Notebook UPF 2018
Maratona de Programação da SBC

Felipe G. Foschiera, Leonardo D. Constantin
felipefoschiera@gmail.com, constantin.leo@gmail.com

17 de agosto de 2018

Sumário

1 Introdução **4**
- 1.1 Bugs do Milênio . . . 4
- 1.2 Os 1010 mandamentos . . . 5
- 1.3 Truques sujos (porém válidos) . . . 5
- 1.4 Limites da representação de dados . . . 6
- 1.5 Quantidade de números primos de 1 até 10^n . . . 6
- 1.6 Triângulo de Pascal . . . 6
- 1.7 Fatoriais . . . 7
- 1.8 Tabela ASCII . . . 7
- 1.9 Primos até 10.000 . . . 8

2 C++ e STL **10**
- 2.1 Macros . . . 10
- 2.2 Compilador GNU . . . 10
- 2.3 C++11 . . . 10
- 2.4 Verificar overflow . . . 10
- 2.5 Complex . . . 10
- 2.6 Pair . . . 11
- 2.7 List . . . 11
- 2.8 Vector . . . 11
- 2.9 Deque . . . 11
- 2.10 Queue . . . 11
- 2.11 Stack . . . 11
- 2.12 Map . . . 11
- 2.13 Set . . . 12
- 2.14 Ordered set . . . 12
- 2.15 Unordered set e map . . . 12
- 2.16 Priority Queue . . . 12
- 2.17 Bitset . . . 12
- 2.18 String . . . 12
- 2.19 Algorithm e numeric . . . 12
- 2.20 Algorithm: Não modificadores . . . 13
- 2.21 Algorithm: Modificadores . . . 13
- 2.22 Algorithm: Partições . . . 13
- 2.23 Algorithm: Ordenação . . . 13
- 2.24 Algorithm: Busca binária . . . 13
- 2.25 Algorithm: Heap . . . 13
- 2.26 Algorithm: Máximo e mínimo . . . 14
- 2.27 Algorithm: Permutações . . . 14
- 2.28 Numeric: Acumuladores . . . 14
- 2.29 Functional . . . 14

3 Estruturas de Dados e Bibliotecas **15**
3.1 Manipulação de Bits . . . 15
3.2 Union–Find . . . 15
3.3 Fenwick Tree . . . 15
3.4 Segment Tree . . . 16
3.5 Segment Tree com Lazy Propagation . . . 16
3.6 Segment Tree com struct . . . 17
3.7 Sparse Table . . . 17
3.8 Operações em Binary Search Tree . . . 18

4 Programação Dinâmica **19**
4.1 Algoritmo da Mochila . . . 19
4.2 Edit Distance . . . 19
4.3 Longest Common Subsequence . . . 19
4.4 Coin Change . . . 20
4.5 Subset Sum . . . 20
4.6 Minimum number of coins . . . 20
4.7 Problema dos Pares mais Próximos . . . 20

5 Grafos **22**
5.1 Floyd-Warshall . . . 22
5.2 Dijkstra . . . 22
5.3 BFS . . . 22
5.4 Flood Fill (DFS) . . . 23
5.5 Kruskal - Minimum Spanning Tree . . . 23
5.6 Encontrar ciclo com DFS . . . 23
5.7 Encontrar ciclo em grafo não-direcionado com Union-Find . . . 23
5.8 Problema do Caixeiro Viajante . . . 23
5.9 Algoritmo de Tarjan (SCC) . . . 24
5.10 Lowest Common Ancestor (LCA) . . . 24
5.11 Ordenação Topológica . . . 24
5.12 Dinic - Max Flow . . . 25

6 Matemática **26**
6.1 Progressão Aritmética . . . 26
6.2 Múltiplos positivos de k num intervalo . . . 26
6.3 Número par ou ímpar de divisores . . . 26
6.4 Número de quadrados perfeitos de A a B . . . 26
6.5 Quadrados e retângulos em um Grid de N lados com K dimensões . . . 26
6.6 Exponenciação binária . . . 26
6.7 Sieve of Eratosthenes . . . 27
6.8 Exponenciação Modular . . . 27
6.9 String para número com mod . . . 27
6.10 Inverso multiplicativo . . . 27
6.11 Probabilidade / Gambler's ruin . . . 27
6.12 Quantidade de pontos inteiros embaixo de uma reta entre dois pontos . . . 27

7 Strings **28**
7.1 Knuth–Morris–Pratt . . . 28
7.2 Rabin-Karp . . . 28
7.3 Repetend: menor período de uma string . . . 29
7.4 Suffix Array e Longuest Common Prefix . . . 29
7.5 Função Z e Algoritmo Z . . . 29
7.6 Algoritmo de Manacher . . . 30
7.7 Aho-Corasick . . . 30
7.8 Autômato de Sufixos . . . 30
7.9 Needleman–Wunsch (Alinhamento de Strings) . . . 31

8 Geometria **33**
8.1 Ponto 2D e segmentos de reta . . . 33
8.2 Linha 2D . . . 34
8.3 Círculo 2D . . . 34
8.4 Triângulo 2D . . . 35
8.5 Polígono 2D . . . 35
8.6 Convex Hull . . . 36
8.7 Ponto dentro de polígono convexo . . . 36
8.8 Ponto dentro de triângulo . . . 36
8.9 Círculo dentro de outro . . . 36

9 Outros **37**
9.1 Maximum subarray sum . . . 37
9.2 Swaps adjacentes para ordenar um array . . . 37
9.3 Swaps não adjacentes para ordenar um array . . . 37
9.4 Inversões de tamanho três . . . 37
9.5 Prefixa e Infixa para Posfixa . . . 38
9.6 Números Romanos . . . 38

Capítulo 1

Introdução

1.1 Bugs do Milênio

Erros teóricos:

- Não ler o enunciado do problema com calma.
- Assumir algum fato sobre a solução na pressa.
- Não reler os limites do problema antes de submeter.
- Quando adaptar um algoritmo, atentar para todos os detalhes da estrutura do algoritmo, se devem (ou não) ser modificados (ex:marcação de vértices/estados).
- O problema pode ser NP, disfarçado ou mesmo sem limites especificados. Nesse caso a solução é bronca mesmo. Não é hora de tentar ganhar o prêmio nobel.

Erros com valor máximo de variável:

- Verificar com calma (fazer as contas direito) para ver se o infinito é tão infinito quanto parece.
- Verificar se operações com infinito estouram 31 bits.
- Usar multiplicacao de *int*'s e estourar 32 bits (por exemplo, checar sinais usando $a * b > 0$).

Erros de casos extremos:

- Testou caso $n = 0$? $n = 1$? $n = MAXN$? Muitas vezes tem que tratar separado.
- Pense em todos os casos que podem ser considerados casos extremos ou casos isolados.
- Casos extremos podem atrapalhar não só no algoritmo, mas em coisas como construir alguma estrutura (ex: lista de adj em grafos).
- Não esquecer de self-loops ou multiarestas em grafos.
- Em problemas de caminho Euleriano, verificar se o grafo é conexo.

Erros de desatenção em implementação:

- Errar ctrl-C/ctrl-V em código. Muito comum.
- Colocar igualdade dentro de if? ($if(a = 0)continue;$)
- Esquecer de inicializar variável.
- Trocar *break* por *continue* (ou vice-versa).
- Declarar variável global e variável local com mesmo nome (é pedir pra dar merda...).

Erros de implementação:

- Definir variável com tipo errado (*int* por *double*, *int* por *char*).
- Não usar variável com nome *max* e *min*.
- Não esquecer que *.size*() é unsigned.
- Lembrar que 1 é *int*, ou seja, se fizer *long long a = 1 << 40;*, não irá funcionar (o ideal é fazer *long long a = 1LL << 40;*).

Erros em limites:

- Qual o ordem do tempo e memória? 10^8 é uma referência para tempo. Sempre verificar rapidamente a memória, apesar de que o limite costuma ser bem grande.
- A constante pode ser muito diminuída com um algoritmo melhor (ex: húngaro no lugar de fluxo) ou com operações mais rápidas (ex: divisões são lentas, bitwise é rápido)?
- O exercício é um caso particular que pode (e está precisando) ser otimizado e não usar direto a biblioteca?

Erros em doubles:

- Primeiro, evitar (a não ser que seja necessário ou mais simples a solução) usar $float/double$. E.g. conta que só precisa de 2 casas decimais pode ser feita com inteiro e depois %100.
- Sempre usar *double*, não $float$ (a não ser que o enunciado peça explicitamente).
- Testar igualdade com tolerância (absoluta, e talvez relativa).
- Cuidado com erros de imprecisão, em particular evitar ao máximo subtrair dois números praticamente iguais.

Outros erros:

- Evitar (a não ser que seja necessário) alocação dinâmica de memória.

- Não usar STL desnecessariamente (ex: vector quando um array normal dá na mesma), mas usar se facilitar (ex: nomes associados a vértices de um grafo - $map < string, int >$) ou se precisar (ex: um algoritmo $O(n \log n)$ que usa $< set >$ é necessário para passar no tempo).
- Não inicializar variável a cada teste (zerou vetores? zerou variável que soma algo? zerou com zero? era pra zerar com zero, com -1 ou com INF?).
- Saída está formatada corretamente?
- Declarou vetor com tamanho suficiente?
- Cuidado ao tirar o módulo de número negativo. Ex.: $x\%n$ não dá o resultado esperado se x é negativo, fazer $(x\%n + n)\%n$.

1.2 Os 1010 mandamentos

Cortesia da PUC-RJ.

0. Não dividirás por zero.
1. Não alocarás dinamicamente.
2. Compararás números de ponto flutuante usando EPS.
3. Verificarás se o grafo pode ser desconexo.
4. Verificarás se as arestas do grafo podem ter peso negativo.
5. Verificarás se pode haver mais de uma aresta ligando dois vértices.
6. Conferirás todos os índices de uma programação dinâmica.
7. Reduzirás o branching factor da DFS.
8. Farás todos os cortes possíveis em uma DFS.
9. Tomarás cuidado com pontos coincidentes e com pontos colineares.

1.3 Truques sujos (porém válidos)

- **Método Steve Halim**: As possíveis saídas do problema cabem no código do problema? Deixe um algoritmo *naive* brutando o problema na máquina por alguns minutos e escreva as respostas direto no código para submeter. Exemplo: problema cuja entrada é um único número da ordem de 10^5. Verificar o tamanho máximo de caracteres de uma submissão.
- **Fatoriais até** 10^9: Deixe um programa na sua máquina brutando os fatoriais até 10^9. A cada 10^3 ou 10^6, imprima. Cole a saída no código e use os valores pré-calculados pra calcular um fatorial com 10^3 ou 10^6 operações.
- **Problemas com constantes**: Se algum valor útil de algum problema for constante (independe da entrada), mas você não sabe, brute ele na sua máquina e cole no código.
- **Debug com assert**: Pode colocar *assert* em código para submeter. Tente usar isso pra transformar um WA em um RTE. É uma forma válida de debug. Usar isso somente no desespero (fica gastando submissões).

1.4 Limites da representação de dados

tipo	scanf	bits	mínimo	..	máximo	precisão decimal
char	%c	8	0	..	255	2
signed char	%hhd	8	−128	..	127	2
unsigned char	%hhu	8	0	..	255	2
short	%hd	16	−32.768	..	32.767	4
unsigned short	%hu	16	0	..	65.535	4
int	%d	32	-2×10^9	..	2×10^9	9
unsigned int	%u	32	0	..	4×10^9	9
long long	%lld	64	-9×10^{18}	..	9×10^{18}	18
unsigned long long	%llu	64	0	..	18×10^{18}	19

tipo	scanf	bits	expoente	precisão decimal
float	%f	32	38	6
double	%lf	64	308	15
long double	%Lf	80	19.728	18

1.5 Quantidade de números primos de 1 até 10^n

É sempre verdade que $n/ln(n) < pi(n) < 1.26 * n/ln(n)$.

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

1.6 Triângulo de Pascal

$n\backslash p$	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	1	1									
2	1	2	1								
3	1	3	3	1							
4	1	4	6	4	1						
5	1	5	10	10	5	1					
6	1	6	15	20	15	6	1				
7	1	7	21	35	35	21	7	1			
8	1	8	28	56	70	56	28	8	1		
9	1	9	36	84	126	126	84	36	9	1	
10	1	10	45	120	210	252	210	120	45	10	1

$C(33,16)$	1.166.803.110	limite do int
$C(34,17)$	2.333.606.220	limite do unsigned int
$C(66,33)$	7.219.428.434.016.265.740	limite do long long
$C(67,33)$	14.226.520.737.620.288.370	limite do unsigned long long

1.7 Fatoriais

Fatoriais até 20 com os limites de tipo.

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 do 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	limite do unsigned long long

1.8 Tabela ASCII

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

1.9 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			

Capítulo 2

C++ e STL

2.1 Macros

```
#include <bits/stdc++.h>
#define DEBUG false
#define debugf if (DEBUG) printf
#define MAXN 200309
#define MAXM 900009
#define ALFA 256
#define MOD 1000000007
#define INF 0x3f3f3f3f
#define INFLL 0x3f3f3f3f3f3f3f3f
#define EPS 1e-9
#define PI 3.14159265358979323846264338327950288 4
#define FOR(x,n) for(int x=0; (x)<int(n); (x)++)
#define FOR1(x,n) for(int x=1; (x)<=int(n); (x)++)
#define REP(x,n) for(int x=int(n)-1; (x)>=0; (x)--)
#define REP1(x,n) for(int x=(n); (x)>0; (x)--)
#define pb push_back
#define pf push_front
#define fi first
#define se second
#define mp make_pair
#define all(x) x.begin(), x.end()
#define mset(x,y) memset(&x, (y), sizeof(x));
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
typedef unsigned int uint;
typedef vector<int> vi;
typedef pair<int, int> ii;
```

2.2 Compilador GNU

Alguns comandos do compilador do GNU traduz para algumas instruções em Assembly (muito rápido).

__builtin_ffs(int) //Retorna 1 + posição do bit 1 menos significativo. Retorna zero para 0.
__builtin_clz(int) //Retorna o número de zeros na frente do bit 1 mais significativo. Não definido para zero.
__builtin_ctz(int) //Retorna o número de zeros atrás do bit 1 menos significativo. Não definido para zero.
__builtin_popcount(int) //Soma dos bits.
__builtin_parity(int) //Soma dos bits módulo 2.

__builtin_ffsl(long) //Retorna 1 + posição do bit 1 menos significativo. Retorna zero para 0.
__builtin_clzl(long) //Retorna o número de zeros na frente do bit 1 mais significativo. Não definido para zero.
__builtin_ctzl(long) //Retorna o número de zeros atrás do bit 1 menos significativo. Não definido para zero.
__builtin_popcountl(long) //Soma dos bits.
__builtin_parityl(long) //Soma dos bits módulo 2.

__builtin_ffsll(long long) //Retorna 1 + posição do bit 1 menos significativo. Retorna zero para 0.
__builtin_clzll(long long) //Retorna o número de zeros na frente do bit 1 mais significativo. Não definido para zero.
__builtin_ctzll(long long) //Retorna o número de zeros atrás do bit 1 menos significativo. Não definido para zero.
__builtin_popcountll(long long) //Soma dos bits.
__builtin_parityll(long long) //Soma dos bits módulo 2.

2.3 C++11

auto a = b //a é o tipo de b.
auto a = b() //a é o tipo de retorno de b.
for(T a : b) //itera sobre todos os elementos de uma coleção iterável b.
for(T & a : b) //itera sobre todas as referências de uma coleção iterável b.
lambda functions: **[] (params) -> type {body}** retorna o ponteiro para uma função **type name(params) {body}**

2.4 Verificar overflow

```
if (b > 0 && a > INFLL-b) //a+b vai dar overflow
if (b < 0 && a < -INFLL-b) //a+b vai dar underflow
if (b < 0 && a > INFLL+b) //a-b vai dar overflow
if (b > 0 && a < -INFLL+b) //a-b vai dar underflow
if (b > INFLL/a) //a*b vai dar overflow
if (b < -INFLL/a) //a*b vai dar underflow
```

2.5 Complex

Exemplo: **#include <complex>**, **complex<double> point**;
Funções: **real**, **imag**, **abs**, **arg**, **norm**, **conj**, **polar**

2.6 Pair

#include <utility>
pair<tipo1, tipo2> P;
tipo1 first, tipo2 second

2.7 List

list<Elem> c //Cria uma lista vazia.
list<Elem> c1(c2) //Cria uma cópia de uma outra lista do mesmo tipo (todos os elementos são copiados).
list<Elem> c(n) //Cria uma lista com *n* elementos definidos pelo construtor default.
list<Elem> c(n,elem) //Cria uma lista inicializada com *n* cópias do elemento *elem*.
list<Elem> c(beg,end) //Cria uma lista com os elementos no intervalo [*beg*,*end*).
c.l̃ist<Elem>() //Destrói todos os elementos e libera a memória.

Membros de list:
begin, **end**, **rbegin**, **rend**, **size**, **empty**, **clear**, **swap**.
front //Retorna o primeiro elemento.
back //Retorna o último elemento.
push_back //Coloca uma cópia de *elem* no final da lista.
pop_back //Remove o último elemento e não retorna ele.
push_front //Insere uma cópia de *elem* no começo da lista.
pop_front //Remove o primeiro elemento da lista e não retorna ele.
swap //Troca duas list's em $O(1)$.
erase (it)//Remove o elemento na posição apontada pelo iterador *it* e retorna a posição do próximo elemento.
erase (beg,end)//Remove todos os elementos no range [*beg*,*end*) e retorna a posição do próximo elemento;
insert (it, pos)//Insere o elemento *pos* na posição anterior à apontada pelo iterador *it*.

2.8 Vector

#include <vector>
vector<tipo> V;

Membros de vector:
begin, **end**, **rbegin**, **rend**, **size**, **empty**, **clear**, **swap**.
reserve //Seta a capacidade mínima do vetor.
front //Retorna a referência para o primeiro elemento.
back //Retorna a referência para o último elemento.
erase //Remove um elemento do vetor.
pop_back //Remove o último elemento do vetor.
push_back //Adiciona um elemento no final do vetor.
swap //Troca dois vector's em $O(1)$.

2.9 Deque

#include <queue>
deque<tipo> Q;
Q[50] //Acesso randômico.

Membros de deque:
begin, **end**, **rbegin**, **rend**, **size**, **empty**, **clear**, **swap**.
front //Retorna uma referência para o primeiro elemento.
back //retorna uma referência para o último elemento.
erase //Remove um elemento do deque.
pop_back //Remove o último elemento do deque.
pop_front //Remove o primeiro elemento do deque.
push_back //Insere um elemento no final do deque.
push_front//Insere um elemento no começo do deque.

2.10 Queue

#include <queue>
queue<tipo> Q;

Membros de queue:
back //Retorna uma referência ao último elemento da fila.
empty //Retorna se a fila está vazia ou não.
front //Retorna uma referência ao primeiro elemento da fila.
pop //Retorna o primeiro elemento da fila.
push //Insere um elemento no final da fila.
size //Retorna o número de elementos da fila.

2.11 Stack

#include <stack>
stack<tipo> P;

Membros de stack:
empty //Retorna se pilha está vazia ou não.
pop //Remove o elemento no topo da pilha.
push //Insere um elemento na pilha.
size //retorna o tamanho da pilha.
top //Retorna uma referência para o elemento no topo da pilha.

2.12 Map

#include <map>
#include <string>
map<string, int> si;

Membros de map:
begin, **end**, **rbegin**, **rend**, **size**, **empty**, **clear**, **swap**, **count**.
erase //Remove um elemento do mapa.
find //retorna um iterador para um elemento do mapa que tenha a chave.
lower_bound //Retorna um iterador para o primeiro elemento maior que a chave ou igual à chave.
upper_bound //Retorna um iterador para o primeiro elemento maior que a chave.

Map é um set de pair, ao iterar pelos elementos de map, $i->first$ é a chave e $i->second$ é o valor.

Map com comparador personalizado: Utilizar **struct** com **bool operator<(tipoStruct s) const** . Cuidado pra diferenciar os elementos!

2.13 Set

#include <set>
set<tipo> S;

Membros de set:
begin, **end**, **rbegin**, **rend**, **size**, **empty**, **clear**, **swap**.
erase //Remove um elemento do set.
find //Retorna um iterador para um elemento do set.
insert //Insere um elemento no set.
lower_bound //Retorna um iterador para o primeiro elemento maior que um valor ou igual a um valor.
upper_bound //Retorna um iterador para o primeiro elemento maior que um valor.

Criando set com comparador personalizado: Utilizar **struct cmp** com **bool operator()(tipo, tipo) const** e declarar **set<tipo, vector<tipo>, cmp()> S**. Cuidado pra diferenciar os elementos!

2.14 Ordered set

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#using namespace __gnu_pbds;
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
ordered_set<tipo> S;

Membros de ordered_set:
find_by_order(p) //Retorna um ponteiro para o p-ésimo elemento do set. Se p é maior que o tamanho de n, retorna o fim do set.
order_by_key(v) //Retorna a posição do elemento v no set. Se é menor que o 0-ésimo elemento, retorna 0. Se maior que n-ésimo, retorna n. Indexado em 0.

Mesmo set com operações de find_by_order e order_by_key.

2.15 Unordered set e map

Igual a set e map, porém usa Hash Table (é mais rápido). Precisa de C++11.
unordered_set<tipo> S;
unordered_map<chave, valor> S;

2.16 Priority Queue

#include <queue>
priority_queue<tipo> pq

Membros: **empty**, **size**, **top**, **push**, **pop**.

Utilizar **struct cmp** com **bool operator ()(tipo, tipo)** e declarar **priority_queue<tipo, vector<tipo>, cmp()> pq** .

Maior vem antes!

2.17 Bitset

#include <bitset>
bitset<MAXN> bs

Membros: **empty**, **size**, **count**, **to_string**, **to_ulong**, **to_ullong**.
set //Seta todos os elementos para 1.
reset //Seta todos os elementos para 0.
flip(n) //Alterna o bit n.
flip //Alterna todos os bits.
operador » //Shift left.
operador « //Shift right.
operador & //And bit a bit.
operador | //Or bit a bit.
operador ^ //Xor bit a bit.
operador ~ //Not bit a bit.
operador == //Totalmente igual.
operador != //Ao menos um bit é diferente.

2.18 String

#include <string>
string a = "hello";

Membros: **begin**, **end**, **rbegin**, **rend**, **size**, **clear**, **empty**
operator + //Concatena string.
operator += ou append(str) //Concatena string.
push_back(c) //Concatena caractere.
push_back(c) //Remove último caractere (C++11).
insert(pos, str) ou insert(it, str) //Concatena caractere.
assign(str) ou assign(n, c) //Atribui string.
erase(pos, len) //Deleta trecho da string.
replace(pos, len, str) //Substitui trecho da string.
swap(str) //Troca conteúdos em $O(1)$.
find(str, pos) //Retorna índice da próxima aparição de str em $O(n)$. Retorna string::npos se não achar.
substr(pos, len) //Retorna substring.

2.19 Algorithm e numeric

#include <algorithm> ou **#include <numeric>**
beg e end podem ser ponteiros para arrays do tipo **T** ou iteradores de uma coleção de container tipo **T**. Quando falarmos

em comparador, falamos de funções **bool comp(T a, T b)**, que simulam "menor que". Quando falarmos em evaluadores, falamos em funções **bool eval(T a)**. Quando falarmos em somadores, falamos em funções **T add(T a, T b)**. Todos os ponteiros de funções usados abaixo podem ser codados com lambda functions em C++11.

2.20 Algorithm: Não modificadores

any_of(beg, end, eval) //Retorna se todos os elementos em [beg,end) são evaluados como true pelo evaluador eval.
all_of(beg, end, eval) //Retorna se algum elemento em [beg,end) é evaluado como true pelo evaluador eval.
none_of(beg, end, eval) //Retorna se nenhum elemento em [beg,end) é evaluado como true pelo evaluador eval.
for_each(beg, end, proc) //Executa a função **void proc(T a)** para cada elemento em [beg, end).
count(beg, end, c) //Conta quantos elementos em [beg, end) são iguais a c.
count_if(beg, end, eval) //Conta quantos elementos em [beg, end) são evaluados como true pelo evaluador eval.

2.21 Algorithm: Modificadores

fill(beg, end, c) //Atribui c a todos os elementos em [beg, end).
generate(beg, end, acum) //Atribui a cada posição em [beg,end) o valor retornado por **T acum()** na ordem (usar variáveis globais ou estáticas para valores distintos).
remove(beg, end, c) //Remove todos os elementos em [beg, end) que são iguais a c, retorna o ponteiro para o novo fim de intervalo ou o novo iterador end.
remove_if(beg, end, eval) //Remove todos os elementos em [beg, end) que forem evaluados como true pelo evaluador eval, retorna o ponteiro para o novo fim de intervalo ou novo iterador end.
replace(beg, end, c, d) //Substitui por d todos os elementos em [beg, end) que são iguais a c.
replace_if(beg, end, eval, c) //Substitui por d todos os elementos em [beg, end) que forem evaluados como true pelo evaluador eval.
swap(a, b) //Troca o conteúdo de a e b. Para a maior parte das coleções do C++, é $O(1)$.
reverse(beg, end) //Inverte a ordem em [beg, end).
rotate(beg, beg+i, end) //Rotaciona [beg, end) de forma que o i-ésimo elemento fique em primeiro.
random_shuffle(beg, end) //Aplica permutação aleatória em [beg, end).
unique(beg, end) //Remove todas as duplicatas de elementos consecutivos iguais em [beg, end), retorna o ponteiro para o novo fim de intervalo o novo iterador end.

2.22 Algorithm: Partições

partition(beg, end, eval) //Reordena [beg,end) de forma a que todos os elementos que sejam evaluados como true pelo evaluador eval venham antes dos que sejam evaluados como false. Ordem de cada parte é indefinida.
stable_partition(beg, end, eval) //Mesmo que acima, mas a ordem de cada partição é preservada.

2.23 Algorithm: Ordenação

is_sorted(beg, end) ou **is_sorted(beg, end, comp)** (C++11) //Verifica se [beg, end) está ordenado de acordo com o operador < ou de acordo com o comparador comp.
sort(beg, end) ou **sort(beg, end, comp)** //Ordena [beg, end) de acordo com o operador < ou de acordo com o comparador comp.
stable_sort(beg, end) ou **stable_sort(beg, end, comp)** /Ordena [beg, end) de acordo com o operador < ou de acordo com o comparador comp. Mantém a ordem de elementos iguais.
nth_element(beg, beg+n, beg) ou **nth_element(beg, beg+n, beg, comp)** //Realiza a partição de [beg, end) de forma a que o n-ésimo fique no lugar, os menores fiquem antes e os maiores, depois. *Expected* $O(n)$. Usa o operador < ou o comparador comp.

2.24 Algorithm: Busca binária

lower_bound(beg, end, c) ou **lower_bound(beg, end, c, comp)** //Retorna o ponteiro ou iterador ao primeiro elemento maior que ou igual a c na array ordenada [beg, end) de acordo com o operador < ou de acordo com o comparador comp.
upper_bound(beg, end, c) ou **upper_bound(beg, end, c, comp)** //Retorna o ponteiro ou iterador ao primeiro elemento maior que c na array ordenada [beg, end) de acordo com o operador < ou de acordo com o comparador comp.
binary_search(beg, end, c) ou **binary_search(beg, end, c, comp)** //Retorna se o elemento c na array ordenada [beg, end) de acordo com o operador < ou de acordo com a função **bool comp(T a, T b)**, que simula "menor que".

2.25 Algorithm: Heap

make_heap(beg, end) ou **make_heap(beg, end, comp)** //-Transforma [beg,end) em uma heap de máximo de acordo com o operador < ou de acordo com o comparador **comp**.
push_heap(beg, end, c) ou **push_heap(beg, end, c, comp)** //Adiciona à heap de máximo [beg,end) o elemento c.
pop_heap(beg, end) ou **pop_heap(beg, end, comp)** //Remove da heap de máximo [beg,end) o maior elemento. Joga ele para o final.
sort_heap(beg, end) ou **sort_heap(beg, end, comp)** //Ordena a heap de máximo [beg,end) de forma crescente.

2.26 Algorithm: Máximo e mínimo

max(a,b) //Retorna o maior valor de a e b.
min(a,b) //Retorna o menor valor de a e b.
max_element(beg, end) ou **max_element(beg, end, comp)** //Retorna o elemento máximo em [beg, end) pelo operador < ou pela comparador comp.
min_element(beg, end) ou **min_element(beg, end, comp)** //Retorna o elemento mínimo em [beg, end) pelo operador < ou pela comparador comp..

2.27 Algorithm: Permutações

Use **sort** para obter a permutação inicial!
next_permutation(beg, end) ou **next_permutation (beg, end, comp)** //Reordena [beg, end) para a próxima permutação segundo a ordenação lexicográfica segundo o operador < ou segundo o comparador comp. $O(n)$. Retorna se existe próxima permutação ou não (bool).
prev_permutation(beg, end) ou **prev_permutation (beg, end, comp)** //Reordena [beg, end) para a permutação anterior segundo a ordenação lexicográfica segundo o operador < ou segundo o comparador comp. $O(n)$. Retorna se existe permutação anterior ou não (bool).

2.28 Numeric: Acumuladores

accumulate(beg, end, st) ou **accumulate(beg, end, st, add)** //Soma todos os elementos em [beg, end) a partir de um valor inicial st usando o operador + ou o somador add.
partial_sum(beg, end) ou **partial_sum(beg, end, add)** //-Transforma [beg, end) em sua array de somas parciais usando o operador + ou o somador add. **partial_sum(beg, end, st)** ou **partial_sum(beg, end, st, add)** //Coloca na array iniciando em st a array de somas parciais de [beg, end) usando o operador + ou o somador add.

2.29 Functional

#include <functional>
Algumas funções binárias úteis, especialmente para as funções acima. Quando falamos em agregar, falamos em funções binárias to tipo **T add(T a, T b)**. Quando falamos em comparadores, falamos em funções binárias do tipo **bool comp(T a, T b)**. Quando falamos em transformações, falamos em funções unárias do tipo **T t(T a)**.
plus<T>() //Agregador pelo + do tipo T.
minus<T>() //Agregador pelo - do tipo T.
multiplies<T>() //Agregador operador * do tipo T.
divides<T>() //Agregador pelo / do tipo T.
modulus<T>() //Agregador pelo % do tipo T.
negate<T>() //Transformador pelo - do tipo T.
equal_to<T>() //Comparador pelo == do tipo T.
not_equal_to<T>() //Comparador pelo != do tipo T.
greater<T>() //Comparador pelo > do tipo T.
less<T>() //Comparador pelo < do tipo T.
greater_equal<T>() //Comparador pelo >= do tipo T.
less_equal<T>() //Comparador pelo <= do tipo T.
logical_and<T>() //Comparador pelo && do tipo T.
logical_or<T>() //Comparador pelo || do tipo T.
bind1st(f, k) //Transforma a função binária em unária fixando o primeiro argumento a k.
bind2nd(f, k) //Transforma a função binária em unária fixando o segundo argumento a k.

Capítulo 3

Estruturas de Dados e Bibliotecas

3.1 Manipulação de Bits

```
#include <cmath>
#include <cstdio>
#include <stack>
using namespace std;

#define isOn(S, j) (S & (1 << j))
#define setBit(S, j) (S |= (1 << j))
#define clearBit(S, j) (S &= ~(1 << j))
#define toggleBit(S, j) (S ^= (1 << j))
#define lowBit(S) (S & (-S))
#define setAll(S, n) (S = (1 << n) - 1)

#define modulo(S, N) ((S) & (N - 1))   // returns S % N
    , where N is a power of 2
#define isPowerOfTwo(S) (!(S & (S - 1)))
#define nearestPowerOfTwo(S) ((int)pow(2.0, (int)((log
    ((double)S) / log(2.0)) + 0.5)))

#define turnOffLastBit(S) ((S) & (S - 1))
#define turnOnLastZero(S) ((S) | (S + 1))
#define turnOffLastConsecutiveBits(S) ((S) & (S + 1))
#define turnOnLastConsecutiveZeroes(S) ((S) | (S - 1))

// in binary representation
void printSet(int vS) {
  printf("S = %2d = ", vS);
  stack<int> st;
  while (vS)
    st.push(vS % 2), vS /= 2;
  // to reverse the print order
  while (!st.empty())
    printf("%d", st.top()), st.pop();
  printf("\n");
}
```

3.2 Union–Find

```
class UnionFind {
private:
  vi p, rank, setSize;
  int numSets;
public:
  UnionFind(int N) {
    setSize.assign(N, 1);
    numSets = N;
    rank.assign(N, 0);
    p.assign(N, 0);
        for (int i = 0; i < N; i++)
      p[i] = i;
  }
  int findSet(int i) {
    return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
  bool isSameSet(int i, int j) {
    return findSet(i) == findSet(j); }
  void unionSet(int i, int j) {
    if (!isSameSet(i, j)) {
      numSets--;
      int x = findSet(i), y = findSet(j);
      if (rank[x] > rank[y]) {
        p[y] = x; setSize[x] += setSize[y]; }
      else {
        p[x] = y; setSize[y] += setSize[x];
        if (rank[x] == rank[y]) rank[y]++; }
    }
  }
  int numDisjointSets() { return numSets; }
  int sizeOfSet(int i) { return setSize[findSet(i)]; }
};
```

3.3 Fenwick Tree

```
class FenwickTree {
private: vi ft;
public:
    FenwickTree(int n) { ft.assign(n+1, 0); }
    int rsq(int b){
        int sum = 0;
        for(; b; b -= (b & (-b))) sum += ft[b];
        return sum;
    }
    int rsq(int a, int b){
        return rsq(b) - (a == 1 ? 0 : rsq(a-1));
    }
    void update(int k, int v){
        for(; k < (int)ft.size(); k += (k & (-k))) ft[k
            ] += v;
    }
};
```

3.4 Segment Tree

```
#include <stdio.h>
#include <vector>
#include <string>
using namespace std;

int comp(int a, int b){
  return a + b;
}

class SegmentTree {
private:
    vector<int> st, A;
    int size;
#define left(p) (p << 1)
#define right(p) ((p << 1) + 1)
    void build(int p, int l, int r){
        if(l == r) st[p] = A[l];
        else {
            build(left(p), l, (l+r)/2);
            build(right(p), (l+r)/2+1, r);
            st[p] = comp(st[left(p)], st[right(p)]);
        }
    }
    int query(int p, int l, int r, int a, int b){
        if(a > r || b < l) return 1;
        if(l >= a && r <= b) return st[p];
        int p1 = query(left(p), l, (l+r)/2, a, b);
        int p2 = query(right(p), (l+r)/2+1, r, a, b);
        return comp(p1, p2);
    }
    void update(int p, int l, int r, int ind, int v){
        if(ind > r || ind < l) return;
        else if(l == ind && r == ind){
            A[ind] = v;
            st[p] = A[l];
        }else {
            update(left(p), l, (l+r)/2, ind, v);
            update(right(p), (l+r)/2+1, r, ind, v);
            st[p] = comp(st[left(p)], st[right(p)]);
        }
    }
public:
    SegmentTree(vector<int> &_A){
        A = _A;
        size = A.size();
        st.assign(4*size, 0);
        build(1, 0, size-1);
    }
    int query(int a, int b){  return query(1, 0, size
        -1, a, b); }
    void update(int ind, int v) { update(1, 0, size-1,
        ind, v); }
};
```

3.5 Segment Tree com Lazy Propagation

Permite consultas em intervalo e atualização de elementos em intervalo.

```
#include <stdio.h>
#include <vector>
#include <string>
using namespace std;

int comp(int a, int b){
     return a + b;
}

class SegmentTree {
private:
    vector<int> st, lazy;
    int size;
#define left(p) (p << 1)
#define right(p) ((p << 1) + 1)
    void push(int p, int l, int r){
        if(l != r){
            lazy[right(p)] += lazy[p];
            lazy[left(p)] += lazy[p];
        }
        st[p] += (r - l + 1) * lazy[p];
        lazy[p] = 0;
    }
    int query(int p, int l, int r, int a, int b){
        push(p, l, r);
        if(a > r || b < l) return 0;
        if(l >= a && r <= b) return st[p];
        int p1 = query(left(p), l, (l+r)/2, a, b);
        int p2 = query(right(p), (l+r)/2+1, r, a, b);
        return comp(p1, p2);
    }
    void update(int p, int l, int r, int a, int b, int
        k){
        push(p, l, r);
        if(a > r || b < l) return;
        else if(l >= a && r <= b) {
            lazy[p] = k;
            push(p, l, r);
        }else {
            update(left(p), l, (l+r)/2, a, b, k);
            update(right(p), (l+r)/2+1, r, a, b, k);
            st[p] = comp(st[left(p)], st[right(p)]);
        }
    }
public:
    SegmentTree(int sz){
        size = sz;
        st.assign(4*size, 0);
        lazy.assign(4*size, 0);
    }
    int query(int a, int b){  return query(1, 0, size
        -1, a, b); }
    void update(int a, int b, int k) { update(1, 0,
        size-1, a, b, k); }
};
```

3.6 Segment Tree com struct

Exemplo de uso no URI 1477 - Homem, Elefante e Rato. O método shift move todos os elementos do vetor N vezes para trás (circular). Modificar na hora do build, e push/update.

```
#include <stdio.h>
#include <vector>
using namespace std;

struct contagem {
    int cnt[3];
    void shift(int n){
        n %= 3;
        int tmp[3];
        for(int i = 0; i < 3; i++) tmp[i] = cnt[i];
        for(int i = 0; i < 3; i++)
            cnt[i] = tmp[(i - n + 3) % 3];
    }
 };

 contagem operator + (const contagem a, const contagem
     b){
    contagem res;
    for(int i = 0; i < 3; i++) res.cnt[i] = a.cnt[i] +
        b.cnt[i];
    return res;
 }


class SegmentTree {
private:
    vector<contagem> st;
    vector<int> lazy;
    int size;

#define left(p) (p << 1)
#define right(p) ((p << 1) + 1)
    void build(int p, int l, int r){
        if(l == r){
            st[p].cnt[0] = 1;
        }else {
            build(left(p), l, (l+r)/2);
            build(right(p), (l+r)/2+1, r);
            st[p] = st[left(p)] + st[right(p)];
        }
    }
    void push(int p, int l, int r){
        st[p].shift(lazy[p]);
        if(l != r){
            lazy[left(p)] += lazy[p];
            lazy[right(p)] += lazy[p];
        }
        lazy[p] = 0;
    }
    contagem query(int p, int l, int r, int a, int b){
        push(p, l, r);
        if(a > r || b < l) return contagem();
        if(l >= a && r <= b) return st[p];
        contagem p1 = query(left(p), l, (l+r)/2, a, b);
        contagem p2 = query(right(p), (l+r)/2+1, r, a,
            b);
        return p1 + p2;
    }
    void update(int p, int l, int r, int a, int b, int
        v){
        push(p, l, r);
        if(a > r || b < l) return;
        else if(l >= a && r <= b){
            lazy[p] += v;
            push(p, l, r);
        }else{
            update(left(p), l, (l+r)/2, a, b, v);
            update(right(p), (l+r)/2+1, r, a, b, v);
            st[p] = st[left(p)] + st[right(p)];
        }
    }
public:
    SegmentTree(int N){
        size = N;
        st.assign(4*size, contagem());
        lazy.assign(4*size, 0LL);
        build(1, 0, size-1);
    }
    contagem query(int a, int b) { return query(1, 0,
        size-1, a, b); }
    void update(int a, int b, int v) { update(1, 0,
        size-1, a, b, v); }
};
```

3.7 Sparse Table

```
#define MAXN 112345
#define LOGMAXN 20
#define INF INT_MAX

int n;
int v[MAXN];
int st[MAXN][LOGMAXN];

void prep(){
    for(int i = n-1; i >= 0; i--){
        st[i][0] = v[i];
        for(int j = 1; j <= LOGMAXN; j++)
            st[i][j] = min(st[i][j-1], st[min(i + (1 <<
                 (j-1)), n-1)][j-1]);
    }
}

int query(int i, int j){
    int m = j - i + 1;
    if(m == 1) return v[i];
    m = (31 - __builtin_clz(m));
    return min(st[i][m], st[j-(1<<m)+1][m]);
}

void change(int i, int x){
    v[i] = x;
    prep();
}
```

3.8 Operações em Binary Search Tree

```
struct node {
  node* left;
  node* right;
  int data;
  node(int info):
    left(0),
    right(0),
    data(info) {}
};

struct node* insert(node *tree, int v){
  if(tree == NULL)
    tree = new node(v);
  else {
    if(v < tree->data) tree->left = insert(tree->left,
        v);
    else if(v > tree->data) tree->right = insert(tree->
        right, v);
  }
  return tree;
}

struct node *maxValueNode(node *tree){
  struct node* current = tree;
  while(current != NULL) current = current->right;
  return current;
}

struct node *deleteNode(node *root, int v){
    if(root == NULL) return root;
    if(v < root->data) root->left = deleteNode(root->
        left, v);
    else if(v > root->data) root->right = deleteNode(
        root->right, v);
    else{
        if(root->left == NULL && root->right == NULL){
            free(root);
            root = NULL;
        }else if(!(root->left && root->right)){
            node *temp = root;
            if(root->left != NULL)
                root = root->left;
            else root = root->right;
            free(temp);
        }else{
            node *temp = maxValueNode(root->left);
            root->data = temp->data;
            root->left = deleteNode(root->left, temp->
                data);
        }
    }
    return root;
}

string ans = "";
void prefixa(node *tree){
  if(tree != NULL) {
    ans += "␣" + to_string(tree->data);
    prefixa(tree->left);
    prefixa(tree->right);
  }
}

void infixa(node *tree){
  if(tree != NULL){
    infixa(tree->left);
    ans += "␣" + to_string(tree->data);
    infixa(tree->right);
  }
}

void posfixa(node *tree){
  if(tree != NULL){
    posfixa(tree->left);
    posfixa(tree->right);
    ans += "␣" + to_string(tree->data);
  }
}

bool search(node *tree, int v){
  if(tree == NULL) return false;
  if(tree->data == v) return true;
  if(v < tree->data) return search(tree->left, v);
  return search(tree->right, v);
}

// Iniciar arvore com node *tree = 0;
```

Capítulo 4

Programação Dinâmica

4.1 Algoritmo da Mochila

Implementação com uso de memória O(W)

```
int ks[MAX], ks2[MAX];
int knapsack(int W, int wt[], int v[], int n){
    memset(ks, 0, sizeof ks);
    memset(ks2, 0, sizeof ks2);
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= W; j++)
            if(j - wt[i-1] >= 0)
                ks2[j] = max(ks[j], ks[j-wt[i-1]] + v[i
                    -1]);
        for(int j = 1; j <= W; j++)
            ks[j] = ks2[j];
    }
    return ks[W];
}
```

Implementação iterativa, com uso de memória O(nW)

```
int ks[MAX], ks2[MAX];
int knapsack(int W, int wt[], int v[], int n){
    memset(ks, 0, sizeof ks);
    memset(ks2, 0, sizeof ks2);
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= W; j++)
            if(j - wt[i-1] >= 0)
                ks2[j] = max(ks[j], ks[j-wt[i-1]] + v[i
                    -1]);
        for(int j = 1; j <= W; j++)
            ks[j] = ks2[j];
    }
    return ks[W];
}
```

4.2 Edit Distance

```
int editDistance(string str1, string str2, int m, int n
    ){
    int dp[m+1][n+1];
    for(int i = 0; i <= m; i++)
        for(int j = 0; j <= n; j++){
            if(i == 0) dp[i][j] = j;
            else if(j == 0) dp[i][j] = i;
            else if(str1[i-1] == str2[j-1])
                dp[i][j] = dp[i-1][j-1];
            else dp[i][j] = 1 + min(dp[i][j-1], min(dp[
                i-1][j], dp[i-1][j-1]));
        }

    return dp[m][n];
}
```

4.3 Longest Common Subsequence

```
int LCS(string a, string b, int m, int n){
    int L[m+1][n+1];
    for(int i = 0; i <= m; i++){
        for(int j = 0; j <= n; j++){
            if(i == 0 || j == 0)
                L[i][j] = 0;
            else if(a[i-1] == b[j-1]){
                L[i][j] = L[i-1][j-1] + 1;
            }else L[i][j] = max(L[i-1][j], L[i][j-1]);
        }
    }
    return L[m][n];
}
```

4.4 Coin Change

```
#define MAX 100000
typedef long long ll;
ll ways[MAX];
int types[] = {1, 2, 3};

void coinChange(){
    ways[0] = 1;
    for(auto c : types)
        for(ll i = c; i <= MAX; i++)
            ways[i] += ways[i-c];
}
```

4.5 Subset Sum

```
vector<int> moedas;

bool isSubsetSum(int sum){
    vector<bool> possible(sum+1, false);
    possible[0] = true;
    for(auto c : moedas)
        for(int i = sum-c; i >= 0; i--)
            if(possible[i])
                possible[i+c] = true;
    return possible[sum];
}
```

4.6 Minimum number of coins

```
int value[MAX];
bool ready[MAX];
int coins[] = {1, 2, 3};

int solve(int x)
{
    if (x < 0) return INF;
    if (x == 0) return 0;
    if (ready[x]) return value[x];
    int best = INF;
    for (auto c : coins)
        best = min(best, solve(x - c) + 1);
    value[x] = best;
    ready[x] = true;
    return best;
}
```

4.7 Problema dos Pares mais Próximos

Implementação $O(n \log n)$ para achar os pares mais próximos segundo a distância euclidiana em uma array de pontos 2D. A implementação original é $O(n \log^2 n)$, mas para muitos pontos, é necessário otimizar com merge sort. Caso precise mudar para pontos inteiros, mudar *dist* para usar quadrado da distância e não esquecer de usar $1+\sqrt{d}$ em vez de d.

```
#include <cmath>
#include <algorithm>
#define MAXN 100309
#define INF 1e+30
using namespace std;

struct point {
    double x, y;
    point() { x = y = 0; }
    point(double _x, double _y) : x(_x), y(_y) {}
};

typedef pair<point, point> pp;

double dist(pp p) {
  double dx = p.first.x - p.second.x;
  double dy = p.first.y - p.second.y;
  return hypot(dx, dy);
}

point strip[MAXN];

pp closest(point *P, int l, int r) {
  if (r == l) return pp(point(INF, 0), point(-INF, 0));
  int m = (l + r) / 2, s1 = 0, s2;
  int midx = (P[m].x + P[m+1].x)/2;
  pp pl = closest(P, l, m);
  pp pr = closest(P, m+1, r);
  pp ans = dist(pl) > dist(pr) ? pr : pl;
  double d = dist(ans);
  for(int i = l; i <= m; i++) {
    if (midx - P[i].x < d) strip[s1++] = P[i];
  }
  s2 = s1;
  for(int i = m+1; i <= r; i++) {
    if (P[i].x - midx < d) strip[s2++] = P[i];
  }
  for(int j = 0, s = s1; j < s1; j++) {
    point p = strip[j];
    for(int i = s; i < s2; i++) {
      point q = strip[i];
      pp cur = pp(p, q);
      double dcur = dist(cur);
      if (d > dcur) {
        ans = cur; d = dcur;
      }
      if (q.y - p.y > d) break;
      if (p.y - q.y > d) s = i+1;
    }
  }
  int i = l, j = m+1, k = l;
  while(i <= m && j <= r) {
    if (P[i].y < P[j].y) strip[k++] = P[i++];
    else strip[k++] = P[j++];
  }
  while(i <= m) strip[k++] = P[i++];
  for(i = l; i < k; i++) P[i] = strip[i];
  return ans;
}
```

```
bool compx(point a, point b) {
    return a.x < b.x;
}
```

```
pp closest(point *P, int n){
  sort(P, P+n, compx);
  return closest(P, 0, n-1);
}
```

Capítulo 5

Grafos

5.1 Floyd-Warshall

```
#include <string.h>
#define MAX 512
#define INF 0x3f3f3f3f

int AG[MAX][MAX];
int N, M;

void floydWarshall(){
  for(int k = 0; k < N; k++)
      for(int i = 0; i < N; i++)
          for(int j = 0; j < N; j++)
              AG[i][j] = min(AG[i][j], AG[i][k] + AG[k
                  ][j]);
}

// memset(AG, INF, sizeof AG);
// AG[u][u] = AG[v][v] = 0;
```

5.2 Dijkstra

```
#include <vector>
#include <queue>
#define INF ((int)1e9)
using namespace std;
typedef pair<int, int> ii;
typedef vector<ii> vii;

int N, M;
vector<int> dist;
vector<vii> LG;

void dijkstra(int s){
  dist.assign(N, INF);
    dist[s] = 0;
    priority_queue<ii, vector<ii>, greater<ii> > Q;
    Q.push(ii(0, s));
    while(!Q.empty()){
      int u = Q.top().second; Q.pop();
        for(auto e : LG[u]){
          int v = e.first, w = e.second;
            if(dist[v] > dist[u] + w){
              dist[v] = dist[u] + w;
                Q.push(ii(dist[v], v));
            }
        }
    }
}
```

5.3 BFS

```
int bfs(int s, int t){
    dist.assign(N, INFTO);
    dist[s] = 0;
    queue<int> Q;
    Q.push(s);
    while(!Q.empty()){
        int u = Q.front();
        Q.pop();
        // if(u == t) break;
        for(int v : LG[u]){
            if(dist[v] > dist[u] + 1){
                dist[v] = dist[u] + 1;
                Q.push(v);
            }
        }
    }
    return dist[t];
}
```

5.4 Flood Fill (DFS)

Exemplo de uso no URI 2317 - Lobo Mau

```
typedef pair<int, int> ii;

int movX[] = {0,  0, 1, -1};
int movY[] = {1, -1, 0,  0};
char grid[MAX][MAX];

bool movValido(int i, int j){
    return i >= 0 && i < R && j >= 0 && j < C && grid[i
        ][j] != '#';
}

ii operator + (const ii a, const ii b){
    return ii(a.first + b.first, a.second + b.second);
}


ii dfs(int i, int j){
    ii count = ii(0, 0);
    if(grid[i][j] == 'v') count.second++;
    if(grid[i][j] == 'k') count.first++;
    grid[i][j] = '#';
    for(int x = 0; x < 4; x++)
        if(movValido(i+movX[x], j+movY[x]))
            count = count + dfs(i+movX[x], j+movY[x]);
    return count;
}
```

5.5 Kruskal - Minimum Spanning Tree

```
// Utilizar Union-Find
typedef pair<int, int> ii;
typedef pair<int, ii> aresta;

int kruskal(vector<aresta> arestas){
    int custo = 0;
    for(auto e : arestas){
        if(!isSameSet(e.second.first, e.second.second))
            {
            unionSet(e.second.first, e.second.second);
            custo += e.first;
        }
    }
    return custo;
}

// Dar sort no vetor de arestas
```

5.6 Encontrar ciclo com DFS

```
typedef vector<int> vi;

vi visited;
vector<vi> LG;
bool cycle = false;

void dfs(int s){
  visited[s] = 1;
    if(cycle) return;
    for(auto v : LG[s]){
      if(visited[v] == 1){
      cycle = true; return;
    }else if(!visited[v]) dfs(v);
    }
    visited[s] = 2;
}
```

5.7 Encontrar ciclo em grafo não-direcionado com Union-Find

```
bool hasCycle(vector<aresta> arestas, int N){
    UnionFind uf(N);
    for(auto e : arestas){
        if(uf.isSameSet(e.first, e.second)){
            return true;
        }else{
            uf.unionSet(e.first,e .second);
        }
    }
    return false;
}
```

5.8 Problema do Caixeiro Viajante

```
int dist[MAX][MAX], memo[MAX][1 << MAX];

int tsp(int pos, int bitmask){
  if(bitmask == (1 << (n+1)) - 1)
    return dist[pos][0];
  if(memo[pos][bitmask] != -1)
    return memo[pos][bitmask];
  int ans = 2000000000;
  for(int nxt = 0; nxt <= n; nxt++)
    if(nxt != pos && !(bitmask & (1 << nxt)))
      ans = min(ans, dist[pos][nxt] + tsp(nxt, bitmask | (1 << nxt)));
  return memo[pos][bitmask] = ans;
}
```

5.9 Algoritmo de Tarjan (SCC)

```
#define UNVISITED -1
typedef vector<int> vi;
int N, dfsNumberCounter, numSCC;
vi num, low, S, visited;
vector<vi> LG;

void tarjanSCC(int u){
    low[u] = num[u] = dfsNumberCounter++;
    S.push_back(u);
    visited[u] = 1;
    for(auto v : LG[u]){
        if(num[v] == UNVISITED) tarjanSCC(v);
        if(visited[v]) low[u] = min(low[u] , low[v]);
    }
    if(low[u] == num[u]) {
        numSCC++;
        while(1){
            int v = S.back(); S.pop_back(); visited[v]
                = 0;
            if(u == v) break;
        }
    }
}
```

5.10 Lowest Common Ancestor (LCA)

$P[i][j]$ = o 2^j-ésimo pai do i-ésimo nó. $D[i][j]$ = distância para o 2^j-ésimo pai do i-ésimo nó. $computeP(root)$ computa as matrizes P e D em $O(n \log n)$. $LCA(u,v)$ retorna um par (LCA, distância) dos nós u e v em $O(\log n)$. CUIDADO: ele usa o tamanho da árvore N e adota indexação em 1!

```
#include <iostream>
#include <string.h>
#include <vector>
using namespace std;
#define MAXN 212345
#define MAXLOGN 20

typedef long long ll;

ll comp(ll a, ll b) { return a + b; }

typedef pair<int, ll> ii;

vector<ii> adjList[MAXN];
int level[MAXN], N;
int P[MAXN][MAXLOGN];
ll D[MAXN][MAXLOGN];

void depthdfs(int u) {
    for(auto i : adjList[u]){
    int v = i.first;
        ll w = i.second;
    if (v == P[u][0]) continue;
    P[v][0] = u; D[v][0] = w;
    level[v] = 1 + level[u];
    depthdfs(v);
  }
}

void computeP(int root) {
  level[root] = 0;
  P[root][0] = root; D[root][0] = 0;
  depthdfs(root);
  for(int j = 1; j < MAXLOGN; j++)
    for(int i = 0; i < N; i++) {
      P[i][j] = P[P[i][j-1]][j-1];
      D[i][j] = comp(D[P[i][j-1]][j-1], D[i][j-1]);
    }
}

ii LCA(int u, int v) {
  if (level[u] > level[v]) swap(u, v);
  int d = level[v] - level[u];
  ll ans = 0;
  for(int i = 0; i < MAXLOGN; i++) {
    if (d & (1<<i)) {
      ans = comp(ans, D[v][i]);
      v = P[v][i];
    }
  }
  if (u == v) return ii(u, ans);
  for(int i = MAXLOGN-1; i >= 0; i--)
    while(P[u][i] != P[v][i]) {
      ans = comp(ans, D[v][i]);
      ans = comp(ans, D[u][i]);
      u = P[u][i]; v = P[v][i];
    }
  ans = comp(ans, D[v][0]);
  ans = comp(ans, D[u][0]);
  return ii(P[u][0], ans);
}
```

5.11 Ordenação Topológica

Inicializar vis como false. toposort guarda a ordenação na ordem inversa!

```
#include <vector>
using namespace std;
#define MAXN 1009

int vis[MAXN];
vector<int> adjList[MAXN];
vector<int> toposort; //Ordem reversa!

void ts(int u) {
  vis[u] = true;
  for (int j = 0, v; j < (int)adjList[u].size(); j++) {
    v = adjList[u][j];
    if (!vis[v]) ts(v);
  }
  toposort.push_back(u);
}
```

5.12 Dinic - Max Flow

```
/*
    Dinic (Max Flow)
    Complexidade O(V^2E)
    -> Nos vao de 1 a V
    -> Setar numero de nos V, source S e sink T
    -> Setar limite de ns MAXV
    -> Adicionar adjacencias usando add_adj()
    -> Chamar clear() antes de cada teste
*/

const int MAXV = 100, INF = 0x7FFFFFFF;

struct edge {
    int dest, rev, cap, f;
    edge(int dest, int rev, int cap) : dest(dest), rev(
        rev), cap(cap) {
        f = 0;
    }
};

int dist[MAXV + 1], ptr[MAXV + 1];
vector<edge> adj[MAXV + 1];
int V, S, T;

void add_edge(int a, int b, int cap) {
    adj[a].push_back(edge(b, adj[b].size(), cap));
    adj[b].push_back(edge(a, adj[a].size() - 1, 0));
}

bool bfs() {
    for(int i = 1; i <= V; i++)
        dist[i] = -1;
    dist[S] = 0;
    queue<int> q;
    q.push(S);
    while(!q.empty()) {
        int v = q.front();
        q.pop();
        for(int i = 0; i < adj[v].size(); i++) {
            edge e = adj[v][i];
            if(dist[e.dest] == -1 && e.cap > e.f) {
                dist[e.dest] = dist[v] + 1;
                q.push(e.dest);
            }
        }
    }
    return dist[T] != -1;
}

int dfs(int v, int f) {
    if(v == T) return f;
    for(; ptr[v] < adj[v].size(); ++ptr[v]) {
        edge &e = adj[v][ptr[v]];
        if(dist[e.dest] == dist[v] + 1 && e.cap > e.f)
            {
            int df = dfs(e.dest, min(f, e.cap - e.f));
            if(df > 0) {
                e.f += df;
                adj[e.dest][e.rev].f -= df;
                return df;
            }
        }
    }
    return 0;
}

int max_flow() {
    int flow = 0;
    while(bfs()) {
        for(int i = 1; i <= V; i++)
            ptr[i] = 0;
        while(true) {
            int df = dfs(S, INF);
            if(df == 0) break;
            flow += df;
        }
    }
    return flow;
}

void clear() {
    for(int i = 1; i <= V; i++)
        adj[i].clear();
}
```

Capítulo 6

Matemática

6.1 Progressão Aritmética

Soma dos termos entre 1 e N: $S_n = \frac{n \times (a_1 + a_n)}{2}$

n-ésimo termo de uma $P.A$: $a_n = a_1 + (n-1) \times r$

6.2 Múltiplos positivos de k num intervalo

O número de múltiplos positivos $m(k)$ de k no intervalo $[1, N]$ é igual a $m(k) = \frac{N}{k}$.

6.3 Número par ou ímpar de divisores

Números que são quadrados perfeitos tem um número ímpar de divisores, enquanto o resto tem um número par.

6.4 Número de quadrados perfeitos de A a B

$N = floor(sqrt(B)) - ceil(sqrt(A)) + 1$

6.5 Quadrados e retângulos em um Grid de N lados com K dimensões

Quadrados = $N^K + (N-1)^K + (N-2)^K$ até 1
Retângulos = $\frac{(N^K * (N+1)^K)}{2^K} - Quadrados$

6.6 Exponenciação binária

```
ll exp_by_squaring(ll x, ll n){
    if(n < 0) {
        x = 1 / x;
        n = -n;
    }
    if(n == 0) return 1;
    ll y = 1;
    while(n > 1) {
        if(n&1) y *= x;
        x *= x;
        n /= 2;
    }
    return x * y;
}
```

6.7 Sieve of Eratosthenes

```
void SieveOfEratosthenes(int n){
  bool prime[n+1];
  memset(prime, true, sizeof prime);
  for(int p = 2; p <= n; p++){
    if(prime[p] == true){
      for(int i = p*2; i <= n; i += p)
        prime[i] = false;
    }
  }
}
```

6.8 Exponenciação Modular

```
typedef long long ll;

ll power(ll a, ll b){
  if(b == 0) return 1;
    if(b&1) return (a * power(a, b-1)) % MOD;
    ll c = power(a, b >> 1);
    return (c*c) % MOD;
}
```

6.9 String para número com mod

Ler números muito grandes como string

```
int mod(string num, int a){
  int res = 0;
    for(int i = 0; i < num.length(); i++)
        res = (res*10 + (int)num[i] - '0') % a;
    return res;
}
```

6.10 Inverso multiplicativo

Se MOD é primo:

```
power(num, MOD-2) % MOD
```

Se MOD e num são coprimos:

```
int modInverse(int a, int m){
  int m0 = m;
  int y = 0, x = 1;
  if(m == 1) return 0;
  while(a > 1){
    int q = a / m;
    int t = m;
    m = a % m, a = t;
    t = y;
    y = x - q*y;
    x = t;
  }
  if(x < 0) x += m0;
  return x;
}
```

6.11 Probabilidade / Gambler's ruin

Fair coin flipping: $P_1 = \frac{n_1}{n_1 + n_2}$

Unfair coin flipping: $P_1 = \frac{1 - (\frac{p}{q})^{n_1}}{1 - (\frac{p}{q})^{n_1+n_2}}$

6.12 Quantidade de pontos inteiros embaixo de uma reta entre dois pontos

$gcd(abs(x2 - x1), abs(y2 - y1)) - 1$

Capítulo 7

Strings

7.1 Knuth–Morris–Pratt

```
#include <cstdio>
#include <cstring>
using namespace std;

#define MAX_N 100010

char T[MAX_N], P[MAX_N]; // T = text, P = pattern
int b[MAX_N], n, m;
// b = back table, n = length of T, m = length of P

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

void kmpSearch() {
  int i = 0, j = 0;
  while (i < n) {
    while (j >= 0 && T[i] != P[j]) j = b[j];
    i++; j++;
    if (j == m) {
      printf("P is found at index %d in T\n", i - j);
      j = b[j];
    }
  }
}
```

7.2 Rabin-Karp

String matching em $O(m)$, construtor $O(n)$. $hash(i,j) = \sum_{k=i}^{j} s[k]p^{k-i} \mod m$, $O(1)$. Usar hashing para verificar igualdade de strings. Se necessário, fazer com 3 ou 5 pares P e M diferentes. Usar M potência de dois é pedir pra ser hackeado. Abaixo, a probabilidade de colisão por complexidade e M para $n = 10^6$ (tabela por Antti Laaksonen). Observe o paradoxo do aniversário: em uma sala com 23 pessoas, a probabilidade que duas tenham aniversário no mesmo dia é de 50%.

Cenário	Probabilidade	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
entre duas strings $O(1)$	$1/M$	0.001000	0.000001	0.000000	0.000000	0.000000	0.000000
uma string contra n $O(n)$	$1-(1-1/M)^n$	1.000000	0.632121	0.001000	0.000000	0.000000	0.000000
pares de n strings $O(n^2)$	$1-M!/(M^n(M-n)!)$	1.000000	1.000000	1.000000	0.393469	0.000500	0.000001

```
#include <vector>
using namespace std;
typedef long long ll;

class RabinKarp {
  ll m;
  vector<int> pw, hsh;
public:
  RabinKarp() {}
  RabinKarp(char str[], ll p, ll _m) : m(_m) {
    int n = strlen(str);
    pw.resize(n); hsh.resize(n);
    hsh[0] = str[0]; pw[0] = 1;
    for(int i = 1; i < n; i++) {
      hsh[i] = (hsh[i-1] * p + str[i]) % m;
      pw[i] = (pw[i-1] * p) % m;
    }
  }
  ll hash(int i, int j) {
    ll ans = hsh[j];
    if (i > 0) ans = (ans - ((hsh[i-1]*1ll*pw[j-i+1])%m
        ) + m) % m;
    return ans;
  }
};
```

7.3 Repetend: menor período de uma string

Retorna o menor que k tal que $k|n$ e a string pode ser decomposta em n/k strings iguais.

```
#include <cstring>
#define MAXN 100009

int repetend(char* s) {
    int n = strlen(s);
    int nxt[n+1];
    nxt[0] = -1;
    for (int i = 1; i <= n; i++) {
        int j = nxt[i - 1];
        while (j >= 0 && s[j] != s[i - 1])
            j = nxt[j];
        nxt[i] = j + 1;
    }
    int a = n - nxt[n];
    if (n % a == 0) return a;
    return n;
}
```

7.4 Suffix Array e Longuest Common Prefix

computeSA computa a Suffix Array em $O(n \log n)$. *computeLCP* computa o Longuest Common Prefix em $O(n)$. $LCP[i]$ guarda o tamanho do maior prefixo comum entre $SA[i]$ e $SA[i-1]$. A Longest Repeated Substring é o valor do maior LCP. CUIDADO: ele coloca '$' no final e ele aparece na posição zero da SA!

```
#define MAXN 100009
#include <algorithm>
#include <cstring>
using namespace std;

class SuffixArray {
  int RA[MAXN], tempRA[MAXN];
  int tempSA[MAXN], c[MAXN], n;
  int Phi[MAXN], PLCP[MAXN]; //para LCP
  void countingSort(int k, int SA[]) { // O(n)
    int i, sum, maxi = max(300, n);
    memset(c, 0, sizeof c);
    for (i = 0; i < n; i++) c[i + k < n ? RA[i + k] :
        0]++;
    for (i = sum = 0; i < maxi; i++) {
      int t = c[i];
      c[i] = sum;
      sum += t;
    }
    for (i = 0; i < n; i++)
      tempSA[c[SA[i]+k < n ? RA[SA[i]+k] : 0]++] = SA[i
          ];
    for (i = 0; i < n; i++) SA[i] = tempSA[i];
  }
public:
  void constructSA(char str[], int SA[]) { // O(nlogn)
    int i, k, r; n = strlen(str);
    str[n++] = '$'; str[n] = 0;
    for (i = 0; i < n; i++) RA[i] = str[i];
    for (i = 0; i < n; i++) SA[i] = i;
    for (k = 1; k < n; k <<= 1) {
      countingSort(k, SA);
      countingSort(0, SA);
      tempRA[SA[0]] = r = 0;
      for (i = 1; i < n; i++)
        tempRA[SA[i]] = (RA[SA[i]] == RA[SA[i-1]] && RA
            [SA[i]+k] == RA[SA[i-1]+k]) ? r : ++r;
      for (i = 0; i < n; i++) RA[i] = tempRA[i];
      if (RA[SA[n-1]] == n-1) break;
    }
  }
  void computeLCP(char str[], int SA[], int LCP[]) { //
       O(n)
    int i, L; n = strlen(str);
    Phi[SA[0]] = -1;
    for (i = 1; i < n; i++)  Phi[SA[i]] = SA[i-1];
    for (i = L = 0; i < n; i++) {
      if (Phi[i] == -1) {
        PLCP[i] = 0; continue;
      }
      while (str[i + L] == str[Phi[i] + L]) L++;
      PLCP[i] = L;
      L = max(L-1, 0);
    }
    for (i = 0; i < n; i++) LCP[i] = PLCP[SA[i]];
  }
};
```

7.5 Função Z e Algoritmo Z

Função Z é uma função tal que $z[i]$ é máximo e $str[j] = str[i+j]$, $0 \le j \le z[i]$. O algoritmo Z computa todos os $z[i]$, $0 \le i < n$, em $O(n)$. $z[0] = 0$.

```
#include <cstring>
#include <algorithm>
using namespace std;

void zfunction(char s[], int z[]) {
  int n = strlen(s);
  fill(z, z+n, 0);
  for (int i=1, l=0, r=0; i<n; ++i) {
    if (i <= r) z[i] = min(r-i+1, z[i-l]);
    while (i+z[i] < n && s[z[i]] == s[i+z[i]])
      ++z[i];
    if (i+z[i]-1 > r)
      l = i,  r = i+z[i]-1;
  }
}
```

7.6 Algoritmo de Manacher

Usado para achar a maior substring palíndromo. A função *manacher* calcula a array L. $L[i]$ é o máximo valor possível tal que $str[i+j] = str[i-j]$, $0 \leq j \leq L[i]$. Pra calcular os palíndromos pares, basta adicionar '|' entre todos os caracateres e calcular o maior valor de L da string.

```
#include <cstring>
#include <algorithm>
using namespace std;

void manacher(char str[], int L[]) {
  int n = strlen(str), c = 0, r = 0;
  for(int i = 0; i < n; i++) {
    if(i < r && 2*c >= i)
      L[i] = min(L[2*c-i], r-i);
    else L[i] = 0;
    while(i-L[i]-1 >= 0 && i+L[i]+1 < n &&
      str[i-L[i]-1] == str[i+L[i]+1]) L[i]++;
    if(i+L[i]>r) { c=i; r=i+L[i]; }
  }
}
```

7.7 Aho-Corasick

Resolve o problema de achar ocorrências de um dicionário em um texto em $O(n)$, onde n é o comprimento do texto. Pré-processamento: $O(mk)$, onde m é a soma do número de caracteres de todas as palavras do dicionário e k é o tamanho do alfabeto. Cuidado: o número de matches tem pior caso $O(n\sqrt{m})$! Guardar apenas o número de matches, se for o que o problema pedir. Caso o alfabeto seja muito grande, trocar nxt por um *map* ou usar lista de adjacência.

```
#include <cstring>
#include <queue>
#include <vector>
using namespace std;
#define ALFA 62
#define MAXS 2000009

typedef pair<int, int> ii;

int nxt[MAXS][ALFA], fail[MAXS], cnt = 0;
vector<ii> pats[MAXS];

class AhoCorasick {
private:
  int root;
  int suffix(int x, int c) {
    while (x != root && nxt[x][c] == 0) x = fail[x];
    return nxt[x][c] ? nxt[x][c] : root;
  }
  int newnode() {
    int x = ++cnt;
    fail[x] = 0; pats[x].clear();
    for(int c = 0; c < ALFA; c++) nxt[x][c] = 0;
    return x;
  }
  inline int reduce(char c) {
    if (c >= 'a' && c <= 'z') return c - 'a';
    if (c >= 'A' && c <= 'Z') return c - 'A' + ('z'-'a'
        +1);
    if (c >= '0' && c <= '9') return c - '0' + 2*('z'-'
        a'+1);
    return -1;
  }
public:
  AhoCorasick() { root = newnode(); }
  void setfails() {
    queue<int> q;
    int x, y;
    q.push(root);
    while (!q.empty()) {
      x = q.front(); q.pop();
      for (int c = 0; c < ALFA; c++) {
        y = nxt[x][c];
        if (y == 0) continue;
        fail[y] = x == root ? x : suffix(fail[x], c);
        pats[y].insert(pats[y].end(),
          pats[fail[y]].begin(), pats[fail[y]].end());
        q.push(y);
      }
    }
  }
  void insert(const char* s, int id) {
    int len = strlen(s);
    int x = root;
    for (int i = 0; i < len; i++) {
      int & y = nxt[x][reduce(s[i])];
      if (y == 0 || y == root) {
        y = newnode();
      }
      x = y;
    }
    pats[x].push_back(ii(id, len));
  }
  vector<ii> match(const char *s) { //(id, pos)
    int x = root;
    vector<ii> ans;
    for (int i = 0; s[i]; i++) {
      x = suffix(x, reduce(s[i]));
      for(int j = 0; j < (int)pats[x].size(); j++) {
        ii cur = pats[x][j];
        ans.push_back(ii(cur.first, i - cur.second + 1)
            );
      }
    }
    return ans;
  }
};
```

7.8 Autômato de Sufixos

Constrói o autômato de sufixos online em $O(n)$ de tempo e $O(nk)$ memória, em que n é a soma dos tamanhos da strings e k é o tamanho do alfabeto. Caso k seja muito grande, trocar nxt por um *map*. Resolve os problemas de string matching com

contagem de aparições em $O(m)$, número de substrings diferentes em $O(n)$, maior string repetida em $O(n)$ e maior substring comum em $O(m)$. Os estados terminais são $last, link(last), link(link(last)), \cdots$.

```
#include <cstring>
#include <queue>
using namespace std;
#define MAXS 500009 // 2*MAXN
#define ALFA 26

class SuffixAutomaton {
  int len[MAXS], link[MAXS], cnt[MAXS];
  int nxt[MAXS][ALFA], sz, last, root;
  int newnode() {
    int x = ++sz;
    len[x] = 0; link[x] = -1; cnt[x] = 1;
    for(int c = 0; c < ALFA; c++) nxt[x][c] = 0;
    return x;
  }
  inline int reduce(char c) { return c - 'a'; }
public:
  SuffixAutomaton() { clear(); }
  void clear() {
    sz = 0;
    root = last = newnode();
  }
  void insert(const char *s) {
    for(int i = 0; s[i]; i++) extend(reduce(s[i]));
  }
  void extend(int c) {
    int cur = newnode(), p;
    len[cur] = len[last] + 1;
    for(p = last; p != -1 && !nxt[p][c]; p = link[p]) {
      nxt[p][c] = cur;
    }
    if (p == -1) link[cur] = root;
    else {
      int q = nxt[p][c];
      if (len[p] + 1 == len[q]) link[cur] = q;
      else {
        int clone = newnode();
        len[clone] = len[p] + 1;
        for(int i = 0; i < ALFA; i++)
          nxt[clone][i] = nxt[q][i];
        link[clone] = link[q];
        cnt[clone] = 0;
        while(p != -1 && nxt[p][c] == q) {
          nxt[p][c] = clone;
          p = link[p];
        }
        link[q] = link[cur] = clone;
      }
    }
    last = cur;
  }
  bool contains(const char *s) {
    for(int i = 0, p = root; s[i]; i++) {
      int c = reduce(s[i]);
      if (!nxt[p][c]) return false;
      p = nxt[p][c];
    }
    return true;
  }
  long long numDifSubstrings() {
    long long ans = 0;
    for(int i=root+1; i<=sz; i++)
      ans += len[i] - len[link[i]];
    return ans;
  }
  int longestCommonSubstring(const char *s) {
    int cur = root, curlen = 0, ans = 0;
    for(int i = 0; s[i]; i++) {
      int c = reduce(s[i]);
      while(cur != root && !nxt[cur][c]) {
        cur = link[cur];
        curlen = len[cur];
      }
      if (nxt[cur][c]) {
        cur = nxt[cur][c];
        curlen++;
      }
      if (ans < curlen) ans = curlen;
    }
    return ans;
  }
private:
  int deg[MAXS];
public: // chamar computeCnt antes!
  void computeCnt() {
    fill(deg, deg+sz+1, 0);
    for(int i=root+1; i<=sz; i++)
      deg[link[i]]++;
    queue<int> q;
    for(int i=root+1; i<=sz; i++)
      if (deg[i] == 0) q.push(i);
    while(!q.empty()) {
      int i = q.front(); q.pop();
      if (i <= root) continue;
      int j = link[i];
      cnt[j] += cnt[i];
      if ((--deg[j]) == 0) q.push(j);
    }
  }
  int nmatches(const char *s) {
    int p = root;
    for(int i = 0; s[i]; i++) {
      int c = reduce(s[i]);
      if (!nxt[p][c]) return 0;
      p = nxt[p][c];
    }
    return cnt[p];
  }
  int longestRepeatedSubstring(int times) {
    int ans = 0;
    for(int i=root; i<=sz; i++) {
      if (cnt[i] >= times && ans < len[i]) {
        ans = len[i];
      }
    }
    return ans;
  }
};
```

7.9 Needleman–Wunsch (Alinhamento de Strings)

```
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

int main() {
  char A[20] = "ACAATCC", B[20] = "AGCATGC";
  int n = (int)strlen(A), m = (int)strlen(B);
  int i, j, table[20][20];
    // Needleman Wunsnch's algorithm
```

```
  memset(table, 0, sizeof table);
  // insert/delete = -1 point
  for (i = 1; i <= n; i++)
    table[i][0] = i * -1;
  for (j = 1; j <= m; j++)
    table[0][j] = j * -1;

  for (i = 1; i <= n; i++)
    for (j = 1; j <= m; j++) {
      // match = 2 points, mismatch = -1 point
      table[i][j] = table[i - 1][j - 1] + (A[i - 1] ==
          B[j - 1] ? 2 : -1); // cost for match or
          mismatches
      // insert/delete = -1 point
      table[i][j] = max(table[i][j], table[i - 1][j] -
          1); // delete
      table[i][j] = max(table[i][j], table[i][j - 1] -
          1); // insert
    }

  printf("DP table:\n");
  for (i = 0; i <= n; i++) {
    for (j = 0; j <= m; j++)
      printf("%3d", table[i][j]);
    printf("\n");
  }
  printf("Maximum Alignment Score: %d\n", table[n][m]);

  return 0;
}
```

Capítulo 8

Geometria

8.1 Ponto 2D e segmentos de reta

Ponto com double em 2D com algumas funcionalidades: distância, produto interno, produto vetorial (componente z), teste counter-clockwise, teste de colinearidade, rotação em relação ao centro do plano, projeção de u sobre v, ponto dentro de segmento de reta, intersecção de retas, teste de paralelidade, teste de intersecção de segmentos de reta, ponto mais próximo ao segmento de reta.

```
#include <cmath>
#include <vector>
using namespace std;
#define EPS 1e-9

struct point {
  double x, y;
  point() { x = y = 0.0; }
  point(double _x, double _y) : x(_x), y(_y) {}
  double norm() { return hypot(x, y); }
  point normalized() {
    return point(x,y)*(1.0/norm());
  }
  double angle() { return atan2(y, x); }
  double polarAngle() {
    double a = atan2(y, x);
    return a < 0 ? a + 2*acos(-1.0) : a;
  }
  bool operator < (point other) const {
    if (fabs(x - other.x) > EPS) return x < other.x;
    else return y < other.y;
  }
  bool operator == (point other) const {
    return (fabs(x - other.x) < EPS && (fabs(y - other.
        y) < EPS));
  }
  point operator +(point other) const {
    return point(x + other.x, y + other.y);
  }
  point operator -(point other) const {
    return point(x - other.x, y - other.y);
  }
  point operator *(double k) const {
    return point(x*k, y*k);
  }
};
double dist(point p1, point p2) {
  return hypot(p1.x - p2.x, p1.y - p2.y);
}
double inner(point p1, point p2) {
  return p1.x*p2.x + p1.y*p2.y;
}
double cross(point p1, point p2) {
  return p1.x*p2.y - p1.y*p2.x;
}
bool ccw(point p, point q, point r) {
  return cross(q-p, r-p) > 0;
}
bool collinear(point p, point q, point r) {
  return fabs(cross(p-q, r-p)) < EPS;
}
point rotate(point p, double rad) {
  return point(p.x * cos(rad) - p.y * sin(rad),
  p.x * sin(rad) + p.y * cos(rad));
}
double angle(point a, point o, point b) {
  return acos(inner(a-o, b-o) / (dist(o,a)*dist(o,b)));
}
point proj(point u, point v) {
  return v*(inner(u,v)/inner(v,v));
}
bool between(point p, point q, point r) {
  return collinear(p, q, r) && inner(p - q, r - q) <=
      0;
}
point lineIntersectSeg(point p, point q, point A, point
     B) {
  double c = cross(A-B, p-q);
  double a = cross(A, B);
  double b = cross(p, q);
  return ((p-q)*(a/c)) - ((A-B)*(b/c));
}
bool parallel(point a, point b) {
  return fabs(cross(a, b)) < EPS;
}
bool segIntersects(point a, point b, point p, point q)
    {
  if (parallel(a-b, p-q)) {
    return between(a, p, b) || between(a, q, b)
      || between(p, a, q) || between(p, b, q);
  }
  point i = lineIntersectSeg(a, b, p, q);
  return between(a, i, b) && between(p, i, q);
}
point closestToLineSegment(point p, point a, point b) {
  double u = inner(p-a, b-a) / inner(b-a, b-a);
  if (u < 0.0) return a;
  if (u > 1.0) return b;
  return a + ((b-a)*u);
}
```

8.2 Linha 2D

Algumas funções de reta e segmento de reta no plano 2D: dois pontos para reta, projeção e distância ponto-reta e lonto-segmento de reta. Reta representada da forma $ax + by + c = 0$. Se possivel fazemos $b = 1$.

```
struct line {
  double a, b, c;
  line() { a = b = c = NAN;}
  line(double _a, double _b, double _c) : a(_a), b(_b),
       c(_c) {}
};

line pointsToLine(point p1, point p2) {
  line l;
  if (fabs(p1.x - p2.x) < EPS && fabs(p1.y - p2.y) <
      EPS) {
    l.a = l.b = l.c = NAN;
  }
  else if (fabs(p1.x - p2.x) < EPS) {
    l.a = 1.0; l.b = 0.0; l.c = -p1.x;
  }
  else {
    l.a = -(p1.y - p2.y) / (p1.x - p2.x);
    l.b = 1.0;
    l.c = -(l.a * p1.x) - p1.y;
  }
  return l;
}

bool areParallel(line l1, line l2) {
  return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) <
      EPS);
}

bool areSame(line l1, line l2) {
  return areParallel(l1 ,l2) && (fabs(l1.c - l2.c) <
      EPS);
}

point intersection(line l1, line l2) {
  if (areParallel(l1, l2)) return point(NAN, NAN);
  point p;
  p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1
      .a * l2.b);
  if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
  else p.y = -(l2.a * p.x + l2.c);
  return p;
}

point projPointToLine(point u, line l) {
  point a, b;
  if (fabs(l.b-1.0)<EPS) {
    a = point(-l.c/l.a, 0.0);
    b = point(-l.c/l.a, 1.0);
  }
  else{
    a = point(0, -l.c/l.b);
    b = point(1, -(l.c+1.0)/l.b);
  }
  return a + proj(u-a, b-a);
}

double distToLine(point p, line l) {
  return dist(p, projPointToLine(p, l));
}
```

8.3 Círculo 2D

Círculo no plano 2D com algumas funcionalidades: área, comprimento de corda, área do setor, teste de intersecção com outro círculo, teste de ponto, pontos de retas tangentes (se o ponto estiver dentro *asin* retorna *nan*), circumcírculo e incírculo (divisão por zero se os pontos forem colineares).

```
struct circle{
  point c;
  double r;
  circle() { c = point(); r = 0; }
  circle(point _c, double _r) : c(_c), r(_r) {}
  double area() { return acos(-1.0)*r*r; }
  double chord(double rad) { return  2*r*sin(rad/2.0);
      }
  double sector(double rad) { return 0.5*rad*area()/
      acos(-1.0); }
  bool intersects(circle other) {
    return dist(c, other.c) < r + other.r;
  }
  bool contains(point p) { return dist(c, p) <= r + EPS
      ; }
  pair<point, point> getTangentPoint(point p) {
    double d1 = dist(p, c), theta = asin(r/d1);
    point p1 = rotate(c-p,-theta);
    point p2 = rotate(c-p,theta);
    p1 = p1*(sqrt(d1*d1-r*r)/d1)+p;
    p2 = p2*(sqrt(d1*d1-r*r)/d1)+p;
    return make_pair(p1,p2);
  }
};

circle circumcircle(point a, point b, point c) {
  circle ans;
  point u = point((b-a).y, -(b-a).x);
  point v = point((c-a).y, -(c-a).x);
  point n = (c-b)*0.5;
  double t = cross(u,n)/cross(v,u);
  ans.c = ((a+c)*0.5) + (v*t);
  ans.r = dist(ans.c, a);
  return ans;
}

int insideCircle(point p, circle c) {
  if (fabs(dist(p , c.c) - c.r)<EPS) return 1;
  else if (dist(p , c.c) < c.r) return 0;
  else return 2;
} //0 = inside/1 = border/2 = outside

circle incircle( point p1, point p2, point p3 ) {
    double m1=dist(p2, p3);
    double m2=dist(p1, p3);
    double m3=dist(p1, p2);
    point c = (p1*m1+p2*m2+p3*m3)*(1/(m1+m2+m3));
    double s = 0.5*(m1+m2+m3);
    double r = sqrt(s*(s-m1)*(s-m2)*(s-m3))/s;
    return circle(c, r);
}
```

8.4 Triângulo 2D

```
struct triangle{
  point a, b, c;
  triangle() { a = b = c = point(); }
  triangle(point _a, point _b, point _c) : a(_a), b(_b)
      , c(_c) {}
  double perimeter() { return dist(a,b) + dist(b,c) +
      dist(c,a); }
  double semiPerimeter() { return perimeter()/2.0; }
  double area() {
    double s = semiPerimeter(), ab = dist(a,b),
      bc = dist(b,c), ca = dist(c,a);
    return sqrt(s*(s-ab)*(s-bc)*(s-ca));
  }
  double rInCircle() {
    return area()/semiPerimeter();
  }
  circle inCircle() {
    return incircle(a,b,c);
  }
  double rCircumCircle() {
    return dist(a,b)*dist(b,c)*dist(c,a)/(4.0*area());
  }
  circle circumCircle() {
    return circumcircle(a,b,c);
  }
    int isInside(point p) {
    double u = cross(b-a,p-a)*cross(b-a,c-a);
    double v = cross(c-b,p-b)*cross(c-b,a-b);
    double w = cross(a-c,p-c)*cross(a-c,b-c);
    if (u > 0.0 && v > 0.0 && w > 0.0) return 0;
    if (u < 0.0 || v < 0.0 || w < 0.0) return 2;
    else return 1;
  } //0 = inside/ 1 = border/ 2 = outside
};

double rInCircle(point a, point b, point c) {
  return triangle(a,b,c).rInCircle();
}

double rCircumCircle(point a, point b, point c) {
  return triangle(a,b,c).rCircumCircle();
}

int isInsideTriangle(point a, point b, point c, point p
    ) {
  return triangle(a,b,c).isInside(p);
} //0 = inside/ 1 = border/ 2 = outside
```

8.5 Polígono 2D

```
#include <vector>
#include <algorithm>
using namespace std;

typedef vector<point> polygon;

double signedArea(polygon & P) {
  double result = 0.0;
  int n = P.size();
  for (int i = 0; i < n; i++) {
    result += cross(P[i], P[(i+1)%n]);
  }
  return result / 2.0;
}

int leftmostIndex(vector<point> & P) {
  int ans = 0;
  for(int i=1; i<(int)P.size(); i++) {
    if (P[i] < P[ans]) ans = i;
  }
  return ans;
}

polygon make_polygon(vector<point> P) {
  if (signedArea(P) < 0.0) reverse(P.begin(), P.end());
  int li = leftmostIndex(P);
  rotate(P.begin(), P.begin()+li, P.end());
  return P;
}

double perimeter(polygon & P) {
  double result = 0.0;
  for (int i = 0; i < (int)P.size()-1; i++) result +=
      dist(P[i], P[i+1]);
  return result;
}

double area(polygon & P) {
  return fabs(signedArea(P));
}

bool isConvex(polygon & P) {
  int n = (int)P.size();
  if (n < 3) return false;
  bool left = ccw(P[0], P[1], P[2]);
  for (int i = 1; i < n; i++) {
    if (ccw(P[i], P[(i+1)%n], P[(i+2)%n]) != left)
      return false;
  }
  return true;
}

bool inPolygon(polygon & P, point p) {
  if (P.size() == 0u) return false;
  double sum = 0.0;
  int n = P.size();
  for (int i = 0; i < n; i++) {
    if (ccw(p, P[i], P[(i+1)%n])) sum += angle(P[i], p,
        P[(i+1)%n]);
    else sum -= angle(P[i], p, P[(i+1)%n]);
  }
  return fabs(fabs(sum) - 2*acos(-1.0)) < EPS;
}

polygon cutPolygon(polygon & P, point a, point b) {
  vector<point> R;
  double left1, left2;
  int n = P.size();
  for (int i = 0; i < n; i++) {
    left1 = cross(b-a, P[i]-a);
    left2 = cross(b-a, P[(i+1)%n]-a);
    if (left1 > -EPS) R.push_back(P[i]);
    if (left1 * left2 < -EPS)
      R.push_back(lineIntersectSeg(P[i], P[(i+1)%n], a,
          b));
  }
  return make_polygon(R);
}
```

8.6 Convex Hull

Dado um conjunto de pontos, retorna o menor polígono que contém todos os pontos em $O(n \log n)$. Caso precise considerar os pontos no meio de uma aresta, trocar o teste *ccw* para ≥ 0. CUIDADO: Se todos os pontos forem colineares, vai dar RTE.

```
#include <algorithm>
using namespace std;

point pivot(0, 0);

bool angleCmp(point a, point b) {
  if (collinear(pivot, a, b))
    return inner(pivot-a, pivot-a) < inner(pivot-b,
        pivot-b);
  return cross(a-pivot, b-pivot) >= 0;
}

polygon convexHull(vector<point> P) {
  int i, j, n = (int)P.size();
  if (n <= 2) return P;
  int P0 = leftmostIndex(P);
  swap(P[0], P[P0]);
  pivot = P[0];
  sort(++P.begin(), P.end(), angleCmp);
  vector<point> S;
  S.push_back(P[n-1]);
  S.push_back(P[0]);
  S.push_back(P[1]);
  for(i = 2; i < n;) {
    j = (int)S.size()-1;
    if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]);
    else S.pop_back();
  }
  reverse(S.begin(), S.end());
  S.pop_back();
  reverse(S.begin(), S.end());
  return S;
}
```

8.7 Ponto dentro de polígono convexo

Dado um polígono convexo no sentido horário, verifica se o ponto está dentro (inclui borda) em $O(\log n)$.

```
bool query(polygon &P, point q) {
  int i = 1, j = P.size()-1, m;
  if (cross(P[i]-P[0], P[j]-P[0]) < -EPS)
    swap(i, j);
  while(abs(j-i) > 1) {
    int m = (i+j)/2;
    if (cross(P[m]-P[0], q-P[0]) < 0) j = m;
    else i = m;
  }
  return isInsideTriangle(P[0], P[i], P[j], q) != 2;
}
```

8.8 Ponto dentro de triângulo

```
struct pt{
    int x, y;
    pt(){ x = y = 0; }
    pt(int _x, int _y): x(_x), y(_y) {}
};

struct triangle{
    pt a, b, c;
    triangle(){}
    triangle(pt _a, pt _b, pt _c): a(_a), b(_b), c(_c)
        {}
    int area() const{
        return abs((a.x*(b.y-c.y) + b.x*(c.y-a.y) + c.x
            *(a.y-b.y)));
    }
};

bool inside(pt x, triangle t){
    int a0 = t.area(),
        a1 = triangle(x, t.b, t.c).area(),
        a2 = triangle(t.a, x, t.c).area(),
        a3 = triangle(t.a, t.b, x).area();
    return a0 == a1 + a2 + a3;
}
```

8.9 Círculo dentro de outro

Um círculo B está dentro de um círculo A se o $R_a >= R_b + d_{ab}$.

Capítulo 9

Outros

9.1 Maximum subarray sum

```
int N, arr[N];
int best = 0, soma = 0;
for(int i = 0; i < N; i++){
    soma = max(arr[i], soma+arr[i]);
    best = max(best, soma);
}
```

9.2 Swaps adjacentes para ordenar um array

```
int ft[MAX];

void update(int k, int v){
  while(k < MAX){
    ft[k] += v;
        k += (k & (-k));
  }
}

int rsq(int b){
  int sum = 0;
    while(b > 0){
    sum += ft[b];
        b -= (b & (-b));
  }
    return sum;
}

int main(){
  int N, count = 0;
    for(int i = 0; i < N; i++){
      int v; scanf("%d", &v);
        count += i - rsq(v);
        update(v, 1);
    }
  return 0;
}
```

9.3 Swaps não adjacentes para ordenar um array

```
int minSwaps(int arr[], int n){
    pair<int, int> pos[n];
    for(int i = 0; i < n; i++){
        pos[i].first = arr[i];
        pos[i].second = i;
    }
    sort(pos, pos+n);
    vector<bool> visited(n, false);
    int ans = 0;
    for(int i = 0; i < n; i++){
        if(visited[i] || pos[i].second == i) continue;
        int cycle_size = 0;
        int j = i;
        while(!visited[j]){
            visited[j] = 1;
            j = pos[j].second;
            cycle_size++;
        }
        ans += cycle_size - 1;
    }
    return ans;
}
```

9.4 Inversões de tamanho três

```
#include <stdio.h>
#include <algorithm>
#define MAX 1000123
using namespace std;
typedef long long ll;

int bit[MAX];

int rsq(int index){
    int sum = 0;
    while (index > 0) {
        sum += bit[index];
        index -= index & (-index);
    }
    return sum;
}
```

```
void update(int index, int val){
    while (index <= MAX) {
       bit[index] += val;
       index += index & (-index);
    }
}

void convert(int arr[], int n){
    int temp[n];
    for (int i=0; i<n; i++) temp[i] = arr[i];
    sort(temp, temp+n);
    for (int i=0; i<n; i++)
        arr[i] = lower_bound(temp, temp+n, arr[i]) -
            temp + 1;
}

ll getInvCount(int arr[], int n){
    convert(arr, n);
    ll greater1[n], smaller1[n];
    for (int i=0; i<n; i++) greater1[i] = smaller1[i] =
         0;

    for (int i=0; i<=n; i++) bit[i]=0;

    for(int i=n-1; i>=0; i--){
        smaller1[i] = rsq(arr[i]-1);
        update(arr[i], 1);
    }
    for (int i=0; i<=n; i++) bit[i] = 0;

    for (int i=0; i<n; i++) {
        greater1[i] = i - rsq(arr[i]);
        update(arr[i], 1);
    }
    ll invcount = 0;
    for (int i=0; i<n; i++) invcount += smaller1[i]*
        greater1[i];
    return invcount;
}
```

9.5 Prefixa e Infixa para Posfixa

```
void posfixa(string s1, string s2){
    char raiz = s1[0];
    int pos_raiz_infixa = s2.find(raiz);

    if(pos_raiz_infixa != 0){
        string esq1 = s1.substr(1, pos_raiz_infixa);
        string esq2 = s2.substr(0, pos_raiz_infixa);
        posfixa(esq1, esq2);
    }
    if(pos_raiz_infixa + 1 != s1.size()){
        string dir1 = s1.substr(pos_raiz_infixa+1);
        string dir2 = s2.substr(pos_raiz_infixa+1);
        posfixa(dir1, dir2);
    }
    cout << raiz;
}
```

9.6 Números Romanos

```
#include <cstdio>
#include <cstdlib>
#include <ctype.h>
#include <map>
#include <string>
using namespace std;

void AtoR(int A) {
  map<int, string> cvt;
  cvt[1000] = "M"; cvt[900] = "CM"; cvt[500] = "D";
  cvt[400] = "CD"; cvt[100]  = "C"; cvt[90]  = "XC";
  cvt[50]  = "L";  cvt[40]  = "XL"; cvt[10]  = "X";
  cvt[9]  = "IX";  cvt[5]   = "V";  cvt[4]  = "IV";
  cvt[1]    = "I";
  // process from larger values to smaller values
  for (map<int, string>::reverse_iterator i = cvt.
      rbegin();
       i != cvt.rend(); i++)
    while (A >= i->first) {
      printf("%s", ((string)i->second).c_str());
      A -= i->first; }
  printf("\n");
}

void RtoA(char R[]) {
  map<char, int> RtoA;
  RtoA['I'] = 1;   RtoA['V'] = 5;   RtoA['X'] = 10;
      RtoA['L'] = 50;
  RtoA['C'] = 100; RtoA['D'] = 500; RtoA['M'] = 1000;

  int value = 0;
  for (int i = 0; R[i]; i++)
    if (R[i+1] && RtoA[R[i]] < RtoA[R[i+1]]) {     //
        check next char first
      value += RtoA[R[i+1]] - RtoA[R[i]];
                          // by definition
      i++; }

          // skip this char
    else value += RtoA[R[i]];
  printf("%d\n", value);
}

int main() {
  char str[1000];
  while (gets(str) != NULL) {
    if (isdigit(str[0])) AtoR(atoi(str)); // Arabic to
        Roman Numerals
    else                 RtoA(str); // Roman to Arabic
        Numerals
  }
  return 0;
}
```