1. Padrões

Template

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <string>
#include <cstring>
#include <cmath>
#include <vector>
#include <stack>
#include <queue>
#include <map>
#include <set>
#include <algorithm>

#define FOR(i,v,n) for(int i=v,_n=n;i<_n;++i)
#define FORD(i,v,n) for(int i=(n-1),_v=v;i>=_v;--i)
#define REP(i,n) FOR(i,0,n)
#define REPD(i,n) FORD(i,0,n)
#define FOREACH(it,c) for(__typeof((c).begin())
it=(c).begin();it!=(c).end();++it)

#define MP(a,b) make_pair((a),(b))
#define ALL(c) (c).begin(),(c).end()
#define PB push_back
#define MAX_V
#define MAX_A

#define TRACE(x...) x
#define PRINT(x...) TRACE(printf(x))
#define WATCH(x) TRACE(cout << #x" = " << x << "\n";)

const int INF = 0x3F3F3F3F;
const double PI = acos(-1.0);

using namespace std;

int main(){
    return 0;
}
```

Definição de struct

```
typedef struct{
...
} TReg;
```

O nome da estrutura deve começar sempre com um 'T' (de tipo) seguida do nome começando em maiúscula. Geralmente quando eu não sei que nome dar, eu ponho TReg mesmo.

Leitura de string até EOF

```
while(cin.getline(str, MAXLINE))
      cout << str << endl;
```

sscanf(char *input, char *format, variables)

Usa a variável input como entrada, usando o formato passado e armazena os valores nas variáveis

```
while(sscanf(line, "%d %d %d", &s, &t, &w) == 3)
      printf("%d %d %d", s, t, w);
```

sprintf(char *output, char *format, variables)

Utiliza as variáveis de entrada, usando o formato passado, para armazenar em output. Obs: não imprime na tela.

```
sprintf(line, "%d %d %d", a, b, c);
printf("%s\n", line);
```

2. Tabela ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

3. STL

3.1. Map

Declaração
```
map<string, int> m;
map<string, int>::iterator it;
```

Inserção (sobrescreve valores existentes)
```
m["abracadabra"] = 10;
m["yabadabadoo"] = 15;
m["hi-yo silver"] = 20;
```

Remoção
```
m.erase("yabadabadoo");
```

Percurso
```
for(it=m.begin(); it != m.end(); it++)
    cout << it->first << " " << it->second << endl;
```

Mostrar valor (se não existir retorna 0)
```
cout << m["abracadabra"] << endl;
```

Contar número de elementos iguais à chave
```
cout << m.count("hi-yo silver") << endl;
```

Esvaziar map
```
m.clear();
```

3.2. Set

Declaração
```
set<int> s;
```

Inserção
```
s.insert(10);
```

Se count > 0, elemento está no Set
```
cout << s.count(10) << endl;
```

Esvaziar Set
```
s.clear();
```

3.3. Queue

Declaração
```
queue<int> Q;
```

Inserção
```
Q.push(10);
```

Mostrar primeiro elemento da fila
```
cout << Q.front() << endl;
```

Remove primeiro elemento da fila (retorna void)
```
Q.pop();
```

Verifica se fila está vazia (retorna true se estiver vazia)
```
if(!Q.empty()){ ... }
```

Retorna tamanho da fila
```
cout << Q.size() << endl;
```

4. Ordenação

4.1. Ordenação crescente de vetor

```
int v[n];
/* -1:  a  <  b
 *  0:  a ==  b
 *  1:  a  >  b  */
int cmp(const void * a, const void * b){
  return ( *(int*)a - *(int*)b );
}

qsort(v, n, sizeof(int), cmp);
```

4.2. Ordenação crescente de *struct*

```
typedef struct {
int u, v, w;
} TAdj;

TAdj adj[n];

int cmp(const void *a, const void *b){
TAdj *x, *y;
    x = (TAdj*)a;
    y = (TAdj*)b;
    if(x->w < y->w)
      return -1;
    if(x->w > y->w)
      return 1;
    return 0;
}

qsort(adj, n, sizeof(TAdj), cmp);
```

5. Números Primos

5.1. Números Primos até 10.000

Existem 1.229 números primos até 10.000

```
 2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71  73  79  83  89  97  101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197
199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311
313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431
433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557
563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661
673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809
811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937
941 947 953 967 971 977 983 991 997 1009 1013 1019 1021 1031 1033 1039 1049
1051 1061 1063 1069 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153
1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277
1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373 1381
1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487
1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597
1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699
1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823
1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949
1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063
2069 2081 2083 2087 2089 2099 2111 2113 2129 2131 2137 2141 2143 2153 2161
2179 2203 2207 2213 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293
2297 2309 2311 2333 2339 2341 2347 2351 2357 2371 2377 2381 2383 2389 2393
2399 2411 2417 2423 2437 2441 2447 2459 2467 2473 2477 2503 2521 2531 2539
2543 2549 2551 2557 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 2927 2939 2953 2957 2963 2969 2971 2999 3001
3011 3019 3023 3037 3041 3049 3061 3067 3079 3083 3089 3109 3119 3121 3137
3163 3167 3169 3181 3187 3191 3203 3209 3217 3221 3229 3251 3253 3257 3259
3271 3299 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 4957 4967 4969 4973 4987 4993 4999 5003 5009
5011 5021 5023 5039 5051 5059 5077 5081 5087 5099 5101 5107 5113 5119 5147
5153 5167 5171 5179 5189 5197 5209 5227 5231 5233 5237 5261 5273 5279 5281
5297 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 6907 6911 6917 6947 6949 6959 6961 6967 6971
6977 6983 6991 6997 7001 7013 7019 7027 7039 7043 7057 7069 7079 7103 7109
7121 7127 7129 7151 7159 7177 7187 7193 7207 7211 7213 7219 7229 7237 7243
7247 7253 7283 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 7927
7933 7937 7949 7951 7963 7993 8009 8011 8017 8039 8053 8059 8069 8081 8087
8089 8093 8101 8111 8117 8123 8147 8161 8167 8171 8179 8191 8209 8219 8221
8231 8233 8237 8243 8263 8269 8273 8287 8291 8293 8297 8311 8317 8329 8353
8363 8369 8377 8387 8389 8419 8423 8429 8431 8443 8447 8461 8467 8501 8513
8521 8527 8537 8539 8543 8563 8573 8581 8597 8599 8609 8623 8627 8629 8641
8647 8663 8669 8677 8681 8689 8693 8699 8707 8713 8719 8731 8737 8741 8747
8753 8761 8779 8783 8803 8807 8819 8821 8831 8837 8839 8849 8861 8863 8867
8887 8893 8923 8929 8933 8941 8951 8963 8969 8971 8999 9001 9007 9011 9013
9029 9041 9043 9049 9059 9067 9091 9103 9109 9127 9133 9137 9151 9157 9161
9173 9181 9187 9199 9203 9209 9221 9227 9239 9241 9257 9277 9281 9283 9293
9311 9319 9323 9337 9341 9343 9349 9371 9377 9391 9397 9403 9413 9419 9421
9431 9433 9437 9439 9461 9463 9467 9473 9479 9491 9497 9511 9521 9533 9539
9547 9551 9587 9601 9613 9619 9623 9629 9631 9643 9649 9661 9677 9679 9689
9697 9719 9721 9733 9739 9743 9749 9767 9769 9781 9787 9791 9803 9811 9817
9829 9833 9839 9851 9857 9859 9871 9883 9887 9901 9907 9923 9929 9931 9941
9949 9967 9973
```

5.2. Quantidade de números primos de 1 a 10^N

```
pi(10^1) =           4
pi(10^2) =          25
pi(10^3) =         168
pi(10^4) =       1.229
pi(10^5) =       9.592
pi(10^6) =      78.498
pi(10^7) =     664.579
pi(10^8) =   5.761.455
pi(10^9) =  50.847.534
```

5.3. Fatorial

```
 0! =                         1
 1! =                         1
 2! =                         2
 3! =                         6
 4! =                        24
 5! =                       120
 6! =                       720
 7! =                     5.040
 8! =                    40.320
 9! =                   362.880
10! =                 3.628.800
11! =                39.916.800
12! =               479.001.600 [limite unsigned int]
13! =             6.227.020.800
14! =            87.178.291.200
15! =         1.307.674.368.000
16! =        20.922.789.888.000
17! =       355.687.428.096.000
18! =     6.402.373.705.728.000
19! =   121.645.100.408.832.000
20! = 2.432.902.008.176.640.000
```

5.4. Limites de tipos de dados

tipo	bits	mínimo .. máximo	precisão decimal
char	8	0 .. 127	2
signed char	8	-128 .. 127	2
unsigned char	8	0 .. 255	2
short	16	-32.768 .. 32.767	4
unsigned short	16	0 .. 65.535	4
int	32	-2x10^9 .. 2x10^9	9
unsigned int	32	0 .. 4x10^9	9
int64_t	64	-9x10^18 .. 9x10^9	18
uint64_t	64	0 .. 18x10^18	19

tipo	bits	expoente	precisão decimal
float	32	38	6
double	64	308	15
long double	80	19.728	18

6. Grafos

Estruturas gerais

```
int grau[MAX_V];     // número de arestas no vértice
int d[MAX_V];        // distância
int p[MAX_V];        // pai do vertice
int visited[MAX_V];  // vértices visitados
```

Grafos com peso

```
typedef struct{
 int v;    // vértice
 double w; // peso
} TAdj;
TAdj adj[MAX_V][MAX_A];
```

Grafos sem peso

```
int adj[MAX_V][MAX_A];
```

Inserção de aresta

```
void aresta(int a, int b, double w){
  adj[a][ grau[a] ].v = b;
  adj[a][ grau[a] ].w = w;
  grau[a]++;

  adj[b][ grau[b] ].v = a;
  adj[b][ grau[b] ].w = w;
  grau[b]++;
}
```

Inicialização do grafo

```
memset(grau, 0, sizeof(grau));
memset(visited, 0, sizeof(visited));
memset(p, -1, sizeof(p));
```

6.1. Depth First Search (DFS)

testado com o problema 459

```
void dfs(int s){
int t;
      visited[s] = 1;
      for(int i=0; i<grau[s]; i++){
         t = adj[s][i];
         if(visited[t] == 0){
           p[t] = s;
           dfs(t);
         }
      }
}

void initializeDfs(){
      memset(visited, 0, sizeof(visited));
      memset(p, -1, sizeof(p));
}
```

6.2. Breadth First Search (BFS)

testado com o problema 627

```
void bfs(int inicio){
int s, t;
queue<int> Q;

      memset(visited, 0, sizeof(visited));
      memset(p, -1, sizeof(p));
      d[inicio] = 0;
      visited[inicio] = 1;

      Q.push(inicio);
      while(!Q.empty()){
            s = Q.front();
            Q.pop();
            for(int i=0; i<grau[s]; i++) {
               t = adj[s][i];
               if(visited[t] == 0){
                 visited[t] = 1;
                 d[t] = d[s] + 1;
                 p[t] = s;
                 Q.push(t);
               }
            }
      }
}
```

6.3. Floyd–Warshall

testado com o problema 10171

```
void initialize(){ // inicializar antes de ler o grafo
     for(int i=0; i<MAX_V; i++){
        for(int j=0; j<MAX_V; j++){
            adj[i][j] = INF;
            p[i][j] = -1;
        }
        adj[i][i] = 0;
     }
}

void floyd(){
     for(int k=0; k<MAX_V; k++)
        for(int i=0; i<MAX_V; i++)
           for(int j=0; j<MAX_V; j++)
              if(adj[i][k] + adj[k][j] < adj[i][j]){
                 adj[i][j] = adj[i][k] + adj[k][j];
                 p[i][j]= p[k][j];
              }
}
```

6.4. Floyd MinMax

testado com o problema 10048

```
void floydMinMax(){
int maior;

    for(int k=0; k<MAX_V; k++)
       for(int i=0; i<MAX_V; i++)
          for(int j=0; j<MAX_V; j++){
              if(adj[i][k] > adj[k][j])
                 maior = adj[i][k];
              else
                 maior = adj[k][j];
              if(adj[i][j] > maior){
                 adj[i][j] = maior;
                 adj[j][i] = maior;
              }
          }
}
```

6.5. Dijkstra

testado com o problema 929

```
void dijkstra(int inicial){
priority_queue< pair<int,int> > heap; // peso, vertice
int s, t, peso;

   for(int i=0; i<MAX_V; i++)
       d[i] = INF;
   memset(p, -1, sizeof(p));

   heap.push(make_pair(d[inicial] = 0, inicial));
   while(!heap.empty()){
         s = heap.top().second;
         heap.pop();
         for(int i=0; i<grau[s]; i++){
             t = adj[s][i].v;
             peso = adj[s][i].w;
             if(d[s] + peso < d[t]){
                d[t] = d[s] + peso;
                p[t] = s;
                heap.push(make_pair(-d[t], t));
             }
         }
    }
}
```

6.6. Bellman-Ford

testado com o problema 558

Obs.: algoritmo de caminhos mais curtos de origem única;
é possível inserir arestas com peso negativo;
retorna falso caso haja ciclo com peso negativo (não existe solução).

```
bool bellmanFord(int inicial, int n){
      memset(p, -1, sizeof(p));
      for(int i=0; i<n; i++)
         d[i] = INF;
      d[inicial] = 0;
      for(int i=0; i<n; i++)        // todos vertices
         for(int j=0; j<n; j++)       /*  todas    */
            for(int k=0; k<grau[j]; k++)/*  arestas  */
               if(d[j] + adj[j][k].w < d[adj[j][k].v]){
                  d[adj[j][k].v] = d[j] + adj[j][k].w;
                  p[adj[j][k].v] = j;
             }
      for(int i=0; i<n; i++)
         for(int j=0; j<grau[i]; j++)
            if(d[adj[i][j].v] > d[i] + adj[i][j].w)
              return false;
      return true;
}
```

6.7. PRIM

testado com o problema 10034

```
int total; //armazena o custo total da MST
void prim(int inicio, int n){
priority_queue< pair<int, int> > heap;
int s, t, arc;

      for(int i=0; i<n; i++)
         d[i] = INF;
      memset(p, -1, sizeof(p));
      memset(visited, 0, sizeof(visited));
      heap.push(make_pair(d[inicio] = 0, inicio));
      total = 0;

      while(!heap.empty()){
            s = heap.top().second;
            heap.pop();
            if(!visited[s])
               total += d[s];
            visited[s] = 1;
            for(int i=0; i<grau[s]; i++){
               t = adj[s][i].v;
               arc = adj[s][i].w;
               if(d[t] > arc && !visited[t]){
                  d[t] = arc;
                  p[t] = s;
                  heap.push(make_pair(-d[t], t));
               }
            }
      }
}
```

6.8. Kruskal

testado com o problema 10034

```
typedef struct{
int u, v, w;
} TAdj;

TAdj adj[MAX_A];
TAdj mst[MAX_A]; //armazena a MST
int nro_arestas, total, posMst;

int findSet(int x){
      if(x != p[x])
        p[x] = findSet(p[x]);
      return p[x];
}
void kruskal(){
int u, v, u_set, v_set;

      posMst = 0;
      for(int i=0; i<MAX_V; i++) // todos vértices
         p[i] = i;
      qsort(adj, nro_arestas, sizeof(TAdj), cmp);

      for(int i=0; i<nro_arestas; i++){
            u = adj[i].u; v = adj[i].v;
            u_set = findSet(u);
            v_set = findSet(v);
            if(u_set != v_set){
              p[v_set] = p[u_set];
              mst[posMst++] = adj[i];
              total += adj[i].w;
            }
      }
}
```

6.9. Ordenação Topológica

testado com o problema 11060
Obs.: atenção pois uma ordenação topológica possui diversas saídas possíveis

```
int indegree[MAX_V]; // grau de entrada do vértice
int used[MAX_V];     // vértice foi usado?
queue<int> L;        // armazena a ordenação topológica

void topological_sort(){
queue<int> z;
int v, w;
      memset(indegree, 0, sizeof(indegree));
      for(int i=0; i<MAX_V; i++)
         if(grau[i])
           for(int j=0; j<grau[i]; j++)
              indegree[adj[i][j]]++;

      for(int i=0; i<MAX_V; i++)
         if(used[i] && !indegree[i])
           z.push(i);

      while(!z.empty()){
            v = z.front(); z.pop();
            L.push(v);
            for(int i=0; i<grau[v]; i++){
               w = adj[v][i];
               indegree[w]--;
               if(!indegree[w])
                 z.push(w);
            }
      }
}
```

6.10. Biconectividade

testado pontos de articulação com o problema 10199
testado pontes com o problema 796

Obs.: algoritmo encontra os pontos de articulação e as pontes de um grafo.
Se o grafo for desconexo executar o algoritmo
novamente a partir de um vértice não visitado

```
int nroVert, contRaiz;
vector< pair<int,int> > bridge;
set<int> artPoint;
int low[MAX_V], num[MAX_V];

void initializeArticulation(){
      memset(visited, 0, sizeof(visited));
      memset(p, -1, sizeof(p));
      contRaiz = 0;
      nroVert = 1;
}
```

```
void findArticulation(int v){
int t;
    visited[v] = 1;
    num[v] = low[v] = nroVert++;
    for(int i=0; i<grau[v]; i++){
       t = adj[v][i];
       if(!visited[t]){
         p[t] = v;
         if(num[v] == 1) // conta filhos da raiz na árvore
           contRaiz++;
         findArticulation(t);
         if(low[t] > num[v])
           bridge.push_back(make_pair(v, t));
         if(low[t] >= num[v] && num[v] != 1)
           artPoint.insert(v);
         else if(num[v] == 1 && contRaiz > 1)
                artPoint.insert(v);
         low[v] = MIN(low[v],low[t]);
       }
       else if(p[v] != adj[v][i])
              low[v] = MIN(low[v],num[t]);
    }
}
```

6.11. Bipartição

Algoritmo determina se um grafo é ou não bipartido, ou seja, se o grafo pode ser pintado em apenas duas cores sem vértices adjacentes com a mesma cor.
Testado com o problema 11080.

```
int particao[MAX_V]; // marca qual conjunto está um vértice
int visited[MAX_V], grau[MAX_V];
bool is_bipartite;

void _dfsVisit(int s, int p){
int t;
      visited[s] = 1;
      particao[s] = p;
      for(int i=0; i<grau[s]; i++){
         t = adj[s][i];
         if(!visited[t])
           _dfsVisit(t, !p);
         else
             if(particao[t] == p)
                  is_bipartite = false;
      }
}

void bipartite(){
      is_bipartite = true;
      memset(visited, 0, sizeof(visited));
      for(int i=0; i<MAX_V; i++) {
         if(is_bipartite == false)
            break;
         if(grau[i] > 0 && !visited[i])
           _dfsVisit(i,0);
      }
}
```

6.12. Número de caminhos distintos em um DAG

testado com o problema 988

```
int countDAGPaths(int s, int t){
int u;

  topological_sort();
  memset(d, 0, sizeof(d));
  d[s] = 1;

  while(!L.empty()){
        u = L.front(); L.pop();
        for(int i=0; i<grau[u]; i++)
      d[adj[u][i]] = d[adj[u][i]] + d[u];
  }
  return d[t];
}
```

6.13. Fluxo Máximo - Edmonds-Karp

testado com o problema 820 (Internet Bandwidth)
Complexidade O(V*E*E) - bom para grafos esparsos

```
#define residue(src, dst) ( c[ (src) ][ (dst) ] -
flow( (src), (dst) ) )
#define flow(src, dst) ( f[ (src) ][ (dst) ] - f[ (dst) ]
[ (src) ])


int f[MAX_V][MAX_V]; // fluxo
int c[MAX_V][MAX_V]; // capacidade = peso das arestas

void initialize(){
     memset(f, 0, sizeof(f));
     memset(c, 0, sizeof(c));

     // se grafo possui arestas paralelas com pesos
     // diferentes, acumula capacidade
      for(int i=0; i<MAXV; i++)
         for(int j=0; j<grau[i]; j++)
             c[i][adj[i][j].v] += adj[i][j].w;
}
```

```
int maxFlowEdmondsKarp(int s, int t){
int F = 0, minCapacity, u, v;
  while(true){
       queue<int> Q;
       memset(p, -1, sizeof(p));
       Q.push(s);
       p[s] = -2;
       // BFS
       while(!Q.empty() && p[t] == -1){
             u = Q.front();
             Q.pop();
             for(int i=0; i<grau[u]; i++){
                 v = adj[u][i].v;
                 if(p[v] == -1 && residue(u, v) > 0){
                    p[v] = u;
                    Q.push(v);
                 }
             }
       }
        // não encontrou caminho até t
       if(p[t] == -1)
          return F;
       // encontra menor capacidade no caminho
       minCapacity = INF;
       for(v = t, u = p[v]; u >= 0; v = u, u = p[v])
           minCapacity = MIN(minCapacity, residue(u, v));
        // incrementa fluxo no caminho
        for(v = t, u = p[v]; u >= 0; v = u, u = p[v])
            f[u][v] += minCapacity;
        F += minCapacity;
  }
  return F;
}
```

6.14. Componentes Fortemente Conectados

É o conjunto de vértices em que para cada vértice u,v existe um caminho de u para v e vice-versa. Matriz invAdj representa a lista de adjacências inversa, ou seja, em vez de u->v, nela definimos v->u. Elementos com o mesmo id no vetor visited pertencem ao mesmo grupo fortemente conectado. Testado com o problema 247.

```
stack<int> S;
void dfsVisit(int v){
     visited[v] = 1;
      for(int i = 0; i < grau[v]; i++)
        if(!visited[adj[v][i]])
            dfsVisit(adj[v][i]);
      S.push(v);
}
void dfsVisit2(int x){
      visited[x] = id;
      for(int i = 0; i < invGrau[x]; i++)
          if(!visited[invAdj[x][i]])
             dfsVisit2(invAdj[x][i]);
}
void kosaraju(int start, int end){
      memset(visited, 0, sizeof(visited));
      for(int i=start; i<end; i++)
          if(!visited[i])
            dfsVisit(i);
      memset(visited, 0, sizeof(visited)); id = 1;
      while(!S.empty()){
            int u = S.top(); S.pop();
            if(!visited[u]){
               dfsVisit2(u); id++;
            }
      }
}
```

6.15. Karp

```
int weight[MAX][MAX], d[MAX+1][MAX];
int n, m;

double MMC(){
   //Initialize
   int k, u, v, s = 0;
   for(k=0 ; k<= n ; k++)
      for(u= 0 ; u<n ; u++)
         d[k][u] = INF;
   d[0][s] = 0;

   //Compute distances
   for(k=1; k<=n ; k++)
      for(v=0 ; v<n ; v++)
         for(u=0 ; u<n ; u++)
            if(weight[u][v] < INF)
               d[k][v] = MIN(d[k][v], d[k-1][u]+weight[u][v]);

   //Compute lambda using karp's theorem
   double lamda = INF;
   for(u=0 ; u<n ; u++){
      double currentLamda = -INF; // changed here
      for(int k=0 ; k<n ; k++)
         if(d[n][u] < INF && d[k][u] < INF)
            currentLamda = MIN(currentLamda, 1.0*(d[n][u]-d[k]
[u])/(n-k) );
      lamda = MIN(lamda, currentLamda);
   }

   return lamda;
}
```

6.16. Ciclo Euleriano

```
//Informação sobre uma aresta
typedef struct{
    int label; //contém um label
    int dest;  //e o destino da aresta
}TInfoAr;

int vertices_visitados[MAX_V], //marca vértices visitados - BFS
    grau[MAX_V],               //grau de cada vértice
    vertice_presente[MAX_V],   //vértices presentes no grafo
    achouCiclo,        //marca se encontrou um ciclo euleriano
    nvt,                        //Número de vértices
    nar,                        //Número de arestas
    inicioCiclo; //Vértice inicial em busca do ciclo euleriano

/* Representação do grafo: Cada elemento guarda um vértice e as
arestas que são adjacentes.
   Ex: 1 -> 0 1 2
          2 3 2
   Vértice 1 está ligado às arestas 0, 1, 2 e estas arestas têm
como destino os vértices
   2, 3, 2, respectivamente
*/
TInfoAr va[MAX_V][MAX_A];

//Ciclo euleriano
deque< int > cicloEuleriano;
//Inicializa o grafo
void ini_grafo(int n=0){
    nar = 0;
    nvt = n;
    memset(vertice_presente, 0, sizeof(vertice_presente));
    memset(grau, 0, sizeof(grau));
}
void printGrafo(){
    cout<<nvt<<" "<<nar<<endl;
    for(int i=0; i<MAX_V; i++){
        if(vertice_presente[i]==0) continue;
        cout<<i<<" -> ";
        for(int j=0; j<grau[i]; j++){
            cout<<"("<<va[i][j].dest<<","<<va[i][j].label<<")";
        }
        cout<<endl;
    }
}

//Insere uma aresta no grafo
void aresta(int u, int v){

    //Verifica se os vértices u e v estão no grafo, atualizando
o
    //número de vértices e o vetor vertice_presente
    if(vertice_presente[u]==0){ vertice_presente[u]=1; nvt++; }
    if(vertice_presente[v]==0){ vertice_presente[v]=1; nvt++; }

    //Insere a aresta u-v e v-u no grafo

    va[u][ grau[u] ].dest = v; va[u][ grau[u] ].label = nar;
    grau[u]++;

    va[v][ grau[v] ].dest = u; va[v][ grau[v] ].label = nar;
    grau[v]++;

    //Incrementa o número de arestas
    nar++;

}
```

```
//Verifica usando o BFS se o grafo é conexo
bool eh_conexo(int inicio){
     queue<int> fila;
     int u, vAux;

     memset(vertices_visitados, 0, sizeof(vertices_visitados));

     fila.push(inicio);
     vertices_visitados[inicio] = 1;

     while(fila.empty()==false){
        u = fila.front(); fila.pop();
        for(int i=0; i<grau[u]; i++){
            vAux = va[u][i].dest;
            if(vertices_visitados[vAux]==0){
                vertices_visitados[vAux]=1;
                fila.push(vAux);
            }
        }
     }
     for(int i=0; i<MAX_V; i++)
        if(vertice_presente[i]==1 && vertices_visitados[i]==0)
          return false;
     return true;
}
```

```
//Rotaciona o ciclo e obtêm o vértice onde deverá encontrar o
novo ciclo
void rotacionaCiclo(){

    //retira o primeiro elemento *1 2 3 4 1 ->  2 3 4 1
    cicloEuleriano.pop_front();

    //Procura um vértice no ciclo euleriano cujo grau seja
maior que 0
    int posRot=0;
    for(int i=0; i<cicloEuleriano.size(); i++)
       if(grau[ cicloEuleriano[i] ]>0){ posRot = i;
inicioCiclo=cicloEuleriano[i]; break; }

    //Coloca o elemento com grau>0 presente no cicloEuleriano
no início do ciclo
    rotate(cicloEuleriano.begin(), cicloEuleriano.begin()
+posRot, cicloEuleriano.end());

    //Põe no final do ciclo o elemento inicial
    cicloEuleriano.push_back( cicloEuleriano[0] );

    //Tira o elemento inicial que será usado para encontrar o
próximo ciclo
    cicloEuleriano.pop_front();
}
```

```
void remove_aresta(int u, int v, int lb){

    for(int i=0; i<grau[u]; i++){
        if(va[u][i].label == lb){
            grau[u]--;
            va[u][i].dest  = va[u][ grau[u] ].dest;
            va[u][i].label = va[u][ grau[u] ].label;
            break;
        }
    }

    for(int i=0; i<grau[v]; i++){
        if(va[v][i].label == lb){
            grau[v]--;
            va[v][i].dest  = va[v][ grau[v] ].dest;
            va[v][i].label = va[v][ grau[v] ].label;
            break;
        }
    }
    nar--; //decrementa o número de arestas
}
```

```
void get_ciclo_euleriano(int u){
    int prox, lb;

    //ainda não achou o ciclo
    achouCiclo=0;

    // se o grafo não tem arestas, termina
    if(nar==0) return;

    //Enquanto não achou um ciclo euleriano
    while(achouCiclo==0){
        //Insere o vértice no ciclo
        cicloEuleriano.push_front(u);

        //Obtém o próximo vértice do ciclo
        prox = va[u][0].dest;
        lb   = va[u][0].label;

        //Remove a aresta do vértice
        remove_aresta(u, prox, lb);

        //atribui ao u o prox
        u = prox;

//Se o vértice destino da aresta sendo visitada é igual ao
vértice inicial
        if(u==inicioCiclo)
          achouCiclo=1; //encontrou um ciclo
     }

     //Coloca no cicloEuleriano o vértice inicial novamente,
fechando o ciclo
     cicloEuleriano.push_front(u);
}
```

```
/* Verifica se o grafo possui ciclo euleriano.

   1. Se o grafo não é conexo, termina.
   2. Se possui vértice com grau ímpar, termina.
   3. Encontra o primeiro ciclo euleriano partindo de u.
   4. Se não encontrar um vértice no ciclo euleriano com
grau>1, termina. Senão,
      atribui o vértice encontrado à variável inicioCiclo
   5. u = inicioCiclo
   6. Volta ao passo 3.
*/
void ciclo_euleriano(int u){
    //Verifica se o grafo é conexo
    if(eh_conexo(u)==false){ return; }
    //Verifica se existe algum vértice com grau ímpar
    for(int i=0; i<MAX_V; i++)
       if(vertice_presente[i]==1 && grau[i]%2!=0){ return; }

    //Limpa o ciclo euleriano
    cicloEuleriano.clear();
    inicioCiclo=u;

    while(1){
        //Não achou ciclo
        achouCiclo = 0;
        //Obtém o ciclo euleriano começando em inicioCiclo
        get_ciclo_euleriano(inicioCiclo);

        //Se acabaram as arestas
        if(nar==0) return;

        //Rotaciona o ciclo e encontra o novo inicioCiclo
        rotacionaCiclo();
    }
}
```

```
int main(){
    int t, n, u, v;
    scanf("%d", &t);
    for(int iT=0; iT<t; iT++){
        scanf("%d", &n);
        ini_grafo();
        for(int iN=0; iN<n; iN++){
            scanf("%d %d", &u, &v);
            aresta(u, v);
        }

        ciclo_euleriano(u);

        printf("Case #%d\n",iT+1);
        if(cicloEuleriano.size()==0)
          printf("some beads may be lost\n");
        else
           for(int iC=0; iC< cicloEuleriano.size() - 1; iC++)
               printf("%d %d\n",cicloEuleriano[iC],
                      cicloEuleriano[iC+1]);

        if(iT!=t-1)
          printf("\n");
    }
}
```

7. Programação Dinâmica

7.1. Longest Increasing Subsequence - LIS

testado com o problema 497
Obs.: algoritmo encontra a maior subsequência crescente em um vetor

```
int length[MAX_V]; // armazena o comprimento maximo na posição
int p[MAX_V];      // pai
int v[MAX_V];      // vetor com a sequência
int n;
void lis(){
      for(int i=0; i<n; i++)
         length[i] = 1;
      memset(p, -1, sizeof(p));
      for(int i=0; i<n-1; i++)
         for(int j=i+1; j<n; j++)
             if(v[j] > v[i])
                if(length[i] + 1 > length[j]){
                   length[j] = length[i] + 1;
                   p[j] = i;
                }
}
```

7.2. Longest Common Subsequence - LCS

testado com o problema 11151
Obs.: algoritmo encontra a maior subsequência comum entre duas strings

```
int table[MAX_V][MAX_V];
char a[MAX_V], b[MAX_V];

int lcs(){
int m, n;
      m = strlen(a); n = strlen(b);
      for(int i=0; i<m; i++)
         for(j=0; j<n; j++)
            table[i][j]=0;
      for(int i=1; i<m+1; i++)
         for(int j=1; j<n+1; j++)
            if(a[i-1]==b[j-1])
              table[i][j] = table[i-1][j-1]+1;
            else
              table[i][j] = MAX(table[i][j-1], table[i-1][j]);
      return table[m][n];
}
void printPath(int i, int j){
      if(i == 0 || j==0)
         return;
      if(a[i-1] == b[j-1]){
        printPath(i-1, j-1);
        printf("%c", a[i-1]);
      }
      else if(table[i-1][j] <= table[i][j-1])
              printPath(i, j-1);
            else
              printPath(i-1, j);
}
```

7.3. Matrix Chain Multiplication - MCM

testado com o problema 348

```
int m[MAX_V][MAX_V];  // custos
int s[MAX_V][MAX_V];  // caminho
int p[MAX_V];         // dimensões das matrizes
void initializeMcm(int n){
      for(int i=1; i<n+1; i++)
         for(int j=i; j<n+1; j++)
             m[i][j] = INF;
}
void printPath(int i, int j){
    if(i==j)
       cout << "A" << i;
    else{
      cout << "(";   printPath(i,s[i][j]);
      cout << " x "; printPath(s[i][j]+1, j);
      cout << ")";
    }
}
int mcm(int i, int j){ // primeira chamada: mcm(1, n)
int k, q;
      if(m[i][j] < INF)
        return m[i][j];
      if(i == j)
        m[i][j] = 0;
      else
        for(int k=i; k<j; k++){
            q = mcm(i,k) + mcm(k+1,j) + p[i-1]*p[k]*p[j];
            if(q < m[i][j]){
               m[i][j] = q; s[i][j] = k;
            }
        }
      return m[i][j];
}
```

7.4. Knapsack - problema da mochila binária

testado com o problema 10130

```
int c[MAX_ITEMS+1][MAX_WEIGHT+1];
int weight[MAX_ITEMS+1], price[MAX_WEIGHT+1];

// Bottom-up: tabulation
void knapsack(int n, int w){

  for(int i=0; i<n+1; i++) c[i][0] = 0; // para todos produtos
  for(int i=0; i<w+1; i++) c[0][i] = 0; // para todos pesos
  for(int i=1; i<n+1; i++)
     for(int j=1; j<w+1; j++)
        if(weight[i] > j)
          c[i][j] = c[i-1][j];
        else
          c[i][j] = MAX(c[i-1][j],
                        c[i-1][j-weight[i]] + price[i]);
}

// Top-down: memoization
int memoKnapsack(int w, int i){
    if(w == 0 || i == 0)
        return 0;

    if(c[i-1][w] > 0)
       return c[i-1][w];

    if(weight[i-1] > w)
        return memoKnapsack(w, i-1);

    return max(memoKnapsack(w, i-1),
       price[i-1] + memoKnapsack(w - weight[i-1], i-1));
}
```

7.5. Edit Distance

testado com o problema 526

Entrada: Duas strings
Saída: O menor custo (em termos de remoção, alteração, inserção) para transformar a primeira string na segunda
Obs: Índices das strings iniciam em 1, sendo str[0]=' '

```
#define MATCH 0
#define INSERT 1
#define DELETE 2
#define MAXLEN 1000

typedef struct{
        int c; //custo
        int p; //pai
}TCelula;

TCelula m[MAXLEN+1][MAXLEN+1]; //tabela da programacao dinamica
char s[MAXLEN], t[MAXLEN]; //strings de entrada
```

```
int comparaStrings(){
int i, j, k, opt[3];

  for(i=0; i<MAXLEN; i++){
            m[0][i].c = m[i][0].c = i;
            m[0][i].p = INSERT;
            m[i][0].p = DELETE;
  }
  m[0][0].p = -1;

  for(i=1; i<strlen(s); i++)
   for(j=1; j<strlen(t); j++){
     //os custos podem ser alterados aqui.
     opt[MATCH] = m[i-1][j-1].c +
                  (s[i]==t[j] ? 0 : 1); //c = 0 ou 1
     opt[INSERT] = m[ i ][j-1].c + 1; //deleta t[j], c = 1
     opt[DELETE] = m[i-1][ j ].c + 1; //deleta s[i], c = 1

    //Agora começa a escolha da opção menos custosa...
     m[i][j].c = opt[MATCH];  m[i][j].p = MATCH;

  for(k = INSERT; k<=DELETE; k++)
     if(opt[k] < m[i][j].c){
        m[i][j].c = opt[k]; m[i][j].p = k;
     }
   }
   i = strlen(s) - 1;
   j = strlen(t) - 1;
   return( m[i][j].c );
}
```

```
// deve ser chamado com i = strlen(s) - 1 e  j = strlen(t) - 1;
void reconstroiCaminho(int i, int j){
         if( m[i][j].p == -1 ) return;
         if( m[i][j].p == MATCH ){
              reconstroiCaminho(i-1, j-1);
          if(s[i]==t[j]) printf("M"); //match!
          else printf("S"); //substitution
              return;
         }
         if( m[i][j].p == INSERT ){
              reconstroiCaminho(i, j-1);
              printf("I");
              return;
         }
         if( m[i][j].p == DELETE ){
              reconstroiCaminho(i-1, j);
              printf("D");
              return;
         }
}
int main(){
int i, j;
  s[0] = t[0] = ' '; //char em 0 nas strings deve ser ' '
  while(true){
        gets(&(s[1]));
        if(feof(stdin)) break;
           gets(&(t[1]));
           printf("Menor custo para transformar %s em %s:
%d\n", &(s[1]), &(t[1]), comparaStrings());
           i = strlen(s) - 1;  j = strlen(t) - 1;
           reconstroiCaminho(i, j);
           printf("\n");
        }
        return 0;
}
```

7.6. Máxima soma de subconjunto <= N

testado com o problema 624

Algoritmo para calcular a soma que mais se aproxima de um valor dado.
Entrada: Vetor de inteiros e a soma n
Saída: Maior soma possível <= n
Imprime os números caso necessário

```
map<pair<int, int>, int > tem_sucessor;
map<pair<int, int>, pair<int, int> > sucessor;
map<pair<int, int>, int> cache;
map<pair<int, int>, int> esta_em_cache;

int max_qtd(int n, int i, const VI &ele){
    if(n<=0 || i<=0 || ele.size()==0)
       return 0;
    if(esta_em_cache[make_pair(n,i)]==1)
       return cache[make_pair(n,i)];
    int aux, maximo=0, iMaximo=-1;
    for(int k=0; k<ele.size(); k++){
       VI temp = ele; temp.erase(temp.begin()+k);
       aux=0;
       if(ele[k]<=n)
           aux=ele[k] + max_qtd(n-ele[k], i-1, temp);
       if(aux>maximo)
           maximo=aux, iMaximo=k;
    }
    if(iMaximo>=0){
      sucessor[make_pair(n,i)]=make_pair(n-ele[iMaximo], i-1);
      tem_sucessor[make_pair(n,i)]=1;
    }
    esta_em_cache[make_pair(n,i)] = 1;
    return (cache[make_pair(n,i)]=maximo);
}
```

```
void imprimeSol(int n, int t){
    int auxN, auxT;
    while(tem_sucessor[make_pair(n,t)]==1){
        cout<<n-sucessor[make_pair(n,t)].first<<" ";
        auxN=sucessor[make_pair(n,t)].first;
        auxT=sucessor[make_pair(n,t)].second;
        n=auxN;
        t=auxT;
    }
}

int main(){
   while(1){
       int n, t, aux;
       vector<int> durations;

       if(!(cin>>n))
          return 0;
       cin>>t;

       for(int i=0; i<t; i++)
          cin>>aux, durations.push_back(aux);

       sucessor.erase(all(sucessor));
       cache.erase(all(cache));
       esta_em_cache.erase(all(esta_em_cache));
       tem_sucessor.erase(all(tem_sucessor));

       int maximo = max_qtd(n, t, durations);
       imprimeSol(n, t);
       cout<<"sum:"<<maximo<<endl;
   }
   return 0;
}
```

7.7. Coins Change

Algoritmo retorna o número mínimo de moedas necessárias para um troco. Na configuração abaixo, usando money=7 retornaria duas moedas (3 + 4). Caso tivesse utilizado um algoritmo guloso a resposta seria errada, retornariam 3 moedas (5 + 1 + 1). Abaixo duas versões, a primeira com memoization e a segunda com DP.

```
int coins[] = {1, 3, 4, 5}, coins_length = 4;
int minCoins[MAX];

int coinsChangeMemo(int money){
      if(minCoins[money] > 0)
         return minCoins[money];
      if(money == 0)
         return 0;
      int v = INF;
      for(int c=0; c<coins_length; c++){
         if(coins[c] > money) continue;
       v = min(v, coinsChangeMemo(money - coins[c]) + 1);
      }
      return minCoins[money] = v;
}

int coinsChangeDP(int money){
  minCoins[0] = 0;
  for(int m=1; m <= money; m++){
   minCoins[m] = INF;
   for(int c=0; c<coins_length; c++){
       if(coins[c] > money) continue;
    minCoins[m] = min(minCoins[m], minCoins[m - coins[c]] + 1);
   }
  }
  return minCoins[money];
}
```

7.8. Coins Ways

Algoritmo retorna o número de maneiras de contar um dinheiro usando as moedas disponíveis.
Testado com o problema 674

```
#define MAXMONEY 7500 // máximo dinheiro que pode trocar
#define MAXCOINS 5    // máximo número de moedas

int ways[MAXMONEY + 1];
int moedas[MAXCOINS] = { 50, 25, 10, 5, 1 }; //vetor ordenado

void coinsWays(){
int c;
      ways[0] = 1;
      for(int i=1; i<=MAXMONEY; i++)
          ways[i]=0;
      for(int i=0; i<MAXCOINS; i++){
          c = moedas[i];
          for(int j=c; j<=MAXMONEY; j++)
              ways[j] += ways[j-c];
      }
}

int main(){
int money;
      coinsWays();
      while(cin>>money)
            printf("%llu\n", ways[money]);
}
```

7.9. Maximum Interval Sum

Algoritmo encontra maior soma contínua em um array v[1-n].
Armazena o intervalo [a-b] e o valor máximo da soma.
Obs.: array deve começar a partir da posição 1

```
#define MAXSIZE 20003
int v[MAXSIZE];

void maxIntervalSum(int n, int &a, int &b, int &maxSum){
int max[MAXSIZE], start[MAXSIZE];
   for(int i=1; i<=n; i++)
       max[i] = start[i] = 0;
   start[0] = 1;
   for(int i=1; i<=n; i++)
      if(max[i-1] >= 0){
         max[i] += max[i-1] + v[i];
         start[i] = start[i-1];
      } else {
         max[i] += v[i];
         start[i] = i;
      }
   b = 1;
   for(int i=2; i<=n; i++)
      if(max[i] > max[b])
         b = i;
    a = start[b];
    maxSum = max[b];
}

int v[10] = { -INF, -2, 1, -3, 4, -1, 2, 1, -5, 4};
int a, b, maxSum;
maxIntervalSum(9, a, b, maxSum);
//retorna: a = 4, b = 7, maxSum = 6
```

7.10. Mochila Fracionária

```
/* Recebe como parametros o numero de objetos n,
   o array de objetos com seus respectivos pesos e valores,
 e a capacidade W da mochila */
double mochilaFracNlogN(int n, Object * obj, int W){
    double weight = 0.0; // Peso armazenado na mochila
    double total = 0.0;  // Valor total armazenado na mochila
    int i;

    // Ordena os objetos com relacao a razao entre valor/peso
    qsort(obj, n, sizeof(Object), ratioCmp);

    /* Coloca o maior numero de objetos de forma completa na
mochila, em ordem de razao valor/peso */
    for(i=0; i<n; i++){
    /* Caso nao seja possivel colocar o objeto atual na
mochila, entao sai do laco */
        if(obj[i].weight + weight > W)
           break;
        obj[i].frac = 1.0; // Coloca todo o objeto i na mochila
        total += obj[i].value; // Atualiza o valor total
        weight += obj[i].weight;// Atualiza o peso total
    }
    /* Se nem todos os objetos couberam na mochila,
       coloca apenas uma fracao do melhor objeto, em termos de
valor/peso, ainda nao considerado */
    if(i < n){
         /* Coloca apenas a fracao possivel do objeto i */
        obj[i].frac = (W - weight)/(double)obj[i].weight;
        // Atualiza o valor total da mochila
        total += ((double) obj[i].value) * obj[i].frac;
    }
    return total;
}
```

8. Geometria

```
#define X first
#define Y second
```

Ponto:
```
typedef pair<double, double> TPoint;
```

Polígono:
```
typedef vector< TPoint > TPolygon;
```

Reta:
```
typedef struct{
   double A, B, C;
}TLine;
```

Circunferência:
```
typedef struct{
   TPoint c;
   double r;
}Tball;
```

8.1. Distância de ponto a ponto
```
double point_point_dist(TPoint a, TPoint b){
   return(sqrt( (a.X-b.X)*(a.X-b.X) + (a.Y-b.Y)*(a.Y-b.Y)));
}
```

8.2. Distância de ponto a reta
```
/* Distância de ponto a reta
   2*A = |AB|*h = |AxB| => h = |AxB|/|AB| */
double signed_point_line_dist(TPoint a, TPoint b, TPoint c){
   return (is_left(a,b,c)/point_point_dist(a,b));
}
```

8.3. Produto Escalar
```
double prod_escalar(TPoint a, TPoint b, TPoint u, TPoint v){
    return ((b.X-a.X)*(v.X-u.X) + (b.Y-a.Y)*(v.Y-u.Y));
}
```

8.4. Produto Vetorial
```
/* Produto vetorial entre ab e ac.
   Retorna =0 se a, b, c sao colineares
           >0 se ab é cw de ac (c está à esquerda de ab)
           <0 se ab é ccw de ac (c está à direita de ab) */
double is_left(TPoint a, TPoint b, TPoint c){
   return ((b.X-a.X)*(c.Y-a.Y) - (c.X-a.X)*(b.Y-a.Y));
}
```

8.5. Área do triângulo com sinal
```
/* Área de um triangulo com sinal
   É o produto vetorial b-a e c-a dividido por 2 */
double signed_triangle_area(TPoint a, TPoint b, TPoint c){
   return (is_left(a,b,c)/2.0);
}
```

8.6. Dois pontos para equação da reta
```
/* Transforma dois pontos em uma equação de reta
   na forma Ax+By=C.
   y=m*x+b -> -mx+y=b -> -(dy/dx)x+y=b ->
   mult por dx ->  (-dy)*x+(dx)*y=(dx)*b
   entao, A=-dy; B=dx; C=A*x1 + B*x2 */
TLine points_to_line(TPoint a, TPoint b){
  TLine l;
  l.A=b.Y-a.Y;
  l.B=a.X-b.X;
  l.C=l.A*a.X+l.B*a.Y;
  return l;
}
```

8.7. Reta perpendicular à reta

```
/* Retorna uma linha perpendicular à linha passada como
    parametro, no ponto especificado.
    Ax+By=C -> -Bx+Ay=D */
 TLine line_perp(TLine l, TPoint p){
   TLine r;
   r.A = -l.B;
   r.B =  l.A;
   r.C =  r.A*p.X+r.B*p.Y;
   return r;
 }
```

8.8. Ponto de intersecção entre duas retas

```
/* Calcula ponto de intersecção de duas retas.
   Retorna true se as retas se intersectam,
   false caso sejam paralelas ou coincidentes. */
TPoint intersection_point(TLine l1, TLine l2){
    TPoint p;
    //  Regra de CRAMER
   double D  = l1.A*l2.B - l1.B*l2.A,
          Dx = l1.C*l2.B - l1.B*l2.C,
          Dy = l1.A*l2.C - l1.C*l2.A;

   // O sistema pode ter nenhuma ou infinitas soluções
   if(D==0) return p; //linhas sao paralelas
   p.X = Dx/D;
   p.Y = Dy/D;
   return p;
}
```

8.9. Intersecção de segmentos

```
/* Pre-conditions: point x is known to be collinear with
   segment p1-p2
   Return:  true  if x lies beetwen p1 and p2
            false otherwise */

bool __onSegment(TPoint p1, TPoint p2, TPoint x){
   if( (x.X>=min(p1.X, p2.X) && x.X<=max(p1.X, p2.X)) &&
       (x.Y>=min(p1.Y, p2.Y) && x.Y<=max(p1.Y, p2.Y)) )
         return true;
   return false;
}

/* Return: true if segments p1-p2 and p2-p3 intersect
           false otherwise */
bool segmentsIntersect(TPoint p1, TPoint p2, TPoint p3,
                       TPoint p4){
double d1, d2, d3, d4;
   d1 = is_left(p3, p4, p1);
   d2 = is_left(p3, p4, p2);
   d3 = is_left(p1, p2, p3);
   d4 = is_left(p1, p2, p4);

   if( ((d1>0&&d2<0) || (d1<0&&d2>0)) &&
       ((d3>0&&d4<0)||(d3<0&&d4>0)) ) return true;
   if( !d1 && __onSegment(p3, p4, p1) ) return true;
   if( !d2 && __onSegment(p3, p4, p2) ) return true;
   if( !d3 && __onSegment(p1, p2, p3) ) return true;
   if( !d4 && __onSegment(p1, p2, p4) ) return true;
   return false;
}
```

8.10. Centro da circunferência dado 3 pontos

```
/* Retorna o centro de uma circunferencia dado 3 ptos.
    Encontra a linha perpendicular à ab e à bc. Calcula o pto
    de intersecção entre elas, q eh o centro da circunferencia.
    Os 3 ptos nao podem ser colineares!.*/
 TBall points_to_ball(TPoint a, TPoint b, TPoint c){
   TBall ball;
   TPoint p_ab, p_bc;
   TLine l_ab, l_bc, l_perp_ab, l_perp_bc;

   l_ab = points_to_line(a,b);
   l_bc = points_to_line(b,c);

   p_ab.X = (a.X+b.X)/2.0;
   p_ab.Y = (a.Y+b.Y)/2.0;
   p_bc.X = (b.X+c.X)/2.0;
   p_bc.Y = (b.Y+c.Y)/2.0;

   l_perp_ab = line_perp(l_ab, p_ab);
   l_perp_bc = line_perp(l_bc, p_bc);

   ball.c = intersection_point(l_perp_ab, l_perp_bc);
   ball.r = point_point_dist(ball.c, a);
   return ball;
 }
```

8.11. Distância esférica

```
#define RADIUS 6378  /* raio da esfera */
#define PI 3.141592653589793

//Primeiro algoritmo
long double spherical_distance(long double p_lat,
    long double p_long, long double q_lat, long double q_long){
      return (acos(sin(p_lat)*sin(q_lat) +
cos(p_lat)*cos(q_lat)*cos(p_long)*cos(q_long) +
cos(p_lat)*cos(q_lat)*sin(p_long)*sin(q_long))*RADIUS);
}

//Algoritmo alternativo
double spherical_distance(double lat1,double lon1,
                          double lat2,double lon2) {
    double dlon = lon2 - lon1;
    double dlat = lat2 - lat1;
    double a = pow((sin(dlat/2)),2) +
               cos(lat1) * cos(lat2) * pow(sin(dlon/2), 2);
    double c = 2 * atan2(sqrt(a), sqrt(1-a));
    double d = RADIUS* c;
    return d;
}
```

8.12. Ponto no polígono

```
bool __onSegment(TPoint p1, TPoint p2, TPoint x){
    if( (x.X>=min(p1.X, p2.X) && x.X<=max(p1.X, p2.X)) &&
        (x.Y>=min(p1.Y, p2.Y) && x.Y<=max(p1.Y, p2.Y)) )
        return true;
    return false;
 }

 /* Verifica se um pto pertence ou nao a um poligono.
    Traça uma reta do ponto até infinito e conta o numero de
    vezes que corta a borda do poligono. Se for par, retorna
    zero (fora), se for impar retorna um (dentro) e se estiver
    na borda retorna dois.
    Casos especiais: reta infinita cruza um vertice
    OBS: ESTE ALGORITMO FUNCIONA APENAS PARA POLIGONO COM
    ARESTAS VERTICAIS E HORIZONTAIS!! (por causa do caso
    especial citado acima) */
 int point_in_polygon(TPolygon h, TPoint a){
    TPoint b; //extremidade da reta a-b
    int c=0;
    b.X = 11000; //X INFINITO
    b.Y = 0; //Y INFINITO

    for(int i=0; i<h.size(); i++){
      //verifica se está na borda
      if(is_left(h[i], h[(i+1)%h.size()], a)==0 &&
    __onSegment(h[i], h[(i+1)%h.size()], a) )
         return 2; //BORDA
       // Se a reta inf. cruza uma aresta do poligono e nao
         cruza um vertice do poligono
      if(segmentsIntersect(h[i], h[(i+1)%h.size()], a, b))
         c++;
    }
    return c%2;
}
```

8.13. Área do Polígono

```
//We will triangulate the polygon
//into triangles with points p[0],p[i],p[i+1]
double polygon_area(TPolygon p){
double area=0.0, cross;
   for(int i = 1; i+1 < p.size(); i++){
       double x1 = p[i].X - p[0].X;
       double y1 = p[i].Y - p[0].Y;
       double x2 = p[i+1].X - p[0].X;
       double y2 = p[i+1].Y - p[0].Y;
       cross = x1*y2 - x2*y1;
       area += cross;
   }
    return ABS(area/2.0);
}
```

8.14. Convex Hull

```
TPoint first_point;

/*  Compara os angulos formados por first_point-a e
first_point_b. A ordenacao deve ser do maior angulo para o
menor angulo (formado com o eixo y).
    Caso 2 ptos possuam o mesmo angulo, pega o pto mais
distante */

bool __smaller_angle(const TPoint &a, const TPoint &b){
 if(is_left(first_point, a, b)==0)

if(point_point_dist(first_point,a)<=point_point_dist(first_poin
t,b))
       return false;
    else
       return true;
 if(is_left(first_point, a, b)>0)
   return false;
 return true;
}

struct __menorQue{
    bool operator()(const TPoint& a, const TPoint& b){
       if(a.Y<b.Y) return true;
       if(a.Y>b.Y) return false;
       if(a.X<b.X) return true;
       if(a.X>b.X) return false;
       return false;
    }
};



/* Convex hull - Algoritmo de Graham Scan
   Recebe como parâmetro uma lista de pontos, os quais podem
ter ptos duplicados.
   Retorna caso n>3 um poligono, com seus ptos seguindo a
direcao horaria.
   O primeiro elemento é o ultimo elemento do hull. {(1,1),
(1,2),(2,2),(2,1),(1,1)} caso n<3 retorna os mesmos pontos. */
void convex_hull(vector< TPoint > &in, TPolygon &hull){
int i, top;
     // Ordena os pontos e remove ptos duplicados
     set< TPoint, __menorQue > tempSet(ALL(in));
     in.erase(ALL(in));
     vector< TPoint > p(ALL(tempSet));

     //Pivô
     first_point = p[0];

     //Ordena de acordo com os angulos formados com o pivô
     sort(p.begin()+1, p.end(), __smaller_angle);

     if(p.size()<=2){
        for(int i=0; i<p.size(); i++)
           hull.push_back(p[i]);
        //hull.push_back(p[0]);
        return;
     }
     hull.push_back(p[0]);
     hull.push_back(p[1]);

     p.push_back(first_point); // copia o primeiro pto no
final, para qdo der a volta completa

     top=1; //Topo da pilha dos ptos
     i=2;
```

```
     //Percorre os ptos ordenados
     while(i<p.size()){
         if(is_left(hull[top-1],hull[top],p[i])>=0){ //Permite
que só vire à direita
            top--;
            hull.pop_back();
         }
         else{
            top++;
            hull.push_back(p[i]);
            i++;
         }
     }
}

int main(){
TPolygon hull;
vector< TPoint > in;
int n;
double x,y;

   while(scanf("%d", &n)==1){
       in.erase(ALL(in));
       hull.erase(ALL(hull));
       for(int i=0; i<n; i++){
           scanf("%lf %lf", &x, &y);
           in.push_back(make_pair(x,y));
       }
       convex_hull(in, hull);
       for(int i=0; i<hull.size(); i++)
           printf("%lf %lf\n", hull[i].X, hull[i].Y);
       printf("\n");
   }
   return 0;
}
```

9. Matemática Geral

9.1. Primalidade

```
int isPrime(int n){
      if(n == 2 || n == 3)
         return 1;
      if(n <= 1 || n%2 == 0)
         return 0;
      for(int i=3; i*i <= n; i += 2)
         if(n%i == 0)
            return 0;
      return 1;
}
```

9.2. Exponenciação

```
long square(long n) { return n*n; }

//Maneira eficiente de calcular a^n
long fastexp(long base, long power) {
        if(power == 0) return 1;
        if(power%2 == 0) return square(fastexp(base,power/2));
        return base*(fastexp(base,(power-1)     ));
}

int main(){
long k, n, p;
        while(cin>>n>>p){
             cout << fastexp(n, p ) << endl;
        }
        return 0;
}
```

9.3. BigInt

```
const int DIG = 4;
const int BASE = 10000;
const int TAM = 2048;

struct bigint{
int v[TAM], n;
bigint(int x = 0): n(1){
      memset(v, 0, sizeof(v));
      v[n++] = x; fix();
}
bigint(char *s): n(1){
      memset(v, 0, sizeof(v));
      int sign = 1;
      while(*s && !isdigit(*s)) if(*s++ == '-') sign *= -1;
      char *t = strdup(s), *p = t + strlen(t);
      while(p > t){
            *p = 0; p = MAX(t, p - DIG);
            sscanf(p, "%d",&v[n]);
            v[n++] *= sign;
      }
      free(t); fix();
}
bigint& fix(int m = 0){
      n = MAX(m,n);
      int sign = 0;
      for(int i=1, e=0; i<=n || e && (n = i); i++){
            v[i] += e; e = v[i]/BASE; v[i] %= BASE;
            if(v[i]) sign = (v[i] > 0) ? 1 : -1;
      }
      for(int i = n-1; i > 0; i--)
         if(v[i]*sign < 0){ v[i] += sign*BASE; v[i+1] -= sign;}
         while(n && !v[n]) n--;
         return *this;
}
```

```
bool operator <(const bigint& x) const { return cmp(x) < 0; }
bool operator ==(const bigint& x) const { return cmp(x) == 0; }
bool operator !=(const bigint& x) const { return cmp(x) != 0; }
operator string() const{
      ostringstream s; s << v[n];
      for(int i = n - 1; i > 0; i--){
            s.width(DIG); s.fill('0'); s << ABS(v[i]);
      }
      return s.str();
}
friend ostream& operator <<(ostream& o, const bigint& x){
      return o << (string) x;
}
bigint& operator +=(const bigint& x){
      for(int i = 1; i <= x.n; i++) v[i] += x.v[i];
      return fix(x.n);
}
bigint operator +(const bigint& x){
        return bigint(*this) += x;}
bigint& operator -=(const bigint &x){
      for(int i = 1; i <= x.n; i++) v[i] -= x.v[i];
      return fix(x.n);
}
bigint operator -(const bigint& x){ return bigint(*this) -=
x; }
bigint operator -(){ bigint r = 0; return r -= *this; }
void ams(const bigint& x, int m, int b){
//  *this += (x * m) << b;
for(int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++){
    v[i+b] += x.v[i] * m + e; e = v[i+b]/BASE; v[i+b] %= BASE;
      }
}
```

```
bigint operator *(const bigint& x) const {
      bigint r;
      for(int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
      return r;
}
bigint& operator *=(const bigint& x){return *this = *this *
x; }
bigint& operator /=(const bigint& x){ return *this = div(x); }
bigint& operator %=(const bigint& x){ div(x); return *this; }
bigint operator /(const bigint& x){
       return bigint(*this).div(x); }
bigint operator %(const bigint& x){ return bigint(*this) %=
x; }
bigint div(const bigint& x){
      if(x == 0) return 0;
      bigint q; q.n = MAX(n - x.n + 1, 0);
      int d = x.v[x.n] * BASE + x.v[x.n-1];
      for(int i = q.n; i > 0; i--){
            int j = x.n + i -1;
            q.v[i] = int((v[j] * double(BASE) + v[j-1])/d);
            ams(x, -q.v[i], i-1);
            if(i == 1 || j == 1) break;
            v[j-1] += BASE * v[j]; v[j] = 0;
      }
      fix(x.n); return q.fix();
}
bigint pow(int x){
   if(x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
      bigint r = 1;
      for(int i = 0; i < x; i++) r *= *this;
      return r;
   }
```

```
bigint root(int x){
      if(cmp() == 0 || cmp() < 0 && x%2 == 0) return 0;
      if(*this == 1 || x == 1) return *this;
      if(cmp() < 0) return -(-*this).root(x);
      bigint a = 1, d = *this;
      while(d != 1){
            bigint b = a + (d /= 2);
            if(cmp(b.pow(x)) >= 0){ d += 1; a = b; }
      }
      return a;
  }
};
```

9.4. Módulo com BigInt

Cada dígito do número é armazenado no vetor. No exemplo abaixo retorna true caso seja divisível por 4.

```
int nbr[MAX];
int len;

bool div4(){
int d = 0, rem;

      for(int i=0; i<len; i++){
         rem = (d*10 + nbr[i])%4;
         d = rem;
      }
      if(d == 0)
         return true;
      return false;
}
```

9.5. BigMod

```
/* Esse algoritmo calcula r = b^p mod n. Recebe como parametros
b, p e n, retornando r.
   Esse algoritmo se baseia na idéia: (A*B*C) mod N == ((A mod
N) * (B mod N) * (C mod N)) mod N. */

unsigned long long square(unsigned long long x){
       return x*x;
}

unsigned long long bigMod(unsigned long long  b, unsigned long
long  p, unsigned long long  m){
      if(p == 0)
         return 1;

      if(p%2 == 0) // square(x) = x * x
         return square(bigMod(b, p/2, m)) % m;
      return ((b % m) * bigMod(b, p-1, m)) % m;
}



int main(){
unsigned long long b, p, m;

       while(cin>>b>>p>>m)
          cout<<bigMod(b, p, m)<<endl;

       return 0;
}
```

9.6. Combinatória C(n,k) com n e k grandes

```
long gcd(long a,long b){
  if (a%b==0) return b;
      return gcd(b, a%b);
}

void Divbygcd(long& a,long& b){
   long g = gcd(a,b);
   a/=g; b/=g;
}

long C(int n,int k){
long numerator=1, denominator=1, toMul, toDiv, i;

   if (k>n/2) k=n-k; /* use smaller k!!!, note
                        C(10,6)==C(10,4)*/
   for (i=k; i; i--){
      toMul=n-k+i;
      toDiv=i;
      Divbygcd(toMul,toDiv);/* always divide before multiply */
      Divbygcd(numerator,toDiv);
      Divbygcd(toMul,denominator);
      numerator*=toMul;
      denominator*=toDiv;
   }
   return numerator/denominator;
}

int main(){
long n, k;
       while(1){
         scanf("%ld %ld", &n, &k);
         printf("C(%ld, %ld)=%ld\n", n, k, C(n, k));
       }
}
```

9.7. Quantidade aproximada de primos <= N

```
long arredonda(double x){
       double i, f;
       f = modf(x, &i);
       if(f>=0.5) return (unsigned long long)ceil(x);
       return (unsigned long long)i;
}

//Retorna a qtd de nros primos menores ou iguais a n
long qtdNrosPrimos(long n){
        return arredonda(n/log(n));
}
```

9.8. Fibonacci com BigNum

```
void inicializaBigNum(char *a){
     a[TAM_MAX-1] = '\0';
     for(int i=TAM_MAX-2; i>=0; i--)  a[i] = '0';
}

void soma(char *a, char *b, char *result){
int d1, d2, s, ch, carry=0;

     result[TAM_MAX-1] = '\0';
     for(int i= TAM_MAX-2; i>=0 ; i--){
          d1 = a[i];
          d2 = b[i];
          s = (d1 - '0') + (d2 - '0') + carry;
          ch = (s%10) + '0';
          carry = s/10;

          result[i] = ch;
     }
}

void inverte(char *str){
unsigned long j, i;
    int ch;
    j = TAM_MAX-1;
    i = strlen(str);
    if( i==j ) return;

    while(i>=0)
       str[j--] = str[i--];
}
```

```
int main(){
char a[TAM_MAX], b[TAM_MAX], f[TAM_MAX];
unsigned long n, i;


    while( scanf("%ld", &n) == 1 ){
        inicializaBigNum(a);
        inicializaBigNum(b);
        inicializaBigNum(f);
        a[TAM_MAX-1] = '\0';
        a[TAM_MAX-2] = '1';
        b[TAM_MAX-1] = '\0';
        b[TAM_MAX-2] = '1';
        f[TAM_MAX-1] = '\0';
        f[TAM_MAX-2] = '1';

        for(i=3; i<=n; i++){
           //f(n) = f(n-1) + f(n-2)
           soma(a, b, f);
           strcpy(a, b);
           strcpy(b, f);
        }
        for(i = 0; i<TAM_MAX-1; i++)
           if(f[i]!='0') break;

        while(i<TAM_MAX-1)
           printf("%c", f[i++]), n++;
    }
    return 0;
}
```

9.9. Manipulação de bits: Decimal para binário

```
int main(){
int n, d, aux, DES = 8 * (sizeof(int)) - 1, MASK = 1 << DES, i;
       while(cin>>n){
             i=0;
             while(i<sizeof(int)*8){
               i++;
               d = (n&MASK) ? 1:0;
               cout<<d<<"";
               if(!(i%8)) cout<<" ";
               n = (n<<1);
             }
             cout << endl;
       }
       return 0;
}
```

9.10. Crivo de Eratóstenes - Sieve

```
// Super fast & Memory-tight Sieve by Yarin
#define MAXSIEVE 100000000 // All prime numbers up to this
#define MAXSIEVEHALF (MAXSIEVE/2)
#define MAXSQRT 5000 // sqrt(MAXSIEVE)/2
char a[MAXSIEVE/16+2];
#define isprime(n) (a[(n)>>4]&(1<<(((n)>>1)&7))) // Works when
n is odd

//Does not verify 1 nor 2
//have to check for even numbers
void sieve(){
   int i,j;
   memset(a,255,sizeof(a));
   a[0]=0xFE;
   for(i=1;i<MAXSQRT;i++)
      if (a[i>>3]&(1<<(i&7)))
         for(j=i+i+i+1;j<MAXSIEVEHALF;j+=i+i+1)
            a[j>>3]&=~(1<<(j&7));
}

int ehPrimo(unsigned long long n){
       if(n<=1) return 0;
       if(n==2 || (n%2) && isprime(n)) return 1;
       return 0;
}
```

9.11. Segment Sieve - Números primos em um intervalo

Obs: L<= U; retorna os números primos no intervalo [L, U]

```
long long *primos, numprimos;
void segmentSieve(long long L,long long U) {
long long i, j, d;
    d=U-L+1; // from range L to U, we have d=U-L+1 numbers.
 // use flag[i] to mark whether (L+i) is a prime number or not
bool *flag = new bool[d];

    for (i=0; i<d; i++) flag[i]=true; // mark all true

    for (i=(L%2!=0); i<d; i+=2)
       flag[i]=false; // mark even numbers as false

    /* sieve by prime factors staring from 3 till sqrt(U) */
    for (i=3; i<=sqrt(U); i+=2) {
      if (i>L && !flag[i-L]) continue;
      /* choose the first number to be sieved -- >=L,
         divisible by i, and not i itself! */
      j=L/i*i;
      if (j<L) j+=i;
      if (j==i) j+=i; // if j prime, start from next one

      j-=L; /* change j to the index representing j */
      for (;j<d; j+=i) flag[j]=false;
    }
    /* mark 1 as false, 2 as true */
    if (L<=1) flag[1-L]=false;
    if (L<=2) flag[2-L]=true;
    /* output the result */
    primos = new long long [d];
    for (i=0, j=0; i<d; i++) if (flag[i]) primos[j++]=(L+i);
    numprimos=j;
}
```

9.12. Fatores Primos de um número

Obs: Usa o segmentSieve

```
#define MAX 47000

long long *primos, numprimos;

int main(){
long long n, i;

    segmentSieve(1, MAX);

    while(cin>>n){
            if(n==0) break;
            cout<<n<<" =";
            if(n<0){ cout<<" -1 x"; n = -n; }
            i=0;
            while(true){
               while(n%primos[i]==0){
                  cout<<" "<<primos[i];
                  n=n/primos[i];
                  if(n!=1)cout<<" x";
               }
               if(n<=1) break;
               i++;
               if(i>=numprimos){ cout<<" "<<n; break; }
            }
            cout<<endl;
    }
    return 0;
}
```

9.13. GCD

```
long long gcd(long long a, long long b){
      if( b == 0)
         return a;
    return gcd(b,a%b);

}
```

9.14. GCD/LCM

```
/* Encontra o gcd(p,q) pelo metodo de Euclides and x,y tal que
   p*x + q*y = gcd(p,q)
   Teorema: Se  p e q são inteiros, nao sendo ambos 0, entao
   o menor elemento positivo do conjunto { px + qy : x, y
   pertencentes a Z}
   O gcd(p, q) é uma combinacao linear.*/

long long gcd(long long p, long long q, long long &x,
              long long &y){
long long x1,y1; /* previous coefficients */
long long g;     /* value of gcd(p,q) */

    if (q > p)
        return( gcd(q, p, y, x) );
    if(q == 0){
       x = 1;
       y = 0;
       return(p);
    }
    g = gcd(q, p%q, x1, y1);
    x = y1;
    y = (x1 - floor(p/q)*y1);

    return(g);
}
```

```
long long lcm(long long x, long long y){
long long a, b;
  //calcula otimizado (x * (...))
  return ( x* (y/gcd(x, y, a, b)) );
}

/* Dois números são primos entre si se o gcd entre eles é 1 */
bool primosEntreSi(long long x, long long y){
        long long t, r;
        if(gcd(x, y, t, r)==1) return true;
        return false;
}

int main(){
    long long x, y, p, q, d, m;
    while(cin>>x>>y){
       d = gcd(x, y, p, q);
       m = lcm(x, y);
       cout<<"gcd("<<x<<", "<<y<<")= "<<d<<endl;
       cout<<"p="<<p<<", "<<"q="<<q<<endl;
       cout<<"lcm("<<x<<", "<<y<<")= "<<m<<endl;
       if(primosEntreSi(x, y))
          cout<<"Sao primos entre si"<<endl;
       else
          cout<<"Nao sao primos entre si"<<endl<<endl;
    }
    return 0;
}
```

9.15. Método Newton-Raphson

Método encontra a raiz de uma equação a partir da fórmula abaixo, onde x0 é um chute inicial, a precisão aumenta conforme a diferença entre x1 e x0 diminui. Método f calcula a f(x) no ponto x0, e método d calcula f'(x) no ponto x0.

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

```
double f(double x0) { ... }

double d(double x0) { ... }

double newton_raphson(double x0){
double x1;
double diff = 1.0;

     while(diff > EPS){
           x1 = x0 - (f(x0)/d(x0));
          diff = abs(x1 - x0);
           x0 = x1;
      }
      return x1;
}
```

10.Busca

10.1. Knuth-Morris-Pratt

Algoritmo procura por uma substring P em uma string S.
Retorna o índice inicial que corresponde ao padrão em S, caso não encontre retorna -1. Testado com o problema 11475.

S=ABCADEGH P=ADE => retorna 3

```
int a[MAX], n, m;
char S[MAX], P[MAX];

void calculatePrefix() {
int i = 0, j = -1;
    a[0] = -1;
    while(i < m){
        while (j >= 0 && P[i] != P[j])
            j = a[j];
        i++; j++;
        a[i] = j;
    }
}

int KMP(){
int i = 0, j = 0;
    calculatePrefix();
    while(i < n){
        while(j >= 0 && S[i] != P[j])
                j = a[j];
        j++; i++;
        if(j == m)
          return i - m;
    }
    return -1;
}
```

10.2. Binary Search

Busca pelo elemento key no vetor ordenado, retorna o índice do vetor, caso não encontre o elemento retorna -1.

```
int v[MAX];
int binarySearch(int start, int end, int key){
      if(start > end)
         return -1;
      int mid = (start + end)/2;
      if(v[mid] > key)
         return binarySearch(start, mid-1, key);
      if(v[mid] < key)
         return binarySearch(mid + 1, end, key);
      return mid;
}
```

10.3. Ternary Search

```
//Função que deverá ser encontrado a solução máxima/mínima
double f(double x){ }

/* Função para encontrar máximo/mínimo em um intervalo.
   A função deve ser estritamente crescente e em seguida
   estritamente decrescente, ou vice versa (Forma um bico,
   que é a solução ótima. */
double ternarySearch(double right, double left){
    if(right-left < EPS) return (left+right)/2.0;
    double leftThird  = (left*2.0 + right)/3.0;
    double rightThird = (left + right*2.0)/3.0;
    if(f(leftThird) < f(rightThird))
        return ternarySearch(leftThird, right);
    else
        return ternarySearch(left, rightThird);
}
```

10.4. Segment Tree

Estrutura de dados utilizada para armazenar informações sobre intervalos de um vetor *v*, por exemplo, soma dos elementos em um intervalo, valor máximo no intervalo, etc. Deve ser usada quando muitas consultas são feitas no intervalo, pois o tempo de busca é de ordem logarítmica, portanto, mais rápido que uma busca linear.

Armazenamento na árvore é semelhante a um *heap*, em que o nó *i* da árvore tem como filhos os nós *2*i + 1* e *2*i + 2*, caso a árvore seja indexada a partir da posição 0. Caso o seja indexada a partir da posição 1, os filhos devem ser *2*i* e *2*i + 1*. Os nós folhas armazenam os valores do vetor com os dados consultados.

10.4.1. Soma em um intervalo

```
int n = 5, tree[10];
int v[] = {10, 20, 30, 40, 50};

/* node: índice nó atual
 * start/end: início/fim do segmento atual */
void build(int node, int start, int end){
    if(start == end)
        tree[node] = v[start];
    else {
        int mid = (start + end)/2;
        build(2*node + 1, start, mid);
        build(2*node + 2, mid + 1, end);
        tree[node] = tree[2*node + 1] + tree[2*node + 2];
    }
}
```

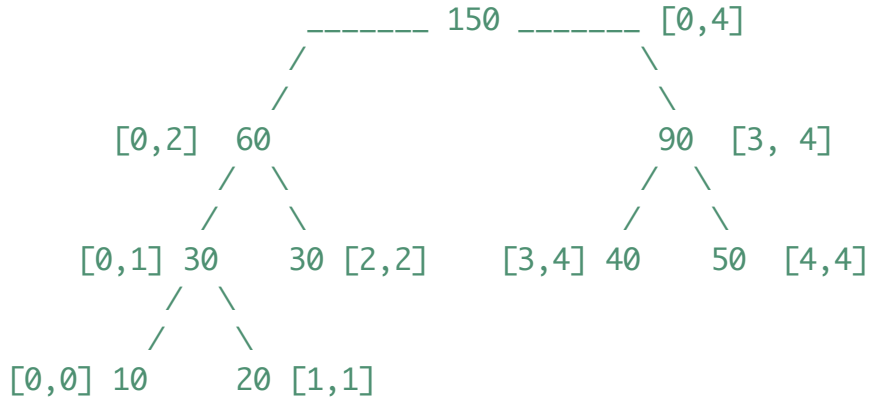
```
/* node: índice nó atual
 * start/end: início/fim do segmento atual
 * i/j: intervalo de busca do array original */
int rangeSum(int node, int start, int end, int i, int j){
    if(start > j || end < i)
       return 0;

    if(i <= start && j >= end)
       return tree[node];

    int mid = (start + end)/2;
    return rangeSum(2*node + 1, start, mid, i, j) +
           rangeSum(2*node + 2, mid + 1, end, i, j);
}

build(0, 0, n-1);
// soma no intervalo [2,4] do vetor v é 120
cout << rangeSum(0, 0, n - 1, 2, 4);

// árvore terá os elementos:
// {150, 60, 90, 30, 30, 40, 50, 10, 20}
// a soma no intervalo [2,4] = [2,2] + [3,4]
```



10.4.2. Menor valor em um intervalo

```
int n = 6;
int tree[12];
int v[] = {1, 97, 45, 100, 300, 70};

void buildMin(int node, int start, int end){
    if(start == end)
        tree[node] = v[start];
    else {
        int mid = (start + end)/2;
        buildMin(2*node + 1, start, mid);
        buildMin(2*node + 2, mid + 1, end);
        tree[node] = min(tree[2*node + 1],
                         tree[2*node + 2]);
    }
}

/* node: índice do nó atual
 * start/end: início/fim do segmento atual
 * i/j: intervalo de busca do array original */
int rangeMin(int node, int start, int end, int i, int j){
    if(start > j || end < i)
        return INF;

    if(i <= start && j >= end)
        return tree[node];

    int mid = (start + end)/2;
    return min(rangeMin(2*node + 1, start, mid, i, j),
               rangeMin(2*node + 2, mid + 1, end, i, j));
}
```

```
buildMin(0, 0, n-1);
// menor valor no intervalo [1,3] do vetor v é 45
cout << rangeMinimum(0, 0, n - 1, 1, 3);
```