Documentación Legendary Grandpupils

1.	Plantilla	3
2.	DP	4
	NK	4
	Longest Common Subsequence	5
	Longest Common Substring	6
	TSP	7
3.	Geometry	8
4.	Graph's Theory	15
	Centroid decomposition	15
	HLD	17
	Isomorphic Rooted Tree Test	19
	Dijkstra	21
	DSU	22
	Find Cycles	24
	Floyd-Warshall	25
	Kruskal	26
	Longest Path in DAG	28
	Maximum Diameter in a Graph	29
	Maximum Bipartite Matching	30
	Strongly Connected Components	31
	Topological Sort	32
5.	Number Theory	33
	Fermat Little's theorem	33
	Fibonacci	34
	Prime Numbers	35
	Triangle of Mahonian	40
6.	Segment Tree	41
	Lazy Propagation	41
	Persistent Segment Tree	42
7.	Treap	43
8.	Strings	45
	Aho-Corasick	45
	KMP	47
	Suffix Array	48

	Z Function	.49
9.	Flows	.50
	Dinic	.50
	Edmons-Karp	.52
	Hungarian	.53
	Min-cost flow	.55
10	O. Variuos	.57
	Gauss-Jordan	.57
	2-Sat	.59
	Bits	.60
	Fast IO	61
	FFT 1	. 62
	FFT 2	. 64
	Fraction	. 66
	Hour	. 68
	Matrix Exponentiation	. 70
	Roman Numbers	.71
	Rotate Matrix	. 72
	Sorts	. 73
11	L. Otros	. 76
	Series	
	First 5000 digits of PI	
	First 150 Fibonacci numbers	
	First 1000 prime numbers	
	First 25 Catalan numbers	
	First 100 powers of 2	
	First 30 Rows of Pascal Triangle	
	Código ASCII	
	Laws and facts	
	LUVY J UTIN TUCKS	· OU

1. Plantilla

```
    #include <bits/stdc++.h>
    #include <algorithm>
    #include <bitset>

4. #include <cmath>5. #include <cstdio>6. #include <cstring>

    #include <deque>
    #include <fstream>
    #include <functional>

10. #include <iomanip>
11. #include <iostream>
12. #include <limits.h>
13. #include <map>
14. #include <math.h>
15. #include <numeric>
16. #include <queue>
17. #include <set>
18. #include <sstream>
19. #include <stack>
20. #include <stdio.h>
21. #include <stdlib.h>
22. #include <string>
23. #include <utility>
24. #include <vector>
25.
26. #define PI 3.14159265358979323846
27. #define EPS 1e-6
28. #define INF 1000000000
29.
30. \#define \_ios\_base::sync\_with\_stdio(0), cin.tie(0), cin.tie(0), cout.tie(0), cout.precision(15);
31. #define FOR(i, a, b) for(int i=int(a); i<int(b); i++)
32. #define RFOR(i, a, b) for(int i=int(a)-1; i>=int(b); i--)
33. #define FORC(cont, it) for(decltype((cont).begin()) it = (cont).begin(); it != (cont).end(); it++)
34. #define RFORC(cont, it) for(decltype((cont).rbegin()) it = (cont).rbegin(); it != (cont).rend();
     it++)
35. #define pb push_back
36.
37. using namespace std;
38.
39. typedef long long 11;
40. typedef pair<int, int> ii;
41. typedef vector<int> vi;
42.
43. #define MAXN 10
44. #define MOD 1000000007
45.
46. int main() { _
47.
48.
           return 0;
49. }
```

2. DP

NK

```
1.
      11 fact[MAXN], inv[MAXN], pascal[MAXN][MAXN];
2.
3.
      11 ppow(11 n, 11 p) {
           ll ret = 1;
4.
5.
           while (p) {
               if (p & 1) ret = (ret*n) % MOD;
6.
               n = (n*n) \% MOD;
7.
8.
               p>>=1;
9.
           return ret;
10.
11.
      }
12.
      void generaFact() {
13.
14.
           fact[0] = 1;
15.
           FOR(i, 1, MAXN)
               fact[i] = (fact[i-1] * i) % MOD;
16.
17.
18.
           inv[MAXN-1] = ppow(fact[MAXN-1], MOD - 2);
           for(int i = MAXN-1; i>0; i--)
19.
20.
               inv[i-1] = inv[i] * i % MOD;
21.
      }
22.
      11 NK(11 n, 11 k) {
    return fact[n] * inv[k] % MOD * inv[n-k] % MOD;
23.
24.
25.
26.
27.
      // pascal i,j corresponde a la formula de combinatoria i!/(j!*(i-j)!), AKA. (N, K)
      void trianguloPascal() {
28.
29.
          pascal[0][0] = 1;
           FOR(i, 1, MAXN) {
30.
31.
               pascal[i][0] = 1;
32.
               FOR(j, 1, i+1) {
                   pascal[i][j] = (pascal[i - 1][j - 1] + pascal[i - 1][j]) % MOD;
33.
34.
35.
          }
36.
      }
37.
38.
      int main(){
39.
           11 n, k;
40.
           generaFact();
           trianguloPascal();
41.
           while (scanf("%11d %11d", &n, &k) != EOF) {
42.
               printf("%lld\n", NK(n, k));
printf("%lld\n", pascal[n][k]);
43.
44.
45.
46.
47.
           return 0;
48.
      }
```

Longest Common Subsequence

```
1.
      #define MAXN 1000005
      #define MOD 1000000007
2.
3.
      /* Returns Length of LCS for X[0..n-1], Y[0..m-1] */
4.
5.
      int lcs(string s1, string s2) {
6.
           int n = s1.length();
           int m = s2.length();
7.
          int L[n + 1][m + 1];
memset(L, 0, sizeof(L));
8.
9.
10.
11.
          /* Following steps build L[n+1][m+1] in bottom up fashion. Note
           that L[i][j] contains Length of LCS of X[0..i-1] and Y[0..j-1] */
12.
          FOR(i, 1, n+1) {
13.
               FOR(j, 1, m+1) {
    if (s1[i-1] == s2[j-1])
        L[i][j] = L[i-1][j-1] + 1;
14.
15.
16.
17.
18.
                        L[i][j] = max(L[i-1][j], L[i][j-1]);
19.
               }
20.
           }
21.
           /* L[n][m] contains length of LCS for X[0..n-1] and Y[0..m-1] */
22.
23.
          return L[n][m];
24.
      }
25.
      int main() {
26.
           string s1, s2;
27.
          while (cin >> s1 >> s2)
28.
29.
           cout << lcs(s1, s2) << endl;</pre>
30.
      }
```

Longest Common Substring

```
#define MAXN 10
1.
      #define MOD 1000000007
2.
3.
      /* Returns Length of Longest common substring of X[0..n-1] and Y[0..m-1] */
4.
      // Create a table to store lengths of longest common suffixes of
5.
6.
      // substrings. Notethat LCSuff[i][j] contains length of longest
      // common suffix of X[0..i-1] and Y[0..j-1]. The first row and
7.
      // first column entries have no logical meaning, they are used only
8.
9.
      // for simplicity of program
10.
     int LCSubStr(string X, string Y) {
          int n = X.length();
int m = Y.length();
11.
12.
13.
14.
          int lcs[n+1][m+1];
15.
          memset(lcs, 0, sizeof(lcs));
          int result = 0; // To store length of the longest common substring
16.
17.
18.
          /* Following steps build lcs[n+1][m+1] in bottom up fashion. */
          FOR(i, 1, n+1) {
19.
              FOR(j, 1, m+1) {
    if (X[i-1] == Y[j-1]) {
20.
21.
                       lcs[i][j] = lcs[i-1][j-1] + 1;
22.
                       result = max(result, lcs[i][j]);
23.
24.
25.
                  else {
                       lcs[i][j] = 0;
26.
27.
28.
              }
29.
          }
30.
31.
          return result;
      }
32.
33.
      int main() {
34.
35.
          string X, Y;
36.
          cin >> X >> Y;
37.
          cout << LCSubStr(X, Y) << endl;</pre>
38.
39.
          return 0;
40.
      }
```

TSP

```
1. #define MAXN 25
2.
int N;
4. int dptsp[MAXN][1 << MAXN], costViajero[MAXN][MAXN];</pre>
5. int maskInicial;
7. // Calcula costos de todos los nodos a todos los nodos.
8. void floyd() {
9.
        FOR(k, 0, N) {
10.
            FOR(i, 0, N) {
11.
                FOR(j, 0, N) {
                     costViajero[i][j] = min(costViajero[i][j],
12.
   costViajero[i][k] + costViajero[k][j]);
13.
14.
            }
15.
        }
16.}
17.
18. // Bits prendidos significa ciudad que aun no se visita.
19. int tsp(int n, int mask) {
20.
        if (!mask) return 0;
21.
        int &c = dptsp[n][mask];
22.
        if (~c) return c;
        c = INF;
23.
24.
        FOR(i, 0, N) {
25.
            if (mask & 1 << i) {</pre>
26.
                c = min(c, tsp(i, mask ^ 1 << i) + costViajero[n][i]);</pre>
27.
28.
29.
        return c;
30.}
31.
32. int main() {
33.
        int ans;
34.
35.
        floyd();
        maskInicial = (1 << N) - 1, ans = INF;</pre>
36.
37.
        FOR(i, 0, N) {
38.
            ans = min(ans, tsp(i, maskInicial ^ 1 << i));</pre>
39.
        }
40.
41.
        return 0;
42.}
```

3. Geometry

```
typedef long long 11;
1.
2.
3.
     double degToRad(double theta){
         return theta * PI / 180.0;
4.
5.
6.
     double radToDeg(double theta){
7.
8.
         return theta * 180 / PI;
9.
10.
11.
      12.
     struct point {
13.
         double x, y;
14.
15.
         point() {}
         point(double xx, double yy) {
16.
17.
             x = xx;
18.
             y = yy;
19.
20.
         point inf() {
21.
             x = INF:
             y = INF;
22.
23.
24.
25.
         double dist(point p) {
26.
             return hypot(x - p.x, y - p.y);
27.
28.
         point rotate(point pivot, point p, double ang) {
29.
             double s = sin(ang);
30.
31.
             double c = cos(ang);
32.
             // translate point back to origin:
33.
34.
             p.x -= pivot.x;
             p.y -= pivot.y;
35.
36.
37.
             // rotate point
             double xaux = p.x * c - p.y * s;
double yaux = p.x * s + p.y * c;
38.
39.
40.
41.
             // translate point back:
             p.x = xaux + pivot.x;
42.
43.
             p.y = yaux + pivot.y;
             return p;
44.
45.
         }
46.
47.
         double getAngle(point pivot, point p) {
             return atan2(p.y - pivot.y, p.x - pivot.x);
48.
49.
50.
51.
         void swap() {
             double aux = x;
52.
53.
             x = y;
54.
             y = aux;
55.
56.
57.
          void swap(point &a, point &b) {
             point aux = a;
59.
             a = b;
             b = aux;
60.
61.
62.
63.
         point punto(point const &p) const {
64.
             return point(x * p.x, y * p.y);
65.
         }
66.
67.
          double cruz(point const &p) const {
68.
             return x * p.y - y * p.x;
69.
70.
71.
         double cruz(point const &p1, point const &p2) const {
72.
             return (p1.x - x) * (p2.y - y) - (p1.y - y) * (p2.x - x);
```

```
73.
74
75.
         point operator +(point const &p) const {
76.
             return point(x + p.x, y + p.y);
77.
78.
79.
         point operator -(point const &p) const {
80.
             return point(x + p.x, y + p.y);
81.
82.
         point operator /(double d) const {
83.
84.
             return point(x / d, y / d);
85.
86.
87.
         bool operator <(point const &p) const {</pre>
88.
             return (x - p.x) > EPS && x < p.x
                     fabs(x - p.x) < EPS && (y - p.y) > EPS && y < p.y;
89.
90.
91.
         bool operator ==(point p) const {
92
93.
             return fabs(x - p.x) < EPS && fabs(y - p.y) < EPS;
94.
95.
96.
         void print() {
           cout << x << "\t" << y << endl;
97.
98.
99.
     };
        ************************
100. //
101
103. struct line {
         point p1, p2;
104
105.
         double a, b, c;
106.
107.
         line() {
108.
             p1.inf();
             p2.inf();
109.
110.
111.
         line(double A, double B, double C) {
112.
             a = A:
             b = B;
113.
114.
             c = C;
115.
116.
             p1.inf();
117.
             p2.inf();
118.
119.
         line(point pp1, point pp2) {
120.
             p1 = pp1;
121.
             p2 = pp2;
122.
123.
             if (p2 < p1)
124.
                 swap(p1, p2);
125.
             if(fabs(p1.x - p2.x) < EPS) {</pre>
126.
                a = 1.0;
127.
128.
                 b = 0.0;
129.
                 c = -p1.x;
130.
131.
             else {
                 a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
132.
                b = 1.0;
133.
134.
                 c = -(double)(a * p1.x) - (b * p1.y);
135.
             }
136.
         }
137.
         line perpendicular(point p) {
138.
             double c = -(a * p.x - b * p.y);
139.
140.
             return line(-b, a, c);
141.
         }
142.
143.
         bool sonParalelas(line 11, line 12){
            return fabs(l1.a - l2.a) < EPS &&
fabs(l1.b - l2.b) < EPS;
144.
145.
146.
         }
```

```
147.
148.
         bool sonIguales(line 11, line 12){
149.
            return sonParalelas(11, 12) &&
150.
                   fabs(11.c - 12.c) < EPS;
151.
152.
         bool insideLine(point p) {
153.
            return ((p1.x - EPS <= p.x && p.x <= p2.x + EPS ||
154
                p2.x - EPS \le p.x & p.x \le p1.x + EPS
155.
156.
157.
                (p1.y - EPS <= p.y && p.y <= p2.y + EPS ||
158.
                p2.y - EPS <= p.y && p.y <= p1.y + EPS));
159.
        }
160
161.
         bool intercectan(line 11, line 12, point &p){
162.
            if (sonParalelas(11, 12))
                return false;
163
164.
            p.x = (12.b * 11.c - 11.b * 12.c) /
165.
166
                    (12.a * 11.b - 11.a * 12.b);
167.
168.
            if(fabs(l1.b) > EPS)
               p.y = -(11.a * p.x + 11.c) / 11.b;
169.
170.
171.
                p.y = -(12.a * p.x + 12.c) / 12.b;
172.
173.
            return 12.insideLine(p);
174.
        }
175. };
    176.
177.
179. struct triangle {
180.
         point p1, p2, p3;
181
         double a, b, c;
182.
         double A, B, C;
183.
         double area, perimetro;
         int type; // equilatero=1, isoseles=2, equilatero=3
184.
185.
         bool rect; // triangulo rectangulo o no
186.
187.
         triangle() {}
188.
         triangle(double aa, double bb, double cc) {
189.
            a = aa;
            b = bb;
190.
191.
            c = cc;
192.
            sort();
193.
194.
            act();
195.
         }
196.
197.
         triangle(point pp1, point pp2, point pp3) {
198.
            p1 = pp1;
199.
            p2 = pp2;
200.
            p3 = pp3;
201
202.
            a = p1.dist(p2);
203.
            b = p1.dist(p3);
            c = p2.dist(p3);
204.
205.
            sort();
206.
207
            act();
208.
209.
         double innerCircleRadio(){
210.
211.
            return area / (perimetro * 2.0);
212.
213.
214.
         double outterCircleRadio(){
215.
            return a * b * c / (4 * area);
216.
217.
218.
         int getType() {
            if(a==b && b==c)
219.
                                  return 1;
220.
            else if(a==b || b==c) return 2;
```

```
221.
             return 3;
222
         }
223.
224.
         bool isRight(){
             return (a*a + b*b - c*c < EPS);
225.
226.
227.
         void sort() {
228
229
             if (a > b)
230.
                 swap(a, b);
231.
             if (a > c)
232.
                 swap(a, c);
             if (b > c)
233.
234
                 swap(b, c);
235.
         }
236.
237.
         double getPerimetro() {
238.
             return a + b + c;
239.
240.
241.
         double getArea() {
242.
             double s = perimetro / 2.0;
             return sqrt(s * (s-a) *(s-b) * (s-c));
243.
244.
245.
         double getAngles() {
246.
247.
             double aa = a * a;
             double bb = b * b;
248.
             double cc = c * c;
249
250.
251.
             A = a\cos(bb + cc - aa) / (2 * b * c);
             B = acos(aa + cc - bb) / (2 * a * c);
252.
253.
             C = acos(aa + bb - cc) / (2 * a * b);
254.
         }
255.
256.
         void act() {
            type = getType();
257.
             rect = isRight();
258.
259.
             perimetro = getPerimetro();
             area = getArea();
260.
261.
             getAngles();
262.
263. };
265.
266.
267. // ************* Struct: circle ************************
268. struct circle {
         point c;
269.
270.
         double r, circ, area;
271.
272.
         circle() {
273.
             c.x = c.y = 0;
274.
             r=1;
275
276.
             act();
277.
         circle(point p, double rr){
278.
279.
             c = p;
             r = rr;
280.
281.
282.
             act();
283.
         }
284.
285.
         int insideCircle(point p){// 0-Dentro, 1-Borde, 2-Fuera
286.
             int dx = p.x - c.x, dy = p.y - c.y;
             int Euc = dx*dx + dy*dy, rSq=r*r;
287.
288.
             double dist = c.dist(p);
             if (fabs(dist - r) < EPS)</pre>
289.
                                        return 1:
             if (dist < r)</pre>
290.
                                        return 0;
291.
             return 2;
292.
         }
293.
294.
         double getArc(double deg) {
```

```
295.
             return circ * deg / 360.0;
296.
297.
         double getChord(double deg) {
             return 2.0 * r * r * (1 - cos(degToRad(deg)));
298.
299.
300.
         double getSector(double deg) {
            return area * deg / 360.0;
301.
302
         double getSegment(double deg) {
303.
304.
             triangle t(r, r, getChord(deg));
305.
             return getSector(deg) - t.getArea();
306.
307.
308
         // 0 - No intercectan
309.
         // 1 - Intersecta 1 punto
         // 2 - Intersecta 2 puntos
310.
         // 3 - Mismo circulo
311
312.
         // 4 - Circulo dentro del otro
313.
         int intersectCirc(circle const &cir, point &p1, point &p2) {
314
             double d = c.dist(cir.c);
315.
316.
             if (d > r + cir.r)
                                                    // No se tocan
317.
                 return 0;
318.
             if ((r - cir.r) < EPS && c == cir.c)</pre>
                                                  // Mismo circulo
319.
                 return 3:
320.
             if (d + min(r, cir.r) < max(r, cir.r)) // Circulo contiene otro</pre>
321.
                 return 4;
322.
             double a = (r * r - cir.r * cir.r + d * d) / (2.0 * d);
double h = sqrt(r * r - a * a);
323.
324.
325.
326.
             // find p2
327.
             point pp1(c.x + (a * (cir.c.x - c.x)) / d, c.y + (a * (cir.c.y - c.y)) / d);
328.
329
             // find intersection points p3
330.
             p1 = point(pp1.x + (h * (cir.c.y - c.y)/d),
                    pp1.y - (h * (cir.c.x - c.x)/d));
331.
332.
333.
             p2 = point(pp1.x - (h * (cir.c.y - c.y)/d),
                     pp1.y + (h * (cir.c.x - c.x)/d));
334.
335.
336.
             if(fabs(d - r - cir.r) < EPS)</pre>
337.
                return 1;
338.
             return 2;
339.
         }
340.
341.
         double getArea() {
342.
             return PI * r * r;
343.
344.
345.
         double getCirc() {
             return PI * 2. * r;
346.
347.
348.
         bool operator ==(circle const &cir) const{
349
350.
            return c == cir.c && fabs(r - cir.r) < EPS;</pre>
351.
352.
353.
         void act() {
             area = getArea();
354.
355.
             circ = getCirc();
356.
357. };
358.
360.
361. typedef vector<point> vp;
362. typedef vector<line> vl;
363. typedef vector<triangle> vt;
364. typedef vector<circle> vc;
367. point p0;
368.
```

```
369. int orientation(point p, point q, point r) {
         int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
370.
371.
372.
         if (val == 0) return 0; // colinear
return (val > 0)? 1: 2; // clock or counterclock wise
373.
374.
375.
     }
376
     bool compAng(const point p1, const point p2) {
377.
378.
          int o = orientation(p0, p1, p2);
379.
380.
          return (o == 0 &&
381.
                  hypot(p0.x - p2.x, p0.y - p2.y) >= hypot(p0.x - p1.x, p0.y - p1.y)) |
382
                  (o == 2);
383.
384.
     bool compY(const point p1, const point p2) {
    return p1.y < p2.y || (p1.y == p2.y && p1.x < p2.x);</pre>
385.
386.
387.
388.
389.
     vp convexHull(vp v) {
390.
          vp c = v;
          int n = v.size(), mini = 0;
391.
392.
         p0 = c[0];
393.
394.
          FOR(i, 1, n) {
395.
              int y = c[i].y;
396.
397
              if (compY(c[i], p0)) {
398.
                  p0 = c[i];
399.
                  mini = i;
400
              }
401.
402.
          swap(c[0], c[mini]);
403.
404.
          sort(c.begin() + 1, c.end(), compAng);
405.
406.
407.
          vp ans;
408.
          ans.pb(c[0]);
409.
          ans.pb(c[1]);
410.
          ans.pb(c[2]);
411.
412.
          FOR (i, 3, n) {
413.
              while (orientation(ans[ans.size()-2], ans[ans.size()-1], c[i]) != 2) {
414.
                  ans.erase(ans.end());
415.
416.
              ans.pb(c[i]);
417.
418.
          ans.pb(ans[0]);
419.
420.
          return ans;
421. }
     422.
423
424.
     425.
426. double cross(const point &O, const point &A, const point &B) {
427.
          return (A.x - 0.x) * (B.y - 0.y) - (A.y - 0.y) * (B.x - 0.x);
428. }
429
430.
     vp convexHull(vp &P) {
          int n = P.size(), k = 0;
431.
          vp H(2*n);
432.
433.
          sort(P.begin(), P.end());
434.
          FOR(i, 0, n) {
              while (k \ge 2 \&\& cross(H[k-2], H[k-1], P[i]) <= 0) k--;
435.
436.
             H[k++] = P[i];
437.
          for (int i = n-2, t = k+1; i >= 0; i--) {
438.
439.
              while (k \ge t \&\& cross(H[k-2], H[k-1], P[i]) \le 0) k--;
440.
             H[k++] = P[i];
441.
442.
         H.resize(k);
```

```
443.
     return H;
444. }
    445.
446.
447.
448.
449. // ************* Revisa si es CW ***********************
450. bool cw(vp v) {
      double ret = 0;
451
452.
      FOR(i, 1, v.size()) {
         ret += v[i].x * v[i-1].y - v[i-1].x * v[i].y;
453.
454.
455.
      return ret < 0;
456. }
    457.
458.
459
460.
    // ****** Revisa si un punto esta dentro en un poligono ************
461. bool PointInPolygon(vp v, point p) {
462
      bool ret = 0;
463.
      int j = 0;
464.
      FOR(i, 1, v.size()) {
465.
         if(((v[i].y > p.y) != (v[j].y > p.y)) &&
466.
            (p.x < (v[j].x - v[i].x) * (p.y - v[i].y) / (v[j].y - v[i].y) + v[i].x)
467.
468.
            ret = !ret:
469.
         j++;
470.
471.
      return ret;
472.
474
475.
477. double perimetro(vp v) {
478.
      double ret = 0;
479.
      FOR(i, 0, v.size() - 1) {
480.
         ret += v[i].dist(v[i + 1]);
481.
482.
      return ret:
483. }
485.
486.
488. double area(vp v) {
489.
      double ret = 0;
490.
      FOR(i, 1, v.size()) {
         ret += v[i].x * v[i-1].y - v[i-1].x * v[i].y;
491.
492.
493.
      return fabs(ret) / 2.0;
494. }
496.
497
498.
499.
500. int main () { _
      point p(1, 2), p2(2,3);
501.
      cout << p.x << " " << p.y << endl;
502.
503.
      double ang = p.getAngle(p, p2);
cout << p.getAngle(p, p2) * 180 / PI << endl;</pre>
504.
505.
506.
      p.rotate(p, p2, -ang).print();
507. }
```

4. Graph's Theory

Centroid decomposition

```
1. //CENTROID DECOMPOSITION BY ADANCITO
3. //NOTES:
4. /*
    -The consider() method contains an algorithm specifically thought of to count pairs
   in trees which fulfill a certain characteristic (for further details go to method).
   However, since this method will be called for all centroids, it can be changed to
   anything the problem needs to do.
   -If consider()'s efficiency is O(F(n)), then the overall complexity is O(F(n) * lgn).
7.
8. const int maxN = 5e5 + 6;
9. vector<int> adj[maxN];
10. int n, sz[maxN], arr[maxN];
12. bool dead[maxN];
14. void precalc(int nd, int an){
15.
        sz[nd] = 1;
        for(int sn: adj[nd]){
16.
17.
            if(sn != an && !dead[sn]){
                precalc(sn, nd);
18.
19.
                sz[nd] += sz[sn];
20.
            }
21.
        }
22. }
23.
24. int getCentroid(int nd, int an, int tot){
        for(int sn: adj[nd]){
25.
26.
            if(sn != an && !dead[sn] && sz[sn] * 2 > tot){
27.
                return getCentroid(sn, nd, tot);
28.
29.
        }
30.
        return nd;
31. }
32.
33. 11 ans = 0;
35. int arr1[maxN], arr2[maxN], curs, ots;
37. void dfs(int nd, int an, int lev){
38.
        // example:
39.
        while(curs <= lev) arr2[curs++] = 0;</pre>
40.
        arr2[lev]++;
        //bucket in arr2[lev] whatever nd represents. Afterwards, just do the same with
41.
   children
42.
43.
        for(int sn: adj[nd]){
44.
            if(sn != an && !dead[sn]){
45.
                dfs(sn, nd, lev);
46.
47.
        }
48.
49. }
50.
51. void mergeVec(){
        //Accumulating arrays to keep the consider() method linear
52.
53.
        while(ots < curs) arr1[ots++] = 0;</pre>
54.
        for(int i = 0; i < curs; i++){</pre>
55.
            arr1[i] += arr2[i];
56.
        }
57.}
```

```
58.
59. /*
60. The most popular CD problem type is pair-counting in trees. This sample is a basic
   technique that uses bucketing and accumulating. Today, we will count pairs of trees
   whose path length is odd.
61. */
62. void consider(int nd){
63.
        //ots is the size for arr1, curs is the size for arr2
64.
65.
        for(int sn: adj[nd]){
66.
67.
            if(!dead[sn]){
68
                curs = 0;
69.
                dfs(sn, nd, 0);
70.
71.
                //example:
72.
73.
                for(int j = 0; j < curs; j++){</pre>
74.
                    ans += ll(arr1[j]) * arr2[j];
75.
76.
                //iterate through arr2, and multiply by frequency in arr1 at the bucket
   place it corresponds
77.
78.
                mergeVec();
79.
80.
            }
81.
        }
82. }
83.
84. void solve(int nd){
        precalc(nd, -1);
86.
        nd = getCentroid(nd, -1, sz[nd]);
87.
88.
        consider(nd);
89.
        dead[nd] = 1;
90.
91.
        for(int sn: adj[nd]){
92.
            if(!dead[sn]){
93.
                solve(sn);
94.
95.
        }
96.}
97.
98. int main() {
99.
        ios::sync_with_stdio(false);
100.
                cin.tie(0), cout.tie(0);
101.
102.
                 cin >> n;
103.
                 for(int i = 0; i < n; i++){
104.
105.
                     cin >> arr[i];
106.
107.
108.
109.
                for(int i = 1; i < n; i++){
110.
                     int u, v;
111.
                     cin >> u >> v, u--, v--;
                    adj[u].push_back(v);
112.
113.
                     adj[v].push_back(u);
114.
                }
115.
                solve(0);
116.
117.
118.
            }
```

HLD

```
1. const int maxN = 1e4 + 3, MOD = 1e9 + 9, LG = 21;
2. //NOTE: maxN>2^LG must be held, remember maxN is twice ur biggest polynom
4. int he[maxN], par[maxN], ind[maxN], cs[maxN], heavy[maxN], head[maxN], n,
    T[maxN << 1];
5. vector<int> adj[maxN];
6. ii ed[maxN];
7.
8. int dfs(int nd){
9.
        int ret = 1, heaS = 0, cr;
10.
        for(int sn: adj[nd]) if(sn != par[nd]){
            par[sn] = nd, he[sn] = he[nd] + 1, cr = dfs(sn), ret += cr;
11.
            if(heaS < cr) heaS = cr, heavy[nd] = sn;</pre>
12.
13.
14.
        return ret;
15. }
16.
17. void setPaths(){
        memset(heavy, -1, sizeof(heavy)), par[0] = -1, he[0] = 0, dfs(0); for(int i = 0, ct = 0; i < n; i++)
18.
19.
            if(par[i] == -1 || heavy[par[i]] != i)
20.
21.
                for(int j = i; j != -1; j = heavy[j])
                     head[j] = i, ind[j] = ct++;
22.
23. }
24.
25. int query(int 1, int r){
26.
        int ret = 0;
27.
        for(1 += n, r += n; 1 < r; 1 >>= 1, r >>= 1){
            if(1 & 1) ret = max(ret, T[1++]);
28.
29.
            if(r & 1) ret = max(ret, T[--r]);
30.
31.
        return ret;
32. }
33.
34. void update(int w, int x){
35.
        w += n, T[w] = x;
36.
        while(w > 1)
            w >>= 1, T[w] = max(T[w << 1], T[w << 1 | 1]);
37.
38. }
39.
40. int ans(int u, int v){
41
        int ret = 0;
42.
        for(; head[u] != head[v]; v = par[head[v]]){
43.
            if(he[head[u]] > he[head[v]]) swap(u, v);
44.
            ret = max(ret, query(ind[head[v]], ind[v] + 1));
45.
46.
        if(he[u] > he[v]) swap(u, v);
47.
        return max(ret, query(ind[u] + 1, ind[v] + 1));
48. }
49.
50. int main(){
        ios::sync with stdio(false);
51.
52.
        cin.tie(0), cout.tie(0);
53.
54.
        int t;
55.
        cin >> t;
56.
        while(t--){
57.
            cin >> n;
58.
59.
            for(int i = 0; i < n; i++) adj[i].clear();</pre>
60.
            for(int i = 1, u, v; i < n; i++){
61.
                cin >> u >> v >> cs[i];
62.
```

```
63.
                 u--, v--;
                 adj[u].push_back(v), adj[v].push_back(u);
64.
65.
                 ed[i] = ii(u, v);
66.
             }
67.
             setPaths();
68.
69.
70.
             for(int i = 1; i < n; i++){</pre>
                 if(he[ed[i].fi] < he[ed[i].se]) swap(ed[i].fi, ed[i].se);</pre>
71.
72.
                 update(ind[ed[i].fi], cs[i]);
             }
73.
74.
75.
             while(true){
76.
                 string inp;
                 int a, b;
cin >> inp;
if(inp == "DONE") break;
77.
78.
79.
                 cin >> a >> b;
if(inp == "CHANGE"){
80.
81.
                      update(ind[ed[a].fi], b);
82.
83.
                 else{
84.
                      cout << ans(--a, --b) << '\n';</pre>
85.
86.
87.
             }
88.
        }
89.
90.
91.}
```

Isomorphic Rooted Tree Test

```
1. const int maxN = 1e5 + 3, maxV = 1e6;
2.
int n, rta, rtb, mla, mlb;
4.
5.
   struct El{
6.
        int uu:
7.
        vector<int> vc;
        El(int uu): uu(uu), vc(vector<int>()){}
8.
        bool operator!=(const El &ot) const{
9
10.
            return vc != ot.vc:
11.
        bool operator<(const El &ot) const{</pre>
12.
13.
            return vc < ot.vc;</pre>
14.
15. };
16. bool notequal(const vector<El> &a, const vector<El> &b){
17.
        if(a.size() != b.size()) return true;
18.
        for(int i = 0; i < a.size(); i++){</pre>
19.
            if(a[i] != b[i]) return true;
20.
21.
        return false;
22. }
23. vector<int> one[maxN], two[maxN];
24. vector<El> vcs[2][maxN];
25. int le[maxN];
26.
27. void getinp(vector<int> *adj, int &rt){
28.
        for(int i = 0, pa; i < n; i++){
29
            cin >> pa;
            if(!pa) rt = i;
30.
31.
            else adj[--pa].push back(i);
32.
33. }
34.
35. void dfs(vector<int> *adj, vector<El> *vcc, int nd, int ct, int &bes){
36.
        bes = max(bes, ct);
        for(int sn: adj[nd]){
37.
38.
            vcc[ct + 1].push_back(El(int(vcc[ct].size() - 1)));
39.
            dfs(adj, vcc, sn, ct + 1, bes);
40.
        }
41. }
42.
43. bool isom(){
44.
        for(int r = max(mla, mlb); r >= 0; r--){
            for(int u = 0; u < 2; u++) sort(vcs[u][r].begin(), vcs[u][r].end());
45.
            //cout << "good " << r << '\n';
46
47.
            if(notequal(vcs[0][r], vcs[1][r])) return false;
            for(int u = 0; u < 2 \&\& r; u++){
48.
                for(int i = 0, j = 0; i < vcs[u][r].size(); i = j){
49
50
                    for(;j<vcs[u][r].size() && vcs[u][r][i].vc == vcs[u][r][j].vc; j++){</pre>
51.
                         int uu = vcs[u][r][j].uu;
52.
                         vcs[u][r - 1][uu].vc.push_back(i);
53.
54.
                }
55.
            }
56.
57.
        return true;
58. }
59.
60. int main(){
61.
        ios::sync with stdio(false);
62.
        cin.tie(0), cout.tie(0);
63.
```

```
int t;
64.
65.
        cin >> t;
66.
67.
        while(t--){
68.
            cin >> n;
69.
            mlb = mla = 0;
70.
            for(int i = 0; i < n; i++)
71.
72.
                 one[i].clear(), two[i].clear(), vcs[0][i].clear(), vcs[1][i].clear();
73.
            getinp(one, rta), getinp(two, rtb);
74.
75.
76.
            vcs[0][0].push_back(El(-1));
            vcs[1][0].push_back(El(-1));
dfs(one, vcs[0], rta, 0, mla), dfs(two, vcs[1], rtb, 0, mlb);
77.
78.
79.
80.
            cout << isom() << '\n';</pre>
81.
82.
        }
83.
84.}
```

Dijkstra

```
typedef long long 11;
1.
      typedef pair<11, 11> ii;
typedef pair<ii, 11> iii;
2.
3.
      #define FIRST first.first
4.
      #define SECOND first.second
5.
      #define THIRD second
6.
      typedef vector<11> vi;
7.
      typedef vector<ii> vii;
8.
9.
10.
      #define MAXN 100000
     #define MOD 1000000007
11.
12.
13.
      vii edges[MAXN];// first = cost, second = where
      ii arr[MAXN];// shortest paths to all nodes will be stored here
ll n; // size of graph
14.
15.
16.
      void dijkstra(ll from, ll to) {
17.
18.
          for(int i = 0; i < n; i++) arr[i].first = INT_MAX;</pre>
          priority_queue<iii, vector<iii>>, greater<iii>> pq;
19.
          pq.push(iii(ii(0, -1), from));
20.
21.
          while(!pq.empty()){
              iii cur = pq.top();
22.
              pq.pop();
23.
              if(arr[cur.THIRD].first == INT_MAX)
24.
25.
                   arr[cur.THIRD] = cur.first;
26.
              else
27.
                   continue;
              for(int i = 0; i < edges[cur.THIRD].size(); i++){</pre>
28.
29.
                   11 cost = edges[cur.THIRD][i].first,
30.
                   where = edges[cur.THIRD][i].second;
                   if(arr[where].first == INT_MAX)
31.
                       pq.push(iii(ii(cur.FIRST + cost, cur.THIRD), where));
32.
33.
              }
34.
          }
35.
     }
```

```
DSU
1.
      #define MAXN 10
      #define MOD 1000000007
2.
3.
      struct DSU {
4.
          int bel[MAXN];
5.
6.
          vi s[MAXN];
7.
          void reset() {
8.
9.
              FOR(i, 0, MAXN)
                  bel[i] = 0, s[i].clear();
10.
11.
          }
12.
13.
          void unir(int a, int b) {
14.
              if (a > b) swap(a, b);
15.
              FOR(i, 0, s[a].size()) {
16.
17.
                   s[b].pb(s[a][i]);
18.
                  bel[a] = b;
19.
              s[a].clear();
20.
21.
22.
23.
          int belongs(int x) {
24.
              return bel[x];
25.
26.
      };
27.
      struct DSU2 {
28.
29.
          int numSets = 0;
30.
          int setSize[MAXN];
          int parent[MAXN];
31.
32.
          int rank[MAXN];
33.
34.
          UnionFind(int n) {
35.
              numSets = n;
              FOR(i, 0, n)
36.
                               parent[i] = i, setSize[i] = rank[i] = 0;
37.
          }
38.
39.
          void make_set(int i) {
40.
              parent[i] = i;
41.
              rank[i] = 0;
42.
43.
44.
          int find_set(int i) {
45.
              if (i != parent[i])
                  parent[i] = find_set(parent[i]);
46.
47.
              return parent[i];
48.
49.
50.
          void unionSet(int i, int j) {
              if (!isSameSet(i, j)) {
51.
52.
                  numSets--;
53.
                  int x = find_set(i), y = find_set(j);
54.
55.
                  if (rank[x] > rank[y]) {
56.
                       parent[y] = x;
                       setSize[x] += setSize[y];
57.
58.
59.
                  else {
                       parent[x] = y;
setSize[y] += setSize[x];
60.
61.
62.
63.
                       if (rank[x] == rank[y])
                                                    rank[y]++;
                  }
64.
65.
              }
66.
          }
67.
68.
          bool isSameSet(int i, int j) {
69.
              return find_set(i) == find_set(j);
```

70.

71. };

Find Cycles

```
#define maxN 100005
1.
2.
3.
      vi edges[maxN];
      bool cur[maxN], visit[maxN];
4.
5.
      stack<int> ans;
6.
      int findCycle(int n) {
7.
                            return n;
8.
          if (cur[n])
9.
          if (visit[n])
                           return -1;
10.
11.
           cur[n] = true;
          visit[n] = true;
12.
13.
          FOR(i, 0, edges[n].size()) {
   if(ans.size()) break;
14.
15.
16.
17.
               int v = findCycle(edges[n][i]);
               if (v != -1) {
    cur[n] = false;
18.
19.
                   ans.push(n);
20.
21.
                   if(v == n) {
                       return -1;
22.
23.
24.
                   return v;
25.
               }
26.
27.
          cur[n] = false;
          return -1;
28.
29.
      }
30.
      int main() {
31.
32.
          int n, m, a, b;
33.
          while (cin >> n >> m) {
               memset(cur, false, sizeof(cur));
34.
35.
               memset(visit, false, sizeof(visit));
36.
37.
               FOR(i, 0, m) {
                   cin >> a >> b;
38.
39.
                   edges[a].pb(b);
               }
40.
41.
42.
               FOR(i, 0, n) {
                   findCycle(i);
43.
44.
45.
               if(! ans.empty()) {
46.
                   cout << "YES\n";</pre>
47.
48.
                   while(! ans.empty()) {
                       int val = ans.top();
49.
50.
                        ans.pop();
51.
                       cout << val << " \n"[ans.empty()];</pre>
                   }
52.
53.
54.
               else {
                   cout << "NO\n";
55.
56.
57.
58.
          return 0;
59.
      }
```

Floyd-Warshall

```
#define MAXN 10005
1.
2.
     #define MOD 1000000007
3.
     int graph[MAXN][MAXN];
4.
5.
     int dist[MAXN][MAXN];
6.
     void floydWarshall(int n) { // O(n^3)
7.
         FOR(i, 0, n) FOR(j, 0, n) dist[i][j] = graph[i][j];
8.
9.
10.
         FOR(k, 0, n) {
11.
             FOR(i, 0, n) {
12.
                  FOR(j, 0, n) {
13.
                     dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
14.
                  }
15.
             }
16.
         }
17.
     }
18.
19.
     int main() {
         FOR(i, 0, MAXN) FOR(j, 0, MAXN) dist[i][j] = INF;
20.
21.
         int n, m;
22.
23.
         cin >> n >> m;
         FOR(i, 0, m) {
24.
25.
             cin >> a >> b >> c;
26.
             graph[a][b] = c;
27.
         }
28.
29.
         floydWarshall(n);
30.
         return 0;
31.
     }
```

```
Kruskal
      #define MAXN 10005
1.
      #define MAXM 10005
2.
3.
      #define MOD 1000000007
4.
      struct Edge {
5.
6.
          int from, to, w;
7.
          Edge() {}
          Edge(int a, int b, int c) : from(a), to(b), w(c) { }
8.
9.
          bool operator <(Edge const& e) const {</pre>
10.
              return w < e.w;
11.
12.
      };
13.
14.
      struct UnionFind {
15.
          int numSets = 0;
          int setSize[MAXN];
16.
          int parent[MAXN];
17.
18.
          int rank[MAXN];
19.
          UnionFind(int n) {
20.
21.
              numSets = n;
              FOR(i, 0, n)
22.
                               parent[i] = i;
23.
          }
24.
25.
          void make_set(int i) {
26.
              parent[i] = i;
27.
              rank[i] = 0;
28.
29.
30.
          int find set(int i) {
              if (i != parent[i])
31.
                  parent[i] = find_set(parent[i]);
32.
33.
              return parent[i];
34.
35.
36.
          void unionSet(int i, int j) {
              if (!isSameSet(i, j)) {
37.
38.
                  numSets--;
39.
                  int x = find_set(i), y = find_set(j);
40.
41.
                  if (rank[x] > rank[y]) {
                       parent[y] = x;
setSize[x] += setSize[y];
42.
43.
44.
45.
                  else {
46.
                       parent[x] = y;
                       setSize[y] += setSize[x];
47.
48.
49
                       if (rank[x] == rank[y])
                                                    rank[y]++;
50.
                  }
51.
              }
          }
52.
53.
          bool isSameSet(int i, int j) {
54.
55.
              return find_set(i) == find_set(j);
56.
57.
      };
58.
59.
      vector<Edge> v;
60.
      vector<Edge> tree;
61.
62.
      int kruskal(int n, int m) { // O(ELogE + ELogV)
          sort(v.begin(), v.end());
63.
64.
65.
          int mst_w = 0;
          UnionFind UF(n);
66.
67.
68.
          FOR(i, 0, m) {
69.
              Edge e = v[i];
70.
              if (! UF.isSameSet(e.from, e.to)) {
71.
72.
                  mst_w += e.w;
```

```
UF.unionSet(e.from, e.to);
tree.pb(Edge(e.from, e.to, e.w));
 73.
 74.
 75.
                                                                                                                                     }
 76.
                                                                                              }
 77.
 78.
                                                                                              return mst_w;
 79.
                                                      }
 80.
                                                         int main() {
81.
                                                                                          maln() {
  int n, m;
  int a, b, c;
  cin >> n >> m;
  FOR(i, 0, m) {
    cin >> a >> b >> c;
    cin >> a >> b >> c;

 82.
 83.
84.
85.
 86.
 87.
                                                                                                                                     v.pb(Edge(a, b, c));
 88.
89.
90.
                                                                                               cout << kruskal(n, m) << endl;</pre>
91.
92.
                                                                                              return 0;
 93.
                                            }
```

Longest Path in DAG

```
#define MAXN 10
1.
      #define MOD 1000000007
2.
3.
4.
      vii graph[MAXN];
5.
      stack<int> s;
6.
      bool v[MAXN];
      int dist[MAXN], n, m;
7.
8.
9.
      void topologicalSortUtil(int act) {
10.
          v[act] = 1;
11.
12.
          FOR(i, 0, graph[act].size())
              if (!v[graph[act][i].second]) topologicalSortUtil(graph[act][i].second);
13.
14
15.
          s.push(act);
      }
16.
17.
18.
      void longestPath(int source) {
19.
          memset(v, 0, sizeof(v));
20.
21.
          FOR(i, 0, n)
              if (!v[i]) topologicalSortUtil(i);
22.
23.
          fill(dist, dist + n, -INF);
24.
25.
          dist[source] = 0;
26.
          while (! s.empty()) {
27.
              int u = s.top();
28.
29.
              s.pop();
30.
              if (dist[u] != -INF) {
31.
                   FOR(i, 0, graph[u].size()) {
32.
33.
                       dist[graph[u][i].second] = max(dist[graph[u][i].second],
      dist[u] + graph[u][i].first);
34
                   }
35.
              }
36.
          }
37.
          FOR(i, 0, n)
    if (dist[i] == -INF)
38.
                                        cout << "INF" << endl;</pre>
39.
40.
                                        cout << dist[i] << endl;</pre>
41.
      }
42.
43.
      int main() { _
44.
45.
          int a, b, c;
46.
          while(cin >> n >> m) {
              FOR(i, 0, m) {
47.
                   cin >> a >> b >> c;
48.
49.
                   graph[a].pb(ii(c, b));
50.
              }
51.
52.
              int source;
53.
              cin >> source;
54.
              longestPath(source);
55.
56.
57.
          return 0;
58.
      }
```

Maximum Diameter in a Graph

```
    #define MAXN 10
    #define MOD 1000000007

3.

    // v es visitados localmente.
    // v2 es todos los que ya he visitado (para grafos no totalmente conectados)
    bool v[MAXN], v2[MAXN];

vi edges[MAXN];
8.
9. ii bfs(int x, bool b) {
           queue<ii> q;
10.
           q.push(ii(x, 0));
11.
12.
          v[x] = !b;
v2[x] |= !b;
13.
14.
15.
           ii last;
16.
           while(! q.empty()) {
17.
                last = q.front();
18.
                q.pop();
19.
                FOR(i, 0, edges[last.first].size()) {
20.
                     if (v[edges[last.first][i]] == b) {
    q.push(ii(edges[last.first][i], last.second + 1));
21.
22.
                           v[edges[last.first][i]] = !b;
v2[edges[last.first][i]] |= !b;
23.
24.
25.
                     }
26.
                }
27.
           }
28.
29.
           return last;
30. }
31.
32. int maxDiameter(int x) {
          ii ret = bfs(x, false);
return bfs(ret.first, true).second;
33.
34.
35. }
```

Maximum Bipartite Matching

```
#define MAXN 10005
1.
      #define MOD 1000000007
2.
3.
4.
      int matchL[MAXN], matchR[MAXN];
5.
6.
      vi edge[MAXN];
      bool v[MAXN];
7.
8.
9.
      bool dfs(int from) {
          if (v[from])
10.
                          return 0;
11.
          v[from] = 1;
12.
13.
          FOR(i, 0, edge[from].size()) {
              int to = edge[from][i];
14.
15.
              if (matchR[to] == -1 || dfs(matchR[to])) {
16.
                   matchL[from] = to;
17.
18.
                   matchR[to] = from;
19.
                   return 1;
20.
              }
21.
22.
          return 0;
23.
      }
24.
25.
      11 MBM() {
          11 ans = 0;
26.
27.
          bool b = 1;
28.
          memset(matchL, -1, sizeof(matchL));
memset(matchR, -1, sizeof(matchR));
29.
30.
31.
          while (b) {
32.
33.
              b = 0;
              memset(v, 0, sizeof(v));
34.
35.
              FOR(i, 0, n) {
36.
                   if (matchL[i] == -1 && dfs(i)) {
37.
                       ans ++;
38.
                       b = 1;
39.
                   }
40.
              }
41.
42.
          return ans;
      }
43.
44.
      int main() { _
45.
          int a, b;
46.
47.
          cin >> n >> m;
          FOR(i, 0, m) {
48.
49.
              cin >> a >> b;
50.
               edge[a].pb(b);
51.
          cout << MBM() << endl;</pre>
52.
53.
54.
          return 0;
55.
      }
```

Strongly Connected Components

```
1.
      #define MAXN 1000
      #define MOD 1000000007
2.
3.
4.
      int n, m;
5.
6.
      vi edge[MAXN];
      int disc[MAXN], low[MAXN], belongs[MAXN];
7.
      bool v[MAXN];
8.
9.
      stack<int> st;
     int ttime, comp;
10.
11.
      void SCCUtil(int u) {
12.
          disc[u] = low[u] = ttime ++;
13.
          st.push(u);
14.
15.
          v[u] = true;
16.
17.
          FOR(i, 0, edge[u].size()) {
18.
              int to = edge[u][i];
19.
              if (disc[to] == -1)
20.
21.
                   SCCUtil(to);
22.
23.
              if (v[to])
24.
                   low[u] = min(low[u], low[to]);
25.
          }
26.
          if (low[u] == disc[u]) {
27.
28.
               comp ++;
29.
              while (1) {
30.
                   int t = st.top();
                   st.pop();
31.
                   belongs[t] = comp;
32.
33.
                   v[t] = false;
34.
35.
                   if (t == u) break;
36.
              }
37.
          }
38.
      }
39.
      void SSC() {
40.
41.
          ttime = comp = 0;
          memset(disc, -1, sizeof(disc));
memset(low, -1, sizeof(low));
42.
43.
          memset(v, 0, sizeof(v));
44.
45.
          FOR(i, 0, n)
46.
              if (disc[i] == -1)
47.
                   SCCUtil(i);
48.
49.
      }
```

Topological Sort

```
1.
      #define MAXN 10005
      #define MOD 1000000007
2.
3.
      vi edges[MAXN];
4.
      stack<int> s;
5.
6.
      bool v[MAXN];
7.
      void topologicalSortUtil(int act) {
8.
9.
          v[act] = 1;
10.
          FOR(i, 0, edges[act].size())
11.
12.
              if (!v[edges[act][i]]) topologicalSortUtil(edges[act][i]);
13.
14.
          s.push(act);
15.
      }
16.
17.
      void topologicalSort(int n) {
18.
          memset(v, 0, sizeof(v));
19.
          FOR(i, 0, n)
20.
21.
              if (!v[i]) topologicalSortUtil(i);
22.
23.
          while (!s.empty()) {
              int a = s.top();
24.
25.
              s.pop();
              cout << a << " \n"[s.empty()];</pre>
26.
27.
28.
      }
29.
      int main() {
30.
          int n, m;
31.
32.
          int a, b;
          while (cin >> n >> m) {
33.
              FOR(i, 0, MAXN)
34.
                                  edges[i].clear();
35.
              FOR(i, 0, m) {
36.
                  cin >> a >> b;
37.
38.
                  edges[a].pb(b);
39.
40.
              topologicalSort(n);
41.
42.
          }
43.
44.
          return 0;
45.
      }
```

5. Number Theory

Fermat Little's theorem

```
1.
     ll mulmod(ll\ a,\ ll\ b,\ ll\ c) { // returns (a * b) % c, and minimize overflow
          return (11)(( int128)(a) * (b) % c);
2.
3.
4.
     11 fastPow(ll x, ll n, ll c) { // returns (a ** b) % c, and minimize overflow
5.
          ll ret = 1;
6.
          while (n) {
7.
8.
              if (n & 1) ret = mulmod(ret, x, c);
9.
              x = mulmod(x, x, c);
10.
              n >>= 1;
11.
12.
          return ret;
      }
13.
14.
15.
16.
17.
      /** return modular multiplicative of: a mod p, assuming p is prime **/
     11 modInverse(ll a, ll p) {
18.
          return fastPow(a, p - 2, p);
19.
20.
21.
      /** return C(n,k) mod p, assuming p is prime **/
22.
23.
     11 modBinomial(ll n, ll k) {
          ll numerator = 1; // n*(n-1)* ... * (n-k+1)
24.
25.
          FOR(i, 0, k)
26.
              numerator = (numerator * (n - i)) % MOD;
27.
28.
          11 denominator = 1; // k!
29.
          FOR (i, 1, k+1)
              denominator = (denominator * i) % MOD;
30.
31.
32.
          11 res = modInverse(denominator, MOD);
33.
          res = (res * numerator) % MOD;
          return res;
34.
35.
     }
36.
      11 extendedGCD(11 a, 11 b, 11 &inva, 11 &invb) {
37.
38.
          if (b) {
39.
              11 g = extendedGCD(b, a%b, invb, inva);
              invb = invb - a / b*inva;
40.
              return g;
41.
42.
43.
          inva = 1, invb = 0;
44.
          return a;
45.
     }
46.
47.
      int main() {
          ll inva=0, invb=0;
48.
49.
          int c = extendedGCD(62424, 13, inva, invb);
          cout << modInverse(62424, 13) << endl;
cout << (inva + 7) % 7 << " " << (invb + 62424) % 13 << " " << c << endl;</pre>
50.
51.
52.
          return 0;
53.
     }
```

Fibonacci

```
const long double PHI ((1.0 + sqrt(5.0)) / 2.0);
1.
2.
3.
      ll 01(ll n) {
          return (11)(floor(pow(PHI, n) / sqrt(5.0) + 0.5));
4.
5.
6.
      7.
8.
          11 y = (F[0][0] * M[0][1]) % MOD + (F[0][1] * M[1][1]) % MOD;
9.
          11 z = (F[1][0] * M[0][0]) % MOD + (F[1][1] * M[1][0]) % MOD;
10.
          11 \text{ W} = (F[1][0] * M[0][1]) \% \text{ MOD} + (F[1][1] * M[1][1]) \% \text{ MOD};
11.
12.
          F[0][0] = x \% MOD;
          F[0][1] = y \% MOD;
13.
          F[1][0] = z \% MOD;
14.
15.
          F[1][1] = w \% MOD;
     }
16.
17.
18.
      void power(11 F[2][2], 11 n) {
          if (n == 0 || n == 1) return;
19.
20.
          11 M[2][2] = {{1, 1}, {1, 0}};
power(F, n / 2);
21.
22.
23.
          mult(F, F);
24.
25.
          if (n % 2)
26.
              mult(F, M);
27.
     }
28.
     11 OlnMat(11 n) {
29.
30.
          if (n == 0) return 0;
31.
          11 F[2][2] = {{1,1},{1,0}};
32.
33.
          power(F, n - 1);
34.
35.
          return F[0][0];
36.
     }
37.
      /** Suma de los primeros n numeros de fibo **/
38.
39.
     11 fiboSuma(11 n) {
          return (MOD + OlnMat(n + 2) - 1) % MOD;
40.
41.
42.
     ll On(ll n) {
43.
44.
          if (n == 1) return 0;
45.
          11 a = 0, b = 1, c = 1;
46.
47.
48.
          FOR(i, 2, n) {
49.
              c = b + a;
50.
              a = b;
              b = c;
51.
52.
53.
          return c;
54.
     }
55.
      11 fibo(ll n) {
          if (n < 72)
if (n < 150)
                          return 01(n);
57.
58.
                          return On(n);
59.
60.
          return OlnMat(n);
61.
      }
62.
63.
     int main(){
64.
          11 n;
65.
          cin >> n;
          cout << fibo(n) << endl;</pre>
66.
```

Prime Numbers

```
    int pc = 0, f[MAXN], primes[MAXN / 10];

2.
3.
   // *****************
                                                         ************
4.
                                          IS PRIME()
5. //Para TODOS los primos
6.
7. 11 mulmod2(11 a, 11 b, 11 c) { // returns (a * b) % c, and minimize overflow
        11 x = 0, y = a \% c;
9.
        while (b) {
10.
            if (b & 1) x = (x + y) \% c;
            y = (y << 1) \% c;
11.
            b >>= 1;
12.
13.
        return x % c;
14.
15. }
16.
17. ll mulmod(ll a, ll b, ll c) {
18.
        return (11)((__int128)(a) * b % c);
19. }
21. ll fastPow(ll x, ll n, ll c) { // returns (a ** b) % c, and minimize overflow
22.
        11 \text{ ret} = 1;
23.
        while (n) {
24
            if (n & 1) ret = mulmod(ret, x, c);
25.
            x = mulmod(x, x, c);
            n >>= 1;
26.
27.
28.
        return ret;
29. }
30.
31. // Miller-Rabin primality test
32. bool millerRabin(ll n) {
        11 d = n - 1;
33.
34.
        int s = 0;
        while (d % 2 == 0) {
35.
36.
            S++;
37.
            d >>= 1;
38.
        ^{\prime}// It's garanteed that these values will work for any number smaller than 2^64
39.
        int a[12] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 };
FOR(i, 0, 12) if(n == a[i]) return true;
40.
41.
42.
43.
        FOR(i, 0, 12) {
44.
            bool comp = fastPow(a[i], d, n) != 1;
            if(comp) FOR(j, 0, s) {
45.
46.
                11 fp = fastPow(a[i], (1LL << (11)j)*d, n);</pre>
                if (fp == n - 1) {
47.
                     comp = false;
48.
49.
                     break;
50.
                }
51.
52.
            if(comp) return false;
53.
54.
        return true;
55. }
56.
57. // Miller-Rabin primality test
58. ll rN(){
        return maxR * rand() + rand();
59.
60.}
61.
62. bool miller(ll n, ll d){
63.
        11 a = 2 + rN() \% (n - 4);
        11 x = fastPow(a, d, n);
64.
        if(x == 1 || x == n - 1)
65.
            return true;
66.
        while(d < n - 1){
67.
            x = _{int128(x)} * x % n;
68.
69.
            if(x == 1)
70.
                return false;
```

```
71.
           if(x == n - 1)
               return true;
72.
73.
           d <<= 1;
74.
       return false;
75.
76. }
77.
78. bool isPrime(ll n, int k){
79.
       if(n == 3 || n == 2)
80.
           return true;
       if(n == 1 || n % 2 == 0)
81.
82.
           return false;
       ll d = n - 1;
83.
       while(d % 2 == 0)
84
85.
           d >>= 1;
86.
       while(k--){
           if(!miller(n, d))
87.
88.
              return false;
89.
90.
       return true;
91. }
      **********************************
92. //
93.
94.
95. // ****************
                                                                 **********
                                      GET PRIME FACTORS()
97. Devuele los factores primos de un numero. No olvidar hacer unique para quitar repetidos
98. int tamano = unique(primeFactors.begin(), primeFactors.end()) - primeFactors.begin();
100.
101.11 A, B;
102.vl primeFactors;
103.
104.11 funcRand(ll x, ll n) {
       return (mulmod(x, (x + A), n) + B) \% n;
106.}
107.
108.void pollardRho(ll n) {
109.
      if (n == 1) return;
110.
       if (millerRabin(n)) {
111.
           primeFactors.pb(n);
           return;
112.
113.
       11 d = n, x, y;
114.
       while(d == n) {
115
116.
           d = 1;
117.
           x = y = 2;
           A = 2 + rand() \% (n - 2);
118.
119.
           B = 2 + rand() \% (n - 2);
120.
121
           while (d == 1) {
122.
               x = funcRand(x, n);
               y = funcRand(funcRand(y, n), n);
123.
               d = \underline{gcd(abs(x - y), n)};
124.
125.
           }
126.
       }
127.
128.
       if (n / d != d)
           pollardRho(n / d);
129.
       pollardRho(d);
130.
131.}
133.
134.// *****************
                                                                  **********
                                   GET SMALLEST DIV PRIME()
135.// Obtener f[t], el primo mas chico que divide a f[t]
137.int divPrime[MAXN];
138.
139.void getSmallestDivPrime() {
140.
       FOR(i, 2, MAXN) {
141.
           if (divPrime[i] == -1) {
142.
               for (int j = 1; i*j < MAXN; j++) if (divPrime[i*j] == -1) divPrime[i*j] = i;</pre>
143.
           }
```

```
144.
145.3
148.
149.// *****************
                                                      ***********
                                    GET DIVISORS()
150.vll factorsOfN:
151.vl divisors;
152.
153.// Usar cuando se necesita sacar muchos numeros
154.void getFactorsOfN2(int n) {
155.
       int t = n;
       int last = -1;
156
157.
       while (divPrime[t] != -1) {
158.
          if (divPrime[t] == last)
                                   factorsOfN[factorsOfN.size() - 1].second ++;
159.
                                   factorsOfN.pb(ii(divPrime[t], 1));
          else.
160.
          last = divPrime[t];
161.
          t /= divPrime[t];
162
       }
163.}
164.
165.// Usar cuando es numeros muy grandes
166.void getFactorsOfN(ll n) {
       pollardRho(n);
167.
       sort(primeFactors.begin(), primeFactors.end());
168
169.
       int ss = unique(primeFactors.begin(), primeFactors.end()) - primeFactors.begin();
170.
       int s = 0:
       11 \text{ num} = n;
171.
172.
       FOR(i, 0, ss) {
173.
          bool b = false:
          while (num % primeFactors[i] == 0) {
174
175.
              if (b) factorsOfN[s].second ++;
176.
              else
                     factorsOfN.pb(pll(primeFactors[i], 1)), b = 1;
177.
178.
              num /= primeFactors[i];
179.
          }
180
          s ++;
181.
       }
182.}
183.
184.void getDivisorsOfN(ll n) {
       primeFactors.clear();
185.
186.
       factorsOfN.clear();
187.
       divisors.clear();
188
189.
       getFactorsOfN(n);
190.
       divisors.push_back(1);
191.
192.
       FOR(i, 0, factorsOfN.size()) {
193.
          int s = divisors.size(), num = 1;
194
195
          FOR(j, 0, factorsOfN[i].second) {
196.
              num *= factorsOfN[i].first;
              FOR(k, 0, s) divisors.push_back(divisors[k] * num);
197.
198.
199.
200.
       sort(divisors.begin(), divisors.end());
201.
       FOR(i, 0, divisors.size()) cout << divisors[i] << endl;</pre>
202.}
      *******************************
203.//
204.
205.
**********
207.void Eratosthenes() {
208
       bool isPrime[MAXN];
       memset(isPrime, 0, sizeof(isPrime));
209.
       FOR(i, 2, MAXN) {
210.
211.
          if (!isPrime[i]) {
212.
              primes[pc ++] = i;
              for (int j = 2; j*i < MAXN; j++) {
213.
214.
                 isPrime[i*j] = true;
215.
              }
216.
          }
```

```
217.
218. }
221.
223.void EratosthenesON(){
      FOR(i, 2, MAXN) {
225.
         if (! f[i]) primes[pc++] = i, f[i] = i;
         for (int j = 0; j < pc && 1LL*i*primes[j] < MAXN && primes[j] <= f[i]; j++)</pre>
226.
227.
             f[i*primes[j]] = primes[j];
228.
229.}
     *******************************
230.//
231.
233.void Atkin() {
234.
      11 sqrtArraySize = sqrt(MAXN);
235.
      11 n;
236.
237
      bool isPrime[MAXN];
238.
      memset(isPrime, false, sizeof(isPrime));
239.
240.
      11 pp[MAXN];
                              pp[i] = i * i;
241.
      FOR(i, 0, sqrtArraySize + 5)
242.
243
      FOR(i, 1, sqrtArraySize + 1) {
244.
         FOR(j, 1, sqrtArraySize + 1) {
            n = 4 * pp[i] + pp[j];
245.
246.
247.
             if (n <= MAXN && (n % 12 == 1 || n % 12 == 5))
                isPrime[n] = !isPrime[n];
248.
249
250.
             n = 3 * pp[i] + pp[j];
251.
252.
             if (n <= MAXN && (n % 12 == 7))
                isPrime[n] = !isPrime[n];
253.
254.
255.
             if (i > j) {
                n = 3 * pp[i] - pp[j];
if(n <= MAXN && n % 12 == 11)
256.
257
258.
                   isPrime[n] = !isPrime[n];
259.
             }
         }
260.
261.
262.
      FOR(i, 5, sqrtArraySize + 1) {
263.
264.
         if (isPrime[i]) {
            ll omit = pp[i];
265.
266
267.
             for (ll j = omit; j <= MAXN; j += omit) {</pre>
268.
                isPrime[j] = false;
269.
270.
         }
271.
      }
272
273.
      if (MAXN >= 2) {
274.
         primes[pc ++] = 2;
275.
         if (MAXN >= 3) {
276.
             primes[pc ++] = 3;
277.
         }
278.
      }
279.
280.
      FOR(i, 2, MAXN) {
281.
         if (isPrime[i]) {
282.
             primes[pc ++] = i;
283.
284.
      }
285.}
286.//
287.
```

```
289.// REGRESA PINCHE CANTIDAD DE COPRIMOS DE N
290.// Para sacar la puta suma de coprimos es n*phi(n) / 2
291.ll eulerTotient(ll n) {
292.  factorsOfN.clear();
293.  getFactorsOfN(n);
294.  FOR(i, 0, factorsOfN.size())
295.  n = n / factorsOfN[i].first * (factorsOfN[i].first - 1);
296.  return n;
297.}
```

Triangle of Mahonian

```
    #define MAXN 35
    #define MAXM 5005

3.
11 arr[MAXN][MAXM];
5.

    int main() {
    memset(arr, 0, sizeof(arr));
    arr[0][0] = arr[1][0] = 1;

9.
            FOR(i, 0, MAXN) {
   for(int m = 0; m < MAXM - 5 && arr[i][m] != 0; m ++) {
10.
11.
                        while(m < 5000 && ) {
    FOR(j, 0, i+1) {
        arr[i+1][j+m] += arr[i][m];
}</pre>
12.
13.
14.
                               }
15.
16.
                        }
17.
                  }
18.
19.
20.
            return 0;
```

6. Segment Tree

Lazy Propagation

```
1.
      const int maxN = 1e5, MOD = 1E9 + 7;
       int HEIGHT, n;
2.
3.
       11 d[maxN], t[maxN << 1];</pre>
4.
       void apply(int p, ll value) {
5.
           t[p] = value;
if (p < n) d[p] = value;
6.
7.
8.
9.
      void build(int p) {
10.
11.
           while (p > 1) p >>= 1, t[p] = max(max(t[p<<1], t[p<<1|1]), d[p]);
12.
13.
14.
       void push(int p) {
           for (int s = HEIGHT; s > 0; --s) {
   int i = p >> s;
15.
16.
                if (d[i] != 0) {
17.
                     apply(i<<1, d[i]);
apply(i<<1|1, d[i]);
18.
19.
20.
                     d[i] = 0;
21.
                }
           }
22.
23.
      }
24.
25.
       void change(int 1, int r, 11 value) {
           1 += n, r += n;
           int 10 = 1, r0 = r;
for (; 1 < r; 1 >>= 1, r >>= 1) {
27.
28.
                if (1&1) apply(1++, value);
if (r&1) apply(--r, value);
29.
30.
31.
           build(10);
32.
           build(r0 - 1);
33.
34.
      }
35.
36.
      11 query(int 1, int r) {
37.
           1 += n, r += n;
38.
           push(1);
           push(r - 1);
ll res = LLONG_MIN;
39.
40.
41.
           for (; 1 < r; \bar{1} >>= 1, r >>= 1) {
                if (1&1) res = max(res, t[1++]);
42.
43.
                if (r\&1) res = max(t[--r], res);
44.
45.
           return res;
      }
46.
```

Persistent Segment Tree

```
const int maxN = 1e6;
2.
3.
      struct Node{
          Node *1, *r;
4.
          int count;
5.
          Node(int count, Node *1, Node *r):count(count), 1(1), r(r){}
6.
          Node * insert(int, int, int);
7.
8.
9.
      Node * segTrees[maxN + 1];
10.
11.
     Node * Node::insert(int left, int right, int pos){
12.
          if(pos < left || pos >= right)
13.
              return this;
          if(left + 1 == right)
14
15.
             return new Node(count + 1, segTrees[0]);
          int m = (left + right) >> 1;
16.
17.
          return new Node(count + 1, 1 -> insert(left, m, pos), r -> insert(m, right, pos));
18.
19.
      int solve(Node * last, Node * first, int left, int right, int k){
          if(left + 1 == right)
20.
21.
             return left;
          int m = (left + right) >> 1, dif = last -> 1 -> count - first -> 1 -> count;
22.
23.
          if(dif < k)</pre>
24.
              return solve(last -> r, first -> r, m, right, k - dif);
25.
          return solve(last -> 1, first -> 1, left, m, k);
26.
     }
27.
28.
      int main() {
          ios::sync_with_stdio(false);
29.
30.
          cin.tie(0);
31.
          cout.tie(0);
32.
          int n, q, x, y, k, arr[maxN], vals[maxN];
33.
          segTrees[0] = new Node(0, NULL, NULL);
          segTrees[0] -> 1 = segTrees[0] -> r = segTrees[0];
34.
35.
          while(cin >> n >> q){
              map<int, int> myMap;
for(int i = 0; i < n; i++){</pre>
36.
37.
38.
                  cin >> arr[i];
39.
                  myMap[arr[i]];
40.
41.
              int valSize = 0;
42.
              for(map<int, int>::iterator it = myMap.begin(); it != myMap.end(); it++){
                  it -> second = valSize;
43.
44.
                  vals[valSize++] = it -> first;
45.
              for(int i = 1; i <= n; i++)</pre>
46.
47.
                  segTrees[i] = segTrees[i - 1] -> insert(0, valSize, myMap[arr[i - 1]]);
48.
              while(q--){
49
                  cin >> x >> y >> k;
50.
                  cout << vals[solve(segTrees[y], segTrees[x - 1], 0, valSize, k)] << '\n';</pre>
51.
              }
52.
          }
53.
    }
```

```
7. Treap
8. const int maxN = 1e5 + 3, MOD = 1e9 + 7, AL = 10;
9.
10. 11 fc[maxN][2];
11.
12. ll perm(int *bu){
        ll den = 1, tot = 0, imp = 0;
13.
        for(int i = 0; i < AL; i++){
14.
15.
             //cout << bu[i] << ' ';
16.
             imp += bu[i] & 1;
17.
             den = den * fc[bu[i] >> 1][1] % MOD;
             tot += bu[i] >> 1;
18.
19.
        }
        //cout << '\n';
if(imp > 1) return 0;
20.
21.
        return fc[tot][0] * den % MOD;
22.
23. }
24.
25. struct Node{
        int ca, pri, bu[AL], cnt;
26.
        Node *1, *r;
27.
28.
        bool rev;
        Node(int ca): ca(ca), l(NULL), r(NULL), rev(false){
29.
30.
             memset(bu, 0, sizeof(bu)), pri = (rand() << 15) + rand();</pre>
31.
32. };
33. typedef Node* pnode;
34.
35. int cnt(pnode nd){
        return nd? nd->cnt: 0;
36.
37. }
38.
39. void upd(pnode nd){
        if(!nd) return;
41.
        nd\rightarrow cnt = 1 + cnt(nd\rightarrow 1) + cnt(nd\rightarrow r);
42.
43.
        for(int i = 0; i < AL; i++)</pre>
44.
             nd\rightarrow bu[i] = (nd\rightarrow l? nd\rightarrow l->bu[i]: 0) + (nd\rightarrow r? nd\rightarrow r->bu[i]: 0);
45.
        nd->bu[nd->ca]++;
46.}
47.
48. void apply(pnode nd){
49.
        if(nd) nd->rev ^= 1;
50.}
51.
52. void push(pnode nd){
53.
        if(nd && nd->rev)
             swap(nd->1, nd->r), apply(nd-> 1), apply(nd -> r), nd->rev = 0;
54.
55.}
57. void split(pnode t, pnode &l, pnode &r, int key, int add){
58.
        if(!t)
59.
             return void(1 = r = NULL);
        push(t);
60.
61.
        int mykey = add + cnt(t->l) + 1;
62.
        if(mykey <= key)</pre>
63.
             split(t->r, t->r, r, key, mykey), l = t;
64.
65.
             split(t->1, 1, t->1, key, add), r = t;
66.
        upd(t);
67.}
68.
69. void merge(pnode &t, pnode 1, pnode r){
70.
        push(1), push(r);
        if(!l || !r)
71.
```

```
t = 1? 1: r;
72.
73.
        else if(l->pri > r->pri)
74.
            merge(1->r, 1->r, r), t = 1;
75.
        else
76.
            merge(r->1, 1, r->1), t = r;
77.
        upd(t);
78.}
79.
80. void insert(pnode &t, pnode it){
        if(!t)
82.
            t = it;
83.
        else if(it->pri > t->pri)
84.
            it\rightarrow 1 = t, t = it;
85.
86.
            insert(t->r, it);
87.
        upd(t);
88.}
89.
90. int n, m;
91. string st;
92.
93. ll fastPow(ll a, ll b){
94.
        11 \text{ ret} = 1;
95.
        while(b){
96.
            if(b & 1)
97.
                ret = ret * a % MOD;
98.
            a = a * a % MOD, b >>= 1;
99.
        }
100.
               return ret;
101.
           }
102.
103.
           int main(){
104.
               ios::sync_with_stdio(false);
105.
               cin.tie(0), cout.tie(0);
106.
107.
               srand(8000000);
108.
109.
               fc[0][0] = fc[0][1] = 1;
110.
               for(ll i = 1; i < maxN; i++)</pre>
                    fc[i][0] = fc[i - 1][0] * i % MOD, fc[i][1] = fastPow(fc[i][0],
   MOD - 2);
112.
113.
               cin >> n >> m >> st;
114.
115.
               pnode head = NULL;
116.
               for(char c: st) insert(head, new Node(c - 'a'));
117.
               while(m--){
118.
119.
                    int ty, 1, r;
                    cin \gg ty \gg 1 \gg r;
120.
121.
                    pnode two, thr;
122.
                    split(head, head, two, l - 1, 0);
123.
                    split(two, two, thr, r - 1 + 1, 0);
124.
                    if(ty == 1)
125.
                        apply(two);
126.
                    else
127.
                        cout << perm(two->bu) << '\n';</pre>
                    merge(head, head, two);
128.
129.
                    merge(head, head, thr);
130.
               }
131.
132.
           }
```

8. Strings

Aho-Corasick

```
1. const int maxN = 1e3 + 3, MOD = 1e9 + 9, LG = 21;
2. //NOTE: maxN>2^LG must be held, remember maxN is twice ur biggest polynom
3. const int dr[8][2] = \{\{1, 0\}, \{0, 1\}, \{-1, 0\}, \{0, -1\}, \{1, 1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1\}, \{-1, -1
          1, 1}, {1, -1}};
4. const string chr = "ECAGDHBF";
5.
string mat[maxN], dic[maxN], ans[maxN];
7. int n, m, w;
8. bool v[maxN][maxN];
9
10. vector<ii> ini[8];
12. bool ins(int x, int y){
                     return x >= 0 && x < n && y >= 0 && y < m;
13.
14. }
16. void findInits(int d, int r, int c){
17.
                     int x = r - dr[d][0], y = c - dr[d][1];
18.
19.
                     v[r][c] = 1;
20.
21.
                     if(!ins(x, y)) ini[d].push back(ii(r, c));
22.
                     else if(!v[x][y]) findInits(d, x, y);
23. }
25. int tr[maxN * maxN][26], dp[maxN * maxN], lf[maxN * maxN], tc, fWord[maxN * maxN],
          lastInd[maxN], sz[maxN];
27. int ins(int par, char c){
                     if(tr[par][c - 'A']) return tr[par][c - 'A'];
28.
29.
                     int k, ret = tr[par][c - 'A'] = tc++;
30.
31.
                     for(k = dp[par]; k && !tr[k][c - 'A']; k = dp[k]);
32.
33.
                     if(k != par) k = tr[k][c - 'A'];
34.
35.
                     if(fWord[k] != -1)
36.
                                 lf[ret] = k;
37.
                     else
                                 lf[ret] = lf[k];
38.
39.
40.
                     dp[ret] = k;
41.
42.
                     return ret;
43. }
44.
45. string bld(int sz, int r, int c, int u){
                     r -= dr[u][0] * sz, c -= dr[u][1] * sz;
return to_string(r) + " " + to_string(c) + " " + chr[u];
46.
47.
48. }
49.
50. int main(){
51.
                      ios::sync with stdio(false);
52.
                     cin.tie(0), cout.tie(0);
53.
                     int t;
54.
55.
                     cin >> t;
56.
                     while(t--){
57.
                                cin >> n >> m >> w;
58.
59.
                                queue<int> q;
```

```
60.
61.
            for(int i = 0; i < n; i++) cin >> mat[i];
62.
            for(int i = 0; i < w; i++) {
63.
                cin >> dic[i];
64.
                sz[i] = int(dic[i].size());
65.
                reverse(dic[i].begin(), dic[i].end());
66.
                q.push(i);
67.
68.
69.
            for(int u = 0; u < 8; u++){
                memset(v, 0, sizeof(v)), ini[u].clear();
70.
                for(int i = 0; i < n; i++)</pre>
71
                     for(int j = 0; j < m; j++)
72
73.
                         if(!v[i][j])
74.
                             findInits(u, i, j);
            }
75.
76.
77.
            memset(tr, 0, tc * sizeof(tr[0])), memset(fWord, -1, sizeof(fWord));
78.
            memset(lastInd, 0, sizeof(lastInd));
79.
            tc = 1;
80.
81.
            while(q.size()){
82.
                int c = q.front(); q.pop();
83.
84.
                lastInd[c] = ins(lastInd[c], dic[c].back());
85.
                //cout << c << ' ' << dic[c].back() << '\n';
86.
87.
88.
                dic[c].pop_back();
89.
90.
                if(!dic[c].empty())
91.
                    q.push(c);
92.
                else
93.
                     fWord[lastInd[c]] = c;
94.
            }
95.
            //for(int i = 0; i < 26; i++) cout << tr[0][i] << ' '; cout << '\n';
96.
97.
            map<int, string> myMap;
98.
99.
            for(int u = 0; u < 8; u++){
                         for(ii el: ini[u]){
100.
101.
                             int c = 0;
102.
103.
                             for(; ins(el.fi, el.se); el.fi += dr[u][0],
   el.se += dr[u][1]){
104.
                                  while(c && !tr[c][mat[el.fi][el.se] - 'A'])
105.
                                      c = dp[c];
106.
                                  c = tr[c][mat[el.fi][el.se] - 'A'];
107.
108.
                                  //cout << c << '\n';
109.
110.
                                  for(int i = c; i; i = lf[i])
111.
                                      if(fWord[i] != -1)
112
                                          myMap[i] = bld(sz[fWord[i]]-1, el.fi, el.se, u);
113.
                             }
114.
                         }
                     }
115.
116.
117.
                     for(int i = 0; i < w; i++)</pre>
118.
                         cout << myMap[lastInd[i]] << '\n';</pre>
                     cout << '\n';</pre>
119.
120.
                }
121.
            }
```

```
KMP
      int lps[MAXN];
1.
2.
      vi v;
3.
      void computeLPSArray(string pat, int M) {
4.
5.
          int len = 0;
6.
           int i = 1;
          lps[0] = 0;
7.
8.
          while (i < M) {
              if (pat[i] == pat[len])
9.
10.
                  lps[i ++] = ++ len;
11.
              else {
    if (len != 0) len = lps[len - 1];
12.
13.
                                    lps[i ++] = 0;
                   else
14.
              }
15.
          }
16.
      }
      void KMPSearch(string pat, string txt) {
17.
18.
          int M = pat.length();
          int N = txt.length();
19.
          int i = 0;
20.
21.
          int j = 0;
22.
          computeLPSArray(pat, M);
23.
24.
          while (i < N) {
25.
26.
              if (pat[j] == txt[i]) {
                  j ++;
i ++;
27.
28.
29.
              }
30.
              if (j == M) {
31.
                   v.pb(i - j);
32.
33.
                   j = lps[j - 1];
34.
35.
              else if (i < N && pat[j] != txt[i]) {</pre>
                                   j = lps[j - 1];
i = i + 1;
36.
                   if (j != 0)
37.
                   else
38.
              }
39.
          }
      }
40.
41.
42.
      int main() {
          string txt = "ABABDABACDABABCABAB";
string pat = "AB";
43.
44.
45.
          KMPSearch(pat, txt);
46.
47.
          FOR(i, 0, v.size())
48.
              cout << v[i] << " \n"[i==v.size()-1];
49.
50.
          return 0;
51.
      }
```

Suffix Array

```
typedef pair<int, int> ii;
      typedef pair<ii, int> iii;
typedef long long ll;
2.
3.
      typedef vector<ll> vd;
4.
      typedef vector<vd> Matrix;
5.
6.
      const int maxN = 1e6;
7.
      iii sa[maxN], aux[maxN];
8.
9.
      int inv[maxN], lcp[maxN + 1], values[maxN + 1], n;
      string st;
10.
11.
12.
      #define a(i) (fir? sa[i].FIRST: sa[i].SECOND)
13.
      void radiixSort(bool fir){
14
15.
          memset(values, 0, sizeof(values));
          for(int i = 0; i < n; i++)
16.
17.
              values[a(i) + 1]++;
18.
          for(int i = 1; i <= n; i++)
              values[i] += values[i - 1];
19.
          for(int i = n - 1; i >= 0; i - -)
20.
21.
              aux[--values[a(i) + 1]] = sa[i];
22.
          for(int i = 0; i < n; i++)
23.
              sa[i] = aux[i];
24.
      }
25.
      void createSuffixArray(){
26.
27.
          for(int i = 0; i < n; i++)
              sa[i] = iii(ii(st[i], 0), i);
28.
29.
          sort(sa, sa + n);
30.
          for(int cnt = 1; cnt <= n; cnt <<= 1){</pre>
              for(int i = 0, j = 0; i < n; i = j)
31.
                   for(ii cur = sa[i].first; j < n && sa[j].first == cur; j++)</pre>
32.
33.
                       sa[j].FIRST = inv[sa[j].THIRD] = i;
              for(int i = 0; i < n; i++)
34.
35.
                   sa[i].SECOND = (sa[i].THIRD + cnt < n)? sa[inv[sa[i].THIRD + cnt]].FIRST: -1;</pre>
36.
              radiixSort(false);
              radiixSort(true);
37.
38.
39.
      }
40.
41.
      void createLCPArray(){
          for(int i = 0; i < n; i++)</pre>
42.
              inv[sa[i].THIRD] = i;
43.
          for(int i = 0, k = 0; i < n; i++, k? k--: k){
44.
              if(inv[i] + 1 == n){
45.
                  k = 1cp[n - 1] = 0;
46.
47.
                   continue;
48.
49
              int cur = sa[inv[i]].THIRD, next = sa[inv[i] + 1].THIRD;
50.
              for(; max(cur, next) + k < n && st[cur + k] == st[next + k]; k++);
51.
              lcp[inv[i] + 1] = k;
          }
52.
53.
     }
54.
55.
      int main() {
56.
          ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
57.
          cin >> n >> st;
          createSuffixArray();
58.
59.
          createLCPArray();
60.
      }
```

Z Function

```
// Devuelve el arreglo Z
vector<int> z_function(string &s){
  int L = 0, R = 0, n = s.length();
1.
2.
3.
              vector<int> z(n);
4.
               for(int i = 1; i < n; i++){
    if(i <= R)
5.
6.
                    z[i] = min(z[i-L], R - i + 1);
while(i + z[i] < n && s[i + z[i]] == s[z[i]])
7.
8.
                         z[i]++;
9.
10.
                    if(i + z[i] - 1 > R){
                         L = i;

R = i + z[i] - 1;
11.
12.
13.
                    }
14.
15.
              return z;
16.
        }
17.
18.
        int main(){ io
              string A, B;
cin >> A >> B;
19.
20.
              string z_Arg = B + '$' + A;
vector<int> z = z_function(z_Arg);
21.
22.
23.
        }
```

9. Flows

Dinic

```
1. /*
        O(V^2E)
2.
3.
        On unit networks O(Esart(V))
4.
        Unit network = edges with capacity = 1
5.
6. struct FlowEdge {
7.
        int v, u;
8.
        long long cap, flow = 0;
        FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
9.
10. };
11.
12. struct Dinic {
        const long long flow_inf = 1e18;
13.
14
        vector<FlowEdge> edges;
15.
        vector<vector<int>> adj;
16.
        int n, m = 0;
17.
        int s, t;
18.
        vector<int> level, ptr;
19.
        queue<int> q;
20.
        Dinic(int n, int s, int t) : n(n), s(s), t(t) {
21.
22.
            adj.resize(n);
23.
            level.resize(n);
24.
            ptr.resize(n);
25.
26.
27.
        void add_edge(int v, int u, long long cap) {
            edges.push_back(FlowEdge(v, u, cap));
28.
29.
            edges.push_back(FlowEdge(u, v, 0));
30.
            adj[v].push_back(m);
31.
            adj[u].push_back(m + 1);
32.
            m += 2;
33.
        }
34.
        bool bfs() {
35.
36.
            while (!q.empty()) {
                int v = q.front();
37.
38.
                q.pop();
39.
                for (int id : adj[v]) {
40.
                     if (edges[id].cap - edges[id].flow < 1)</pre>
41.
                         continue;
                     if (level[edges[id].u] != -1)
42.
43.
                         continue;
44.
                     level[edges[id].u] = level[v] + 1;
45.
                     q.push(edges[id].u);
46.
            }
47.
48.
            return level[t] != -1;
49.
50.
51.
        long long dfs(int v, long long pushed) {
52.
            if (pushed == 0)
53.
                return 0;
54.
            if (v == t)
55.
                return pushed;
            for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {</pre>
56.
                int id = adj[v][cid];
57.
58.
                int u = edges[id].u;
59.
                if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)</pre>
60.
                    continue;
                long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
61.
```

```
62.
                if (tr == 0)
63.
                   continue;
64.
                edges[id].flow += tr;
                edges[id ^ 1].flow -= tr;
65.
66.
                return tr;
67.
68.
            return 0;
69.
70.
       long long flow() {
71.
            long long f = 0;
72.
            while (true) {
73.
74.
                fill(level.begin(), level.end(), -1);
                level[s] = 0;
75.
                q.push(s);
76.
77.
                if (!bfs())
78.
                    break;
79.
                fill(ptr.begin(), ptr.end(), 0);
80.
                while (long long pushed = dfs(s, flow_inf)) {
                   f += pushed;
81.
82.
83.
            return f;
84.
85.
86. };
```

Edmons-Karp

```
1. /*
2. Edmonds-Karp algorithm O(VE^2)
3. Max flow
4. The matrix capacity stores the capacity for every pair of vertices.
5. adj is the adjacency list of the undirected graph,
6. since we have also to use the reversed of directed edges when we are looking for
   augmenting paths.
7.
8. The function maxflow will return the value of the maximal flow.
9. During the algorithm the matrix capacity will actually store the residual capacity of
   the network.
10. The value of the flow in each edge will actually no stored,
11. but it is easy to extent the implementation - by using an additional matrix - to also
   store the flow and return it.
12. */
13.
14. int n;
15. vector<vector<int>> capacity;
16. vector<vector<int>> adj;
17.
18. int bfs(int s, int t, vector<int>& parent) {
       fill(parent.begin(), parent.end(), -1);
19.
20.
       parent[s] = -2;
21.
       queue<pair<int, int>> q;
22.
       q.push({s, INF});
23.
24.
       while (!q.empty()) {
25.
            int cur = q.front().first;
            int flow = q.front().second;
26.
27.
            q.pop();
28.
29.
            for (int next : adj[cur]) {
30.
                if (parent[next] == -1 && capacity[cur][next]) {
31.
                    parent[next] = cur;
32.
                    int new flow = min(flow, capacity[cur][next]);
33.
                    if (next == t)
34.
                        return new_flow;
35.
                    q.push({next, new_flow});
36.
                }
37.
            }
38.
       }
39
40.
       return 0;
41. }
42.
43. int maxflow(int s, int t) {
       int flow = 0;
44.
       vector<int> parent(n);
45.
46.
       int new_flow;
47
       while (new_flow = bfs(s, t, parent)) {
48.
49.
            flow += new flow;
50.
            int cur = t;
            while (cur != s) {
51.
52.
                int prev = parent[cur];
                capacity[prev][cur] -= new flow;
53.
54.
                capacity[cur][prev] += new flow;
55.
                cur = prev;
56.
            }
57.
       }
58.
       return flow;
59.
60.}
```

Hungarian

```
1. const int maxN = 50 + 9, MOD = 1e9 + 7, INF = 2e7;
3. int n, m, a[maxN][maxN];
4.
5. /*
6.
7.
    HUNGARIAN ALGORITHM
8.

    9. -efficiency O(n ^ 2 * m)
    10. -input matrix a[1..n][1..m] ONE BASED!

11. -n <= m
12. -need to fix an INF value!</pre>
13. -it can handle negative values :0
14. -returns MINIMUM cost (just *= -1 to change it to maximum cost)
16. Such an incredible implementation demands credit for the author. Thanks again, Andrei
   Lopatin, we remember your genious every time we look at this beautiful piece of code!
17.
18. "This implementation was actually developed by Andrei Lopatin several years ago. It
   is distinguished by an amazing brevity: the whole algorithm is placed in 30 lines of
   code ."
19.
20. */
21.
22. int hungarian(){
23.
        vector<int> u (n+1), v (m+1), p (m+1), way (m+1);
24.
        for (int i=1; i<=n; ++i) {
25.
            p[0] = i;
            int j0 = 0;
26.
27.
            vector<int> minv (m+1, INF);
            vector<char> used (m+1, false);
28.
29.
            do {
30.
                 used[j0] = true;
                int i0 = p[j0], delta = INF, j1;
31.
                for (int j=1; j<=m; ++j)</pre>
32.
                     if (!used[j]) {
33.
                         int cur = a[i0][j]-u[i0]-v[j];
34.
                         if (cur < minv[j])</pre>
35.
                             minv[j] = cur, way[j] = j0;
36.
                         if (minv[j] < delta)</pre>
37.
38.
                             delta = minv[j], j1 = j;
39
40.
                for (int j=0; j<=m; ++j)</pre>
                     if (used[j])
41.
                         u[p[j]] += delta, v[j] -= delta;
42.
43
                     else
44.
                         minv[j] -= delta;
45.
                i0 = i1;
            } while (p[j0] != 0);
46.
47
            do {
                 int j1 = way[j0];
48.
49.
                p[j0] = p[j1];
50.
                j0 = j1;
            } while (jo);
51.
52.
        }
53.
54.
         Recovery of the answer in a more usual form, i.e. finding for each row the i = 1
55.
    \ ldots nnumber of the column selected in it ans [i]is done as follows:*/
        vector<int> ans (n+1);
56.
57.
        for (int j=1; j<=m; ++j)</pre>
58.
            ans[p[j]] = j;
59.
```

```
60.
61.
           The value of the matching found can simply be taken as the potential of the zero
62.
    column (taken with the opposite sign). In fact, how easy it is to trace the code -v [0] contains a sum of all quantities delta, i.e. total potential change. Although at each potential change several values could change at once u [i] and v [j], the total
    change in the potential value is exactly the same delta, since as long as there is no
    increasing chain, the number of reachable rows is exactly one more than the number of
    reachable columns (only the current row idoes not have a pair in the form of a visited
    column )*/
63.
64.
          return -v[0];
65.}
66.
67. int main(){
          ios::sync_with_stdio(false);
68.
          cin.tie(0);
69.
70.
          cout.tie(0);
71.
          cin >> n >> m;
72.
73.
74.
          for(int i = 1; i <= n; i++)</pre>
               for(int j = 1; j <= m; j++)
    cin >> a[i][j];
75.
76.
77.
78.
          cout << hungarian();</pre>
79.}
```

Min-cost flow

```
1. /*
2. Minimum-cost flow
3. O(n^3m)
4. */
5.
6. struct Edge
7. {
8.
        int from, to, capacity, cost;
9. };
10.
11. vector<vector<int>> adj, cost, capacity;
12.
13. const int INF = 1e9;
15. void shortest paths(int n, int v0, vector<int>& d, vector<int>& p) {
16.
        d.assign(n, INF);
17.
        d[v0] = 0;
        vector<int> m(n, 2);
18.
19.
        deque<int> q;
        q.push_back(v0);
20.
21.
        p.assign(n, -1);
22.
23.
        while (!q.empty()) {
24.
            int u = q.front();
25.
            q.pop_front();
26.
            m[u] = 0;
            for (int v : adj[u]) {
27.
28.
                if (capacity[u][v] > 0 \&\& d[v] > d[u] + cost[u][v]) {
29.
                    d[v] = d[u] + cost[u][v];
30.
                    p[v] = u;
                    if (m[v] == 2) {
31.
32.
                        m[v] = 1;
                        q.push_back(v);
33.
34.
                    } else if (m[v] == 0) {
35.
                        m[v] = 1;
36.
                        q.push_front(v);
37.
38.
                }
39.
            }
40.
        }
41. }
42.
43. int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
44.
        adj.assign(N, vector<int>());
        cost.assign(N, vector<int>(N, 0));
45.
46.
        capacity.assign(N, vector<int>(N, 0));
47.
        for (Edge e : edges) {
48.
            adj[e.from].push back(e.to);
49.
            adj[e.to].push_back(e.from);
50
            cost[e.from][e.to] = e.cost;
51.
            cost[e.to][e.from] = -e.cost;
52.
            capacity[e.from][e.to] = e.capacity;
53.
54.
55.
        int flow = 0;
        int cost = 0;
56.
57.
        vector<int> d, p;
        while (flow < K) {</pre>
58.
59.
            shortest_paths(N, s, d, p);
            if (d[t] == INF)
60.
61.
                break;
62.
            // find max flow on that path
63.
```

```
64.
            int f = K - flow;
65.
            int cur = t;
66.
            while (cur != s) {
                f = min(f, capacity[p[cur]][cur]);
67.
68.
                cur = p[cur];
            }
69.
70.
            // apply flow
71.
            flow += f;
cost += f * d[t];
72.
73.
            cur = t;
74.
75.
            while (cur != s) {
76.
                capacity[p[cur]][cur] -= f;
77.
                capacity[cur][p[cur]] += f;
78.
                cur = p[cur];
79.
            }
80.
        }
81.
        if (flow < K)</pre>
82.
83.
            return -1;
84.
        else
85.
            return cost;
86.}
```

10. Variuos

Gauss-Jordan

```
1. // Regresa el vector con los valores de cada variable.
2. // Si no es posible obtener solucion, regresa un vector vacio.
3. vector<double> gaussJordan(vector<vector<double> > v) {
        int n = v.size();
4.
5.
        double val;
6.
        FOR(i, 0, n) {
7.
            if (v[i][i] == 0) {
                bool b = 1;
8.
                FOR(j, i+1, n) {
9.
10.
                     if (v[i][j] != 0) {
11.
                         swap(v[i], v[j]);
12.
                         b = 0;
13.
                         break;
14.
                     }
15.
16.
                if (b) return vector<double>();
17.
            }
18.
19.
            val = v[i][i];
20.
            FOR(j, i, n+1) {
21.
                v[i][j] /= val;
22.
23.
24.
            FOR(k, 0, n) {
25.
                if (i == k)
                                 continue;
                val = -v[k][i];
26.
27.
                FOR(j, i, n+1) {
28.
                     v[k][j] += v[i][j] * val;
29.
30.
            }
31.
        }
32.
33.
        vector<double> vd(n);
34.
        FOR(i, 0, n)
                       vd[i] = v[i][n];
35.
        return vd;
36. }
37.
38. int main() {
39. vector<double> v(4);
        vector<vector<double> > vv;
40.
41.
        v[0] = 1;
42.
        v[1] = 1;
43.
        v[2] = 1;
        v[3] = 5;
44.
45.
        vv.pb(v);
46.
47.
        v[0] = 2;
        v[1] = 3;
48.
49.
        v[2] = 5;
50.
        v[3] = 8;
51.
        vv.pb(v);
52.
53.
        v[0] = 4;
54.
        v[1] = 0;
55.
        v[2] = 5;
56.
        v[3] = 2;
57.
        vv.pb(v);
58.
59.
        FOR(i, 0, 3) {
            FOR(j, 0, 4)
60.
61.
                cout << vv[i][j] << " ";</pre>
```

```
2-Sat
      #define MAXN 100005
1.
      #define MOD 1000000007
2.
3.
      struct TwoSAT{
4.
5.
          int n;
          vector<int> G[MAXN*2];
6.
          int S[MAXN*2], c;
7.
          bool mark[MAXN*2];
8.
9.
10.
          bool dfs(int x){
11.
              if(mark[x^1]) return false;
12.
              if(mark[x]) return true;
13.
              mark[x] = true;
14
              S[c++] = x;
15.
16.
              FOR(i, 0, G[x].size())
17.
                  if(!dfs(G[x][i])) return false;
18.
              return true;
19.
          }
20.
          void init(int n){
21.
              this->n = n;
FOR(i, 0, 2*n) G[i].clear();
22.
23.
24.
              memset(mark, 0, sizeof(mark));
25.
          }
26.
27.
           * Each clause is in the form x or y
28.
           * x is abs(x) - 1 and xval is x < 0 ? 1 : 0
29.
30.
31.
          void add_clause(int x, int xval, int y, int yval){
32.
              x = x*2 + xval;
33.
              y = y*2 + yval;
              G[x^1].push_back(y);
34.
35.
              G[y^1].push_back(x);
36.
37.
          bool solve(){
38.
39.
              for(int i = 0; i < 2*n; i += 2){
                  if(!mark[i] && !mark[i+1]){
40.
41.
                       c = 0;
                       if(!dfs(i)){
42.
                           while(c > 0) mark[S[--c]] = false;
43.
44.
                           if(!dfs(i + 1)) return false;
45.
                      }
                  }
46.
47.
48.
              return true;
49.
          }
50.
      }TS;
51.
      int main(){
52.
53.
          int n, m;
          while(scanf("%d %d", &n, &m) != EOF){
54.
55.
              TS.init(n);
56.
              // scan m clauses
              FOR(i, 0, m){
57.
58.
                  int a, b;
                  scanf("%d %d", &a, &b);
59.
60.
                  TS.add_clause(abs(a) - 1, a < 0 ? 1 : 0, abs(b) - 1, b < 0 ? 1 : 0);
61.
62.
              printf("%d\n", TS.solve()? 1 : 0);
63.
64.
          return 0;
65.
     }
```

Bits 1. typedef unsigned int ui; typedef unsigned long long ull; 2. 3. 4. ull BitCount(ull u) { ull uCount = u - ((u >> 1) & 03333333333) - ((u >> 2) & 0111111111111);5. return ((uCount + (uCount >> 3)) & 030707070707) % 63; 6. 7. } 8. ull flipBits(ull n, int k) { 9. 10. ull mask = (1 << k) - 1;return ~n & mask; 11. 12. 13. ull flipBits(ull n) { 14. 15. return ~n; 16. 17. ull differentBits(ull a, ull b) { 18. 19. return BitCount(a ^ b); 20. 21. void getEvenOddBits(ull n) { 22. // Para ui, hacerlo con 8 A's o 5's 23. ull evenBits = n & 0xAAAAAAAAAAAAAA; 24. 25. ull oddBits = n & 0x555555555555555; } 26. 27. ull rotateBits(ull n, int d) { // d negative for left rotation, positive for right. 28. 29. 30. d % = 64;31. return (n >> d) | (n << (64 - d)); 32. } 33. string toBinary(ull n) { 34. 35. string s = 36. while(n) { if (n & 1) s = "1" + s;37. s = "0" + s; 38. else 39. n >>= 1; 40. 41. return s; 42. } 43. ui getSetBitsFromOneToN(ull n){ 44. 45. ui two = 2, ans = 0; ull N = n; 46. 47. while(n) { 48. ans += (N / two) * (two >> 1);if ((N & (two - 1)) > (two >> 1) - 1)49. ans += (N & (two - 1)) - (two >> 1) + 1;50. 51. two <<= 1; n >>= 1; 52. 53. 54. return ans; 55. }

Fast IO

```
1.
      const int BUFFSIZE = 10240;
      char BUFF[BUFFSIZE + 1], *ppp = BUFF;
2.
3.
      int RR, CHAR, SIGN, BYTES = 0;
      #define GETCHAR(c) { \
4.
          if(ppp-BUFF==BYTES && (BYTES==0 || BYTES==BUFFSIZE)) { BYTES = fread(BUFF,1,BUFFSIZE,stdin);
5.
      ppp=BUFF; } \
          if(ppp-BUFF==BYTES && (BYTES>0 && BYTES<BUFFSIZE)) { BUFF[0] = 0; ppp=BUFF; }\</pre>
6.
7.
          c = *ppp++; \setminus
8.
9.
      #define DIGIT(c) (((c) >= '0') && ((c) <= '9'))
      #define MINUS(c) ((c)== '-')
10.
      #define GETNUMBER(n) { \
11.
          n = 0; SIGN = 1; do { GETCHAR(CHAR); } while(!(DIGIT(CHAR) || MINUS(CHAR))); \
12.
          if(MINUS(CHAR)) { SIGN = -1; GETCHAR(CHAR); } \
13.
          while(DIGIT(CHAR)) { n = ((n << 3) + (n << 1)) + CHAR-'0'; GETCHAR(CHAR); } if(SIGN == -1) { n =
14.
      -n; } \
15.
     }
```

FFT 1

```
1. struct Complex{
2.
        double real, imag;
3.
        Complex(){}
4.
        Complex(const complex<double> &a): real(a.real()), imag(a.imag()){}
5.
        Complex(double real, double imag): real(real), imag(imag){}
        Complex operator*(const Complex & a){
6.
7.
            return Complex(real * a.real - imag * a.imag, real * a.imag + a.real * imag);
8.
9
        Complex operator+(const Complex & a){return Complex(real + a.real,
    imag + a.imag);}
10. };
11. Complex conj(const Complex & a){return Complex(a.real, -a.imag);}
13. const int maxN = 1e6:
14. int maxL;
15.
16. int reverses[maxN << 2];</pre>
17. bool v[maxN << 2];</pre>
18. Complex steps[30], unityRoots[maxN << 2], first[maxN << 2], second[maxN << 2];</pre>
19. 11 sums[maxN << 2];</pre>
20.
21. int reverse(int num, int exp){
22.
        int reverse = 0, pot = 1, inv = 1 << (exp - 1);</pre>
23.
        while(inv >= 1){
24.
            if(num & pot) reverse |= inv;
25.
            inv >>= 1, pot <<= 1;
26.
27.
        return reverse;
28. }
29.
30. void sqRoot(Complex *vec, int len, int loog){
        for(int i = len - 1; i >= 0; i--){
32.
            vec[i << 1] = vec[i];
            vec[(i << 1) + 1] = vec[i] * steps[loog];</pre>
33.
34.
        }
35. }
36.
37. void FFT(Complex *coef, int arrS){
        const int saiz = ceil(log2(arrS));
38.
39.
40.
        memset(v, false, sizeof(v));
41
        for(int i = 0; i < (1 << saiz); i++)</pre>
42.
            if(!v[reverses[i] >> (maxL - saiz)])
43.
                swap(coef[i], coef[reverses[i] >> (maxL - saiz)]), v[i] = true;
44.
        unityRoots[0] = Complex(1, 0);
45
46.
        //DIVIDE AND CONQUER...
47.
        for(int T = 1, u = 0; T < (1 << saiz); T <<= 1, u++){
48.
            sqRoot(unityRoots, T, u);
49.
            for(int i = 0; i < (1 << saiz); i += (T << 1)){}
50.
                for(int j = 0; j < T; j++){
                    Complex lTmp = coef[i + j], rTmp = coef[i + j + T];
51.
52.
                    coef[i + j] = lTmp + rTmp * unityRoots[j];
53.
54.
                    coef[i + j + T] = lTmp + rTmp * unityRoots[j + T];
55.
                }
56.
            }
57.
        }
58.}
60. //one & two = coefficient arrays w/sizes, res is where results are stored
61. void polynomMultiplication(int *one, int *two, int *res, int oS, int tS){
        const int saiz = 1 << int(ceil(log2(oS + tS)));</pre>
62.
```

```
63.
64.
        for(int i = 0; i < saiz; i++)</pre>
65.
            first[i] = Complex(i < oS? double(one[i]): 0.0, 0.0);
66.
        for(int i = 0; i < saiz; i++)</pre>
67.
            second[i] = Complex(i < tS? double(two[i]): 0.0, 0.0);</pre>
68.
69.
        FFT(first, saiz), FFT(second, saiz);
70.
71.
        //INVERSE FFT = FFT(conj(C1 * C2) / N)
72.
        for(int i = 0; i < saiz; i++)</pre>
73.
74.
            first[i] = conj(first[i] * second[i]);
75.
        FFT(first, saiz);
76.
77.
        for(int i = 0; i < saiz; i++)</pre>
            res[i] = int(round(first[i].real / saiz));
78.
79.}
```

FFT 2

```
1. const int maxN = 2.1e6 + 3, MOD = 1e9 + 7, LG = 21;
2. //NOTE: maxN>2^LG must be held, remember maxN is twice ur biggest polynom
cd ur[maxN], res[maxN], ors[maxN], cf[maxN];
5. //assumes sz and step are 2-powers
6. void FFT(int cfs, int step, int his, int rs, int sz, cd *res){
        if(cfs == 1)
7.
            for(int i = 0; i < sz; i++) res[i] = cf[his];
R
9
        else{
10.
11.
            FFT(ceil(cfs/2.0), step+1, his, rs<<1, sz>>1, res);
            FFT(cfs>>1, step+1, 1 << step | his, rs<<1, sz>>1, res + (sz>>1));
12.
13.
14.
            for(int i = 0, m = sz >> 1; i < m; i++){
15.
                cd cr = res[i];
16.
                res[i] = cr + res[i + m] * ur[i * rs];
17.
                res[i + m] = cr + res[i + m] * ur[(i + m) * rs];
18.
            }
19.
        }
20.}
21.
22. void mu(const vector<ll> &a, const vector<ll> &b, vector<ll> &ans){
        int sz = 1<<int(ceil(log2(a.size() + b.size() - 1)));</pre>
23.
24.
        for(int i = 0; i < a.size(); i++) cf[i] = a[i];</pre>
25.
26.
        FFT((int)a.size(), 0, 0, (1<<LG)/sz, sz, res);</pre>
27.
28.
        for(int i = 0; i < b.size(); i++) cf[i] = b[i];</pre>
29.
        FFT((int)b.size(), 0, 0, (1<<LG)/sz, sz, ors);</pre>
30.
31.
        for(int i = 0; i < sz; i++) cf[i] = conj(res[i] * ors[i]);</pre>
32.
        FFT(sz, 0, 0, (1<<LG)/sz, sz, res);</pre>
33.
34.
35.
36.
        for(int i = 0; i < a.size() + b.size() - 1; i++)</pre>
37.
            ans.push_back(ll(round(res[i].real() / sz)));
38.
39. }
40.
41. string st;
42. int main(){
        ios::sync_with_stdio(false);
43.
44.
        cin.tie(0), cout.tie(0);
45.
46.
        ur[0] = 1, ur[1] = polar(1.0, M_PI * 2 / (1 << LG));
47.
48.
        for(int i = 2; i < (1 << LG); i++)
            ur[i] = ur[i - 1] * ur[1];
49
50.
51.
        cin >> st;
52.
53.
        vector<ll> a, b, ans;
54.
        for(int i = 0; i < st.size(); i++)</pre>
55.
            a.push back((st[i] == 'B')? 1: 0);
        for(int i = int(st.size() - 1); i >= 0; i--)
56.
57.
            b.push back((st[i] == 'A')? 1: 0);
58.
59.
        mu(a, b, ans);
        for(int i = int(st.size() - 2); i >= 0; i--) cout << ans[i] << '\n';
60.
61.
62. }
63.//1010
```

- 64. //1010
- 65.
- 66. //coefficients can be handled with an integer for how many, and a pair of ints: step and history, which will tell the index of the coefficient in the base case (the only time it will be used)
- 67. //now, for the unity roots, use formula x * 2 % sz to get square simply, in this way we can keep only one array of unity roots.
- 68. //for m current unity roots, when we squere them, all indices will have at least one zero to left more in binary notation. So we just need to calculate squares of the upper side of argand's diagram. So its new size will be a power of two and the step unity will be twice bigger.
- 69. //args: cfs(how many coefficients), step(for coef), history(parities defined),
 rootstep(size of unity root step), rootam(amount of roots to consider)
- 70. //so process is simple now:
- 71. //base case: return coefficient rootam times
- 72. //regular case: rec(ceil(n/2), step+1, history, rootstep*2, next2pow(rootam))
- 73. // rec(floor(n/2), step+1, 1<< st|history, rootstep*2, next2pow(rootam))
- 74.
- 75. //now where should we store them?
- 76. //create array that will store evaluations. So add new parameter Foo *pl, where we will store results of shit. When we call recursively, store at end all even- coef evals. Then put in current and forget, so use same place for unevens

Fraction

```
1.
     struct Fraction {
2.
          11 num, den;
3.
4.
          Fraction() {num = den = 1;}
5.
          Fraction(ll numm, ll denn) : num(numm), den(denn) {
6.
              simplify();
7.
          }
8.
9.
10.
          void simplify() {
              bool signo = false;
11.
                              signo = !signo;
              if (num < 0)
12.
                              signo = !signo;
              if (den < 0)
13.
              11 g = __gcd(num, den);
if (g) {
14
15.
                  num /= g;
16.
17.
                  den /= g;
18.
19.
              if (signo) num = -num;
          }
20.
21.
          Fraction operator + (const Fraction &r) const {
22.
23.
              Fraction ret:
              ret.num = num*r.den + r.num*den;
24.
25.
              ret.den = r.den*den;
              ret.simplify();
26.
27.
              return ret;
28.
29.
30.
          Fraction operator + (const 11 &r) const {
31.
              Fraction ret;
              ret.num = num + r*den;
32.
33.
              ret.den = den;
34.
              return ret;
35.
          }
36.
          Fraction operator - (const Fraction &r) const {
37.
38.
              Fraction ret;
39.
              ret.num = num*r.den - r.num*den;
40.
              ret.den = r.den*den;
              ret.simplify();
41.
42.
              return ret;
43.
          }
44.
45.
          Fraction operator - (const 11 &r) const {
46.
              Fraction ret:
              ret.num = num - r * den;
47.
48.
              ret.den = den;
49.
              return ret;
50.
          }
51.
          Fraction operator * (const Fraction &r) const {
52.
53.
              Fraction ret;
              ret.num = num*r.num;
ret.den = den*r.den;
54.
55.
56.
              ret.simplify();
57.
              return ret;
          }
58.
59.
60.
          Fraction operator * (const 11 &r) const {
61.
              Fraction ret:
62.
              ret.num = num*r;
63.
              ret.den = den;
              ret.simplify();
64.
65.
              return ret;
66.
67.
          Fraction operator / (const Fraction &r) const {
68.
69.
              Fraction ret:
70.
              ret.num = num*r.den;
71.
              ret.den = den*r.num;
```

```
72.
              ret.simplify();
73.
              return ret;
74.
75.
          Fraction operator / (const 11 &r) const {
76.
77.
              Fraction ret;
78.
              ret.num = num;
              ret.den = den*r;
79.
80.
              ret.simplify();
81.
              return ret;
          }
82.
83.
84.
          Fraction pow(int n) const {
85.
              Fraction ret(1, 1);
86.
              Fraction x(num, den);
87.
              while (n) {
                  if (n & 1) ret = ret * x;
88.
                  x = x * x;
89.
90.
                  n >>= 1;
91.
92.
              return ret;
93.
          }
94.
95.
          bool operator <(const Fraction &r) const {</pre>
              return num * r.den < den * r.num;
96.
97.
98.
99.
          bool operator ==(const Fraction &r) const {
100.
              return num == r. num && den == r.den;
101.
102.
          string to_str() {
103.
              return to_string(num) + "/" + to_string(den);
104.
105.
106. };
107.
108. int main() { _
109. Fraction f;
110.
          while(cin >> f.num >> f.den) {
              f.simplify();
111.
              cout << f.num << "/" << f.den << endl;</pre>
112.
113.
114.
          return 0;
115.
116. }
```

```
Hour
      struct Hora {
1.
          int h, m, s, t;
2.
3.
          Hora() { t = h = m = s = 0; }
4.
5.
6.
          Hora(int hh, int mm, int ss) : h(hh), m(mm), s(ss) {
7.
              simplify();
8.
9.
10.
          Hora(string ss) {
              int f1 = ss.find(":");
11.
12.
              h = atoi(ss.substr(0, f1).c_str());
13.
              int f2 = ss.find(":", f1+1);
14.
15.
              if (f2 != -1) {
                  m = atoi(ss.substr(f1+1, f2-f1).c_str());
16.
                   s = atoi(ss.substr(f2+1).c_str());
17.
18.
              else {
19.
                   m = atoi(ss.substr(f1+1).c_str());
20.
21.
                   s = 0;
22.
              }
23.
24.
              simplify();
25.
          }
26.
27.
          Hora operator +(const Hora &r) const {
28.
              Hora ret;
29.
30.
              ret.h = h + r.h;
              ret.m = m + r.m;
31.
              ret.s = s + r.s;
32.
33.
34.
              ret.simplify();
35.
36.
              return ret;
37.
          }
38.
39.
          Hora operator -(const Hora &r) const {
40.
              Hora ret;
41.
              ret.h = h - r.h;
ret.m = m - r.m;
42.
43.
              ret.s = s - r.s;
44.
45.
46.
              ret.simplify();
47.
48.
              return ret;
49.
          }
50.
51.
          bool operator <(const Hora &r) const {</pre>
              return t < r.t;</pre>
52.
53.
54.
55.
          bool operator ==(const Hora &r) const {
56.
              return t == r.t;
57.
58.
59.
          void simplify() {
              t = 3600 * h + 60 * m + s;
60.
              t = (t + 86400) % 86400;
61.
62.
              int tt = t;
h = tt / 3600;
63.
64.
65.
              tt %= 3600;
66.
              m = tt / 60;
              s = tt \% 60;
67.
68.
69.
70.
          string to_str() {
              string ret = "";
71.
72.
```

```
73.
               if (h < 10) ret = "0";
               ret += std::to_string(h) + ":";
74.
75.
76.
               if (m < 10) ret += "0";
77.
               ret += std::to_string(m) + ":";
78.
79.
               if (s < 10) ret += "0";
               ret += std::to_string(s);
80.
81.
82.
               return ret;
83.
           }
84.
85.
           string to_str(const bool b) {
   string ret = "";
86.
87.
               if (h < 10) ret = "0";
ret += to_string(h) + ":";</pre>
88.
89.
90.
91.
               if (m < 10) ret += "0";
               ret += to_string(m);
92.
93.
94.
               return ret;
95.
           }
96.
      };
97.
      int main() {
    Hora h1, h2;
98.
99.
100.
           string s1, s2;
101.
           while(cin >> s1 >> s2) {
102.
103.
               h1 = Hora(s1);
               h2 = Hora(s2);
104.
105.
106.
               cout << (h1 - h2).to_str() << endl;</pre>
107.
           }
108.
109.
           return 0;
110. }
```

Matrix Exponentiation

```
#define FIRST first
1.
       #define SECOND second.first
#define THIRD second.second
2.
3.
4.
5.
       using namespace std;
       typedef pair<int, int> ii;
typedef vector<double> vd;
6.
7.
       typedef vector<vd> Matrix;
typedef long long ll;
8.
9.
10.
11.
       const int maxN = 100;
12.
       Matrix operator *(const Matrix &first, const Matrix &second){
   Matrix ret(first.size(), vd(second[0].size(), 0.0));
13.
14.
15.
            for(int i = 0; i < first.size(); i++)</pre>
                 for(int j = 0; j < second[0].size(); j++)
16.
                       for(int k = 0; k < second.size(); k++)</pre>
17.
18.
                            ret[i][j] += first[i][k] * second[k][j];
19.
            return ret;
20.
       Matrix operator^(Matrix mat, int coef){
21.
            Matrix ret(mat.size(), vd(mat.size(), 0.0));
for(int i = 0; i < mat.size(); i++) ret[i][i] = 1.0;
22.
23.
            while(coef){
24.
                 if(coef & 1)
25.
26.
                      ret = ret * mat;
                 coef >>= 1, mat = mat * mat;
27.
28.
            return ret;
29.
30.
```

Roman Numbers

```
#define MAXN 10
1.
           #define MOD 1000000007
2.
3.
4.
           int mil[MAXN];
5.
6.
           string fill(char c, int n) {
                   string s;
7.
8.
                  while (n--) s += c;
9.
                   return s;
10.
           }
11.
12.
           string toRoman(ll n, int nivel) {
                  if (n == 0) return "";
13.
                  if (n < 4) return fill('I', n);
if (n < 6) return fill('I', 5 - n) + "V";</pre>
14
15.
                  if (n < 9) return string("V") + fill('I', n - 5);</pre>
16.
                 if (n < 9) return string("V") + fill('I', n - 5);
if (n < 11) return fill('I', 10 - n) + "X";
if (n < 40) return fill('X', n / 10) + toRoman(n % 10, nivel);
if (n < 60) return fill('X', 5 - n / 10) + 'L' + toRoman(n % 10, nivel);
if (n < 90) return string("L") + fill('X', n / 10 - 5) + toRoman(n % 10, nivel);
if (n < 110) return fill('X', 10 - n / 10) + "C" + toRoman(n % 10, nivel);
if (n < 400) return fill('C', n / 100) + toRoman(n % 100, nivel);
if (n < 900) return string("D") + fill('C', n / 100 - 5) + toRoman(n % 10, nivel);
if (n < 1100) return fill('C', 10 - n / 100) + "M" + toRoman(n % 100, nivel);</pre>
17.
18.
19.
20.
21.
22.
23.
24.
                  if (n < 1100) return fill('C', 10 - n / 100) + "M" + toRoman(n % 100, nivel); if (n < 4000) return fill('M', n / 1000) + toRoman(n % 1000, nivel);
25.
26.
27.
28.
                   string ret = toRoman(n / 1000, nivel + 1);
29.
                   string ret2 = toRoman(n % 1000, nivel);
                  mil[nivel] = ret.length();
if (ret2 == "")    return ret;
return ret + " " + toRoman(n % 1000, nivel);
30.
31.
32.
33.
           }
34.
35.
           string toRoman(ll n) {
                  string ret = toRoman(n, 0);
for(int i = 0; mil[i]; i ++) {
    ret = fill('_', mil[i]) + "\n" + ret;
36.
37.
38.
39.
40.
41.
                   return ret;
42.
           }
43.
44.
           int main() { _
45.
                   11 n;
46.
                   while (cin >> n) {
47.
                          cout << toRoman(n) << endl;</pre>
48.
49.
                   return 0;
50.
           }
```

Rotate Matrix

```
1. #define MAXN 10
2. #define MOD 1000000007
3.
4. int mat[MAXN][MAXN];
5.
     void giraMat(int k, int n) {
6.
           k %= 4;
7.
8.
9.
           FOR(p, 0, k) {
10.
                 FOR(i, 0, n/2) {
                      (1, 0, n/2) {
FOR(j, i, n - i - 1) {
    ii p1(i, j);
    ii p2(j, n - i - 1);
    ii p3(n - i - 1, n - j - 1);
    ii p4(n - j - 1, i);
11.
12.
13.
14.
15.
16.
                            int aux = mat[p1.first][p1.second];
17.
                            mat[p1.first][p1.second] = mat[p4.first][p4.second];
mat[p4.first][p4.second] = mat[p3.first][p3.second];
18.
19.
                            mat[p3.first][p3.second] = mat[p2.first][p2.second];
20.
21.
                            mat[p2.first][p2.second] = aux;
                      }
22.
23.
                 }
24.
           }
25. }
26.
27. int main() { _
28.    int cant = 1;
29.    FOR(i, 0, MAXN) FOR(j, 0, MAXN) {
30.
                 mat[i][j] = cant;
31.
                 cant ++;
           }
32.
33.
34.
           giraMat(6, MAXN);
35.
           FOR(i, 0, MAXN) FOR(j, 0, MAXN)
     cout << mat[i][j] << "\t\n"[j == MAXN-1];</pre>
36.
37.
           return 0;
38.
39. }
```

```
Sorts
1. #define MAXN 15
2. #define MOD 1000000007
3.
4. int arr[MAXN];
5. int comp, inter;
6.
    void print() {
7.
        FOR(i, 0, MAXN) {
8.
             cout << arr[i] << " \n"[i == MAXN-1];
9.
10.
11.
         cout << endl;</pre>
12. }
13.
14. void bubbleSort() {
15.
         int n = MAXN;
        bool b = true;
16.
17.
18.
         for(int i = 0; i < n - 1 && b; i ++){}
             b = false;
19.
20.
             FOR(j, 0, n - 1 - i) {
                 if (arr[j+1] < arr[j]) {</pre>
21.
22.
                      swap(arr[j], arr[j + 1]);
23.
                      b = true;
24.
25.
                      inter ++;
26.
27.
                 comp ++;
28.
             }
29.
         }
30. }
31.
32. void selectionSort() {
33.
        int n = MAXN;
         FOR(i, 0, n - 1) {
34.
35.
             int mini = i;
36.
             FOR(j, i + 1, n) {
37.
                 comp ++;
                 if (arr[j] < arr[mini]) mini = j;</pre>
38.
39.
             }
40.
41.
             if (i != mini) {
42.
                 swap(arr[i], arr[mini]);
43.
44.
                 inter ++;
45.
             }
46.
        }
47. }
48.
49. void intercambioSort() {
50.
         int n = MAXN;
        FOR(i, 0, n - 1) {
    FOR(j, i + 1, n) {
        if (arr[i] > arr[j]) {
51.
52.
53.
                      swap(arr[i], arr[j]);
54.
55.
                      inter++;
56.
57.
                 comp++;
58.
             }
59.
        }
60.}
61.
62. void insertionSort() {
63.
        int j, key, n = MAXN;
64.
         FOR(i, 1, n) {
65.
             key = arr[i];
             j = i - 1;
66.
67.
             while (j >= 0 && arr[j] > key) {
                 inter ++;
arr[j+1] = arr[j];
68.
69.
70.
                 j = j - 1;
71.
             }
72.
```

```
73.
             comp = inter;
74.
             if(j) comp ++;
75.
76.
             arr[j+1] = key;
77.
        }
78. }
80. void insertionBinarySort() {
81.
        int n = MAXN;
         vi v;
        FOR(i, 0, n) {
83.
84.
             v.insert(upper_bound(v.begin(), v.end(), arr[i]), arr[i]);
85.
86
87.
        copy(v.begin(), v.end(), arr);
88. }
29
90.
91. void Une(int ini, int mitad, int fin) {
        int aux[fin - ini + 1];
int i = ini, j = mitad + 1, k = 0;
92.
93.
94.
95.
        while(i <= mitad && j <= fin) {</pre>
96.
             if (arr[i] < arr[j]) {</pre>
                 aux[k] = arr[i];
97.
98.
                 i ++;
99.
100.
             else {
101
                 aux[k] = arr[j];
102.
                 j ++;
103.
             k ++;
104
105.
106.
107.
        for(; i <= mitad; i ++, k ++)</pre>
                                            aux[k] = arr[i];
108.
        for(; j <= fin; j ++, k ++)
                                            aux[k] = arr[j];
109.
110.
        for(k = 0; ini <= fin; ini ++, k ++)</pre>
111.
             arr[ini] = aux[k];
112.}
113.
114.void mergeSort(int ini, int fin){
        if(ini < fin){</pre>
115.
             int mitad = (ini + fin) / 2;
116.
117.
             mergeSort(ini, mitad);
             mergeSort(mitad + 1, fin);
118.
119.
             Une(ini, mitad, fin);
120.
        }
121.}
122.
123.
124.void heapify(int n, int i) {
125.
        int maxi = i;
         int 1 = 2*i + 1;
126.
        int r = 2*i + 2;
127.
128.
129.
        if (1 < n && arr[1] > arr[maxi])
130.
             maxi = 1:
131.
        if (r < n && arr[r] > arr[maxi])
132.
133.
             \max i = r;
134.
        if (maxi != i) {
    swap(arr[i], arr[maxi]);
135.
136.
137.
             heapify(n, maxi);
138.
        }
139.}
140.
141.void heapSort() {
142.
        int n = MAXN;
143.
         for (int i = n / 2 - 1; i >= 0; i--)
144.
             heapify(n, i);
145.
146.
        for (int i = n-1; i >= 0; i--) {
```

```
147.
             swap(arr[0], arr[i]);
148.
             heapify(i, 0);
149.
150.}
151.
152.int part(int low, int high) {
153.
        int pivot = arr[high];
         int i = low - 1;
154.
155.
156.
         FOR(j, low, high) {
157.
             if (arr[j] <= pivot) {</pre>
158.
159.
                 swap(arr[i], arr[j]);
160.
             }
161.
162.
         swap(arr[i + 1], arr[high]);
163.
        return (i + 1);
164.}
165.
166.// QuickSort with less recursion calls.
167.void quickSortNoTail(int low, int high) {
        while (low < high) {</pre>
             int pi = part(low, high);
169.
170.
171.
             if (pi - low < high - pi) {</pre>
                 quickSortNoTail(low, pi - 1);
172.
173.
                 low = pi + 1;
174.
             else {
175.
                 quickSortNoTail(pi + 1, high);
176.
177.
                 high = pi - 1;
178.
179.
180.}
181.
182.void quickSort(int low, int high) {
       if (low < high) {</pre>
183.
             int pi = part(low, high);
184.
             quickSort(low, pi - 1);
quickSort(pi + 1, high);
185.
186.
187.
188.}
```

11. Otros

Series

$$\sum_{i=0}^{n} 1 = n$$

$$\sum_{i=0}^{n} i = \frac{n * (n+1)}{2}$$

$$\sum_{i=0}^{n} i^2 = \frac{n * (n+1) * (2n+1)}{6}$$

$$\sum_{i=0}^{n} i^3 = \frac{n^2 * (n+1)^2}{4}$$

$$\sum_{i=0}^{n} i^k \approx \frac{1}{k+1} * n^{k+1}$$

$$\sum_{i=0}^{n} a^{i} = \frac{a^{n+1} - 1}{a - 1}$$

$$\sum_{i=0}^{n} 2^{i} = 2^{n+1} - 1$$

$$\sum_{i=0}^{n} {i \choose n} = 2^n$$

$$\log x^y = y \, \log x$$

$$\log xy = \log x + \log y$$

$$\log \frac{x}{y} = \log x - \log y$$

First 5000 digits of PI

First 150 Fibonacci numbers

```
1:1
2:1
3:2
4:3
5:5
6 : 8 = 2^3
7:13
8 : 21 = 3 \times 7
9:34=2\times17
10 : 55 = 5 \times 11
11 : 89
12 : 144 = 2^4 \times 3^2
13 : 233
14 : 377 = 13 x 29
15 : 610 = 2 \times 5 \times 61
16 : 987 = 3 x 7 x 47
17 : 1597
18 : 2584 = 2^3 \times 17 \times 19
19 : 4181 = 37 x 113
20 : 6765 = 3 \times 5 \times 11 \times 41
21 : 10946 = 2 x 13 x 421
22 : 17711 = 89 x 199
23 : 28657
24 : 46368 = 2^5 \times 3^2 \times 7 \times 23
25 : 75025 = 5^2 \times 3001
26 : 121393 = 233 \times 521
27 : 196418 = 2 x 17 x 53 x 109
28 : 317811 = 3 x 13 x 29 x 281
29 : 514229
30 : 832040 = 2^3 \times 5 \times 11 \times 31 \times 61
31 : 1346269 = 557 x 2417
32 : 2178309 = 3 \times 7 \times 47 \times 2207
33 : 3524578 = 2 x 89 x 19801
34 : 5702887 = 1597 x 3571
35 : 9227465 = 5 \times 13 \times 141961
36 : 14930352 = 2^4 \times 3^3 \times 17 \times 19 \times 107
37 : 24157817 = 73 x 149 x 2221
38 : 39088169 = 37 \times 113 \times 9349
39 : 63245986 = 2 \times 233 \times 135721
40 : 102334155 = 3 x 5 x 7 x 11 x 41 x 2161
41 : 165580141 = 2789 x 59369
42 : 267914296 = 2^3 \times 13 \times 29 \times 211 \times 421
43 : 433494437
44 : 701408733 = 3 \times 43 \times 89 \times 199 \times 307
45 : 1134903170 = 2 \times 5 \times 17 \times 61 \times 109441
46 : 1836311903 = 139 x 461 x 28657
47 : 2971215073
48 : 4807526976 = 2^6 \times 3^2 \times 7 \times 23 \times 47 \times 1103
49 : 7778742049 = 13 x 97 x 6168709
50 : 12586269025 = 5^2 \times 11 \times 101 \times 151 \times 3001
51 : 20365011074 = 2 x 1597 x 6376021
52 : 32951280099 = 3 x 233 x 521 x 90481
53 : 53316291173 = 953 \times 55945741
54 : 86267571272 = 2^3 \times 17 \times 19 \times 53 \times 109 \times 5779
55 : 139583862445 = 5 \times 89 \times 661 \times 474541
56 : 225851433717 = 3 \times 7^2 \times 13 \times 29 \times 281 \times 14503
57 : 365435296162 = 2 x 37 x 113 x 797 x 54833
58 : 591286729879 = 59 x 19489 x 514229
59 : 956722026041 = 353 \times 2710260697
60 : 1548008755920 = 2^4 \times 3^2 \times 5 \times 11 \times 31 \times 41 \times 61 \times 2521
61 : 2504730781961 = 4513 x 555003497
62 : 4052739537881 = 557 \times 2417 \times 3010349
63 : 6557470319842 = 2 \times 13 \times 17 \times 421 \times 35239681
64 : 10610209857723 = 3 x 7 x 47 x 1087 x 2207 x 4481
65 : 17167680177565 = 5 \times 233 \times 14736206161
66: 27777890035288 = 2^3 \times 89 \times 199 \times 9901 \times 19801
67 : 44945570212853 = 269 \times 116849 \times 1429913
68 : 72723460248141 = 3 \times 67 \times 1597 \times 3571 \times 63443
69 : 117669030460994 = 2 \times 137 \times 829 \times 18077 \times 28657
70 : 190392490709135 = 5 \times 11 \times 13 \times 29 \times 71 \times 911 \times 141961
71 : 308061521170129 = 6673 x 46165371073
72 : 498454011879264 = 2^5 \times 3^3 \times 7 \times 17 \times 19 \times 23 \times 107 \times 103681
73 : 806515533049393 = 9375829 x 86020717
```

```
74 : 1304969544928657 = 73 x 149 x 2221 x 54018521
75 : 2111485077978050 = 2 \times 5^2 \times 61 \times 3001 \times 230686501
76 : 3416454622906707 = 3 x 37 x 113 x 9349 x 29134601
77 : 5527939700884757 = 13 \times 89 \times 988681 \times 4832521
78 : 8944394323791464 = 2^3 \times 79 \times 233 \times 521 \times 859 \times 135721
79 : 14472334024676221 = 157 x 92180471494753
80 : 23416728348467685 = 3 x 5 x 7 x 11 x 41 x 47 x 1601 x 2161 x 3041
81 : 37889062373143906 = 2 \times 17 \times 53 \times 109 \times 2269 \times 4373 \times 19441
82 : 61305790721611591 = 2789 x 59369 x 370248451
83: 99194853094755497
84 : 160500643816367088 = 2^4 \times 3^2 \times 13 \times 29 \times 83 \times 211 \times 281 \times 421 \times 1427
85 : 259695496911122585 = 5 \times 1597 \times 9521 \times 3415914041
86 : 420196140727489673 = 6709 x 144481 x 433494437
87 : 679891637638612258 = 2 x 173 x 514229 x 3821263937
88 : 1100087778366101931 = 3 \times 7 \times 43 \times 89 \times 199 \times 263 \times 307 \times 881 \times 967
89 : 1779979416004714189 = 1069 x 1665088321800481
90 : 2880067194370816120 = 2<sup>3</sup> x 5 x 11 x 17 x 19 x 31 x 61 x 181 x 541 x 109441
91 : 4660046610375530309 = 13<sup>2</sup> x 233 x 741469 x 159607993
92 : 7540113804746346429 = 3 \times 139 \times 461 \times 4969 \times 28657 \times 275449
93 : 12200160415121876738 = 2 x 557 x 2417 x 4531100550901
94 : 19740274219868223167 = 2971215073 \times 6643838879
95 : 31940434634990099905 = 5 \times 37 \times 113 \times 761 \times 29641 \times 67735001
96 : 51680708854858323072 = 2^7 \times 3^2 \times 7 \times 23 \times 47 \times 769 \times 1103 \times 2207 \times 3167
97 : 83621143489848422977 = 193 x 389 x 3084989 x 361040209
98 : 135301852344706746049 = 13 \times 29 \times 97 \times 6168709 \times 599786069
99 : 218922995834555169026 = 2 \times 17 \times 89 \times 197 \times 19801 \times 18546805133
100 : 354224848179261915075 = 3 \times 5^2 \times 11 \times 41 \times 101 \times 151 \times 401 \times 3001 \times 570601
101 : 573147844013817084101 = 743519377 x 770857978613
102 : 927372692193078999176 = 2^3 x 919 x 1597 x 3469 x 3571 x 6376021
103 : 1500520536206896083277 = 519121 x 5644193 x 512119709
104 : 2427893228399975082453 = 3 \times 7 \times 103 \times 233 \times 521 \times 90481 \times 102193207
105 : 3928413764606871165730 = 2 \times 5 \times 13 \times 61 \times 421 \times 141961 \times 8288823481
106 : 6356306993006846248183 = 953 x 55945741 x 119218851371
107 : 10284720757613717413913 = 1247833 x 8242065050061761
108 : 16641027750620563662096 = 2^4 \times 3^4 \times 17 \times 19 \times 53 \times 107 \times 109 \times 5779 \times 11128427
109 : 26925748508234281076009 = 827728777 x 32529675488417
110 : 43566776258854844738105 = 5 \times 11^2 \times 89 \times 199 \times 331 \times 661 \times 39161 \times 474541
111 : 70492524767089125814114 = 2 \times 73 \times 149 \times 2221 \times 1459000305513721
112 : 114059301025943970552219 = 3 \times 7^2 \times 13 \times 29 \times 47 \times 281 \times 14503 \times 10745088481
113 : 184551825793033096366333 = 677 x 272602401466814027129
114 : 298611126818977066918552 = 2^3 x 37 x 113 x 229 x 797 x 9349 x 54833 x 95419
115 : 483162952612010163284885 = 5 x 1381 x 28657 x 2441738887963981
116 : 781774079430987230203437 = 3 x 59 x 347 x 19489 x 514229 x 1270083883
117 : 1264937032042997393488322 = 2 \times 17 \times 233 \times 29717 \times 135721 \times 39589685693
118 : 2046711111473984623691759 = 353 x 709 x 8969 x 336419 x 2710260697
119 : 3311648143516982017180081 = 13 x 1597 x 159512939815855788121
120 : 5358359254990966640871840 = 2^5 \times 3^2 \times 5 \times 7 \times 11 \times 23 \times 31 \times 41 \times 61 \times 241 \times 2161 \times 2521 \times 20641
121 : 8670007398507948658051921 = 89 x 97415813466381445596089
122 : 14028366653498915298923761 = 4513 x 555003497 x 5600748293801
123 : 22698374052006863956975682 = 2 x 2789 x 59369 x 68541957733949701
124 : 36726740705505779255899443 = 3 x 557 x 2417 x 3010349 x 3020733700601
125 : 59425114757512643212875125 = 5^3 \times 3001 \times 158414167964045700001
126 : 96151855463018422468774568 = 2<sup>3</sup> x 13 x 17 x 19 x 29 x 211 x 421 x 1009 x 31249 x 35239681
127 : 155576970220531065681649693 = 27941 x 5568053048227732210073
128 : 251728825683549488150424261 = 3 \times 7 \times 47 \times 127 \times 1087 \times 2207 \times 4481 \times 186812208641
129 : 407305795904080553832073954 = 2 \times 257 \times 5417 \times 8513 \times 39639893 \times 433494437
130 : 659034621587630041982498215 = 5 \times 11 \times 131 \times 233 \times 521 \times 2081 \times 24571 \times 14736206161
131 : 1066340417491710595814572169
132 : 1725375039079340637797070384 = 2^4 x 3^2 x 43 x 89 x 199 x 307 x 9901 x 19801 x 261399601
133 : 2791715456571051233611642553 = 13 x 37 x 113 x 3457 x 42293 x 351301301942501
134 : 4517090495650391871408712937 = 269 \times 4021 \times 116849 \times 1429913 \times 24994118449
135 : 7308805952221443105020355490 = 2 \times 5 \times 17 \times 53 \times 61 \times 109 \times 109441 \times 1114769954367361
136 : 11825896447871834976429068427 = 3 x 7 x 67 x 1597 x 3571 x 63443 x 23230657239121
137 : 19134702400093278081449423917
138 : 30960598847965113057878492344 = 2^3 \times 137 \times 139 \times 461 \times 691 \times 829 \times 18077 \times 28657 \times 1485571
139 : 50095301248058391139327916261 = 277 x 2114537501 x 85526722937689093
140 : 81055900096023504197206408605 = 3 x 5 x 11 x 13 x 29 x 41 x 71 x 281 x 911 x 141961 x 12317523121
141: 131151201344081895336534324866 = 2 \times 108289 \times 1435097 \times 142017737 \times 2971215073
142 : 212207101440105399533740733471 = 6673 \times 46165371073 \times 688846502588399
143 : 343358302784187294870275058337 = 89 x 233 x 8581 x 1929584153756850496621
144 : 555565404224292694404015791808 = 2^6 \times 3^3 \times 7 \times 17 \times 19 \times 23 \times 47 \times 107 \times 1103 \times 103681 \times 10749957121
145 : 898923707008479989274290850145 = 5 x 514229 x 349619996930737079890201
146 : 1454489111232772683678306641953 = 151549 x 9375829 x 86020717 x 11899937029
147 : 2353412818241252672952597492098 = 2 x 13 x 97 x 293 x 421 x 3529 x 6168709 x 347502052673
148 : 3807901929474025356630904134051 = 3 x 73 x 149 x 2221 x 11987 x 54018521 x 81143477963
149 : 6161314747715278029583501626149 = 110557 x 162709 x 4000949 x 85607646594577
150 : 9969216677189303386214405760200 = 2^3 \times 5^2 \times 11 \times 31 \times 61 \times 101 \times 151 \times 3001 \times 12301 \times 18451 \times 23068650
```

First	1000	prime	numl	oers
	2	3	5	7

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 1229	1117	1123 1237	1129	1151 1259	1153 1277	1163 1279	1171	1181 1289	1187 1291
1193 1297	1201 1301	1213 1303	1217 1307	1223 1319	1321	1231 1327	1361	1249 1367	1373	1381	1399	1283 1409	1423	1427
1429	1433	1439	1447	1451	1453	1459	1471	1481	1483	1487	1489	1493	1423	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
2579	2591	2593	2609	2617	2621	2633	2647	2657	2659	2663	2671	2677	2683	2687
2689	2693	2699	2707	2711	2713	2719	2729	2731	2741	2749	2753	2767	2777	2789
2791	2797	2801	2803	2819	2833	2837	2843	2851	2857	2861	2879	2887	2897	2903
2909	2917 3049	2927	2939 3067	2953	2957	2963	2969	2971	2999	3001	3011	3019	3023	3037 3181
3041 3187	3191	3061 3203	3209	3079 3217	3083 3221	3089 3229	3109 3251	3119 3253	3121 3257	3137 3259	3163 3271	3167 3299	3169 3301	3307
3313	3319	3323	3329	3331	3343	3347	3359	3361	3371	3373	3389	3391	3407	3413
3433	3449	3457	3461	3463	3467	3469	3491	3499	3511	3517	3527	3529	3533	3539
3541	3547	3557	3559	3571	3581	3583	3593	3607	3613	3617	3623	3631	3637	3643
3659	3671	3673	3677	3691	3697	3701	3709	3719	3727	3733	3739	3761	3767	3769
3779	3793	3797	3803	3821	3823	3833	3847	3851	3853	3863	3877	3881	3889	3907
3911	3917	3919	3923	3929	3931	3943	3947	3967	3989	4001	4003	4007	4013	4019
4021	4027	4049	4051	4057	4073	4079	4091	4093	4099	4111	4127	4129	4133	4139
4153	4157	4159	4177	4201	4211	4217	4219	4229	4231	4241	4243	4253	4259	4261
4271	4273	4283	4289	4297	4327	4337	4339	4349	4357	4363	4373	4391	4397	4409
4421	4423	4441	4447	4451	4457	4463	4481	4483	4493	4507	4513	4517	4519	4523
4547	4549	4561	4567	4583	4591	4597	4603	4621	4637	4639	4643	4649	4651	4657
4663	4673	4679	4691	4703	4721	4723	4729	4733	4751	4759	4783	4787	4789	4793
4799	4801	4813	4817	4831	4861	4871	4877	4889	4903	4909	4919	4931	4933	4937
4943	4951 5059	4957	4967	4969 5087	4973 5099	4987	4993 5107	4999 5113	5003 5119	5009 5147	5011 5153	5021	5023 5171	5039 5179
5051 5189	5197	5077 5209	5081 5227	5231	5233	5101 5237	5261	5273	5279	5281	5297	5167 5303	5309	5323
5333	5347	5351	5381	5387	5393	5399	5407	5413	5417	5419	5431	5437	5441	5443
5449	5471	5477	5479	5483	5501	5503	5507	5519	5521	5527	5531	5557	5563	5569
5573	5581	5591	5623	5639	5641	5647	5651	5653	5657	5659	5669	5683	5689	5693
5701	5711	5717	5737	5741	5743	5749	5779	5783	5791	5801	5807	5813	5821	5827
5839	5843	5849	5851	5857	5861	5867	5869	5879	5881	5897	5903	5923	5927	5939
5953	5981	5987	6007	6011	6029	6037	6043	6047	6053	6067	6073	6079	6089	6091
6101	6113	6121	6131	6133	6143	6151	6163	6173	6197	6199	6203	6211	6217	6221
6229	6247	6257	6263	6269	6271	6277	6287	6299	6301	6311	6317	6323	6329	6337
6343	6353	6359	6361	6367	6373	6379	6389	6397	6421	6427	6449	6451	6469	6473
6481	6491	6521	6529	6547	6551	6553	6563	6569	6571	6577	6581	6599	6607	6619
6637	6653	6659	6661	6673	6679	6689	6691	6701	6703	6709	6719	6733	6737	6761
6763	6779	6781	6791	6793	6803	6823	6827	6829	6833	6841	6857	6863	6869	6871
6883	6899 7013	6907	6911	6917	6947	6949	6959	6961	6967	6971	6977 7121	6983	6991	6997
7001 7159	7013 7177	7019 7187	7027 7193	7039 7207	7043 7211	7057 7213	7069 7219	7079 7229	7103 7237	7109 7243	7121 7247	7127 7253	7129 7283	7151 7297
7307	7309	7321	7331	7333	7349	7351	7369	7393	7411	7417	7433	7451	7457	7459
7477	7481	7487	7489	7499	7507	7517	7523	7529	7537	7541	7547	7549	7559	7561
7573	7577	7583	7589	7591	7603	7607	7621	7639	7643	7649	7669	7673	7681	7687
7691	7699	7703	7717	7723	7727	7741	7753	7757	7759	7789	7793	7817	7823	7829
7841	7853	7867	7873	7877	7879	7883	7901	7907	7919	-	-		-	

First 25 Catalan numbers

1	1
2	1 2
4	5
5	14
6	42
7	132
8	429
9	1430
10	4862
11	16796
12	58786
13	208012
14	742900
15	2674440
16	9694845
17	35357670
18	129644790
19	477638700
20	1767263190
21	6564120420
22	24466267020
23	91482563640
24	343059613650
25	1289904147324
26	4861946401452
27	18367353072152
28	69533550916004
29	263747951750360
30	1002242216651360
31	3814986502092300

First 100 powers of 2 277 278 279 279 279 279 279 279 271 2				
2**P\$ 1	First 100	nowers of 2		
2-12		•		
2-1 8 2-78 3022145490857273678544 2-1 8 2-79 64462998671458731388 2-4 16 2-80 12892581964629174786176 2-8 2 16 2-80 12892581964629174786176 2-8 2 16 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2				
2-4 16 2-79 604-62/9989731458735888 2-4 16 2-78 12892191462917458735888 2-5 32 2-81 12891462917458735888 2-7 6 62 2-82 2-81 2-81 247881635292758849412567 2-8 2-8 2-8 2-8 2-8 2-8 2-8 2-8 2-8 2-8				
2-76 64 2.2-81 24178163922958349412352 2-76 64 2.2-82 4857927848516088847094 2-77 128 2.7-81 96744655691793137649488 2-78 276 2.2-81 96744655691793137649488 2-78 276 2.2-81 967446556917931377649488 2-78 276 2.2-81 96744655691793137649488 2-78 2.2-81 96744655691793137649488 2-78 2.2-81 96744655691793137649488 2-78 2.2-81 96744655691793137649488 2-78 2.2-82 96 18975712245335627181195264 2-71 968 2.2-88 2.2-89 777712245395627481195264 2-71 968 2.2-88 3094569687114568677478166 2-71 97 18192 2.2-99 618976919564269817440952112 2-71 97 18192 2.2-99 618976919564269817440952112 2-71 97 18192 2.2-99 618976919564269817440952112 2-71 97 18192 2.2-91 968569181959696969696969696969696969696969696969	2^3		2^79	604462909807314587353088
2-76	2^4		2^80	1208925819614629174706176
2-79 128 256 2-84 103242111383406579520816 1 2-96 512 2-86 3068502622766813359657652 2 2-10 1024 2-86 7737125455336271315264 2 2-11 1024 2-86 7737125455336271315264 2 2-12 1036 2-86 136736556 2 2-13 1036 2-86 136736556 2 2-14 1036 2-86 136736556 2 2-15 1036 2-86 2-96 1237940825388274895124224 2 2-15 10376 2-86 2-91 10372468978538827895124224 2 2-16 65536 2-92 123794082538827895124224 2 2-16 65536 2-92 123794082538827895124224 2 2-17 131877 2-93 90825382857895124224 2 2-18 1036 2-99 12379408295383827895124204 2 2-19 1048 2-99 1048	2^5	32	2^81	2417851639229258349412352
2-9 512 2-85 382-85 2-96 1924 1934281111383406795;29816 2 2-10 1024 2-86 7737125245335627181195264 3 2-11 2048 2-86 7737125245335627181195264 3 2-11 2048 2-86 32-86 32-97 1524269910672534629052 3 2-11 4069 2-86 32-86 32-97 152426991067253462905 3 2-12 4069 32-86 32-98 329-9805928121456852478105 3 2-13 4069 32-86 32-98 329-980592821456852478105 3 2-14 41834 32-98 329-9805928284448 3 2-15 45536 2-99 12475880078778706459780348448 3 2-16 65536 2-92 4951706157141512009596489806 3 2-17 131072 2-93 390853201428384119912299792 3 2-18 7214 2-94 198074062535680489883895784 3 2-19 544288 2-95 390448125713216879677197168 3 2-10 54428 2-99 41886742 3-99 390853201428384119912299792 3 2-18 7214 34434 3-99 318128515215245787777168 3 2-10 144344 3-99 418127124 3 2-10 144344 3-				
1924				
2-19				
2-11 2-048 2-18 30-048-509-012-18-18-18-18-18-18-18-18-18-18-18-18-18-				
2-12				
2-13 8192 2-189 618970819942698137449565112 2-114 16384 2-296 12379408932858927489912424 2-115 37768 2-291 247588067857976894978248448 2-115 37768 2-291 247588067857976894978248448 2-115 24758807857976894978248448 2-115 2475880785797894978248448 2-115 2475880785797894978248448 2-115 2475880785797894978484848 2-115 2475880785797894978484848 2-115 24758807857949484848 2-115 24758807857949484848 2-115 247588078579494848 2-115 24758807857949484848 2-115 247588078579494848 2-115 247588078579494848 2-115 247588078579494848 2-115 247588078579494848 2-115 247588078579494848 2-115 247588078579494848 2-115 24758807857894848 2-115 24758807857894848 2-115 2475880785788848 2-115 24758807857894848 2-115 24758807857894848 2-115 24758807857894848 2-115 24758807857894848 2-115 24758807857894848 2-115 24758807857894848 2-115 2475880785807858 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247588078578948 2-115 247580				
2°15 32768 2°91 24758807785769694798248448 2°16 65536 2°92 2°93 9981520314283042199102993792 2°18 26144 2°94 9981520314283042199102993792 2°19 524288 2°95 39614680157132168796771375168 2°20 1988576 2°96 79228162734624337593437983336 2°11 2897152 2°97 158456128628528518718708790677 2°12 2997152 2°97 158456128628528518718708790667 2°12 2997152 2°97 15845612862852867818708790667 2°12 2997152 2°97 158456128628528678187097906677 2°12 2997152 2°97 158456128628528678738774179593336 2°12 2997172 2014 12675666002282284149679226737 2°25 3535422 2°10 12675666002282284149679226537 2°26 367683456 2°104 20282409693561574423397251286616 2°27 362437728 2°104 20282409693561574423397251286606 2°38 26648545456 2°104 202824096935616744223972527802			2^89	618970019642690137449562112
2°16 65336 2°92 4951761571415214099596496896 2°17 318072 2°93 9985283142834915919299727 2°18 262144 2°94 1980764062856668439838987584 2°19 52428 2°95 7922810521426433759354396836 2°20 1048576 2°96 7922810521426433759354396836 2°21 2097152 2°97 31845632828286751878936374075881344 2°22 4194304 2°98 31651265867587352374175881344 2°23 388608 2°99 3163125680781141770874831302088 2°24 16777212 2°108 2°108 1276566822282284784782823874 2°24 16777212 2°108 2°108 1276766474844 2°27 14027728 2°108 127676674844 2°27 14027728 2°138 18141204881428283521197365543088 2°28 268473656 2°109 495648192673634484825572832 2°29 536870912 2°109 495648192673634684789482577682 2°29 13673741824 2°106 4956481926736368974784825257682	2^14	16384	2^90	1237940039285380274899124224
2°17 131072 2°93 99852831428384219912999792 2°18 26144 2°94 19887466255566684938358957584 2°19 524288 2°95 39614881257132168796771975168 2°20 1848576 2°96 79281653146343759394396336 2°21 299152 2°97 158456315928828687818787968672 2°22 419434 2°98 638253808114114780768735634715801344 2°23 8388688 2°99 638253808114114780768351602688 2°24 16777216 2°108 1267055606222894014957921635767 2°25 33356432 2°101 25538128045645880293486410752 2°26 6708804 2°102 2876822809936017074238472125601 2°27 134217728 2°103 18141264801223832511973625643808 2°29 13421344 2°104 282244967206 2°29 13421444 2°108 281219481238382119736256438080 2°12 1392345656 2°108 281219481253564360601764238472126601 2°13 1382345656 2°108 28121948125365660176423472756808	2^15	32768	2^91	2475880078570760549798248448
2-18 262144 2-94 1986/78-085666849898385987584 2-19 124288 2-95 39614881573715168 2-20 1048576 2-96 792281652142643375935393369336 2-21 2097152 2-97 71584653252978996672 2-22 4104304 2-98 3166125696787353937475801344 2-23 388608 2-99 31661265985787353937475801344 2-24 16777216 2-100 126705606022822540149070320376 2-25 33554122 2-101 32535012046458802939404040752 2-26 07108864 2-102 5278662406912017665988312821548 2-27 07108864 2-102 5278662406912017665988312821548 2-28 28435456 2-102 52786824067212136816 2-29 536879012 2-105 49564812927933348847894802572832 2-31 2147483568 2-104 16225927682921336339157801288818 2-32 2494667296 2-108 812293841646665816957890514664 2-33 258934592 2-109 64968731067856566362967656 <t< td=""><td></td><td></td><td></td><td></td></t<>				
2-19 524288 2-95 39514891257132168796771975168 2-20 1048576 2-96 7922181214643375935933636 2-21 2497152 2-97 158456315982582675187887996977 2-22 2149344 2-98 31691156068768753937473816144 2-23 3388688 2-99 633825360114114760748531602688 2-24 16777216 2-100 12675666002322924041693026376 2-25 67168864 2-102 55766600232924041693205376 2-26 67168864 2-102 5576660024021716769893205376 2-27 154217728 2-103 1041204801825353211973625643088 2-28 26413455 2-104 20523406960151674213921251664 2-29 35887891124 2-105 4554419287333468748452572614 2-12 210 3588789124 2-108 4554419287333468748545272614 2-12 3494967206 2-108 3245185536842672678315600576256 2-13 3893934592 2-109 2-1096731653435660130571362408236902 2-13 1547476736 2-11 5595626607				
2°20 1048576 2°96 79228162514264337593543969336 2°21 2997152 2°98 31691256982852675182887437881444 2°23 388668 2°99 6382233614114786974851626288 2°24 16777216 2°10e 127656606228222924801496793205376 2°25 33554432 2°101 2533361200645643880293537564308 2°26 6718864 2°102 5976602480912917669986312821504 2°27 71217728 2°103 192282409663651676423947251286016 2°28 268435456 2°104 29282409663651676423947251286016 2°29 36867912 2°106 8111963841466068169578906144064 2°31 117483648 2°106 8111963841466068169578906144064 2°32 4294067296 2°188 32431855365842672678783150620576256 2°33 8399345592 2°108 3453185536584267267883150620576256 2°34 17179869184 2°110 129867421463378699713262486236826 2°35 453579385856 2°111 1296742146337869971362482268262 2°34 17179869184 2°110				
2°21 297152 2°9 1584563250285269751878979906772 2°22 4194344 2°98 633825380114114769748551602688 2°41 1777216 2°109 633825380114114769748551602688 2°25 31554432 2°101 12675696002232940404572205376 2°26 67168864 2°102 5970602408129170699981221564 2°27 14217728 2°133 1611204881825385211973625643088 2°28 26485456 2°109 4965641920739334964789452572832 2°30 1973741824 2°106 480564192073933496478945572932 2°31 2147483648 2°107 16225927662921336391578810288128 2°32 2294967296 2°108 32451855565642672678162022267266 2°34 17778661184 2°110 129807421463376699132656612662236202409 2°35 34359738366 2°111 25961484292674188416523481646104848 2°36 8714470736 2°112 2596148429267438814652481265481264104848 2°39 548757906944 2°114 2976914734341931165112463441975816366192494894 2°39 54875780644416				
2°22 4194364 2°98 3169125698757573861344 2°23 838668 2°99 633825361411476978351602688 2°24 16777216 2°109 12765968022822940149670205376 2°25 3554432 2°101 25336120084564388029229401496702059376 2°26 6718864 2°102 57676602400912917665986812821504 2°27 12417728 2°103 10141264681825835111965543698 2°28 266435456 2°104 20222406063651670423947251286016 2°29 56870912 2°106 81129638414606661695789905144064 2°31 127483648 2°107 1022952762823133639182028128 2°32 4294907296 2°188 32451855368426726783156020576256 2°33 8599345592 2°109 649937189731655435566401152512 2°34 17179869184 2°110 129807431854576600713624081288128 2°35 435597388063 2°111 51922288853482762833694632922006 2°36 4374786736 2°112 51922886461415512 2°373 859178690377776 2°10 159807477765				
2°23 838668 2°99 633825360114114708743815026088 2°24 1°777216 2°100 1267566060238229440197526757 2°25 33554432 2°101 2535381206456458882993406410752 2°26 6°108864 2°102 50766060282294091217562643088 2°27 134217728 2°103 10141248881825835211973625643088 2°28 268435456 2°104 202824496963651679439861466 2°30 1073741824 2°106 81125638414666661879879806144664 2°31 2147483648 2°107 162259276829213363391578010288128 2°32 2294967296 2°108 3245188555584267598965144664 2°34 17179869184 2°110 12988741463376691156028756256 2°34 17179869184 2°110 1298874146337669136246192624 2°35 34359738368 2°111 2596148429267413814265248164616048 2°36 6871947676 2°112 519222968853482726349639323760768 2°37 137438953472 2°113 1038459371769665525766999268440192 2°38 2748779666944 2°114				
2°25 33554432 2°191 5275662468882993465410752 2°26 1678864 2°192 5076662409912917688618281594 2°8 268435456 2°194 40282499693515749751286916 2°29 536879912 2°195 4656481929739334884789450257282 2°39 1973741824 2°196 81129639444666661879879334846464 2°31 2147483648 2°197 16225927682921336339157801288128 2°32 429467296 2°198 3245186555654267289895144064 2°31 2147483648 2°197 16225927682921336339157801288128 2°32 429467296 2°198 6498371073168534535631204115561255 2°34 17179669184 2°119 19282674128145024524614618048 2°35 34359738388 2°111 59522688553482759496329229906 2°37 13748953472 2°113 18384593717966652575669992463639229696 2°38 274877966044 2°113 183846937416756652576699924693292996 2°38 274877969644 2°115 41538374681827461192853168914 2°39 594755813888 2°115				
2°6 67108864 2°192 56706024099129176659386812821594 2°77 142117728 2°183 101412488818283517625643088 2°28 268435456 2°184 20282409608351670423947251286016 2°29 556879912 2°196 48156481292739348489452727292 2°191 11747483648 2°196 16125297682921336391578810288128 2°12 4294967296 2°188 324518853658426726783156026976256 2°33 8589934592 2°190 6498371073168534331624813294072 2°34 17179869184 2°110 12988742146337760967132624882396924 2°35 34399738368 2°111 2°59148429267413813614048 2°36 68719476736 2°112 5'912296885534827628534965329226996 2°37 173438953472 2°113 13884374862786218243996337906926 2°39 497575813888 2°114 20769187441393105141219851088084 2°40 1099511627776 2°116 838764973655742656487941267521536 2°41 129922325552 2°117 166153494756143497365744397365774578853496493493444444849 2°43	2^24	16777216	2^100	1267650600228229401496703205376
2°27 134217728 2°194 2028284969693515632943947521886016 2°29 536879012 2°196 405648139207303340847894582572032 2°39 536879012 2°196 41563841468678578095144064 2°31 2147483648 2°196 11269384146867878905114064 2°31 2147483648 2°196 112693841468678311560297576256 2°32 4294967296 2°108 3245185555684267831560296756256 2°33 8589934592 2°109 649637107316553453566312041152512 2°41 1179869184 2°111 259614842926741381426524816410848 2°35 34359738388 2°111 259614842926741381426524816410848 2°36 68193476736 2°112 519229685534382634045329220806 2°37 137438953472 2°113 181834539717669555257660992658444019 2°38 274877906944 2°114 2976918743139141212188531688084 2°39 549755813888 2°115 415238746227218283495927275768 2°40 4099511627776 2°116 836747497365577251536 2°41 219962255555	2^25	33554432	2^101	2535301200456458802993406410752
2°28 26845456 2°105 4965481392873334781286815 2°29 1673741824 2°106 811296388143967333193157881028212 2°31 1274783648 2°107 161252957858291391575810288128 2°32 4294967296 2°108 324518553658426726783156902676256 2°33 8589934592 2°109 4699371073168535636121241152512 2°34 171798691384 2°110 1288074214633766997132624802308904 2°35 34359738368 2°111 25961484292267413814265248164610048 2°36 68719476736 2°112 519229688534827628834965329220096 2°37 137438953472 2°113 1638459377966955257869992584448192 2°38 27487796644 2°114 2076918743735757688 2°39 549755813888 2°115 415383748662786210224399633766788 2°40 1099511627776 2°16 83976493625759268687941267521536 2°41 2199623255552 2°117 166153499473114484112975882535843927 2°42 239804651104 2°118 33236959894225295517555709608068444 2°43 8764		67108864		
2°29 536870912 2°106 815093140847894502572032 2°30 1073741824 2°106 81209384146869578095144064 2°31 2147483648 2°107 16225927682921336339157810288128 2°32 4294967296 2°188 3245185536584281556926767055 2°33 8589934592 2°109 64093710731655345356631241152512 2°34 17179669184 2°110 12980742146337669971326240823050024 2°35 3459738368 2°111 2596148422674181812652248164610048 2°36 68719476736 2°112 5192296858534276285304925220096 2°37 373438953472 2°113 1838459371706965552760992658440192 2°39 294755813888 2°115 415383746807828243976637360768 2°40 1099511627776 2°116 83076749736557242056487941267521536 2°41 199962325552 2°117 16615349947731441197958825364972 2°42 4338846511104 2°118 33323669398946228968225951756970886144 2°43 4759693922208 2°119 664631999782457141975860808144776 2°44 1				
2~39 1073741824 2~106 8112593844146666816957899695144664 2~31 2147483648 2~107 1622592768929133633391578810288128 2~32 4294967296 2~108 324518553658426726783156620976256 2~33 8589344592 2~109 6409371873163256612041155112 2~34 17179865184 2~110 1298074214633766997132624823365024 2~35 34559738368 2~111 25961484292678334664329220996 2~37 137438953472 2~113 16384593717669655527669992658446192 2~38 274877986944 2~114 34596718743413931651412198531688834 2~39 549755813888 2~115 41538374868278621028243976633766768 2~40 1099511627776 2~116 383967699358278617657678 2~42 3389646511104 2~113 333366938462786212828243976633767678 2~42 3398046511104 2~118 33336693846228654875156570886144 2~43 3796093022208 2~119 664613997813873164814127958253504072 2~42 3398046511104 2~118 333369938462472593645918925976769885144				
2-31 2147483648 2-107 162259276829213363391578010288128 2-32 4294967296 2-108 64963716276783156820576256 2-34 17179869184 2-119 129807416353653612041152512 2-35 34359738368 2-111 2596148429267413814265248164610048 2-36 68719476736 2-112 51922968585348276282949696 2-37 137438953472 2-113 10884593717069555257066995658440192 2-38 274877969544 2-114 20769187434139310514121985316880384 2-39 59475581388 2-115 415383746882761282439796123760768 2-40 1099511627776 2-116 83076749736557242056487941267521536 2-41 199602355552 2-117 61653499473144112975882535943972 2-42 4398046511104 2-118 332306999846228968225951765970806144 2-43 75966939252208 2-119 6661399789274344112975882535943972 2-44 17592186044416 2-120 1329227995784915875979364198758708087662880344576 2-45 513843720888832 2-121 656845599156926832455964416522824111213786764 <				
2-22 4.294967296 2.108 3.245185536584267267831156620976256 2-33 8589934592 2.109 6.4093710971633566312041155112 2-34 17179869184 2.110 1.2980742146337669097132624883365024 2-35 34359738368 2.111 2596148429267413814265248164610948 2-36 68719476736 2.112 519229888548248276285304953292200996 2-38 274877906944 2.113 1838459371766955527060992658440192 2-39 549755813888 2.115 41538374866278621028243976633760768 2-40 1099511627776 2.116 383676497356524966487941267521536 2-41 1299923255552 2.117 166153499473114484112975882536043072 2-42 3380465911104 2.118 3332366998462286547576567086144 2-43 8796093022208 2.119 6646199789743731448411275882536043072 2-44 175921186044416 2.120 139227999788062259517566986289344576 2-45 73534778088832 2.121 2.58845599187296831745807614126560889152 2-46 70368744177664 2.122 5316919831396679326983234946488224				
2-23 858934592 2'109 649037167316835453566312641152512 2-34 17179869184 2'111 1298672416826823692624 2-35 34359738368 2'111 2596148429267413814265248164610048 2-36 68719476736 2'112 519229685853487768283696329220096 2-37 137438953472 2'113 19384593777609655257609922658440192 2-38 274877996044 2'114 20769187434139316514121985316880384 2-39 549755818388 2'115 41538374868278621028243979633769768 2-40 1099511627776 2'116 830767497365557242056487941267521536 2-41 1299623255552 2'117 166153499411448411297588235043972 2-42 4398646511164 2'118 3323669894622896825295176597086144 2-43 8796093022208 2'119 65641398796739365199353411472288 2-44 17592186044416 2'120 13292279957849158729380706028044576 2-45 3518437268832 2'121 2656455991560831935366482242756688 2-46 70368744177664 2'122 31931831319663493153224 <t< td=""><td></td><td></td><td></td><td></td></t<>				
2°35 34359738368 2°111 2596148429267413814265248164610048 2°36 68719476736 2°112 519229688427726820920996 2°37 137438953472 2°113 10384593717069655257060992658440192 2°39 549755813888 2°116 830767497365572496114121985316880384 2°40 1099511627776 2°116 8307674973655724266487941267521536 2°41 12199023255552 2°117 166135499743736557324285235943707 2°42 4398046511164 2°118 332306998946228968225951765970886144 2°43 879609302208 2°119 6646139978479364511936336140172288 2°44 17592186944416 2°120 1329227995784915872903807860288344576 2°45 3518472088832 2°121 265845599156981745897614128560689152 2°46 70368744177664 2°122 2558455991569821745897614128660689152 2°47 140737488355328 2°123 16633823966279325983329456482242756688 2°48 281474997619656 2°124 4215644937251868539464892132114 2°49 562949953421311 2°125 4255329586511330734615887492658				649037107316853453566312041152512
2^36 68719476736 2^112 519229685853482762833496329290696 2^37 137438953472 2*114 2087918769655576069992658440192 2*38 274877966944 2*114 208791873415121985316880384 2*39 549755813888 2*115 41538374868278621028243976633760768 2*41 0199511627776 2*116 839676795557242864847941267521536 2*41 1199023255552 2*117 166153499473114484112975882535443072 2*42 4398046511104 2*118 3232899842289842228951755597086144 2*43 8796693022208 2*119 66461399789245793645199353140172288 2*44 17592186644416 2*120 13329277995784915872903807660288344576 2*45 3518437288832 2*121 255845991568817475807614129556689152 2*46 79368744177664 2*122 53169119831395663916152282411213738304 2*47 140737488355328 2*123 16338239662793255865396646991296485513216 2*48 281474976710656 2*124 2126764793255865396646991296485513216 2*49 5524499534471312 2*125 45351392185929321852	2^34	17179869184	2^110	1298074214633706907132624082305024
2^33 274877906944 2^113 1208693717669655257660992658440192 2^38 274877906944 2^114 120769187434139310514121985316880384 2^39 549755813888 2^115 4158374868278821282243976637660768 2^40 1699511627776 2^116 83976749736557242656487941267521536 2^41 219902325552 2^1117 16615349473114484112975882535434072 2^42 4398846511104 2^118 33236699894622886825951765076086144 2^43 879609302208 2^119 6646139982457936451903530140172288 2^44 17592186044416 2^120 1329227995784915872903807660280344576 2^44 17592186044416 2^121 26584599156931745807614120566689152 2^46 79468744177664 2^1212 25169313745807614120566689152 2^47 140737488355328 2^123 10633823966279326683230456482224756608 2^48 281474976710656 2^1214 216567666912964485691296448513216 2^49 562949953421312 2^125 4253529586511730793291825928971026432 2*51 12517998136685248 2^1212 42535295865176	2^35	34359738368	2^111	2596148429267413814265248164610048
2^38 274877966944 2^114 49769187434139310514121985316880384 2^39 549755813888 2^115 415583486278621028243970633760768 2^40 1099511627776 2^116 83076749736557240286487941267521536 2^41 2199023255552 2^111 166153499473114484112975882535043072 2^42 4398846511104 2^118 332306998846228968225951765970886144 2^43 8796093022208 2^119 66461399782457936451993530140172288 2^44 1759218604416 2^120 132927993789188757936841982394766 2^45 35184372088832 2^121 265845599156983174880741120560689152 2^46 70368744177664 2^122 25132914383339663491615228241121378304 2^47 140737488355328 2^123 16033823966279325883230456482244755608 2^48 281474976710656 2^124 212567499354121312 2^125 4253559865117367939713267346158673472622864 2^551 125899906842624 2^126 85070591730234615865843651857942052864 2^551 251693813685248 2^127 17914141874666631733633434713687393715884105728 2^	2^36	68719476736		
2^93 54975813888 2^115 41538374863278621028243970633766768 2^40 1099511627776 2^116 8307674973655724205648794126752136 2^41 2199023255552 2^117 16615349947311448411297588253643072 2^42 4398846511104 2^118 33230699884622199632757565970806144 2^43 8796093022208 2^119 664613997892457936451993530140172288 2^44 1759186044416 2^120 13292279957849158729038076628834576 2^45 35184372088832 2^121 2658455991569831745807614120560689152 2^46 70368744177664 2^122 5316911983139663491615228241121378394 2^47 140737488355328 2^123 10633823966279326883230456482242756668 2^48 281474976710856 2^124 21267479325586531964485513216 2^49 562949953421312 2^125 42535295865117387932921825928971026432 2^59 1125899906842624 2^1216 850759373730234615865846511587942972672844 2^51 2251799813685248 2^127 1701411834604692317316873371888105728 2^52 45035996273764969 2^128				
2-48 1099511627776 2-116 83076749736557242056487941267521536 2-41 2199023255552 2-117 16615349947311448411297582535843072 2-42 4398046511104 2-118 332306998946228968225951765070806144 2-43 8796093022208 2-119 664613997892457936451993530140172288 2-44 17592186044116 2-120 13292799578491587299887066289344576 2-45 35184372088832 2-121 2658455991569831745807614120560689152 2-46 79368744177664 2-122 5316911983139663491615228241121378304 2-47 140737488355328 2-123 106338239662793258851321666822424756608 2-48 281474976710656 2-124 21267647932558653966460912964485513216 2-49 562949953423112 2-125 4253259865117397393291282598870266432 2-51 2251799813685248 2-127 170141183460469231731687393715884105728 2-52 25252 4593599627370496 2-128 3462336692093846443374607431768211456 2-53 36028797018863968 2-131 2752258935365767707706996859454145691648 2-55 360287970188963968				
2^41 219902325552 2°117 1661534994733144848112975882535043072 2°42 4398046511104 2°118 3323069989462289617650709086144 2°43 879609322208 2°119 664613997882457936451903530140172288 2°44 17592186044416 2°120 1329227995784915872903807060280344576 2°45 53184372088832 2°121 256845599156983174588704126560689152 2°46 70368744177664 2°122 5316911983139663491615228241121378304 2°47 140737488355328 2°123 10633823966279326983230456482242756608 2°48 281474976710656 2°124 21257647932558653966460912964485513216 2°49 562949953421312 2°125 42555229886117307932921825928971026432 2°50 1125899906842624 2°126 859059917303246158658346518579426952864 2°51 2251799813685248 2°127 170141183460460231731687303715884105728 2°52 4593599627370496 2°128 340282366920938463463374607431768211456 2°53 9007199254740992 2°129 68064733841876926926749214863536422912 2°54 18014398599481984				
2^42 439846511104 2^118 33236698846228968225951765070866144 2^43 8796093022208 2^119 664613997892457936451903530140172288 2^44 17592186044416 2^120 1329227995784915872993887060280344576 2^45 35184372088832 2^121 12658455991569831745807614120566689152 2^46 70368744177664 2^122 3516911983139663491615228241121378304 2^47 140737488355328 2^123 10633823966279326983230456482242756608 2^48 281474976710656 2^124 41256743732558653966460912964485513216 2^49 56294995421312 2^125 42535295865117307973291285928971026432 2^50 1125893906842624 2^126 85976991730234615865843651857942052864 2^51 2251799813685248 2^127 1701411834604609231731687303715884105728 2^52 4503599627376496 2^128 340228236692093846346343374607431768211456 2^53 9007199254740992 2^129 680564733841876926926749214863536422912 2^54 180143958594184 2^130 15024578735838549849829720772284824 2^55 36028797018963968 <td></td> <td></td> <td></td> <td></td>				
2^43 8796093022208 2^119 664613997892457936451903530140172288 2^44 17592186044416 2^120 1329227995784915872903807060280344576 2^445 35184372088832 2^121 2658455991569831745807614120560689152 2^46 70368741177664 2^122 5316911983139663491615228241121378304 2^47 140737488355328 2^123 1166382396627932698320456482242756608 2^48 281474976710656 2^124 2126764793255865396640912964485513216 2^49 562949953421312 2^125 425352995865117307932921825928971026432 2^59 112589996842624 2^126 85959951730243615855943651857942052864 2^51 2951799813685248 2^127 170141183460469231731687303715884105728 2^52 4563599627376496 2^128 34028236692693846346337487692413768211456 2^553 9067199254740992 2^129 68654733841876926926749214863536422912 2^554 18014398509481984 2^130 1361129467683753853853498429727072845824 2^55 360287970189639568 2^131 472225893536750770766996859545145691648 2^56 72657				
2^45 35184372088832 2^121 2658455991569831745807614120560689152 2^46 70368744177664 2^122 5316911983139663491615228241121378304 2^47 140737488355328 2^123 196338239662793269832384548822472756608 2^48 281474976716656 2^124 21267647932558653966469912964485513216 2^49 562949953421312 2^125 425352958651173097932921825928971026432 2^56 1125899968842624 2^126 85076591730234615865843651857942652864 2^51 2251799813685248 2^127 170141183460469231731687303715884165728 2^52 4593599627370496 2^128 340282366920938463463374607431768211456 2^53 9007199254740992 2^129 688564733841876926926749214863536422912 2^543 18014398509481984 2^130 13611294676837538538353498429727072845824 2^55 36028797018963968 2^131 2722258935367507707706996859454145691648 2^55 36028797018963968 2^131 2722258935367507707706996859454145691648 2^57 144115188075855872 2^133 10889305741470603803827987437816582766592 2^575 <td></td> <td></td> <td></td> <td>664613997892457936451903530140172288</td>				664613997892457936451903530140172288
2^46 79368744177664 2^122 5316911983139663491615228241121378304 2^47 140737488355328 2^123 10633823966279326983230456482242756608 2^48 281474976710656 2^124 2126764793255865396640912964485513216 2^49 562949953421312 2^125 42535295865117307932291825928971026432 2^56 1125899906842624 2^126 85976991730234615865843651857942052864 2^51 2251799813685248 2^127 17041183460469231731687393715888105728 2^52 4503599627370496 2^128 340282366920938463463374607431768211456 2^53 9007199254740992 2^129 680564733841876926926749214863536422912 2^54 18014398509481984 2^131 2722258935367507707706996859454145691648 2^55 36028797018963968 2^131 2722258935367507707706996859454145691648 2^55 720575594037927936 2^132 5444517870735015415413993718908291383296 2^57 144115188075855872 2^133 14859474470736981547541393938829679874378165682766592 2^58 288230376151711744 2^136 43752447482940061661655974875633165533184	2^44	17592186044416	2^120	1329227995784915872903807060280344576
2^47 140737488355328 2^123 10633823966279326983230456482242756608 2^48 281474976710656 2^124 21167647932558653956640912964485513216 2^49 562249953421312 2^125 42535295865117307932921825928971026432 2^50 1125899906842624 2^126 85070591730234615865843651857942052864 2^51 2251799813685248 2^127 170141183460469231731687393715884105728 2^52 4503599627370496 2^128 340282366920938463463374607431768211456 2^53 9007199254740992 2^129 680564733841876926926749214863536422912 2^54 18014398509481984 2^130 1361129467683753853853498429727072845824 2^55 36028797018963968 2^131 2722589935367567770706996859454145691648 2^55 72657594037927936 2^132 5444517870735015415413993718908291383296 2^57 144115188075855872 2^133 10889035741470030830827987437816582766592 2^58 288230376151711744 2^134 2177807148294061661655974875633165533184 2^59 576406752304423488 2^135 43556142965880123323311949751266331066368 <t< td=""><td>2^45</td><td>35184372088832</td><td></td><td>2658455991569831745807614120560689152</td></t<>	2^45	35184372088832		2658455991569831745807614120560689152
2^48 281474976710656 2^124 21267647932558653966460912964485513216 2^49 562949953421312 2^125 42535295865117307932921825928971026432 2^50 112589996842624 2^126 85970591730234615865843651857942052864 2^51 2251799813685248 2^127 170141183460469231731687303715884105728 2^52 4563599627370496 2^128 340282366920938463463374607431768211456 2^53 9007199254740992 2^129 680564733841876926926749214863536422912 2^54 18014398509481984 2^130 1361129467683753853853498429727072845824 2^55 36028797018963968 2^131 2722258935367567707706996859454145691648 2^56 72057594037927936 2^132 5444517870735015415413993718908291383296 2^57 144115188075855872 2^133 10889935741470930830827987437816582766592 2^58 288230376151711744 2^134 21778071482940061661659748756331665331 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 1152921504606846976 2^136 87112285931760246646623899502532662132736				
2^49 562949953421312 2^125 42535295865117307932921825928971026432 2^50 1125899966842624 2^126 85076591730234615865843651857942052864 2^51 2251799813685248 2^127 170141183460469231731687303715884105728 2^52 4503599627370496 2^128 340282366920938463463374607431768211456 2^53 9007199254740992 2^129 680564733841876926926749214863536422912 2^54 18014398509481984 2^130 1361129467683753853853498429727072845824 2^55 36028797018963968 2^131 2722258935367507707706996859454145691648 2^55 72057594037927936 2^132 5444517870735015415413993718908291383296 2^57 144115188075855872 2^133 10889035741470030830827987437816582766592 2^58 288230376151711744 2^134 21778071482940061661655974875633165533184 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 1152921504606846976 2^136 87112285931760246646623899502532662132736 2^61 2305843009213693952 2^137 1742245718635204932932477990050653242665472				
2^50 1125899906842624 2^126 85070591730234615865843651857942052864 2^51 2251799813685248 2^127 170141183460469231731687393715884105728 2^52 4503599627379496 2^128 340282366929929384634633746074317682114156 2^53 9007199254740992 2^129 680564733841876926926749214863536422912 2^54 18014398569481984 2^130 1361129467683753853853498429727072845824 2^55 36028797018963968 2^131 2722258935367507707706996859454145691648 2^56 72657594037927936 2^132 5444517870735015415413993718908291383296 2^57 144115188075855872 2^133 10889035741470030830827987437816582766592 2^58 288230376151711744 2^134 21778071482940061661655974875633165533184 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 1152921504606846976 2^136 87112285931760246646623899502532662132736 2^61 2305843009213693952 2^137 174224571863520493293247799005065324265472 2^62 4611686018427387904 2^138 34844914372704998658649559801191366485399444				
2^51 2251799813685248 2^127 170141183460469231731687303715884105728 2^52 4503599627370496 2^128 340282366920938463463374607431768211456 2^53 9007199254740992 2^129 680564733841876926926749214863536422912 2^54 18014398509481984 2^130 1361129467683753853853498429727072845824 2^55 36028797018963968 2^131 2722258935367507707706996859454145691648 2^56 72057594037927936 2^132 5444517870735015415413993718908291383296 2^57 144115188075855872 2^133 10889035741470030830827987437816582766592 2^58 288230376151711744 2^134 21778071482940061661655974875633165533184 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 1152921504606846976 2^136 87112285931760246646623899502532662132736 2^62 4611686018427387904 2^138 34844914372704098658649598010130648530944 2^63 9223372036854775808 2^139 696898287454081973172991196020261297061888 2^64 18466744073709551616 2^140 13937965749081633498145982392040522594123776 </td <td></td> <td></td> <td></td> <td></td>				
2^52 4503599627370496 2^128 340282366920938463463374607431768211456 2^53 9007199254740992 2^129 686564733841876926926749214863536422912 2^54 18014398509481984 2^130 1361129467683753853853498429727072845824 2^55 36028797018963968 2^131 2722258935367507707706996859454145691648 2^56 72057594037927936 2^132 5444517870735015415413993718908291383296 2^57 144115188075855872 2^133 10889035741470030830827987437816583165533184 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 11529215046068846976 2^136 87112285931760246646623889502532662132736 2^61 2305843009213693952 2^137 174224571863520493293247799005065324265472 2^62 4611686018427387904 2^138 3484491437270409865864955980113130648530944 2^63 9223372036854775808 2^139 6968982875408173172991196020261297061888 2^64 18446744073709551616 2^140 1393796574908163946345982392040522594123776 2^65 36893488147419103232 2^141 27875931498163278926919647840810				
2^54 18014398509481984 2^130 1361129467683753853853498429727072845824 2^55 36028797018963968 2^131 2722258935367507707706996859454145691648 2^56 72057594037927936 2^132 5444517870735015415413993718908291383296 2^57 144115188075855872 2^133 10889035741470030830827987437816582766592 2^58 288230376151711744 2^134 21778071482940061661655974875633165533184 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 1152921504606846976 2^136 87112285931760246646623899502532662132736 2^61 2305843009213693952 2^137 174224571863520493293247799005065324265472 2^62 4611686018427387904 2^138 3484491437727040986586495598010130648530944 2^63 9223372036854775808 2^139 696898287454081973172991196020261297061888 2^64 18446744073709551616 2^140 1393796574908163946345982392040522594123776 2^65 36893488147419103232 2^141 2787593149816327892691964784081045188247552 2^66 73786976294838206464 2^142 257518629963265311570767859136324180755799200808 2^68 2951479051793				
2^55 36028797018963968 2^131 2722258935367507707706996859454145691648 2^56 72057594037927936 2^132 54445178707350154154131993718908291383296 2^57 144115188075855872 2^133 10889035741470030830827987437816582766592 2^58 288230376151711744 2^134 217780771482940061661655974875633165533184 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 1152921504606846976 2^136 87112285931760246646623899502532662132736 2^61 2305843009213693952 2^137 174224571863520493293247799005065324265472 2^62 4611686018427387904 2^138 348449143727040986586495598010130648530944 2^63 9223372036854775808 2^139 696898287454081973172991196020261297061888 2^64 18446744073709551616 2^140 1393796574908163946345982392040522594123776 2^65 36893488147419103232 2^141 27875931498163278926919647884081045188247552 2^66 73786976294838206464 2^142 5575186299632655785383929568162099376495104 2^67 147573952589676412928 2^143 11150372599265311570767859136324180752990208 2^68 29514790	2^53	9007199254740992	2^129	680564733841876926926749214863536422912
2^56 72057594037927936 2^132 5444517870735015415413993718908291383296 2^57 144115188075855872 2^133 10889035741470030830827987437816582766592 2^58 288230376151711744 2^134 217780714829400616616555774875633165533184 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 1152921504606846976 2^136 87112285931760246646623899502532662132736 2^61 2305843009213693952 2^137 174224571863520493293247799005065324265472 2^62 4611686018427387904 2^138 348449143727040986586495598010130648530944 2^63 9223372036854775808 2^139 696898287454081973172991196020261297061888 2^64 18446744073709551616 2^140 1393796574998163946345982392040522594123776 2^65 36893488147419103232 2^141 2787593149816327892691964784081045188247552 2^66 73786976294838206464 2^142 5575186299632655785383929568162099376495104 2^67 147573952589676412928 2^143 11150372599265311570767859136324180752990208 2^68 295147905179352825856 2^144 22300745198530623141535718272648361505980416 2^69 59	2^54	18014398509481984	2^130	
2^57 144115188075855872 2^133 10889035741470030830827987437816582766592 2^58 288230376151711744 2^134 21778071482940061661655974875633165533184 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 1152921504606846976 2^136 87112285931760246646623899502532662132736 2^61 2305843009213693952 2^137 1742245718635204932293247799005065324265472 2^62 4611686018427387904 2^138 348449143727040986586495598010130648530944 2^63 9223372036854775808 2^139 696898287454081973172991196020261297061888 2^64 18446744073709551616 2^140 1393796574908163946345982392040522594123776 2^65 36893488147419103232 2^141 2787593149816327892691964784081045188247552 2^66 73786976294838206464 2^142 5575186299632655785383929568162090376495104 2^67 147573952589676412928 2^143 1150372599265311570767859136324180752990208 2^68 295147905179352825856 2^144 22300745198530623141535718272648361505980416 2^69 590295810358705651712 2^145 44601490397061246283071436545296723011960832 2^70				
2^58 288230376151711744 2^134 21778071482940061661655974875633165533184 2^59 576460752303423488 2^135 43556142965880123323311949751266331066368 2^60 1152921504606846976 2^136 87112285931760246646623899502532662132736 2^61 2305843009213693952 2^137 174224571863520493293247799005065324265472 2^62 4611686018427387904 2^138 348449143727040986586495598010130648530944 2^63 9223372036854775808 2^139 696898287454081973172991196020261297061888 2^64 18446744073709551616 2^140 1393796574908163946345982392040522594123776 2^65 36893488147419103232 2^141 2787593149816327892691964784081045188247552 2^66 73786976294838206464 2^142 5575186299632655785383929568162090376495104 2^67 147573952589676412928 2^143 11150372599265311570767859136324180752990208 2^68 295147905179352825856 2^144 22300745198530623141535718272648361505980416 2^69 5902958103587055651712 2^145 44601490397061246283071436545296723011960832 2^70 1180591620717411303424 2^146 89202980794122492566142873090593446023921664 2^71 <td></td> <td></td> <td></td> <td></td>				
2^595764607523034234882^135435561429658801233233119497512663310663682^6011529215046068469762^136871122859317602466466238995025326621327362^6123058430092136939522^1371742245718635204932932477990050653242654722^6246116860184273879042^1383484491437270409865864955980101306485309442^6392233720368547758082^1396968982874540819731729911960202612970618882^64184467440737095516162^14013937965749081639463459823920405225941237762^65368934881474191032322^14127875931498163278926919647840810451882475522^66737869762948382064642^14255751862996326557853839295681620903764951042^671475739525896764129282^143111503725992653115707678591363241807529902082^682951479051793528258562^144223007451985306231415357182726483615059804162^695902958103587056517122^145446014903970612462830714365452967230119608322^7011805916207174113034242^146892029807941224925661428730905934460239216642^7123611832414348226068482^1471784059615882449851322857461811868920478433282^7247223664828696452136962^1483568119231764899702645714923623737840956866562^7394447329657392904273922^1497136238463529799405291429847247475568191373312				
2^6011529215046068469762^136871122859317602466466238995025326621327362^6123058430092136939522^1371742245718635204932932477990050653242654722^6246116860184273879042^1383484491437270409865864955980101306485309442^6392233720368547758082^1396968982874540819731729911960202612970618882^64184467440737095516162^14013937965749081639463459823920405225941237762^65368934881474191032322^14127875931498163278926919647840810451882475522^66737869762948382064642^14255751862996326557853839295681620903764951042^671475739525896764129282^143111503725992653115707678591363241807529902082^682951479051793528258562^144223007451985306231415357182726483615059804162^695902958103587056517122^145446014903970612462830714365452967230119608322^7011805916207174113034242^146892029807941224925661428730905934460239216642^7123611832414348226068482^1471784059615882449851322857461811868920478433282^7247223664828696452136962^148356811923176489970264571492362373784095686562^7394447329657392904273922^149713623846352979940529142984724747568191373312				
2^61 2305843009213693952 2^137 174224571863520493293247799005065324265472 2^62 4611686018427387904 2^138 348449143727040986586495598010130648530944 2^63 9223372036854775808 2^139 696898287454081973172991196020261297061888 2^64 18446744073709551616 2^140 1393796574908163946345982392040522594123776 2^65 36893488147419103232 2^141 2787593149816327892691964784081045188247552 2^66 73786976294838206464 2^142 5575186299632655785383929568162090376495104 2^67 147573952589676412928 2^143 11150372599265311570767859136324180752990208 2^68 295147905179352825856 2^144 22300745198530623141535718272648361505980416 2^69 590295810358705651712 2^145 44601490397061246283071436545296723011960832 2^70 1180591620717411303424 2^146 89202980794122492566142873090593446023921664 2^71 2361183241434822606848 2^147 178405961588244985132285746181186892047843328 2^72 4722366482869645213696 2^148 35681192317648997026457149236237378409568656 2^73 9444732965739290427392 2^149 713623846352979940529142984724747568191373312 <td></td> <td></td> <td></td> <td></td>				
2^6246116860184273879042^1383484491437270409865864955980101306485309442^6392233720368547758082^1396968982874540819731729911960202612970618882^64184467440737095516162^14013937965749081639463459823920405225941237762^65368934881474191032322^14127875931498163278926919647840810451882475522^66737869762948382064642^14255751862996326557853839295681620903764951042^671475739525896764129282^143111503725992653115707678591363241807529902082^682951479051793528258562^144223007451985306231415357182726483615059804162^695902958103587056517122^145446014903970612462830714365452967230119608322^7011805916207174113034242^146892029807941224925661428730905934460239216642^7123611832414348226068482^1471784059615882449851322857461811868920478433282^7247223664828696452136962^1483568119231764899702645714923623737840956866562^7394447329657392904273922^149713623846352979940529142984724747568191373312				
2^64184467440737095516162^14013937965749081639463459823920405225941237762^65368934881474191032322^14127875931498163278926919647840810451882475522^66737869762948382064642^14255751862996326557853839295681620903764951042^671475739525896764129282^143111503725992653115707678591363241807529902082^682951479051793528258562^144223007451985306231415357182726483615059804162^695902958103587056517122^145446014903970612462830714365452967230119608322^7011805916207174113034242^146892029807941224925661428730905934460239216642^7123611832414348226068482^1471784059615882449851322857461811868920478433282^7247223664828696452136962^1483568119231764899702645714923623737840956866562^7394447329657392904273922^149713623846352979940529142984724747568191373312				
2^65368934881474191032322^14127875931498163278926919647840810451882475522^66737869762948382064642^14255751862996326557853839295681620903764951042^671475739525896764129282^143111503725992653115707678591363241807529902082^682951479051793528258562^144223007451985306231415357182726483615059804162^695902958103587056517122^145446014903970612462830714365452967230119608322^7011805916207174113034242^146892029807941224925661428730905934460239216642^7123611832414348226068482^1471784059615882449851322857461811868920478433282^7247223664828696452136962^1483568119231764899702645714923623737840956866562^7394447329657392904273922^149713623846352979940529142984724747568191373312	2^63	9223372036854775808	2^139	696898287454081973172991196020261297061888
2^66 73786976294838206464 2^142 5575186299632655785383929568162090376495104 2^67 147573952589676412928 2^143 11150372599265311570767859136324180752990208 2^68 295147905179352825856 2^144 22300745198530623141535718272648361505980416 2^69 590295810358705651712 2^145 44601490397061246283071436545296723011960832 2^70 1180591620717411303424 2^146 89202980794122492566142873090593446023921664 2^71 2361183241434822606848 2^147 178405961588244985132285746181186892047843328 2^72 4722366482869645213696 2^148 356811923176489970264571492362373784095686656 2^73 9444732965739290427392 2^149 713623846352979940529142984724747568191373312	2^64	18446744073709551616		
2^67 147573952589676412928 2^143 11150372599265311570767859136324180752990208 2^68 295147905179352825856 2^144 22300745198530623141535718272648361505980416 2^69 590295810358705651712 2^145 44601490397061246283071436545296723011960832 2^70 1180591620717411303424 2^146 89202980794122492566142873090593446023921664 2^71 2361183241434822606848 2^147 178405961588244985132285746181186892047843328 2^72 4722366482869645213696 2^148 356811923176489970264571492362373784095686656 2^73 9444732965739290427392 2^149 713623846352979940529142984724747568191373312				
2^68 295147905179352825856 2^144 22300745198530623141535718272648361505980416 2^69 590295810358705651712 2^145 44601490397061246283071436545296723011960832 2^70 1180591620717411303424 2^146 89202980794122492566142873090593446023921664 2^71 2361183241434822606848 2^147 178405961588244985132285746181186892047843328 2^72 4722366482869645213696 2^148 356811923176489970264571492362373784095686656 2^73 9444732965739290427392 2^149 713623846352979940529142984724747568191373312				
2^69 590295810358705651712 2^145 44601490397061246283071436545296723011960832 2^70 1180591620717411303424 2^146 89202980794122492566142873090593446023921664 2^71 2361183241434822606848 2^147 178405961588244985132285746181186892047843328 2^72 4722366482869645213696 2^148 356811923176489970264571492362373784095686656 2^73 9444732965739290427392 2^149 713623846352979940529142984724747568191373312				
2^70 1180591620717411303424 2^146 89202980794122492566142873090593446023921664 2^71 2361183241434822606848 2^147 178405961588244985132285746181186892047843328 2^72 4722366482869645213696 2^148 356811923176489970264571492362373784095686656 2^73 9444732965739290427392 2^149 713623846352979940529142984724747568191373312				
2^71 2361183241434822606848 2^147 178405961588244985132285746181186892047843328 2^72 4722366482869645213696 2^148 356811923176489970264571492362373784095686656 2^73 9444732965739290427392 2^149 713623846352979940529142984724747568191373312				
2^72 4722366482869645213696 2^148 356811923176489970264571492362373784095686656 2^73 9444732965739290427392 2^149 713623846352979940529142984724747568191373312				
2^73 9444732965739290427392 2^149 713623846352979940529142984724747568191373312				
2^150 1427247692705959881058285969449495136382746624	2^73	9444732965739290427392		
			2^150	1427247692705959881058285969449495136382746624

First 30 Rows of Pascal Triangle

1																				
1	1																			
1	2	1																		
1	3	3	1																	
1	4	6	4	1																
1	5	10	10	5	1															
1	6	15	20	15	6	1														
1	7	21	35	35	21	7	1													
1	8	28	56	70	56	28	8	1												
1	9	36	84	126	126	84	36	9	1											
1	10	45	120	210	252	210	120	45	10	1										
1	11	55	165	330	462	462	330	165	55	11	1									
1	12	66	220	495	792	924	792	495	220	66	12	1								
1	13	78	286	715	1287	1716	1716	1287	715	286	78	13	1							
1	14	91	364	1001	2002	3003	3432	3003	2002	1001	364	91	14	1						
1	15	105	455	1365	3003	5005	6435	6435	5005	3003	1365	455	105	15	1					
1	16	120	560	1820	4368	8008	11440	12870	11440	8008	4368	1820	560	120	16	1				
1	17	136	680	2380	6188	12376	19448	24310	24310	19448	12376	6188	2380	680	136	17	1			
1	18	153	816	3060	8568	18564	31824	43758	48620	43758	31824	18564	8568	3060	816	153	18	1		
1	19	171	969	3876	11628	27132	50388	75582	92378	92378	75582	50388	27132	11628	3876	969	171	19	1	
1	20	190	1140	4845	15504	38760	77520	125970	167960	184756	167960	125970	77520	38760	15504	4845	1140	190	20	1
1	21	210	1330	5985	20349	54264	116280	203490	293930	352716	352716	293930	203490	116280	54264	20349	5985	1330	210	21
1																				
1	22	231	1540	7315	26334	74613	170544	319770	497420	646646	705432	646646	497420	319770	170544	74613	26334	7315	1540	231
22	1																			
1	23	253	1771	8855	33649	100947	245157	490314	817190	1144066	1352078	1352078	1144066	817190	490314	245157	100947	33649	8855	1771
253	23	1																		
1	24	276	2024	10626	42504	134596	346104	735471	1307504	1961256	2496144	2704156	2496144	1961256	1307504	735471	346104	134596	42504	10626
2024	276	24	1																	
1	25	300	2300	12650	53130	177100	480700	1081575	2042975	3268760	4457400	5200300	5200300	4457400	3268760	2042975	1081575	480700	177100	53130
12650	2300	300	25	1																
1	26	325	2600	14950	65780	230230	657800	1562275	3124550	5311735	7726160	9657700	10400600	9657700	7726160	5311735	3124550	1562275	657800	230230
65780	14950	2600	325	26	1															
1	27	351	2925	17550	80730	296010	888030	2220075	4686825	8436285	13037895	17383860	20058300	20058300	17383860	13037895	8436285	4686825	2220075	888030
296010	80730	17550	2925	351	27	1														
1	28	378	3276	20475	98280	376740	1184040	3108105	6906900	13123110	21474180	30421755	37442160	40116600	37442160	30421755	21474180	13123110	6906900	
3108105	1184040	376740	98280	20475	3276	378	28	1												
1	29	406	3654	23751	118755	475020	1560780	4292145	10015005	20030010	34597290	51895935	67863915	77558760	77558760	67863915	51895935	34597290	20030010	
10015005	4292145	1560780	475020	118755	23751	3654	406	29	1											
1	30	435	4060	27405	142506	593775	2035800	5852925	14307150	30045015	54627									

1716 1716 1287 715 3432 3003 6435 6435 8008 11440 12870 11440 8008 6188 12376 19448 24310 24310 19448 12376 8568 18564 31824 43758 48620 43758 31824 18564 3876 11628 27132 50388 75582 92378 92378 75582 50388 27132 11628 3876 969 1140 4845 15504 38760 77520 125970 167960 184756 167960 125970 77520 38760 15504 4845 1140 190

Código ASCII

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	32	20	(space)	64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	Α	97	61	а
2	2	STX	34	22	11	66	42	В	98	62	b
3	3	ETX	35	23	#	67	43	С	99	63	С
4	4	EOT	36	24	\$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	Ε	101	65	е
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	1	71	47	G	103	67	g
8	8	BS	40	28	(72	48	Н	104	68	h
9	9	TAB	41	29)	73	49	1	105	69	i
10	Α	LF	42	2A	*	74	4A	J	106	6A	j
11	В	VT	43	2B	+	75	4B	K	107	6B	k
12	С	FF	44	2C	,	76	4C	L	108	6C	1
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	Ε	SO	46	2E		78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	О	111	6F	0
16	10	DLE	48	30	0	80	50	Р	112	70	р
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	S
20	14	DC4	52	34	4	84	54	Т	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	V
23	17	ETB	55	37	7	87	57	W	119	77	W
24	18	CAN	56	38	8	88	58	Χ	120	78	X
25	19	EM	57	39	9	89	59	Υ	121	79	У
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	Z
27	1B	ESC	59	3B	;	91	5B	[123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	
29	1D	GS	61	3D	=	93	5D]	125	7D	}
30	1E	RS	62	3E	>	94	5E	٨	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	?

Laws and facts

Sum of squares

An integer greater than one can be written as a sum of two squares if and only if its prime decomposition contains no prime congruent to 3 (mod 4) raised to an odd power.

<u>Goldbach's conjecture</u>

Every even integer greater than 2 can be expressed as the sum of two primes.

Is Divisible by prime

Given a number M, we want to know if it is divisible by a prime P.

- 1. Find the smallest K for this condition (KP + 1) MOD 10.
- 2. N = (KP + 1) / 10
- 3. Multiply the last digit of M times N and add the result to M without the last digit. (You might use (N-P) as well for the multiplication instead of N, note that it would be a negative number).
- 4. Repeat until M is short enough.

$$\sum_{i=0}^{a} {a \choose i} {b \choose i} = {a+b \choose a}$$
 when $a \le b$

- **Euler's formula** states that if a finite, <u>connected</u>, planar graph is drawn in the plane without any edge intersections, and v is the number of vertices, e is the number of edges and f is the number of faces (regions bounded by edges, including the outer, infinitely large region), then v e + f = 2
- Inclusion-exclusion

$$\left| \bigcup_{i=1}^n A_i \right| = \sum_{i=1}^n |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \cdots$$

$$\cdots + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n-1} |A_1 \cap \cdots \cap A_n|.$$

•