FN}

run: ${FN}
	${FN} < testdata | tee testoutput
	diff testoutput goodoutput

init:
	cp -b -i ${TP} ./${FN}.cpp
	touch testdata goodoutput
```

C++ Template

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <map>
#include <set>
#include <queue>
#include <list>
#include <cmath>
#include <algorithm>
#include <complex>

#ifdef DD
const bool debug = true;
#else
const bool debug = false;
#endif

#define D(x) if (debug) { x; };
#define DS if (debug) {
#define DE };

#define xrange(i,a,b) for(int i=(a),_b=(b);i<_b;i++)
#define kforeach(it,c) for(__typeof((c).begin()) it=(c).begin();it!
=(c).end();++it)
#define kfe(it,c) kforeach(it,c)

using namespace std;

template<class T> string x2s(T x) { ostringstream o; o << x; return
o.str(); }
int s2i(string s) { istringstream i(s); int x; i>>x; return x; }


int main()
{
    return EXIT_SUCCESS;
}
```

.vimrc:

```
set nocompatible
syntax on
filetype on

set vb lbr wrap mh ai si et sta fen spr nohls is ru lz sc
set ts=8 sw=4 sts=4 ch=2
set so=5 siso=5 history=200 mouse=a backspace=eol,start,indent
set ul=1000 uc=100 foldlevel=99999

ino {<CR> {<CR>}<Esc>O
com W w

function InsertTabWrapper()
    let col = col('.') - 1
    if !col || getline('.')[col - 1] !~ '\k'
        return "\<tab>"
    else
        return "\<c-p>"
    endif
endfunction
inoremap <tab> <c-r>=InsertTabWrapper()<cr>
```

algorithms:

Matrix Chain Order

```
Matrix-Chain-Order(int p[])
{
    n = p.length - 1;
    for (i = 1; i <= n; i++)
       m[i,i] = 0;

    for (L=2; L<=n; L++) { // L is chain length
        for (i=1; i<=n-L+1; i++) {
            j = i+L-1;
            m[i,j] = MAXINT;
```

```
            for (k=i; k<=j-1; k++) {
                //Matrix Ai has the dimension  p[i-1] x p[i].
                q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
                if (q < m[i,j]) {
                    m[i,j] = q;
                    s[i,j] = k;
                }
            }
        }
    }
}
```

bfs/dfs

```
Notes: SIZE should be #defined, matrix should use negative number
to denote no edge

void bfs(int graph[SIZE][SIZE], int maxIndex, start)
{
    int i, curIndex, k;
    int notSeen[maxIndex];
    queue<int> q;

    for (i = 0; i < maxIndex; i++)
    {
        notSeen[i] = 1;
    }

    q.push(start);

    while (!q.empty())
    {
        curIndex = q.front(); // priority queue and stack use top()

        if (notSeen[curIndex] == 0)
        {
            q.pop();
            continue;
        }

    notSeen[curIndex] = 0;
        q.pop();

        // process current node here

        for (k = 0; k < maxIndex; k++)
            if (graph[curIndex][k] >= 0 && notSeen[k] == 1)
            {
                q.push(k);
            }
    }

}
```

Recursive dfs

```
void doSearch(graph g, node currentNode)
{
    vector<node> adj = getAdjacentNodes(currentNode);

    // Process currentNode here for pre-order

    FOREACH(it, adj)
        if (!it->visited())
            doSearch(g, *it);

    // Process currentNode here for post-order
}

void dfs(graph g)
{
    doSearch(g, g.root());
}

    lcs TODO
    graph coloring TODO
    cycle detection TODO
```

convex hull

```
// Convex hull algorithm: removes all collinear/coincident points
// modifies the input n, p[]
point P0;

double cp(point a,point b) {
    return a.x*b.y-a.y*b.x;
}

bool cmp(point p1, point p2)
{
    double s = cp(p1 - P0, p2 - P0);
    if(fabs(s) > eps)
        return s>0;
    return abs(p1 - P0) < abs(p2 - P0) - eps;
}
```

```
void graham(int & n, point * p)
{
    point ret[maxn];
    int i, m = 0;
    if (!n)
        return;
    for(i = 1; i < n; i++)
        if(p[i].x<p[0].x || p[i].x==p[0].x && p[i].y<p[0].y)
            swap(p[i], p[0]);
    P0=p[0];
    sort(p+1, p+n, cmp);
    ret[m++] = p[0];
    for(i = 1; i < n; i++)
    {
        while(m > 1 && cp(ret[m-2]-ret[m-1],p[i]-ret[m-1]) > -eps)
            m--;
        if (abs(p[i]-ret[m-1]) > eps && abs(p[i]-ret[0]) > eps)
            ret[m++]=p[i];
    }
    for(n = m, i = 0; i < n; i++)
        p[i] = ret[i];
}
```

area of polygon

```
double cp(point a,point b) {
    return a.x*b.y-a.y*b.x;
}

double area_polygon(int n,point* p){
    double s=0;
    for(int i=0; i<n; i++) s+=cp(p[i],p[(i+1)%n]);
    return fabs(s)/2;
}
```

biconnected components (max biconnected [connected and non-seperable] subgraph)

```
// Note that component excludes bridges
int cutvertex[maxn],low[maxn],number[maxn],parent[maxn],I;
vector<pair<int,int> >  bridge;
vector<vector<pair<int,int> > >  component;
stack<pair<int,int> > s;
void biconnected(int u){
    low[u] = number[u] = ++I;
    int child = 0;
    for (int i = 0; i < a[u].size(); i++) {
        int v = a[u][i];
        if (!number[v]) {
            s.push(make_pair(u,v));
            parent[v] = u, child++;
            biconnected(v);
            low[u] = min(low[u], low[v]);
            if(parent[u]!=-1 && low[v]>=number[u])
                cutvertex[u]=1;
            if(low[v]>number[u])
                bridge.push_back(make_pair(u,v)), s.pop();
            else if(low[v]==number[u]) {
                component.resize(component.size()+1);
                while(s.top()!=make_pair(u,v) && s.top()!
=make_pair(v,u))
                    component.back().push_back(s.top()), s.pop();
                component.back().push_back(make_pair(u,v));
                s.pop();
            }
        }
        else if(number[v]<number[u] && v!=parent[u])
            low[u] = min(low[u], number[v]),
s.push(make_pair(u,v));
    }
    if(parent[u]==-1 && child>=2) cutvertex[u]=1;
}
void doit(int n) {
    I=0;
    component.clear(),bridge.clear();
    memset(number,0,sizeof(number));
    memset(cutvertex,0,sizeof(cutvertex));
    memset(parent,0,sizeof(parent));
    for(int i=0; i<n; i++)
        if(!number[i])
            parent[i]=-1, biconnected(i);
}
```

minimum cost spanning trees

```
struct Edge{
    int u, v; int w;
    Edge(){}
    Edge(int x, int y, int z):u(x), v(y),w(z){}
    int operator<(const Edge &e) const { return w < e.w; }
};
```

```
int root(int i, int *tree){
    int temp, ret = i;
    while (tree[ret] != ret) ret = tree[ret];
    while (i != ret) temp = tree[i], tree[i] = ret , i = temp;
    return ret;
}

int kruskal(int n, vector<Edge> e, vector<Edge> &t){
    int ret = 0;
    int i, j, r[2], tree[n];

    sort(e.begin(), e.end());
    for (i = 0; i < n; i++) tree[i] = i;

    for (i = 0; i < e.size(); i++){
        r[0] = root(e[i].u, tree);
        r[1] = root(e[i].v, tree);

        if (r[0] != r[1]){
            ret+= e[i].w;
            t.push_back(e[i]);
            if (rand() % 2) tree[r[0]] = r[1];
            else tree[r[1]] = r[0];
        }
    }

    if (t.size()+1 < n) return -1;
    return ret;
}


int prim(int n, vector<pair<int, int> > a[maxn], int *t, int *pre){
    int ret = 0, w, d;
    int i, u, v, visited[maxn];
    priority_queue<pair<int, int> > pq;

    pq.push(make_pair(0, 0));
    memset(visited, 0, sizeof(visited));
    memset(pre, -1, sizeof(pre));
    for (i = 0; i < n; i++) t[i] = inf;

    while (!pq.empty()){
        w = -pq.top().first;
        u = pq.top().second;
        pq.pop();

        if (!visited[u]){
            visited[u] = 1;
            for (i = 0; i < a[u].size(); i++) {
                v = a[u][i].first;
                d = a[u][i].second;

                if (!visited[v]){
                    pre[v] = u;
                    pq.push(make_pair(-d, v));
                }
            }
            ret += w;
        }
    }

    for (i = 0; i < n; i++)
        if (!visited[i]) return -1;

    return ret;
}
```

Find the highest bit

```
// This is 0 based, not 1 based, ie. highBit(1) == 0, highBit(2) ==
1
unsigned int highBit(unsigned int v)
{
    const unsigned int b[] = {0x2, 0xC, 0xF0, 0xFF00, 0xFFFF0000};
    const unsigned int S[] = {1, 2, 4, 8, 16};
    register unsigned int r = 0; // result of log2(v) will go here

    for (int i = 4; i >= 0; i--) // unroll for speed...
    {
        if (v & b[i])
        {
            v >>= S[i];
            r |= S[i];
        }
    }
    return r;
}
```

Fast GCD

```
// Returns the greatest common divisor of a and b.
unsigned int gcd(register unsigned int a, register unsigned int b)
{
    // Neat in-place alternating technique.
    for (;;)
    {
        if (!b)
            return a;
        a = b % a;
```

```
        if (!a)
            return b;
        b = a % b;
    }
}
```

Extended Euclidean

```
/*
 * Extended Euclidean algorithm.
 * Given m and n, finds gcd g and numbers r, s such
 * that r*m + s*n == g.
 */
void extendedEuclidean(int m, int n, int &g, int &r, int &s)
{
    if (&g == &r || &g == &s || &r == &s)
        throw "int extendedEuclidean: Outputs are aliased";
    int r1 = 1, s1 = 0, r2 = 0, s2 = 1, q;
    /*
     * Invariants:
     * r1*m(orig) + s1*n(orig) == m(current)
     * r2*m(orig) + s2*n(orig) == n(current)
     */
    for (;;)
    {
        if (!n)
        {
            r = r1; s = s1; g = m;
            return;
        }
        // Subtract q times the second invariant from the first
invariant.
        q = m;
        m = n % m;

        r1 -= q*r2; s1 -= q*s2;

        if (!m)
        {
            r = r2; s = s2; g = n;
            return;
        }
        // Subtract q times the first invariant from the second
invariant.
        q = n;
        n = m % n;

        r2 -= q*r1; s2 -= q*s1;
    }
}
```

ModPow (modular exponentiation)

```
// Returns (base ^ exponent) % modulus.
unsigned int modpow(int base, unsigned int exponent, unsigned int
modulus)
{
    unsigned int result = 1;
    while (exponent > 0)
    {
        if (exponent & 1)
            result = (result * base) % modulus;
        exponent = exponent >> 1;
        base = (base * base) % modulus;
    }
    return result;
}
```

Is Power of Two

```
// True if v is a power of 2
#define IS_POW_TWO(v) v && !(v &(v - 1))
```

Count the ones in an int

```
inline unsigned int countOnes(unsigned int v)
{
    unsigned int c; // c accumulates the total bits set in v
    for (c = 0; v; c++)
        v &= v - 1; // clear the least significant bit set
    return c;
}
```

Reverse Bits in int

```
inline unsigned int reverseBits(register unsigned int v)
{
    register unsigned int r = v; // r will be reversed bits of v
    int s = sizeof(v) * CHAR_BIT - 1; // extra shift needed at end

    for (v >>= 1; v; v >>= 1)
    {
        r <<= 1;
        r |= v & 1;
        s--;
    }
    r <<= s; // shift when v's highest bits are zero
    return r;
}
```

Log base 10

```
inline int logTen(unsigned int v)
{
    int r; // result goes here

    r = (v >= 1000000000) ? 9 : (v >= 100000000) ? 8 : (v >=
10000000) ? 7 :
            (v >= 1000000) ? 6 : (v >= 100000) ? 5 : (v >= 10000) ?
4 :
                (v >= 1000) ? 3 : (v >= 100) ? 2 : (v >= 10) ? 1 :
0;
    return r;
}
```

Number of Trailing Zeros

```
// Produces a warning sometimes, this is okay
// Returns -127 if v == 0
inline int numTrailingZeros(unsigned int v)
{
    float f = (float)(v & -v); // cast the lsb -> float
    return (*(unsigned int *)&f >> 23) - 0x7f;
}
```

From large html cheat sheet:

GRAPH THEORY

Floyd-Warshall (Minimum distance between pairs)

```
// Takes nxn adjency mtx. converts it to shortest path mtx
// graph[i][j] = weight from i -> j. 9999999 for no-link..
void floydWarshall(vector<vector<int> > graph)
{
    int n = graph.size();
    for(int k = 0; k < n; k++)
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                graph[i][j] = min(graph[i][j], graph[i][k]+graph[k]
[j]);
}
```

Max Flow

Maximum flow through graph given edge capacities

```
vector<int> flow[maxn],adj[maxn],cap[maxn],rev[maxn],cost[maxn];
void init() {
    for(int i=0;i<maxn;i++)
        flow[i].clear(),adj[i].clear(),cap[i].clear(),rev[i].clear(
),
            cost[i].clear();
}
void add(int a,int b,int c) {
    int an=adj[a].size(),bn=adj[b].size();
    adj[a].push_back(b), flow[a].push_back(0),
        rev[a].push_back(bn), cap[a].push_back(c);
    adj[b].push_back(a), flow[b].push_back(0),
        rev[b].push_back(an), cap[b].push_back(0);
}
int max_flow(int n,int s, int t) {
    int pre[maxn],y[maxn],i,j,x,v;
    queue<int> que;
    if(s==t) return inf;
    while(1) {
        memset(pre,-1,sizeof(pre));
        que.push(s), pre[s]=-2, y[s]=inf;
        while(que.size()) {
            x=que.front(), que.pop();
            for(i=0;i<adj[x].size();i++)
                if(pre[v=adj[x][i]]==-1&&(j=cap[x][i]-flow[x][i]))
                    que.push(v),pre[v]=rev[x][i],y[v]=min(y[x],j);
        }
        if(pre[t]==-1) break;
        for(i=t;i!=s;i=v)
            flow[v=adj[i][pre[i]]][rev[i][pre[i]]]+=y[t],flow[i]
[pre[i]]-=y[t];
    }
    for(j=i=0;i<adj[s].size();j+=flow[s][i++]);
    return j;
}
```

Strongly connected components

Directed graph, path from any node to another

```
stack<int> s;
int I, Cs, low[maxn],number[maxn],removed[maxn], id[maxn];
void strongly_connected(int u){
    int v; low[u] = number[u] = ++I;
    s.push(u);
    for (int i = 0; i < a[u].size(); i++) {
        int v = a[u][i];
        if (!removed[v])
            if (!number[v]) {
                strongly_connected(v);
                low[u] = min(low[u], low[v]);
            } else low[u] = min(low[u], number[v]);
    }
    if (number[u] == low[u]) {
        while (1){
            v = s.top(); s.pop();
            id[v] = Cs;
            removed[v] = 1;
            if (u == v) break;
        }
```

```
        Cs++;
    }
}
void doit(int n){
    I = Cs = 0;
    memset(removed,0,sizeof(removed));
    memset(number,0,sizeof(number));
    REP(i, n)
        if (!removed[i]) strongly_connected(i);
}
```

Stable matching or stable marriage

```
/*
   find an unmarried man, and his most preferred woman not yet
considered
   if the woman not married, then marry man and woman.
   otherwise the woman already married with someone else,
   so marry them only if it is more preferred for the woman.

   X is the matrix for the man, Y is the matrix for woman
 */

void stable_matching(int n,int X[][maxn],int Y[][maxn],int M[]
[maxn]) {
    int usedX[maxn],usedY[maxn],i,j,k,t,w,flag=1;
    memset(usedX,0,sizeof(usedX)),memset(usedY,0,sizeof(usedY));
    for(i=0;i<n;i++) for(j=0;j<n;j++) M[i][j]=-1;
    while(flag) {
        flag = 0;
        for(i=0; i<n; i++) {
            if(usedX[i]) continue;
            flag = 1, w=0, j=-1;
            for(t=0; t<n; t++)
                if(M[i][t]==-1)
                    if(X[i][t]>w) w=X[i][t],j=t;
            if(j==-1) continue;
            if(!usedY[j]) usedY[j]=usedX[i]=1, M[i][j]=1;
            else
                for(k=0; k<n; k++)
                    if(M[k][j]==1) {
                        if(Y[j][i] > Y[j][k])
                            usedX[i]=1, usedX[k]=0, M[k][j]=0, M[i]
[j]=1;
                        else M[i][j]=0;
                        break;
                    }
        }
    }
}
```

Bipartite matching (matching on bipartite graph)

Definition: matching - a set of non-adjacent edges

Definition: bipartite - graph with two sets of vertices which connect only to vertices in the other set

```
#define _clr(x) memset(x,0xff,sizeof(int)*maxn)
int matching(int m,int n,int a[][maxn],int* x,int* y){
    int s[maxn+1],t[maxn],p,q,ret=0,i,j,k;
    for (_clr(x),_clr(y),i=0;i<m;ret+=(x[i++]>=0))
        for (_clr(t),s[p=q=0]=i;p<=q&&x[i]<0;p++)
            for (k=s[p],j=0;j<n&&x[i]<0;j++)
                if (a[k][j]&&t[j]<0){
                    s[++q]=y[j],t[j]=k;
                    if (s[q]<0)
                        for (p=j;p>=0;j=p)
                            y[j]=k=t[j],p=x[k],x[k]=j;
                }
    return ret;
}
```

Maximum matching (matching with most edges)

```
int aug(int n,int mat[][maxn],int* match,int* v,int now){
  int i,ret=0;
  v[now]=1;
  for (i=0;i<n;i++)
    if (!v[i]&&mat[now][i]){
      if (match[i]<0)
        match[now]=i,match[i]=now,ret=1;
      else{
        v[i]=1;
        if (aug(n,mat,match,v,match[i]))
          match[now]=i,match[i]=now,ret=1;
        v[i]=0;
      }
      if (ret) break;
    }
  v[now]=0;
  return ret;
}
int graph_match(int n,int mat[][maxn],int* match){
  int v[maxn],i,j;
  for (i=0;i<n;i++)
    v[i]=0,match[i]=-1;
  for (i=0,j=n;i<n&&j>=2;)
    if (match[i]<0&&aug(n,mat,match,v,i))
      i=0,j-=2;
    else i++;
```

```
  for (i=j=0;i<n;i++)
    j+=(match[i]>=0);
  return j/2;
}
```

Eulerian circuit (visit every node exactly once)

An undirected graph has a closed Euler tour iff it is connected and each vertex has an even degree.
An undirected graph has an open Euler tour iff it is connected, and each vertex, except for exactly two vertices, has an even degree. The two vertices of odd degree have to be the endpoints of the tour.
A directed graph has a closed Euler tour iff it is strongly connected and the in-degree of each vertex is equal to its out-degree.
Similarly, a directed graph has an open Euler tour iff it is strongly connected and for each vertex the difference between its in-degree and out-degree is 0, except for two vertices. In one of them the difference has to be +1 (this will be the beginning of the tour) and in the other one the difference has to be -1 (this will be its end).

```
// finding euler circuit (undirected) (output reverse order if
directed)
void euler(int u) {
  REP(v, n)
    while(a[u][v])
      a[u][v]--, a[v][u]--, euler(v);
  sol.push_back(u);
}
```

Minimum cut (cut with lowest cut-set)

Definition: cut - partition the edges into 2 partitions. Cut-set (size) is set of edges whose endpoints are in separate partitions

```
int M[maxn][maxn];
int P[maxn];

void contract(int n,int u,int v) {
    // delete edge P[u],P[v]
    for(int i=0; i<n; i++) {
        if(i!=u && i!=v) {
            M[P[i]][P[u]]+=M[P[i]][P[v]];
            M[P[u]][P[i]]+=M[P[v]][P[i]];
        }
    }
    P[v]=P[n-1];
}

int mincutfunc(int n,int a) {
    int s=a,t,cp;
    int A[maxn],D[maxn];
    memset(A,0,sizeof(A));
    memset(D,0,sizeof(D));
    A[a] = 1;
    priority_queue<pair<int,int> > Q;
    for(int i=0; i<n; i++) {
        if(i!=a) {
            D[i] = M[P[i]][P[a]];
            Q.push(make_pair(D[i],i));
        }
    }

    for(int k=1; k<n; k++) {
        while(A[Q.top().second])
            Q.pop();
        int x = Q.top().second;
        Q.pop();

        if(k==n-2)
            s = x;
        else if(k==n-1) {
            t = x;
            cp = D[x];
        }
        A[x] = 1;
        for(int i=0; i<n; i++) {
            if(!A[i]) {
                D[i]+=M[P[i]][P[x]];
                Q.push(make_pair(D[i],i));
            }
        }
    }
    contract(n,s,t);
    return cp;
}

int mincut(int n,int a) {
    int mcp = inf;
    if(n<2)
        return 0;
    if(n==2)
        return M[P[0]][P[1]];
    while(n>1) {
        int cp = mincutfunc(n,a);
        mcp = min(mcp,cp);
        n--;
    }
    return mcp;
}
```

Push-relabel max flow... (just another max flow)

```
int height[maxn],excess[maxn],seen[maxn];
void push(int u,int v,int a[][maxn],int flow[][maxn]) {
    int send = min(excess[u],a[u][v]-flow[u][v]);
```

```
    flow[u][v]+=send, flow[v][u]-=send, excess[u]-=send, excess[v]
+=send;
}
void relabel(int n,int u,int a[][maxn],int source,int flow[][maxn])
{
    int oldh = height[u], minh = height[u], H[maxn]={};
    REP(v,n) if(a[u][v]-flow[u][v]>0) minh<?=height[v];
    height[u]=minh+1;
    REP(v,n) if(height[v]<n) H[height[v]]++;
    FOR(h,1,n-1) if(!H[h]) {
        REP(v,n) if(height[v]>h && v!=source) height[v]>?=n+1;
        break;
    }
}
    void discharge(int n,int u,int a[][maxn],int source,int flow[]
[maxn]) {
        while(excess[u]>0)
            if(seen[u]<n) {
                int v = seen[u];
                if(a[u][v]-flow[u][v]>0 && height[u]>height[v])
push(u,v,a,flow);
                else seen[u]++;
            }
            else {
                relabel(n,u,a,source,flow); seen[u]=0;
            }
    }
int flow(int n,int a[][maxn],int source,int sink,int flow[][maxn]){
    if(source==sink) return inf;
    int ret=0,u=0,oldh,L[maxn],next[maxn],head,prev;
    REP(i,n) REP(j,n) flow[i][j]=0;
    REP(i,n) height[i]=excess[i]=seen[i]=0;
    height[source]=n, excess[source]=inf;
    REP(i,n) if(i!=source && i!=sink) {
        if(u) next[u-1]=u;
        L[u]=i, next[u]=-1, u++;
    }
    REP(v,n) push(source,v,a,flow);
    head=-1;
    if(u) head=0;
    u=head,prev=-1;
    while(u!=-1) {
        int v = L[u];
        oldh=height[v];
        discharge(n,v,a,source,flow);
        if(height[v]>oldh) {
            // move to front of L
            if(prev!=-1) next[prev]=next[u], next[u]=head, head=u;
        }
        prev=u, u=next[u];
    }

    REP(i,n) ret+=flow[source][i];
    return ret;
}
```

Chromatic number (vertex coloring)

```
/*
   11154.cpp vertex/edge coloring

   C[j] is bitmask for vertices with color j
   N[i] is bitmask for vertices adjacent to i
   we sort vertices in nonincreasing degree order
   color[i] = 1..y+1 where y is the number of distinct color used
for vertex
   0..i-1, where y<=i
   c = maximum color used for vertex 0..i-1
   Once we have a solution, we can restrict our search to
colors<maxc
   where maxc is the number of colors in a solution
 */

LL N[maxn],C[maxn];
void backtrack(int c,int y,int i) {
    if(i==n) {
        maxc = c;
        return;
    }
    for(int j=1; j<maxc && j<=y+1; j++) {
        if(!(C[j]&N[i])) {
            C[j]|=1LL<<i;
            backtrack(max(c,j),max(y,j),i+1);
            C[j]&=~(1LL<<i);
        }
        if(max(c,j)>=maxc) return;
    }
}
```

Minimum vertex cover

min num of vertices so each edge is represented

```
/*
   11095.cpp Minimum vertex cover
   for each vertex u, either take u or take all neighbors of u
 */
void search(int mask,int i,int k) {
    if(i==n) {
        if(k<sol) {
            v = mask;
```

```
            sol = k;
        }
        return;
    }
    if(k>=sol)
        return;
    if(mask&(1<<i)) {
        search(mask,i+1,k);
        return;
    }
    search(mask|(1<<i),i+1,k+1);
    int mm = mask, kk=0;
    for(int j=0; j<n; j++)
        if(a[i][j])
            mm|=1<<j;
    for(int j=0; j<n; j++)
        if(mm&(1<<j))
            kk++;
    search(mm,i+1,kk);
}
```

Kth shortest path (1st shortest, 2nd shortest...)

```
int dijkstra(int n,vector<pair<int,int> >* a, int k, int x, int y)
{
    int u,d,v,i,c;
    priority_queue<pair<int,int> > Q;
    int dist[n][k], visit[n];
    memset(visit,0,sizeof(visit));
    Q.push(make_pair(0,x));
    while(!Q.empty()) {
        u = Q.top().second, d = -Q.top().first, Q.pop();
        if(visit[u]<k) dist[u][visit[u]++] = d;
        else continue;
        for(i=0; i<a[u].size(); i++)
            v=a[u][i].first, c=a[u][i].second, Q.push(make_pair(-
(d+c),v));
    }
    return visit[y]<k?-1:dist[y][k-1];
}
```

Graphic sequence

nonincreasing list of degrees [number of edges at a vertex], does the graph exist?

```
/*
   given degree, graph exists iff sum of degree is even, for n=1,
iff degree=0
   sum(d[i],i=1..r) <= r(r-1) + sum(min(r,d[i]),i=r+1..n) for all
1<=r<=n-1
   for each r, find largest i>=r+1 such that r<=d[i]
 */
```

Number of spanning trees

```
/*
   found by determinant of any cofactor of matrix L, where
   L[i][i] = deg(v[i]),
   L[i][j] = -1 if i!=j and v[i],v[j] adjacent,
   L[i][j] = 0 otherwise
 */
```

Directed minimum spanning tree

```
/*
   11183.cpp Directed spanning tree problem (arborescence)
   n-1 edges is MST iff no cycle, and for each node except root,
   exactly one arc enters it

   Chu-Liu/Edmonds algorithm

   1. For each node other than root, select incoming edge with min
cost
   2. If no cycle is formed, then done.
   3. For each cycle, contract the cycle, and modify each edge
entering the
   cycle at node j by mat[i][j] = mat[i][j]-(mat[M[j]][j]-Y) where
   i is a node outside the cycle, Y is the min cost of the arcs
inside cycle,
   M[j],j is the edge that enters j in the cycle
   4. For each cycle, pick the incoming edge with minimum modified
cost
   5. repeat step 2.

 */
```

Minimum diameter spanning tree

```
/*
   0. diam = inf
   1. get BFS tree for each vertex u.
   for each tree do
   a. let d be max(d(u,v)) over all v in tree
   b. let S = {v : d(u,v)=d}
   c. if exists v such that d(u,v)=1 and d = d(v,w)+1 for all w in
S
   then diam = min(diam,2*d-1)
   else diam = min(diam,2*d)

 */
```

Mixed Eulerian circuit

```
/*
   solve with flow (matching) to find an orientation
   idea: create bipartite graph, where A=edges, B=vertices
   u->v iff u is directed and v is destination,
```

```
   or u is undirected and v is either endpoint
   capacity for each vertex B is deg/2 where deg is degree in
underlying graph
   if value of flow is not m, or deg is not even for any vertices,
then no sol
   otherwise use normal euler on the new oriented graph
 */
```

NUMBER THEORY

Sieve of Eratothenes (find primes)

```
// works in under 3s for numbers up to 5,000,000
void runEratosthenesSieve(int upperBound, vector<int> &primes)
{
    int upperBoundSquareRoot = (int)sqrt((double)upperBound);
    vector<bool> isComposite(upperBound + 1, false);

    for (int m = 2; m <= upperBound; m++)
        if (!isComposite[m])
        {
            primes.push_back(m);
            if (m <= upperBoundSquareRoot)
                for (int k = m * m; k <= upperBound; k += m)
                    isComposite[k] = true;
        }
}
```

Euler's totient function

(phi(n) = set of x : 1 < x < n coprime to n)

```
int phi[maxp];
void computephi() {
    phi[1]=1;
    for(int i=2; i<maxp; i++) {
        if(p[i]==i) phi[i]=i-1;
        else if(q[i]==1) phi[i]=p[i]*phi[i/p[i]];
        else phi[i]=phi[i/q[i]]*phi[q[i]];
    }
}
```

GCD, LCM

```
LL gcd(LL a, LL b){ return b?gcd(b, a % b):a; }
LL lcm(LL a, LL b){ return a/gcd(a,b)*b; }

// gcd(a,b) = ax+by, a,b >= 0
LL ext_gcd(LL a,LL b,LL& x,LL& y){
    LL t,ret;
    if (!b){
        x=1,y=0;
        return a;
    }
    ret=ext_gcd(b,a%b,x,y);
    t=x,x=y,y=t-a/b*y;
    return ret;
}

//binary gcd algorithm
//assumes a >= 0, b >= 0, a*b > 0
LL gcd(LL a,LL b) {
    LL g = 0;
    while(a%2==0 && b%2==0)
        g++, a /= 2, b /= 2;

    while(a != 0 && b != 0) {
        while(a%2==0) a /= 2;
        while(b%2==0) b /= 2;
        if(a > b) a -= b;
        else b -= a;
    }
    if(a == 0) return b << g;
    else return a << g;
}
```

Jacobi Symbol (see modular sqrt below)

```
// compute jacobi symbol for odd m>=3, 1 if QR, -1 if not QR, 0 if
n%m=0
// equal to a^((p-1)/2) mod p
LL jacobi(LL n,LL m) {
    int k = 0, ret, m8=m%8;
    if(n%m==0) return 0;
    if(n%2==0) {
        while(n%2==0) k++, n/=2;
        if(m8==1||m8==7) ret=1;
        else ret=-1;
        if(k%2==0) ret*=ret;
        return ret*jacobi(n,m);
    }
    if(n>=m) return jacobi(n%m,m);
    if(n==1) return 1;
    if(n%4==3 && m%4==3) return -jacobi(m,n);
    else return jacobi(m,n);
}
```

Primality test (is n prime?)

```
int modular_exponent(int a,int b,int n){ //a^b mod n
    for (int ret=1;b;b>>=1,a=(int)((long long)a)*a%n)
        if (b&1) ret=(int)((long long)ret)*a%n;
    return ret;
}
int miller_rabin(int n,int time=10){
    if (n==1||(n!=2&&!(n%2))||(n!=3&&!(n%3))||
```

```
            (n!=5&&!(n%5))||(n!=7&&!(n%7))) return 0;
    while (time--)
        if (modular_exponent(((rand()&0x7fff<<16)+rand()&0x7fff+
                    rand()&0x7fff)%(n-1)+1,n-1,n)!=1) return 0;
    return 1;
}
```

Linear modular (a * x mod n = y mod m)

```
// ax=b mod n, a,n > 0
// returns number of solutions
// ext_gcd from GCD/LCM
int modular_linear(LL a,LL b,LL n,LL* sol){
    LL d,e,x,y,i;
    d=ext_gcd(a,n,x,y);
    if (b%d) return 0;
    e=(x*(b/d)%n+n)%n;
    for (i=0;i<d;i++) sol[i]=(e+i*(n/d))%n;
    return d;
}
```

Chinese Remainder Theorem (system of modular equations)

```
/*
   solve x=b[i] mod w[i], 0<=b[i]<w[i], w[i]>=1, 0<=i<k
   sol = solution, n = modulus
   the solution is unique mod n=lcm(w[i])
   returns 1 iff solvable
 */
// ext_gcd from GCD/LCM
int _modular_linear_system(LL b0,LL b1,LL w0,LL w1,LL& a,LL& n) {
    LL x,y,d,g,i,m,w[2],b[2],r[2];
    w[0]=w0,w[1]=w1,b[0]=b0,b[1]=b1,a=0;
    g = ext_gcd(w0,w1,x,y);
    for(i=0;i<2;i++)
        r[i]=b[i]%g, b[i]=(b[i]-r[i])/g, w[i]/=g;
    if(r[0]!=r[1]) return 0;
    n=w[0]*w[1];
    for(i=0;i<2;i++) {
        m=n/w[i];
        d=ext_gcd(w[i],m,x,y);
        a=(a+y*m*b[i])%n;
    }
    a=(a+n)%n;
    a=a*g+r[0], n*=g;
    return 1;
}
int modular_linear_system(int k,LL b[],LL w[],LL& sol,LL& n){
    sol=b[0], n=w[0];
    for(int i=1;i<k;i++)
        if(!_modular_linear_system(sol,b[i],n,w[i],sol,n))
            return 0;
    return 1;
}
```

modular square root (quadratic residues: x^2 = q mod n)

```
LL sqrtmod(LL a) {
    if(p==2) return a;
    if(p%4==3) return powmod(a,(p+1)/4);
    if(p%8==5) {
        LL v = powmod(mult(2,a),(p-5)/8);
        LL i = mult(mult(2,a),mult(v,v));
        return mult(mult(a,v),i-1);
    }

    LL q = p-1, e = 0;
    while(q%2==0) q/=2,e++;
    LL z;
    while(1) {
        LL x = rand()%p;
        if(x<=1) continue;
        z = powmod(x,q);
        if(powmod(z,1LL<<(e-1))!=1) break;
    }
    LL y=z, r=e, x=powmod(a,(q-1)/2), v=mult(a,x), w=mult(v,x);
    while(w!=1) {
        int k;
        LL t = w, d;
        for(k=0;;k++) {
            if(t==1) break;
            t = mult(t,t);
        }
        d = y;
        REP(i,r-k-1) d=mult(d,d);
        y=mult(d,d), r=k, v=mult(d,v), w=mult(w,y);
    }
    return v;
}
```

OTHER MATH

Gaussian elimination (solve linear equations; row reduction)

```
/*
   partial pivot solve a[m][n]x[n]=b[m], modifies a,b
   Using Gauss-Jordan method where a[r][c] is pivot,
   returns -1 (no solution), or k (dimension, number of free var)
   special case: 0 means unique soln
   f indicates the free variables
   computes determinant for square matrices
 */
int gauss(int m,int n,double a[][maxn],double* b,double* x,
        int* f,double& det){
```

```
    int i,j,k,r,c;
    double p;
    det=1;
    for(r=c=0;r<m&&c<n;c++){
        for(p=0,i=r;i<m;i++)
            if(fabs(a[i][c])>fabs(p)) p=a[k=i][c];
        if(fabs(p)<eps) {det=0; continue; }
        if(k!=r){
            for(j=c;j<n;j++) swap(a[r][j],a[k][j]);
            swap(b[r],b[k]), det=-det;
        }
        for(j=c+1;j<n;j++) a[r][j]/=p;
        a[r][c]=1,b[r]/=p,det*=p;
        for(i=0;i<m;i++) {
            if(i==r) continue;
            for(j=c+1;j<n;j++) a[i][j]-=a[i][c]*a[r][j];
            b[i]-=a[i][c]*b[r], a[i][c]=0;
        }
        r++;
    }
    for(i=r;i<m;i++) if(fabs(b[i])>eps) return -1;
    for(j=0;j<n;j++) f[j]=1,x[j]=0;
    for(i=j=0;i<m&&j<n;j++)
        if(a[i][j]==1) f[j]=0,x[j]=b[i++];
    return n-r;
}
```

Simplex algorithm (solve linear programming problems)

```
/*
   simplex algorithm on augmented matrix a of dimension (m+1)x(n+1)
   returns 1 if feasible, 0 if not feasible, -1 if unbounded
   returns solution in b[] in original var order, max(f) in ret
   form: maximize sum_j(a_mj*x_j)-a_mn s.t. sum_j(a_ij*x_j)<=a_in
   in standard form.

   to convert into standard form:
   1. if exists equality constraint, then replace by both >= and <=
   2. if variable x doesn't have nonnegativity constraint, then
replace by
   difference of 2 variables like x1-x2, where x1>=0, x2>=0
   3. for a>=b constraints, convert to -a<=-b

   note: watch out for -0.0 in the solution, algorithm may cycle
   eps = 1e-7 may give wrong answer, 1e-10 is better
 */
void pivot(int m,int n,double a[maxm][maxn],
        int B[maxm],int N[maxn],int r,int c) {
    int i,j;
    swap(N[c],B[r]);
    a[r][c]=1/a[r][c];
    for(j=0;j<=n;j++) if(j!=c) a[r][j]*=a[r][c];
    for(i=0;i<=m;i++)
        if(i!=r) {
            for(j=0;j<=n;j++) if(j!=c) a[i][j]-=a[i][c]*a[r][j];
            a[i][c] = -a[i][c]*a[r][c];
        }
}
int feasible(int m,int n,double a[maxm][maxn],int B[maxm],int
N[maxn]) {
    int r,c,i;
    double p,v;
    while(1) {
        for(p=inf,i=0; i<m; i++) if(a[i][n]<p) p=a[r=i][n];
        if(p>-eps) return 1;
        for(p=0,i=0; i<n; i++) if(a[r][i]<p) p=a[r][c=i];
        if(p>-eps) return 0;
        p = a[r][n]/a[r][c];
        for(i=r+1; i<m; i++)
            if(a[i][c]>eps) {
                v = a[i][n]/a[i][c];
                if(v<p) r=i,p=v;
            }
        pivot(m,n,a,B,N,r,c);
    }
}
int simplex(int m,int n,double a[maxm][maxn],double b[maxn],double&
ret) {
    int B[maxm],N[maxn],r,c,i;
    double p,v;
    for(i=0; i<n; i++) N[i]=i;
    for(i=0; i<m; i++) B[i]=n+i;
    if(!feasible(m,n,a,B,N)) return 0;
    while(1) {
        for(p=0,i=0; i<n; i++) if(a[m][i]>p) p=a[m][c=i];
        if(p<eps) {
            for(i=0; i<n; i++) if(N[i]<n) b[N[i]]=0;
            for(i=0; i<m; i++) if(B[i]<n) b[B[i]]=a[i][n];
            ret = -a[m][n];
            return 1;
        }
        for(p=inf,i=0; i<m; i++)
            if(a[i][c]>eps) {
                v = a[i][n]/a[i][c];
                if(v<p) p=v,r=i;
            }
        if(p==inf) return -1;
        pivot(m,n,a,B,N,r,c);
    }
```

```
}
```

Roots of polynomial

```
/*
   Finding all roots of polynomial using Durand-Kerner or
Weierstrass
   method (based on newton), assuming no repeated roots
   the polynomial is defined as sum(a[i]*x^i, 0<=i<=n), n is degree
 */
void solve(int n, complex<double>* a, complex<double>* root) {
    complex<double> temp, val(0.4,0.9);
    root[0] = 1;
    for(int i=1; i<n; i++) root[i] = root[i-1]*val;
    while(1) {
        double dx = 0;
        for(int i=0; i<n; i++) {
            temp = 0;
            for(int j=n; j>=0; j--)
                temp *= root[i], temp += a[j];
            for(int j=0; j<n; j++)
                if(i!=j) temp /= root[i]-root[j];
            temp = root[i] - temp;
            dx >?= abs(temp-root[i]);
            root[i] = temp;
        }
        if(dx<eps) break;
    }
}
```

Pythagoras triple

```
/*
   Pythagoras triple (a^2 + b^2 = c^2)
   given: gcd(r,s) = 1
          r>s, one odd, one even

   a = r^2-s^2
   b = 2*r*s
   c = r^2+s^2

 */
```

GEOMETRY

Pick's Formula (area of polygon on integer grid)

```
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}
int grid_onedge(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++)
        ret+=gcd(abs(p[i].x-p[(i+1)%n].x),abs(p[i].y-
p[(i+1)%n].y));
    return ret;
}
int grid_inside(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++)
        ret+=p[(i+1)%n].y*(p[i].x-p[(i+2)%n].x);
    return (abs(ret)-grid_onedge(n,p))/2+1;
}
```

Ham sandwich (cut n-dimensional object in half)

```
/*
   10797.cpp
   Ham Sandwich cut
   O((log X)*(n log n))

   assuming computing median in O(nlogn) using stl which is slow
   the dual of points left of y-axis has negative slope, right of
y-axis
   has positive slope, and the median of them gives the bisector
   We need to find the intersection of the 2 medians to find the
cut.
   Given x, we need to find the medians at x
   The function f(x) = Lmedian(x)-Rmedian(x) is monotonic
decreasing
   so we use binary search to find x where Lmedian(x)=Rmedian(x)
 */

struct point {
    double x,y;
};

int m,n;
point P[maxn],Q[maxn];

double median(int n,point* P,double x,int& k) {
    pair<double,int> Q[n];
    REP(i,n) Q[i] = make_pair(P[i].x*x+P[i].y,i);
    sort(Q,Q+n);
    k = Q[n/2].second;
    return Q[n/2].first;
}

void doit(int &si,int &sj) {
    double a=-inf, b=inf;
    while(b-a>eps) {
        double x,y1,y2;
        x = (a+b)/2;
```

```
        y1 = median(m,P,x,si);
        y2 = median(n,Q,x,sj);
        if(y1>y2) a = x;
        else b = x;
    }
}
```

DATA STRUCTURES

Binary index tree (specific operations are fast)

```
// 1d dynamic sum a[0..n-1] (query(i) returns cumulative sum
a[0..i-1])
#define lowbit(x) ((x)&((x)^((x)-1)))
struct sum{
    int a[maxn],c[maxn],ret;
    int n;
    void init(int i)
{memset(a,0,sizeof(a));memset(c,0,sizeof(c));n=i;}
    void update(int i,int v){for (v-=a[i],a[i++]+=v;i<=n;c[i-
1]+=v,i+=lowbit(i));}
    int query(int i){for (ret=0;i;ret+=c[i-1],i^=lowbit(i));return
ret;}
};

// 2d dynamic submatrix sum a[0..m-1][0..n-1]
#define lowbit(x) ((x)&((x)^((x)-1)))
struct sum{
    int a[maxn][maxn],c[maxn][maxn],ret;
    int m,n,t;
    void init(int i,int j){
        memset(a,0,sizeof(a));memset(c,0,sizeof(c));m=i,n=j;}
    void update(int i,int j,int v){
        for (v-=a[i][j],a[i++][j++]+=v,t=j;i<=m;i+=lowbit(i))
            for (j=t;j<=n;c[i-1][j-1]+=v,j+=lowbit(j));
    }
    int query(int i,int j){
        for (ret=0,t=j;i;i^=lowbit(i))
            for (j=t;j;ret+=c[i-1][j-1],j^=lowbit(j));
        return ret;
    }
};
```

Interval tree

quick to find overlapping intervals and stuff

```
// stores binary tree in [1..2*maxn-1], the leafs are
[maxn..2*maxn-1]
// assumes maxn is a power of 2, query searches between [x..y]
struct tree {
    int a[2*maxn];
    void init() {for(int i=0;i<2*maxn;i++) a[i]=inf;}
    void update(int x,int i,int b,int v) {
        if(b>1) {
            if(x>=(b>>1)) update(x-(b>>1),(i<<1)+1,b>>1,v);
            else update(x,i<<1,b>>1,v);
            a[i]=a[i<<1]<?a[(i<<1)+1];
        }
        else a[i]=v;
    }
    void update(int x,int v) {update(x,1,maxn,v);}
    int query(int x,int y,int i,int b,int c) {
        if(x<=b && c<=y) return a[i];
        if(y<b || c<x) return inf;
        int m = (b+c+1)>>1;
        return query(x,y,i<<1,b,m-1)<?query(x,y,(i<<1)+1,m,c);
    }
    int query(int x,int y) {return query(x,y,1,0,maxn-1);}
};
```

union find (search or merge partitions of a set)

```
// union find indexed (1..maxn-1)
#define _ufind_run(x) for(;p[t=x];x=p[x],p[t]=(p[x]?p[x]:x))
#define _run_both _ufind_run(i);_ufind_run(j)
struct ufind{
    int p[maxn],t;
    void init(){memset(p,0,sizeof(p));}
    void set_friend(int i,int j){_run_both;p[i]=(i==j?0:j);}
    int is_friend(int i,int j){_run_both;return i==j&&i;}
};

// friend / enemy
#define sig(x) ((x)>0?1:-1)
#define abs(x) ((x)>0?(x):-(x))
#define _ufind_run(x)
for(;p[t=abs(x)];x=sig(x)*p[abs(x)],p[t]=sig(p[t])*(p[abs(x)]?
p[abs(x)]:abs(p[t])))
#define _run_both _ufind_run(i);_ufind_run(j)
#define _set_side(x) p[abs(i)]=sig(i)*(abs(i)==abs(j)?0:(x)*j)
#define _judge_side(x) (i==(x)*j&&i)
struct ufind{
    int p[maxn],t;
    void init(){memset(p,0,sizeof(p));}
    int set_friend(int i,int j){
        _run_both;_set_side(1);return !_judge_side(-1);}
    int set_enemy(int i,int j){
        _run_both;_set_side(-1);return !_judge_side(1);}
    int is_friend(int i,int j){_run_both;return _judge_side(1);}
    int is_enemy(int i,int j){_run_both;return _judge_side(-1);}
};
```

Suffix tree:

String search

Longest repeated substring

longest common substring

longest palindrome

```
// a,b denotes transition on string s[a..b-1]
// assumes the string s is already normalized (alphabet = 0..maxk-
1)
// maxk is the special end of string marker appended to string
// len denotes the length of substring that state corresponds to
#define maxk 26
char s[maxn];
int n,I;
struct node {
    int child[maxk+1],slink,a,b,parent,len;
};
node tree[2*maxn];

void clearnode(int u) {
    REP(i,maxk+1) tree[u].child[i]=0;
    tree[u].slink=tree[u].a=tree[u].b=0,tree[u].parent=0;
    tree[u].len=-1;
}

bool test(int& u,int a,int b,char c) {
    if(a==b) return tree[u].child[c];
    int v = tree[u].child[s[a]], aa = tree[v].a, bb = tree[v].b;
    if(c==s[aa+b-a]) return true;
    int r = I++;
    clearnode(r);
    tree[u].child[s[aa]]=r, tree[r].a=aa, tree[r].b=aa+b-a,
tree[r].parent=u;
    tree[r].child[s[aa+b-a]] = v, tree[v].a+=b-a, tree[v].parent=r;
    u = r;
    return false;
}

int canonize(int u,int& a,int b) {
    if(a==b) return u;
    int v = tree[u].child[s[a]], d = tree[v].b-tree[v].a;
    while(d<=b-a) {
        a += d, u = v;
        if(a<b) {
            v = tree[u].child[s[a]], d = tree[v].b-tree[v].a;
        }
    }
    return u;
}

void update(int& u,int& a,int i) {
    int root=1, oldr=root, r=u;
    while(!test(r,a,i,s[i])) {
        int rr = I++;
        clearnode(rr);
        tree[r].child[s[i]]=rr, tree[rr].a=i, tree[rr].b=n,
tree[rr].parent=r;
        if(oldr!=root) tree[oldr].slink = r;
        oldr = r;
        u = canonize(tree[u].slink,a,i);
        r = u;
    }
    if(oldr!=root) tree[oldr].slink = u;
}

void marknode(int u) {
    int v  = tree[u].parent;
    if(tree[v].len==-1)
        marknode(v);
    tree[u].len = tree[v].len + tree[u].b-tree[u].a;
}

void buildtree() {
    // 0 is the auxillary node, 1 is the root
    I = 2;
    clearnode(0);
    clearnode(1);
    REP(i,maxk+1) tree[0].child[i] = 1;
    tree[0].parent = -1, tree[0].len=-1;
    tree[1].a=-1, tree[1].b=0, tree[1].parent=0, tree[1].len=0;
    int u,k,i;
    s[n++] = maxk;
    for(i=0,k=0,u=1;i<n;i++) {
        update(u,k,i);
        u = canonize(u,k,i+1);
    }
    FOR(i,1,I-1) if(tree[i].len==-1)  marknode(i);
}
```

Segment tree (intervals again)

```
struct segtree{
    int n,cnt[maxn],len[maxn],cut[maxn],bl[maxn],br[maxn];
    segtree(int t):n(t){
        for (int i=1;i<=t;i++)
            cnt[i]=len[i]=cut[i]=bl[i]=br[i]=0;
    };
    void update(int t,int l,int r);
```

```
    void inc_seg(int t,int l0,int r0,int l,int r);
    void dec_seg(int t,int l0,int r0,int l,int r);
    int seg_len(int t,int l0,int r0,int l,int r);
    int seg_cut(int t,int l0,int r0,int l,int r);
};

int length(int l,int r){
    return r-l;
};

void segtree::update(int t,int l,int r){
    if (cnt[t]||r-l==1)
        len[t]=length(l,r),cut[t]=bl[t]=br[t]=1;
    else{
        len[t]=len[t+t]+len[t+t+1];
        cut[t]=cut[t+t]+cut[t+t+1];
        if (br[t+t]&&bl[t+t+1])
            cut[t]--;
        bl[t]=bl[t+t],br[t]=br[t+t+1];
    }
}
void segtree::inc_seg(int t,int l0,int r0,int l,int r){
    if (l0==l&&r0==r)
        cnt[t]++;
    else{
        int m0=(l0+r0)>>1;
        if (l<m0)
            inc_seg(t+t,l0,m0,l,m0<r?m0:r);
        if (r>m0)
            inc_seg(t+t+1,m0,r0,m0>l?m0:l,r);
        if (cnt[t+t]&&cnt[t+t+1]){
            cnt[t+t]--;
            update(t+t,l0,m0);
            cnt[t+t+1]--;
            update(t+t+1,m0,r0);
            cnt[t]++;
        }
    }
    update(t,l0,r0);
}
void segtree::dec_seg(int t,int l0,int r0,int l,int r){
    if (l0==l&&r0==r)
        cnt[t]--;
    else if (cnt[t]){
        cnt[t]--;
        if (l>l0)
            inc_seg(t,l0,r0,l0,l);
        if (r<r0)
            inc_seg(t,l0,r0,r,r0);
    }
    else{
        int m0=(l0+r0)>>1;
        if (l<m0)
            dec_seg(t+t,l0,m0,l,m0<r?m0:r);
        if (r>m0)
            dec_seg(t+t+1,m0,r0,m0>l?m0:l,r);
    }
    update(t,l0,r0);
}
int segtree::seg_len(int t,int l0,int r0,int l,int r){
    if (cnt[t]||(l0==l&&r0==r))
        return len[t];
    else{
        int m0=(l0+r0)>>1,ret=0;
        if (l<m0)
            ret+=seg_len(t+t,l0,m0,l,m0<r?m0:r);
        if (r>m0)
            ret+=seg_len(t+t+1,m0,r0,m0>l?m0:l,r);
        return ret;
    }
}
int segtree::seg_cut(int t,int l0,int r0,int l,int r){
    if (cnt[t])
        return 1;
    if (l0==l&&r0==r)
        return cut[t];
    else{
        int m0=(l0+r0)>>1,ret=0;
        if (l<m0)
            ret+=seg_cut(t+t,l0,m0,l,m0<r?m0:r);
        if (r>m0)
            ret+=seg_cut(t+t+1,m0,r0,m0>l?m0:l,r);
        if (l<m0&&r>m0&&br[t+t]&&bl[t+t+1])
            ret--;
        return ret;
    }
}
```

Color range tree (no clue)

```
struct node{
    int a, b;
    int color;
    int left, right;
};

int I, ans, v[maxn][2], visited[maxn];
node tree[maxn];
```

```
int construct(int a, int b){
    if (a == b) return 0;

    int i = I++;
    tree[i].a = a;
    tree[i].b = b;
    if (a + 1 == b) tree[i].left = tree[i].right = 0;
    else {
        tree[i].left = construct(a, (a+b) >> 1);
        tree[i].right = construct((a+b) >> 1, b);
    }

    if (tree[tree[i].left].color == tree[tree[i].right].color)
        tree[i].color = tree[tree[i].left].color;
    else tree[i].color = 0;

    return i;
}

void update(int i, int a, int b, int color){
    a = max(a, tree[i].a);
    b = min(b, tree[i].b);

    if (a >= b) return;
    if (!i) return;

    if (tree[i].a >= b || tree[i].b <= a) return;
    if (a <= tree[i].a && tree[i].b <= b) tree[i].color = color;
    else {
        int c = tree[i].color;

        if (c){
            update(tree[i].left, tree[i].a, a, c);
            update(tree[i].left, b, tree[i].b, c);

            update(tree[i].right, tree[i].a, a, c);
            update(tree[i].right, b, tree[i].b, c);
        }

        update(tree[i].left, a, b, color);
        update(tree[i].right, a, b, color);

        if (tree[tree[i].left].color == tree[tree[i].right].color)
            tree[i].color = tree[tree[i].left].color;
        else tree[i].color = 0;
    }
}

void count(int i){
    if (!tree[i].color && i){
        count(tree[i].left);
        count(tree[i].right);
    } else {
        if (tree[i].color && !visited[tree[i].color]) {
            ans++;
            visited[tree[i].color] = 1;
        }
    }
}
```

MISC

KMP (search text for a string [efficiently])

```
// s is the string to be searched, t is the string to search for
void init(string t,int* fail) {
    int i=2, j=0, m=t.length();
    fail[0]=-1, fail[1]=0;
    while(i<m) {
        if(t[i-1]==t[j]) fail[i]=j+1, i++, j++;
        else if(j) j=fail[j];
        else fail[i]=0, i++;
    }
}
int kmp_match(string s,string t) {
    int n=s.length(), m=t.length(), k, i, fail[m];
    init(t,fail);
    for(k=i=0;k+i<n;) {
        if(s[k+i]==t[i]) { if(++i==m) return k; }
        else {
            k += i-fail[i];
            if(i) i=fail[i];
        }
    }
    return n;
}
```

Longest increasing subsequence

```
#define _cp(a,b) ((a)<(b)) // increase
int subseq(int n,T* a,T* t, T* pre){
    int b[maxn], i,l,r,m,ret=0;

    for (i=0;i<n;t[i] = l, pre[b[l]=i++]=b[l-1],ret+=(l>ret))
        for (m=((l=1)+(r=ret))>>1;l<=r;m=(l+r)>>1)
            if (_cp(a[b[m]],a[i])) l=m+1;
            else r=m-1;

    return ret;
}
```

2d sub max (dunno)

```
int maxsum(int m,int n,int mat[][maxn],int& s1,int& s2,int& e1,int&
e2){
    int matsum[maxn][maxn+1],ret,sum;
    int i,j,k,s;
    for (i=0;i<m;i++)
        for (matsum[i][j=0]=0;j<n;j++)
            matsum[i][j+1]=matsum[i][j]+mat[i][j];
    for (ret=mat[s1=e1=0][s2=e2=j=0];j<n;j++)
        for (k=j;k<n;k++)
            for (sum=0,s=i=0;i<m;i++,s=(sum>0?s:i))
                if ((sum=(sum>0?sum:0)+matsum[i][k+1]-matsum[i]
[j])>ret)
                    ret=sum,s1=s,s2=i,e1=j,e2=k;
    return ret;
}
```

Josepheus (avoid suicide)

```
// josepheus n people, count k
int joseph(int n,int k){
    int ret=-1,i;
    for (i=1;i<=n;i++)
        ret=(ret+k)%i;
    return ret+1;
}
```

Inversion (not sure what they're inverting)

```
#define _cp(a,b) ((a)<=(b))
int _tmp[maxn];
int inv(int n,int* a){
    int l=n>>1,r=n-l,i,j;
    int ret=(r>1?(inv(l,a)+inv(r,a+l)):0);
    for (i=j=0;i<=l;_tmp[i+j]=a[i],i++)
        for (ret+=j;j<r&&(i==l||!
_cp(a[i],a[l+j]));_tmp[i+j]=a[l+j],j++);
    memcpy(a,_tmp,sizeof(int)*n);
    return ret;
}
```

Kth element (not sure)

```
// kth element expected O(n), k=0..n-1, modifies a[]
#define _cp(a,b) ((a)<(b))
int kth_element(int n,int* a,int k){
    int t,key;
    int l=0,r=n-1,i,j;
    while (l<r){
        for (key=a[((i=l-1)+(j=r+1))>>1];i<j;){
            for (j--;_cp(key,a[j]);j--);
            for (i++;_cp(a[i],key);i++);
            if (i<j) t=a[i],a[i]=a[j],a[j]=t;
        }
        if (k>j) l=j+1;
        else r=j;
    }
    return a[k];
}
```

Latin square

sudoku with only rows and columns

```
// magic square, l!=2
void dllb(int l,int si,int sj,int sn,int d[][maxn]){
    int n,i=0,j=l/2;
    for (n=1;n<=l*l;n++){
        d[i+si][j+sj]=n+sn;
        if (n%l){
            i=(i)?(i-1):(l-1);
            j=(j==l-1)?0:(j+1);
        }
        else
            i=(i==l-1)?0:(i+1);
    }
}
void magic_odd(int l,int d[][maxn]){
    dllb(l,0,0,0,d);
}
void magic_4k(int l,int d[][maxn]){
    int i,j;
    for (i=0;i<l;i++)
        for (j=0;j<l;j++)
            d[i][j]=((i%4==0||i%4==3)&&(j%4==0||j%4==3)||
                    (i%4==1||i%4==2)&&(j%4==1||j%4==2))?(l*l-
(i*l+j)):(i*l+j+1);
}
void magic_other(int l,int d[][maxn]){
    int i,j,t;
    dllb(l/2,0,0,0,d);
    dllb(l/2,l/2,l/2,l*l/4,d);
    dllb(l/2,0,l/2,l*l/2,d);
    dllb(l/2,l/2,0,l*l/4*3,d);
    for (i=0;i<l/2;i++)
        for (j=0;j<l/4;j++)
            if (i!=l/4||j)
                t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;
    t=d[l/4][l/4],d[l/4][l/4]=d[l/4+l/2][l/4],d[l/4+l/2][l/4]=t;
```

```
    for (i=0;i<l/2;i++)
        for (j=l-l/4+1;j<l;j++)
            t=d[i][j],d[i][j]=d[i+l/2][j],d[i+l/2][j]=t;
}
    void generate(int l,int d[][maxn]){
        if (l%2)
            magic_odd(l,d);
        else if (l%4==0)
            magic_4k(l,d);
        else
            magic_other(l,d);
    }
```

Gray code (only change one bit at a time)

```
// reflected gray code (start 000...000, end 100...000)
void gray(int n,int *code){
    int t=0,i;
    for (i=0;i<n;t+=code[i++]);
    if (t&1)
        for (n--;!code[n];n--);
    code[n-1]=1-code[n-1];
}
```

Comb & Perm

```
// int <-> C(n,m), n>=m
int comb(int n,int m){
    int ret=1,i;
    m=m<(n-m)?m:(n-m);
    for (i=n-m+1;i<=n;ret*=(i++));
    for (i=1;i<=m;ret/=(i++));
    return m<0?0:ret;
}
int comb2num(int n,int m,int *c){
    int ret=comb(n,m),i;
    for (i=0;i<m;i++)
        ret-=comb(n-c[i],m-i);
    return ret;
}
void num2comb(int n,int m,int* c,int t){
    int i,j=1,k;
    for (i=0;i<m;c[i++]=j++)
        for (;t>(k=comb(n-j,m-i-1));t-=k,j++);
}

// int <-> permutation
int perm2num(int n,int *p){
    int i,j,ret=0,k=1;
    for (i=n-2;i>=0;k*=n-(i--))
        for (j=i+1;j<n;j++)
            if (p[j]<p[i])
                ret+=k;
    return ret;
}
void num2perm(int n,int *p,int t){
    int i,j;
    for (i=n-1;i>=0;i--)
        p[i]=t%(n-i),t/=n-i;
    for (i=n-1;i;i--)
        for (j=i-1;j>=0;j--)
            if (p[j]<=p[i])
                p[i]++;
}

// generate perm, comb
int count;
void dummy(int* a,int n){
    int i;
    cout<<count++<<": ";
    for (i=0;i<n-1;i++)
        cout<<a[i]<<' ';
    cout<<a[n-1]<<endl;
}
void _gen_perm(int* a,int n,int m,int l,int* temp,int* tag){
    int i;
    if (l==m)
        dummy(temp,m);
    else
        for (i=0;i<n;i++)
            if (!tag[i]){
                temp[l]=a[i],tag[i]=1;
                _gen_perm(a,n,m,l+1,temp,tag);
                tag[i]=0;
            }
}
void gen_perm(int n,int m){
    int a[maxn],temp[maxn],tag[maxn]={0},i;
    for (i=0;i<n;i++)
        a[i]=i+1;
    _gen_perm(a,n,m,0,temp,tag);
}
void _gen_comb(int* a,int s,int e,int m,int& count,int* temp){
    int i;
    if (!m)
        dummy(temp,count);
    else
        for (i=s;i<=e-m+1;i++){
            temp[count++]=a[i];
```

```
            _gen_comb(a,i+1,e,m-1,count,temp);
            count--;
        }
}
void gen_comb(int n,int m){
    int a[maxn],temp[maxn],count=0,i;
    for (i=0;i<n;i++)
        a[i]=i+1;
    _gen_comb(a,0,n-1,m,count,temp);
}
void _gen_perm_swap(int* a,int n,int l,int* pos,int* dir){
    int i,p1,p2,t;
    if (l==n)
        dummy(a,n);
    else{
        _gen_perm_swap(a,n,l+1,pos,dir);
        for (i=0;i<l;i++){
            p2=(p1=pos[l])+dir[l];
            t=a[p1],a[p1]=a[p2],a[p2]=t;
            pos[a[p1]-1]=p1,pos[a[p2]-1]=p2;
            _gen_perm_swap(a,n,l+1,pos,dir);
        }
        dir[l]=-dir[l];
    }
}
void gen_perm_swap(int n){
    int a[maxn],pos[maxn],dir[maxn],i;
    for (i=0;i<n;i++)
        a[i]=i+1,pos[i]=i,dir[i]=-1;
    _gen_perm_swap(a,n,0,pos,dir);
}
```

Parsing (expression parsing, like a calculator)

```
/*
   assign = var=expr
   expr = expr+term | expr-term | term
   term = term*factor | term/factor | factor
   factor = -factor | +factor | (expr) | const | var

   be careful of -0.00

 */
double M[256];
string s;
int I;
double expr();

double factor() {
    char c = s[I++];
    if(c=='-') return -factor();
    else if(c=='+') return factor();
    else if(c=='(') {
        double r = expr();
        I++;
        return r;
    }
    else if(islower(c)) return M[c];
    else if(isdigit(c)) {
        string t;
        for(I--;I<s.length() && isdigit(s[I]); I++) t+=s[I];
        return atof(t.c_str());
    }
}
double term() {
    double r = factor();
    while(I<s.length() && (s[I]=='*'||s[I]=='/')) {
        if(s[I++]=='*') r *= factor();
        else r /= factor();
    }
    return r;
}
double expr() {
    double r = term();
    while(I<s.length() && (s[I]=='+'||s[I]=='-')) {
        if(s[I++]=='+') r += term();
        else r -= term();
    }
    return r;
}
void assign() {
    int k = I;
    I+=2;
    M[s[k]] = expr();
}
```

BW Transform (compression)

```
/*
   k is position of the initial string in (sorted) last col
 */
string IBW_transform(string s,int k) {
    int n=s.length(),i;
    string t;
    pair<char,int> a[n];
    for(i=0;i<n;i++) a[i] = make_pair(s[i],i);
    sort(a,a+n);
    for(i=k;n--;t+=a[i].first,i=a[i].second);
```

```
    return t;
}
```

Inverse of phi(n)

```
/*
   Find inverse of euler phi
   find all n such that phi(n) = x for a given 1<=x<=1e9
 */

#define maxp 100000
int nsol,nprime,P[maxp],B[maxp];
long long m,sol[maxp];

bool isprime(int n) {
    if(n<maxp) return !B[n];
    for(int i=0; i<nprime && P[i]*P[i]<=n; i++)
        if(n%P[i]==0) return false;
    return true;
}
void recurse(int n,int p) {
    // we know we found solution
    if(n==1 && P[p]>2) {
        sol[nsol++] = m;
        return;
    }
    // if p odd, product of p-1 must be even
    if(n%2==1 && P[p]>2)
        return;
    // check for the form phi = (p-1)(q-1) or phi=p-1
    if(n < (P[p]-1)*P[p] && n>P[p]-1) {
        if(n%(P[p]-1)==0) {
            m *= P[p], n/= P[p]-1;
            if(isprime(n+1) && n+1>P[p])
                m *= n+1, sol[nsol++] = m, m /= n+1;
            n *= P[p]-1, m/= P[p];
        }
        if(isprime(n+1))
            m *= n+1, sol[nsol++] = m, m /= n+1;
        return;
    }
    // n too small
    if(n < P[p]-1)
        return;
    // general backtracking
    if(n % (P[p]-1)==0) {
        m *= P[p], n /= P[p]-1;
        recurse(n,p+1);
        int t;
        for(t=0; n%P[p]==0; t++) {
            n /=P[p], m *= P[p];
            recurse(n,p+1);
        }
        while(t--)
            m/=P[p], n*=P[p];
        m /= P[p], n *= P[p]-1;
    }
    recurse(n,p+1);
}

void init() {
    B[0]=B[1]=1;
    for(int i=2; i*i<maxn; i++) {
        if(!B[i])
            for(int j=i*i; j<maxn; j+=i)
                B[j] = 1;
    }
    nprime = 0;
    for(int i=2; i<maxn; i++)
        if(!B[i]) P[nprime++] = i;
}

int main() {
    int n;
    init();
    while(cin>>n) {
        nsol = 0;
        m = 1;
        recurse(n,0);
        sort(sol,sol+nsol);
        if(!nsol)
            cout << "No solution.";
        else
            for(int i=0; i<nsol; i++)
                if(i)
                    cout << ' '<<sol[i];
                else
                    cout << sol[i];
        cout <<endl;
        //cout << nsol << endl;
    }

}
```

de Bruijn

sequence for alphabet A s.t. every subsequence of length n appears exactly once

```
/*
```

```
   10040.cpp
   Computes lex smallest de bruijn's sequence using euler cycle
method
   uses stack instead of recursion to avoid stack overflow
 */
int n,k;
char M[1<<maxn];
char sol[1<<(maxn+1)];

void dfs(char p,int s) {
    stack<pair<char,int> > S;
    S.push(make_pair(p,s));
    memset(M,-1,sizeof(M));
    int len = 0;
    while(S.size()) {
        p = S.top().first, s = S.top().second;
        char x = -1;
        int t;
        REP(i,2)
            if(M[s]&(1<<i)) {
                M[s]&=~(1<<i);
                t = s<<1;
                t |= i;
                t &= (1<<(n-1))-1;
                x = i;
                break;
            }
        if(x!=-1) S.push(make_pair(x,t));
        else {
            if(p!=-1) {
                sol[(1<<n)+(1<<n)-1-len] = p;
                len++;
            }
            S.pop();
        }
    }
}
int main(){
    for(n=1; n<=maxn; n++) {
        int s = (1<<(n-1))-1;
        dfs(-1,s);
    }
    int T;
    cin>>T;
    while(T--) {
        cin>>n>>k;
        int x = 0;
        REP(i,n) x<<=1, x+=sol[(1<<n)+(k+i)%(1<<n)];
        cout << x << endl;
    }
}
```

Poker

```
string W[]={"","highest-card","one-pair","two-pairs","three-of-a-
kind","straight","flush","full-house","four-of-a-kind","straight-
flush"};
string ranks="0023456789TJQKA";
string suits="CDHS";

struct card_t{
    int rank, suit;
    card_t(){}
    card_t(string s) {
        rank=ranks.find(s[0]);
        suit=suits.find(s[1]);
    }
    bool operator<(const card_t &b) const{
        return (rank!=b.rank)?rank<b.rank:suit<b.suit;
    }
};

struct node{
    int rank, freq;
    bool operator<(const node &b) const {
        return (freq!=b.freq)?freq>b.freq:rank>b.rank;
    }
};

bool straight(vector<card_t> U, vector<int> &ret) {
    ret.clear();
    for (int i=1;i<U.size()-1;i++)
        if (U[i].rank!=U[i-1].rank+1) return false;
    if (U[4].rank!=U[3].rank+1&&(U[3].rank!= 5||U[4].rank!=14))
return false;
    if (U[4].rank!=U[3].rank+1) {
        ret.push_back(5);
        ret.push_back(U[3].rank);
    }
    ret.push_back(5);
    ret.push_back(U[4].rank);
    return true;
}

bool flush(vector<card_t> U, vector<int> &ret) {
    ret.clear();
    for (int i=1;i<U.size();i++)
        if (U[i].suit != U[i-1].suit) return false;
```

```
    ret.push_back(6);
    for (int i=U.size()-1;i>=0;i--) ret.push_back(U[i].rank);
    return true;
}

bool sort2(vector<card_t> U, vector<node> &V) {
    int freq[15]; node temp;
    V.clear();
    memset(freq,0,sizeof(freq));
    for (int i=0;i<U.size();i++) freq[U[i].rank]++;
    for (temp.rank=0;temp.rank<15;temp.rank++)
        if (freq[temp.rank]>0) {
            temp.freq=freq[temp.rank];
            V.push_back(temp);
        }
    sort(V.begin(),V.end());
}

vector<int> type(vector<card_t> U) {
    vector<int> ret, ret2;
    sort(U.begin(),U.end());
    if (straight(U,ret)) {
        if (flush(U, ret2))ret[0]=9;
        return ret;
    }
    else if (flush(U,ret)) {
        return ret;
    }

    vector<node> V;
    sort2(U,V);
    ret.resize(1);
    for (int i=0;i<V.size();i++) {
        ret.push_back(V[i].rank);
    }
    switch(V[0].freq) {
        case 4:
            ret[0]=8; break;
        case 3:
            if (V[1].freq==2) ret[0]=7;
            else ret[0]=4;
            break;
        case 2:
            if (V[1].freq==2) ret[0]=3;
            else ret[0]=2;
            break;
        case 1:
            ret[0]=1;
            break;
    }
    return ret;
}

int compare(vector<card_t> U, vector<card_t> V) {
    vector<int> ans1=type(U);
    vector<int> ans2=type(V);
    for (int i=0;i<U.size();i++)
        if (ans1[i]!=ans2[i]) return ans1[i]-ans2[i];
    return 0;
}

vector<card_t> process(vector<string> V) {
    vector<card_t> U;
    for (int i=0;i<5;i++) U.push_back(card_t(V[i]));
    return U;
}
```

The first 375 prime numbers

There are 15 consecutive primes in each of the 25 rows.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
2	3	5	7	11	13	17	19	23	29	31	37	41	43	47
53	59	61	67	71	73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173	179	181	191	193	197
199	211	223	227	229	233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349	353	359	367	373	379
383	389	397	401	409	419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541	547	557	563	569	571
577	587	593	599	601	607	613	617	619	631	641	643	647	653	659
661	673	677	683	691	701	709	719	727	733	739	743	751	757	761
769	773	787	797	809	811	821	823	827	829	839	853	857	859	863
877	881	883	887	907	911	919	929	937	941	947	953	967	971	977
983	991	997	1009	1013	1019	1021	1031	1033	1039	1049	1051	1061	1063	1069
1087	1091	1093	1097	1103	1109	1117	1123	1129	1151	1153	1163	1171	1181	1187
1193	1201	1213	1217	1223	1229	1231	1237	1249	1259	1277	1279	1283	1289	1291
1297	1301	1303	1307	1319	1321	1327	1361	1367	1373	1381	1399	1409	1423	1427
1429	1433	1439	1447	1451	1453	1459	1471	1481	1483	1487	1489	1493	1499	1511
1523	1531	1543	1549	1553	1559	1567	1571	1579	1583	1597	1601	1607	1609	1613
1619	1621	1627	1637	1657	1663	1667	1669	1693	1697	1699	1709	1721	1723	1733
1741	1747	1753	1759	1777	1783	1787	1789	1801	1811	1823	1831	1847	1861	1867
1871	1873	1877	1879	1889	1901	1907	1913	1931	1933	1949	1951	1973	1979	1987
1993	1997	1999	2003	2011	2017	2027	2029	2039	2053	2063	2069	2081	2083	2087
2089	2099	2111	2113	2129	2131	2137	2141	2143	2153	2161	2179	2203	2207	2213
2221	2237	2239	2243	2251	2267	2269	2273	2281	2287	2293	2297	2309	2311	2333
2339	2341	2347	2351	2357	2371	2377	2381	2383	2389	2393	2399	2411	2417	2423
2437	2441	2447	2459	2467	2473	2477	2503	2521	2531	2539	2543	2549	2551	2557