Alguns critérios de divisibilidade

Divisibilidade por 2
Um número é divisível por 2 se ele é par, ou seja, termina em 0, 2, 4, 6 ou 8.
Exemplos: O número 563**4** é divisível por 2, pois o seu último algarismo é 4, mas 13**5** não é divisível por 2, pois é um número terminado com o algarismo 5 que não é par.

Divisibilidade por 3
Um número é divisível por 3 se a soma de seus algarismos é divisível por 3.
Exemplos: 18 é divisível por 3 pois 1+8=9 que é divisível por 3, 576 é divisível por 3 pois: 5+7+6=18 que é divisível por 3, mas 134 não é divisível por 3, pois 1+3+4=8 que não é divisível por 3.

Divisibilidade por 4
Um número é divisível por 4 se o número formado pelos seus dois últimos algarismos é divisível por 4.
Exemplos: 43**12** é divisível por 4, pois 12 é divisível por 4, mas 16**35** não é divisível por 4 pois 35 não é divisível por 4.

Divisibilidade por 5
Um número é divisível por 5 se o seu último algarismo é 0 (zero) ou 5.
Exemplos: 7**5** é divisível por 5 pois termina com o algarismo 5, mas 10**7** não é divisível por 5 pois o seu último algarismo não é 0 (zero) nem 5.

Divisibilidade por 6
Um número é divisível por 6 se é par e a soma de seus algarismos é divisível por 3.
Exemplos: 756 é divisível por 6, pois 756 é par e a soma de seus algarismos: 7+5+6=18 é divisível por 3, 527 não é divisível por 6, pois não é par e 872 é par mas não é divisível por 6 pois a soma de seus algarismos: 8+7+2=17 não é divisível por 3.

Divisibilidade por 7
Um número é divisível por 7 se o dobro do último algarismo, subtraído do número sem o último algarismo, resultar um número divisível por 7. Se o número obtido ainda for grande, repete-se o processo até que se possa verificar a divisão por 7.
Exemplo: 165928 é divisível por 7 pois:

16592	Número sem o último algarismo
-16	Dobro de 8 (último algarismo)
16576	Diferença

Repete-se o processo com este último número.

1657	Número sem o último algarismo
-12	Dobro de 6 (último algarismo)
1645	Diferença

Repete-se o processo com este último número.

164	Número sem o último algarismo
-10	Dobro de 5 (último algarismo)
154	Diferença

Repete-se o processo com este último número.

15	Número sem o último algarismo
-8	Dobro de 4 (último algarismo)
7	Diferença

A diferença é divisível por 7, logo o número dado inicialmente também é divisível por 7.

Exemplo: 4261 não é divisível por 7, pois:

426	Número sem o último algarismo
-2	Dobro do último algarismo
424	Diferença

Repete-se o processo com este último número.

42	Número sem o último algarismo
-8	Dobro do último algarismo
34	Diferença

A última diferença é 34 que não é divisível por 7, logo o número 4261 dado inicialmente não é divisível por 7.

Divisibilidade por 8

Um número é divisível por 8 se o número formado pelos seus três últimos algarismos é divisível por 8.

Exemplos: 45**128** é divisível por 8 pois 128 dividido por 8 fornece 16, mas 45**321** não é divisível por 8 pois 321 não é divisível por 8.

Divisibilidade por 9

Um número é divisível por 9 se a soma dos seus algarismos é um número divisível por 9.

Exemplos: 1935 é divisível por 9 pois: 1+9+3+5=18 que é divisível por 9, mas 5381 não é divisível por 9 pois: 5+3+8+1=17 que não é divisível por 9.

Divisibilidade por 10

Um número é divisível por 10 se termina com o algarismo 0 (zero).

Exemplos: 542**0** é divisível por 10 pois termina em 0 (zero), mas 634**2** não termina em 0 (zero).

Divisibilidade por 11

Um número é divisível por 11 se a soma dos algarismos de ordem par Sp menos a soma dos algarismos de ordem ímpar Si é um número divisível por 11. Como um caso particular, se Sp-Si=0 ou se Si-Sp=0, então o número é divisível por 11.

Exemplo: 1353 é divisível por 11, pois:

Número	1	3	5	3
Ordem	ímpar	par	ímpar	par

O primeiro e o terceiro algarismos têm ordem impar e a sua soma é: Si=1+5=6, o segundo e o quarto algarismos têm ordem par e a sua soma é: Sp=3+3=6, assim a soma dos algarismos de ordem par Sp é igual à soma dos algarismos de ordem ímpar Si, logo o número é divisível por 11.

Exemplo: 29458 é divisível por 11, pois:

Número	2	9	4	5	8
Ordem	ímpar	par	ímpar	par	ímpar

A soma dos algarismos de ordem ímpar, Si=2+4+8=14, a soma dos algarismos de ordem par, Sp=9+5=14 e como ambas as somas são iguais, o número 29458 é divisível por 11.

Exemplo: 2543 não é divisível por 11, pois:

Número	2	5	4	3
Ordem	**ímpar**	**par**	**ímpar**	**par**

A soma dos algarismos de ordem impar é Si=2+4=6, a soma dos algarismos e ordem par é Sp=5+3=8 e como a diferença Si-Sp não é divisível por 11, o número original também não é divisível por 11.

Exemplo: 65208 é divisível por 11, pois:

Número	6	5	2	0	8

Ordem	ímpar	par	ímpar	par	ímpar

A soma dos algarismos de ordem impar é Si=6+2+8=16, a soma dos algarismos de ordem par é Sp=5+0=5. Como a diferença Si-Sp=11, o número 65208 é divisível por 11

Divisibilidade por 13

Um número é divisível por 13 se o quádruplo (4 vezes) do último algarismo, somado ao número sem o último algarismo, resultar um número divisível por 13. Se o número obtido ainda for grande, repete-se o processo até que se possa verificar a divisão por 13. Este critério é semelhante àquele dado antes para a divisibilidade por 7, apenas que no presente caso utilizamos a soma ao invés de subtração.

Exemplo: 16562 é divisível por 13? Vamos verificar.

1656	Número sem o último algarismo
+8	Quatro vezes o último algarismo
1664	Soma

Repete-se o processo com este último número.

166	Número sem o último algarismo
+16	Quatro vezes o último algarismo
182	Soma

Repete-se o processo com este último número.

18	Número sem o último algarismo
+8	Quatro vezes o último algarismo
26	Soma

Como a última soma é divisível por 13, então o número dado inicialmente também é divisível por 13.

Divisibilidade por 16

Um número é divisível por 16 se o número formado pelos seus quatro últimos algarismos é divisível por 16.

Exemplos: 5**4096** é divisível por 16 pois 4096 dividido por 16 fornece 256, mas 45**321** não é divisível por 16 pois 5321 não é divisível por 16.

Divisibilidade por 17

Um número é divisível por 17 quando o quíntuplo (5 vezes) do último algarismo, subtraído do número que não contém este último algarismo, proporcionar um número

divisível por 17. Se o número obtido ainda for grande, repete-se o processo até que se possa verificar a divisão por 17.

Exemplo: 18598 é divisível por 17 pois:

1859	Número sem o último algarismo
-40	Cinco vezes o último algarismo
1819	Diferença

Repete-se o processo com este último número.

181	Número sem o último algarismo
-45	Cinco vezes o último algarismo
136	Diferença

Repete-se o processo com este último número.

13	Número sem o último algarismo
-30	Cinco vezes o último algarismo
-17	Diferença

A diferença, embora negativa, é divisível por 17, logo o número dado inicialmente também é divisível por 17.

Divisibilidade por 19

Um número é divisível por 19 quando o dobro do último algarismo, somado ao número que não contém este último algarismo, proporcionar um número divisível por 19. Se o número obtido ainda for grande, repete-se o processo até que se possa verificar a divisão por 19.

Exemplo: 165928 é divisível por 19? Vamos verificar.

16592	Número sem o último algarismo
+16	Dobro do último algarismo
16608	Soma

Repete-se o processo com este último número.

1660	Número sem o último algarismo
+16	Dobro do último algarismo
1676	Soma

Repete-se o processo com este último número.

167	Número sem o último algarismo
+12	Dobro do último algarismo
179	Soma

Repete-se o processo com este último número.

17	Número sem o último algarismo
+18	Dobro do último algarismo
35	Soma

Como a última soma não é divisível por 19, então o número dado inicialmente também não é divisível por 19.

Exemplo: 4275 é divisível por 19, pois:

427	Número sem o último algarismo
+10	Dobro do último algarismo
437	Soma

Repete-se o processo com este último número.

43	Número sem o último algarismo
+14	Dobro do último algarismo
57	Soma

Repete-se o processo com este último número.

5	Número sem o último algarismo
+14	Dobro do último algarismo
19	Soma

Como a última Soma é o próprio 19, segue que é divisível por 19, então o número 4275 dado inicialmente é divisível por 19.

Divisibilidade por 23

Um número é divisível por 23 quando o héptuplo (7 vezes) do último algarismo, somado ao número que não contém este último algarismo, proporcionar um número divisível por 23. Se o número obtido ainda for grande, repete-se o processo até que se possa verificar a divisão por 23.

Exemplo: 185909 é divisível por 23? Vamos verificar.

18590	Número sem o último algarismo
+63	Dobro do último algarismo
18653	Soma

Repete-se o processo com este último número.

1865	Número sem o último algarismo
+21	Dobro do último algarismo
1886	Soma

Repete-se o processo com este último número.

188	Número sem o último algarismo
+42	Dobro do último algarismo
230	Soma

Como a última soma é divisível por 23, então o número dado inicialmente também é divisível por 23.

Divisibilidade por 29

Um número é divisível por 29 quando o triplo (3 vezes) do último algarismo, subtraído do número que não contém este último algarismo, proporcionar um número divisível por 29. Se o número obtido ainda for grande, repete-se o processo até que se possa verificar a divisão por 29.

Exemplo: O número 8598 é divisível por 29?

859	Número sem o último algarismo
-24	Dobro do último algarismo
835	Diferença

Repete-se o processo com este último número.

83	Número sem o último algarismo
-15	Dobro do último algarismo
68	Diferença

Repete-se o processo com este último número.

6	Número sem o último algarismo
-24	Dobro do último algarismo

-18	Diferença

A diferença, embora negativa, não é divisível por 29, logo o número dado inicialmente também não é divisível por 29.

Divisibilidade por 31

Um número é divisível por 31 quando o triplo (3 vezes) do último algarismo, somado ao número que não contém este último algarismo, proporcionar um número divisível por 31. Se o número obtido ainda for grande, repete-se o processo até que se possa verificar a divisão por 31.

Exemplo: 8598 é divisível por 31?

859	Número sem o último algarismo
+24	Triplo do último algarismo
883	Soma

Repete-se o processo com este último número.

88	Número sem o último algarismo
+9	Triplo do último algarismo
97	Soma

Repete-se o processo com este último número.

9	Número sem o último algarismo
+21	Triplo do último algarismo
30	Soma

A soma não é divisível por 31, logo o número dado inicialmente também não é divisível por 31.

Divisibilidade por 49

Um número é divisível por 49 quando o quíntuplo (5 vezes) do último algarismo, somado ao número que não contém este último algarismo, proporcionar um número divisível por 49. Se o número obtido ainda for grande, repete-se o processo até que se possa verificar a divisão por 49.

Exemplo: 8598 é divisível por 49?

859	Número sem o último algarismo
+40	Cinco vezes o último algarismo

899	Soma

Repete-se o processo com este último número.

89	Número sem o último algarismo
+45	Cinco vezes o último algarismo
134	Soma

Repete-se o processo com este último número.

13	Número sem o último algarismo
+20	Cinco vezes o último algarismo
33	Soma

A soma não é divisível por 49, logo o número dado inicialmente também não é divisível por 49.

Pontes no grafo

```
int lbl[MAXV];
int low[MAXV];
int parent[MAXV];
int count;
int brs;

void
bridgeR(int v)
{
	int i, w;
	vector<int>& refv = adj[v];
	lbl[v] = count++;
	low[v] = lbl[v];
	for(i = 0; i < refv.size(); ++i)
	{
		w = refv[i];
		if(lbl[w] == -1)
		{
			parent[w] = v;
			bridgeR(w);
			if(low[v] > low[w])
				low[v] = low[w];
			if(low[w] == lbl[w])
			{
				++brs;
				if(v < w)
					bridges.insert(edge(v, w));
				else
					bridges.insert(edge(w, v));
			}
		}
		else if(w != parent[v] && low[v] > lbl[w])
			low[v] = lbl[w];
	}
}


void
all_bridges()
{
```

```
        int v;
        count = brs = 0;

        memset(lbl, -1, (N + 1) * sizeof(int));
        memset(parent, 0, (N + 1) * sizeof(int));
        memset(low, INT_MAX, (N + 1) * sizeof(int));

        for(v = 0; v < N; ++v)
        {
                if(lbl[v] == -1)
                {
                        parent[v] = v;
                        bridgeR(v);
                }
        }
}
```

Pontos de articulação

```
int lbl[MAXV];
int low[MAXV];
int parent[MAXV];
int nchild[MAXV];
bool arts[MAXV];
int count;
int cams;
int root;

int N, R;

void
set_art(int v)
{
	if(!arts[v])
	{
		if(v != root || nchild[v] >= 2)
		{
			arts[v] = true;
			++cams;
			results.insert(namesr[v]);
		}
	}
}

void
bridgeR(int v)
{
	int i, w;
	vector<int>& refv = adj[v];
	lbl[v] = count++;
	low[v] = lbl[v];
	for(i = 0; i < refv.size(); ++i)
	{
		w = refv[i];
		if(lbl[w] == -1)
		{
			parent[w] = v;
			++nchild[v];
			bridgeR(w);
```

```
                        if(low[v] > low[w])
                                low[v] = low[w];

                        if(low[w] >= lbl[v])
                        {
                                set_art(v);
                        }
                }
                else if(w != parent[v] && low[v] > lbl[w])
                        low[v] = lbl[w];
        }
}

void
all_arts()
{
        int v;
        count = cams = 0;

        memset(lbl, -1, N * sizeof(int));
        memset(parent, -1, N * sizeof(int));
        memset(nchild, 0, N * sizeof(int));
        memset(arts, false, N * sizeof(bool));
        memset(low, INT_MAX, N * sizeof(int));

        for(v = 0; v < N; ++v)
        {
                root = v;
                if(lbl[v] == -1)
                {
                        parent[v] = v;
                        bridgeR(v);
                }
        }
}
```

Subsequência crescente máxima

SSCTF-Max-PD (A, n)
1 para m ← 1 até n faça
2 c[m] ← 1
3 para i ← m−1 decrescendo até 1 faça
4 se A[i] ≤ A[m] e c[i]+1 > c[m]
5 então c[m] ← c[i]+1
6 devolva c[1..n]

```
int
main()
{
	int T, N, i, j, m, c = 0, inc, dec, r;

	vector<int> heights;
	vector<int> widths;
	vector<int> cinc;
	vector<int> cdec;

	cin >> T;
	while(T--)
	{
		heights.clear();
		widths.clear();
		cinc.clear();
		cdec.clear();
		cin >> N;
		for(i = 0; i < N; ++i)
		{
			cin >> j;
			heights.push_back(j);
			cinc.push_back(0);
			cdec.push_back(0);
		}
		for(i = 0; i < N; ++i)
		{
			cin >> j;
			widths.push_back(j);
		}
```

```
		inc = dec = 0;
		for(m = 0; m < N; ++m)
		{
			cinc[m] = widths[m];
			if(cinc[m] > inc)
				inc = cinc[m];
			cdec[m] = widths[m];
			if(cdec[m] > dec)
				dec = cdec[m];
			for(i = m - 1; i >= 0; --i)
			{
				if((heights[i] < heights[m]) && ((cinc[i] + widths[m]) > cinc[m])
)
				{
					cinc[m] = cinc[i] + widths[m];
					if(cinc[m] > inc)
						inc = cinc[m];
				}
				if((heights[i] > heights[m]) && ((cdec[i] + widths[m]) > cdec[m
]))
				{
					cdec[m] = cdec[i] + widths[m];
					if(cdec[m] > dec)
						dec = cdec[m];
				}
			}
		}

		++c;

		if(inc >= dec)
			cout << "Case " << c << ". Increasing (" << inc << "). Decreasing ("
<< dec << ")." << endl;
		else
			cout << "Case " << c << ". Decreasing (" << dec << "). Increasing ("
<< inc << ")." << endl;
	}

	return 0;
}
```

Parentização de matrizes

```
#include <climits>
#include <iostream>

using namespace std;

#define MAXN 11

int mat[MAXN][MAXN];
int best[MAXN][MAXN];
int r[MAXN + 1];
int N;

void
print(int i, int j)
{
	if(i == j)
		cout << 'A' << i;
	else
	{
		cout << '(';
		print(i, best[i][j]);
		cout << " x ";
		print(best[i][j] + 1, j);
		cout << ')';
	}
}

int
main()
{
	int i, j, k, t, n, b, l, q, c;

	c = 0;
	while(1)
	{
		cin >> n;
		if(n == 0)
			break;

		N = n;
```

```
            ++c;

            for(i = 1; i <= n; ++i)
                cin >> r[i] >> b;
            r[i] = b;

            for(i = 1; i <= n; ++i)
                mat[i][i] = 0;

            for(l = 2; l <= n; ++l)
            {
                for(i = 1; i <= (n - l + 1); ++i)
                {
                    j = i + l - 1;
                    mat[i][j] = INT_MAX;
                    for(k = i; k <= (j - 1); ++k)
                    {
                        q = mat[i][k] + mat[k + 1][j] + r[i] * r[k + 1] * r[j + 1];
                        if(q < mat[i][j])
                        {
                            mat[i][j] = q;
                            best[i][j] = k;
                        }
                    }
                }
            }

            cout << "Case " << c << ": ";
            print(1, n);
            cout << endl;

        }

        return 0;
}
```

Optimal BST

```
#include <algorithm>
#include <climits>
#include <iostream>

using namespace std;

#define MAXN 251

int mat[MAXN][MAXN];
int acc[MAXN][MAXN];
int f[MAXN];

int
main()
{
	int n, i, j, k, t, l;

	while(cin >> n)
	{
		for(i = 1; i <= n; ++i)
			cin >> f[i];

		for(i = 1; i <= n; ++i)
			acc[i][i] = f[i];

		for(i = 1; i <= n; ++i)
			for(j = i + 1; j <= n; ++j)
				acc[i][j] = acc[i][j - 1] + f[j];

		for(i = 1; i <= n; ++i)
			mat[i][i] = 0;

		for(l = 1; l <= n - 1; ++l)
		{
			for(i = 1; i <= n - l; ++i)
			{
				j = i + l;
				mat[i][j] = mat[i + 1][j] + acc[i + 1][j];
				for(k = i + 1; k < j; ++k)
```

```
                                        mat[i][j] = min(mat[i][j], mat[i][k - 1] + acc[i][k - 1] +
mat[k + 1][j] + acc[k + 1][j]);
                                    mat[i][j] = min(mat[i][j], mat[i][j - 1] + acc[i][j - 1]);
                                }
                        }

                        cout << mat[1][n] << endl;
                }

                return 0;
}
```

Geometria plana (stacking cylinders)

```
#include <climits>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <iostream>

#define MAXP 11

using namespace std;

struct _point
{
	double x;
	double y;
};

typedef struct _point point;

point mat[MAXP][MAXP];

static int
compare(const void* p1, const void* p2)
{
	return ((struct _point*)p1)->x < ((struct _point*)p2)->x;
}

double dist(double x0, double y0, double x1, double y1)
{
	return sqrt(((x1 - x0) * (x1 - x0)) + ((y1 - y0) * (y1 - y0)));
}

double z(double x0, double y0, double x1, double y1, double x2)
{
	double xm = (x0 + x1) / 2.;
	double ym = (y0 + y1) / 2.;
	double dm = dist(xm, ym, x1, y1);
	double dh = 2.;
	double dz = sqrt(dh * dh - dm * dm);
	return sqrt(dz * dz - ((x2 - xm) * (x2 - xm))) + ym;
}
```

```
int main()
{
	int n, i, j;

	while(1)
	{
		cin >> n;
		if(n == 0)
			break;

		for(i = 1; i <= n; ++i)
		{
			cin >> mat[n][i].x;
			mat[n][i].y = 1.;
		}

		mat[n][0].x = (double)INT_MAX;
		qsort(mat[n], n + 1, sizeof(struct _point), compare);

		for(i = (n - 1); i >= 1; --i)
		{
			for(j = 1; j <= i; ++j)
			{
				mat[i][j].x = (mat[n][j].x + mat[n][(n - i) + j].x) / 2.;
				mat[i][j].y = z(mat[i + 1][j].x, mat[i + 1][j].y, mat[i + 1][j + 1].x,
mat[i + 1][j + 1].y, mat[i][j].x);
			}
		}

		printf("%.4f %.4f", mat[1][1].x, mat[1][1].y);
		cout << endl;
	}

	return 0;
}
```

Union Find (componentes conexas)

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <cstring>
#define MAXV 1000001
using namespace std;

struct _head
{
	int comp;
	int qtt;
	struct _head* nexth;
};

typedef struct _head head;

head* heads[MAXV];
head* heads_comp[MAXV];

int main()
{
	int ca, ia, T, F, i, count, nfrom, nto, con_comp;
	char com;
	bool bfrom, bto;

	cin >> T;
	cin >> F;
	while(T--)
	{
		count = con_comp = ca = ia 0;
		for(i = 0; i <= F; ++i) heads[i] = NULL;

		while(scanf("%d", &F) == 0)
		{
			cin >> com >> nfrom >> nto;

			if(com == 'c')
			{
				bfrom = heads[nfrom] != NULL;
				bto = heads[nto] != NULL;
```

```
if(!bfrom && !bto)
{
	head* h = (head*)malloc(sizeof(struct _head));
	h->comp = con_comp++;
	heads_comp[h->comp] = h;
	h->qtt = 2;
	h->nexth = NULL;
	heads[nfrom] = h;
	heads[nto] = h;
}
else if(!bfrom && bto)
{
	head* h = heads[nto];
	while(h->nexth != NULL) h = h->nexth;
	heads[nfrom] = h;
	++h->qtt;
}
else if(bfrom && !bto)
{
	head* h = heads[nfrom];
	while(h->nexth != NULL) h = h->nexth;
	heads[nto] = h;
	++h->qtt;
}
else
{
	head* hfrom = heads[nfrom];
	while(hfrom->nexth != NULL) hfrom = hfrom->nexth;
	head* hto = heads[nto];
	while(hto->nexth != NULL) hto = hto->nexth;
	head* tmp;

	if(hfrom != hto)
	{
		if(hto->qtt < hfrom->qtt)
		{
			tmp = hfrom;
			hfrom = hto;
			hto = tmp;
		}
```

```
                                        hto->qtt += hfrom->qtt;
                                        hfrom->nexth = hto;
                                }
                        }
                }
                else
                {
                        if(heads[nfrom] == NULL || heads[nto] == NULL)
                                ++ia;
                        else
                        {
                                head* hfrom = heads[nfrom];
                                while(hfrom->nexth != NULL) hfrom = hfrom->nexth;
                                head* hto = heads[nto];
                                while(hto->nexth != NULL) hto = hto->nexth;
                                if(hfrom->comp == hto->comp)
                                        ++ca;
                                else
                                        ++ia;
                        }
                }
        }

        cout << ca << ',' << ia << '\n' << endl;

        for(i = 0; i < con_comp; ++i)
                if(heads_comp[i])
                        free(heads_comp[i]);
    }

    return 0;
}
```

Segment Tree

```
void
initialize( int node, int b, int e )
{
	if ( b == e ) H[node] = A[b];
	else
	{
		int m = ( b + e ) / 2;
		initialize( 2 * node, b, m );
		initialize( 2 * node + 1, m + 1, e );
		H[node] = ( mod_exp(B, e - m, P) * H[2 * node] + H[2 * node + 1] ) % P;
	}
}

unsigned long long
hash(int node, int l, int r, int i, int j)
{
	int h, nj, ex;

	if(i > r || j < l)
		return 0;

	if(l >= i && r <= j)
		return vet[node];

	h = (r + l) / 2;

	unsigned long long r0 = hash(2 * node, l, h, i, j);
	unsigned long long r1 = hash(2 * node + 1, h + 1, r, i, j);
	if(r0 != 0)
		r0 = (r0 * Bpow(min(r, j) - h));
	return (r0 + r1) % P;
}

void
insert(int idx, int inl, int inr, int pos, unsigned long long value)
{
	if(pos > inr || pos < inl)
		return;

	if(inl == inr)
```

```
        {
                vet[idx] = value;
        }
        else
        {
                int h = (inl + inr) / 2;
                insert(idx * 2, inl, h, pos, value);
                insert(idx * 2 + 1, h + 1, inr, pos, value);
                vet[idx] = (Bpow(inr - h) * vet[2 * idx] + vet[2 * idx + 1]) % P;
        }
}
```

Modular Exp

```
unsigned long long
mod_exp(unsigned long long b, unsigned long long e, unsigned long long m)
{
	unsigned long long r = 1;
	while(e > 0)
	{
		if(e & 1)
			r = (r * b) % m;
		e >>= 1;
		b = (b * b) % m;
	}

	return r;
}
```

Maximum sum torus (2D maximum sum - PD)

```
#include <algorithm>
#include <cstring>
#include <iostream>

using namespace std;

int
main()
{
	int T, N, i, j, k, l, ksum, csum, m;

	int mat[150][150];
	int acc[150][150];
	cin >> T;
	while(T--)
	{
		cin >> N;
		m = 0;

		for(i = 0; i < N; ++i)
			for(j = 0; j < N; ++j)
			{
				cin >> mat[i][j];
				mat[i][j + N] = mat[i + N][j] = mat[i + N][j + N] = mat[i][j];
			}

		N <<= 1;

		memcpy(acc[0], mat[0], N * sizeof(int));

		for(i = 1; i < N; ++i)
			for(j = 0; j < N; ++j)
				acc[i][j] = mat[i][j] + acc[i - 1][j];

		for(k = 1; k < N / 2 + 1; ++k)
		{
			for(i = 0; i < N - k; ++i)
			{
				for(j = 0; j < N / 2; ++j)
				{
```

```
                                        ksum = 0;
                                        for(l = j; l < j + N / 2; ++l)
                                        {
                                                csum = acc[i + k][l] - acc[i][l];
                                                ksum = (ksum >= 0 ? ksum : 0) + csum;
                                                m = max(ksum, m);
                                        }
                                }
                        }
                }

                cout << m << endl;
        }

        return 0;
}
```

Topological Sorting

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edges
while S is non-empty do
    remove a node n from S
    insert n into L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
if graph has edges then
    return error (graph has at least one cycle)
else
    return L (a topologically sorted order)
```

Algoritmos de LFA

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <list>
#include <stack>
#include <map>
#include <list>
#include <set>
#include <vector>
#include <queue>

#define MAXN 1000000

using namespace std;

class State;

typedef pair< char, State * > transition;

int N;
int realadj[MAXN][2];
bool realfinal[MAXN];
string alphabet;
int Nalpha;
int state_count;

class State
{
public:
	State(int i) : id(i) {}

	void add_transition(char c, State* s)
	{
		transitions.insert(transition(c, s));
	}

	multimap< char, State* > transitions;
	int id;
};
```

```
class Automata
{
public:
	Automata(char a)
	{
		State* s0 = new State(state_count++);
		State* s1 = new State(state_count++);

		s0->add_transition(a, s1);

		states.push_back(s0);
		states.push_back(s1);

		istate = s0;
		fstates.push_back(s1);
	}

	void conc(Automata* b)
	{
		State* s;
		list<State*>::iterator it = fstates.begin();
		for(; it != fstates.end(); ++it)
		{
			s = *it;
			s->add_transition('E', b->istate);
		}
		fstates = b->fstates;
		states.insert(states.end(), b->states.begin(), b->states.end());
		delete b;
	}

	void uni(Automata* b)
	{
		State* s;
		list<State*>::iterator it;

		State* si = new State(state_count++);
		State* sf = new State(state_count++);

		si->add_transition('E', istate);
		si->add_transition('E', b->istate);
```

```
        istate = si;
        states.push_front(si);

        it = fstates.begin();
        for(; it != fstates.end(); ++it)
        {
                s = *it;
                s->add_transition('E', sf);
        }

        it = b->fstates.begin();
        for(; it != b->fstates.end(); ++it)
        {
                s = *it;
                s->add_transition('E', sf);
        }

        states.insert(states.end(), b->states.begin(), b->states.end());

        fstates.clear();
        fstates.push_back(sf);
        states.push_back(sf);

        delete b;
}

void star()
{
        list<State*>::iterator it;
        State* si = new State(state_count++);
        State* sf = new State(state_count++);

        si->add_transition('E', istate);
        si->add_transition('E', sf);

        sf->add_transition('E', istate);

        istate = si;
        states.push_front(si);

        for(it = fstates.begin(); it != fstates.end(); ++it)
```

```
                        (*it)->add_transition('E', sf);

                fstates.clear();
                fstates.push_back(sf);
                states.push_back(sf);
        }

        list<State*> states;
        State* istate;
        list<State*> fstates;
};

void
calculate(char c, stack<Automata*>& operands)
{
        Automata* a;
        Automata* b;

        switch(c)
        {
        case '.':
                b = operands.top();
                operands.pop();
                a = operands.top();
                a->conc(b);
                break;
        case '|':
                b = operands.top();
                operands.pop();
                a = operands.top();
                a->uni(b);
                break;
        case '*':
                a = operands.top();
                a->star();
                break;
        }
}

Automata*
regex2nfa(string regex)
{
```

```
	Automata* a;
	stack<Automata*> operands;
	stack<char> operators;
	int i;
	char c;

	state_count = 0;

	for(i = 0; i < regex.size(); ++i)
	{
		c = regex[i];

		if(c >= 'a' && c <= 'z')
			operands.push(new Automata(c));
		else if(c != ')')
			operators.push(c);
		else
		{
			while((c = operators.top()) != '(')
			{
				operators.pop();
				calculate(c, operands);
			}
			operators.pop();
		}
	}

	a = operands.top();

	return a;
}

void
Eclosure(set<State*>& T, set<State*>& R)
{
	set<State*>::iterator it;
	stack<State*> st;
	State* t;
	State* u;
	multimap< char, State* >::iterator itm;

	for(it = T.begin(); it != T.end(); ++it)
```

```
                st.push(*it);

        R = T;

        while(!st.empty())
        {
                t = st.top();
                st.pop();

                for(itm = t->transitions.find('E'); itm != t->transitions.end() && itm->first ==
'E'; ++itm)
                {
                        u = itm->second;
                        if(R.find(u) == R.end())
                        {
                                R.insert(u);
                                st.push(u);
                        }
                }
        }
}

void
move(set<State*>& T, char c, set<State*>& R)
{
        set<State*>::iterator it;
        State* t;
        State* u;
        multimap< char, State* >::iterator itm;

        for(it = T.begin(); it != T.end(); ++it)
        {
                t = (*it);
                for(itm = t->transitions.find(c); itm != t->transitions.end() && itm->first == c;
++itm)
                {
                        u = itm->second;
                        R.insert(u);
                }
        }
}
```

```
void
nfa2dfa(Automata* nfa)
{
	map< set<State*>, int > nfastates;
	queue< set<State*> > nfaq;
	set<State*> cl;
	set<State*> T;
	set<State*>::iterator it;
	int i, count = 0, from;
	char c;
	bool b;

	memset(realadj, -1, sizeof(realadj));
	memset(realfinal, false, sizeof(realfinal));

	T.insert(nfa->istate);
	Eclosure(T, cl);
	nfaq.push(cl);

	b = false;
	for(it = cl.begin(); it != cl.end(); ++it)
		if((*it)->id == (*(nfa->fstates.begin()))->id)
		{
			b = true;
			break;
		}

	nfastates[cl] = count;
	realfinal[count++] = b;

	while(!nfaq.empty())
	{
		set<State*> pset = nfaq.front();
		nfaq.pop();
		from = nfastates[pset];

		for(i = 0; i < Nalpha; ++i)
		{
			set<State*> R, S;
			c = alphabet[i];
			move(pset, c, R);
			Eclosure(R, S);
```

```
                        if(R.empty() || S.empty())
                                continue;

                        if(nfastates.find(S) == nfastates.end())
                        {
                                nfastates[S] = count;
                                b = false;
                                for(it = S.begin(); it != S.end(); ++it)
                                        if((*it)->id == (*(nfa->fstates.begin()))->id)
                                        {
                                                b = true;
                                                break;
                                        }
                                realfinal[count] = b;
                                realadj[from][c - 'a'] = count++;
                                nfaq.push(S);
                        }
                        else
                        {
                                realadj[from][c - 'a'] = nfastates[S];
                        }
                }
        }

        N = count;
}

bool
execute(string word)
{
        int i, state = 0;
        char c;

        for(i = 0; i < word.length(); ++i)
        {
                c = word[i];
                state = realadj[state][c - 'a'];
                if(state == -1)
                        return false;
        }
```

```
	return realfinal[state];
}

int
main()
{
	int count = 0, Q;
	Automata* a;
	string regex, word;
	char c;

	alphabet = "ab";
	Nalpha = alphabet.length();

	while(cin >> regex)
	{
		a = regex2nfa(regex);
		nfa2dfa(a);
		cin >> Q;
		while(getchar() != '\n');
		while(Q--)
		{
			word = "";
			while(true)
			{
				c = getchar();
				if(c != 'a' && c != 'b')
					break;
				word += c;
			}
			if(c != '\n')
				while(getchar() != '\n');
			cout << (execute(word) ? 'S' : 'N') << endl;
		}
		cout << endl;
	}

	return 0;
}
```