Notebook UPF 2023 para a Maratona SBC de Programação

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

1 de setembro de 2023

Sumário

Capítulo 1

Introdução

1.1 Recomendações gerais

Cortesia da PUC-RJ (adaptado).

Aquecimento:

- Procure tirar todas as suas dúvidas quanto ao ambiente computacional e às regras da competição, para aproveitar melhor o tempo quando a competição começar.
- Use o editor de texto ou IDE de sua preferência. Caso não encontre, procure as ferramentas mais adequadas à equipe.
- Teste o sistema de submissão eletrônica, as impressões de código-fonte, e principalmente, aproveite para errar à vontade.

Antes da prova:

- Revisar os algoritmos disponíveis na biblioteca.
- Revisar a referência STL.
- Reler o roteiro (na próxima página).
- Ouvir o discurso motivacional do técnico.

Antes de implementar um problema:

- Quem for implementar deve relê-lo antes.
- Peça todas as clarifications que forem necessárias.
- Teste o algoritmo no papel e convença outra pessoa de que ele funciona.
- Planeje a resolução para os problemas grandes: a equipe se junta para definir as estruturas de dados, mas cada pessoa escreve uma função.

Debugar um programa:

- Ao encontrar um bug, escreva um caso de teste que o dispare.
- Reimplementar trechos de programas entendidos errados.
- Em caso de *runtime error*, procure todos os [, / e %.

1.2 Truques sujos (porém válidos)

- **Método Steven 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 *Wrong Answer* em um *Runtime Error*. É uma forma válida de debug. Use isso somente no desespero, pois fica gastando submissões.

1.3 Roteiro de prova

Cortesia da PUC-RJ (adaptado).

300 minutos: INÍCIO DE PROVA

- Abram os seus cadernos de prova e dividam-se para procurar o problema mais fácil: um integrante procura no início, o outro no meio, e o terceiro no fim.
- Quando surgir um problema fácil, todos discutem se ele deve ser o primeiro problema a ser resolvido.
- Quando o primeiro problema for escolhido, quem digita mais rápido o implementa, possivelmente trocando de lugar com quem está no computador.
- Se surgir um problema ainda mais fácil que o primeiro, passe a implementar esse problema.
- Enquanto um integrante resolve o primeiro problema, os outros leem os demais.
- A medida em que os problemas forem lidos (com atenção, sem pular detalhes), preencham a tabela de problemas.
  - AC: recebeu YES (Accepted)? Marque (pinte) a letra do problema na tabela.
  - Ordem: ordem de resolução dos problemas (pode ser infinito).
  - Escrito: se já há código escrito neste problema, mesmo que no papel.
  - Leitores: pessoas que já leram o problema.
  - Complexidade: complexidade da solução implementada.
  - Resumo: resumo sobre o problema.
- Assim que o primeiro problema começar a ser compilado, quem está na frente do PC avisa os outros dois para escolherem o próximo problema mais fácil.
- Assim que o primeiro problema for submetido, mande imprimir o código-fonte e saia do computador.
- Quem for para o computador implementa o segundo problema mais fácil.
- Fora do computador, os outros dois integrantes escolhem a ordem e os resolvedores dos problemas, com base no tempo de implementação.
- Se ninguém tiver alguma ideia para resolver um problema, empurrem-o para o final (ou seja, a ordem desse problema será infinito).
- Quando o segundo problema for submetido (primeiro para avaliação e depois para impressão), saia do computador e reveja a ordenação dos problemas com quem ficou fora do computador.

200 minutos: MEIA-PROVA

- A equipe deve resolver no máximo três problemas ao mesmo tempo.
- Escreva o máximo possível de código no papel.
- Depure com o código do problema e prints de debug (cerr, fprintf(stderr, ...), etc).
  - Explique seu código para outra pessoa da equipe.
  - Acompanhe o código linha por linha, anotando os valores das variáveis e redesenhando as estruturas de dados à medida que forem alteradas.
- Momentos nos quais quem estiver no computador deve avisar os outros membros da equipe:
  - Quando estiver pensando ou debugando.
  - Quando estiver prestes a submeter, para que os outros membros possam fazer testes extras e verificar o formato da saída.
- Logo após submeter, imprima o código.
- Jogue fora as versões mais antigas do código impresso de um programa.
- Jogue fora todos os papéis de um problema quando receber Accepted.
- Mantenha todos os papéis de um problema juntos.

100 minutos: FINAL DE PROVA

- A equipe deve resolver apenas um problema no final da prova.
- Use os balões das outras equipes para escolher o último problema:
  - Os problemas mais resolvidos provavelmente são mais fáceis que os outros problemas.
  - Uma equipe mais bem colocada só é informativa quando as demais não o forem.
- Quem digita mais rápido é quem deve ficar o tempo todo no computador.
- Os outros dois colegas sentam ao lado e ficam dando sugestões para o problema.

60 minutos: PLACAR CONGELADO

- Prestem atenção nos melancias e nas comemorações das outras equipes: os balões continuam vindo (até faltar 30 minutos para o final da prova)

30 minutos: SEM MAIS BALÕES

- Quando terminar um problema, teste com o exemplo de entrada, submeta e só depois pense em mais casos de teste.
- Nos últimos cinco minutos, faça alterações pequenas no código, remova os prints de debug e submeta.

1.4 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 multiplicação 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 (EPS) — absoluta, e talvez relativa.
- Cuidado com erros de imprecisão, em particular evitar ao máximo subtrair dois números praticamente iguais.

Outros erros:

- Tomar cuidado com possíveis divisões por zero (é *runtime error* na certa).
- 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 $(x\%n + n)\%n$.

1.5 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

1.6 Quantidade de números primos de 1 até 10^n

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

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

1.7 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

```
for(int i = 0; i < MAX; i++) {
    C[i][0] = C[i][i] = 1;
    for (int j = 1; j < i; j++)
        C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
}
```

1.8 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

1.9 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
```

1.10 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 3.141592653589793238462643383279502884
#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 ˆ //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

```cpp
#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)

// returns S % N, where N is a power of 2
#define modulo(S, N) ((S) & (N - 1))
#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 Disjoint Set Union (Union–Find)

```cpp
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 (Binary Indexed Tree)

```cpp
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 Fenwick Tree com range updates e queries

Resolve queries do tipo RSQ de i a j (1-indexed) em $O(\log n)$. Range updates ($a[i...j]+=v$) em $O(\log n)$.

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

class FenwickTree {
private:
    vector<int> ft1, ft2;
    int rsq(vector<int> & ft, int i) {
        int sum = 0;
        for(; i; i -= (i & -i)) sum += ft[i];
        return sum;
    }
    void update(vector<int> & ft, int i, int v) {
        for(; i < (int)ft.size(); i += (i & -i))
            ft[i] += v;
    }
public:
    FenwickTree(int n) {
        ft1.assign(n + 1, 0); //1-indexed
        ft2.assign(n + 1, 0); //1-indexed
    }
    void update(int i, int j, int v) {
        update(ft1, i, v);
        update(ft1, j+1, -v);
        update(ft2, i, v*(i-1));
        update(ft2, j+1, -v*j);
    }
    int rsq(int i) {
        return rsq(ft1, i)*i - rsq(ft2, i);
    }
    int rsq(int i, int j) {
        return rsq(j) - rsq(i-1);
    }
};
```

2

3.5 2D Fenwick Tree

```
#include <vector>
using namespace std;
const int neutral = 0;
int comp(int a, int b) {
    return a+b;
}
class FenwickTree2D {
private:
    vector< vector<int> > ft;
public:
    FenwickTree2D(int n, int m) {
        ft.assign(n + 1, vector<int>(m + 1, 0));    //
            1-indexed
    }
    int rsq(int i, int j) { // returns RSQ((1,1), (i,j)
        )
        int sum = 0, _j = j;
        while(i > 0) { j = _j;
            while(j > 0) {
                sum = comp(sum, ft[i][j]);
                j -= (j & -j);
            }
            i -= (i & -i);
        }
        return sum;
    }
    void update(int i, int j, int v) {
        int _j = j;
        while(i < (int)ft.size()) { j = _j;
            while(j < (int)ft[i].size()) {
                ft[i][j] = comp(v, ft[i][j]);
                j += (j & -j);
            }
            i += (i & -i);
        }
    }
};
```

3.6 KD Fenwick Tree

Cortesia do BRUTE-UDESC. Fenwick Tree em K dimensões.

- Complexidade de update: $O(log^k(N))$.
- Complexidade de query: $O(log^k(N))$.

```
const int MAX = 10;
ll tree [MAX][MAX][MAX][MAX][MAX][MAX][MAX][MAX];
// insira a quantidade necessária de dimensões

int lsONE(int x) { return x & (-x); }

ll query(vi s, int pos){
    ll sum = 0;
    while(s[pos] > 0){
        if(pos < s.size()-1)
            sum += query(s, pos+1);
        else
            sum += tree[s[0]][s[1]][s[2]][s[3]][s[4]]
                [s[5]][s[6]][s[7]];
        s[pos] -= lsONE(s[pos]);
    }
    return sum;
}

void update(vi s, int pos, int v){
    while(s[pos] < MAX+1){
        if(pos < s.size()-1)
            update(s, pos+1, v);
        else
            tree[s[0]][s[1]][s[2]][s[3]][s[4]]
                [s[5]][s[6]][s[7]] += v;
        s[pos] += lsONE(s[pos]);
    }
}
```

3.7 LiChao Tree

Uma árvore de Funções. Retorna o F(x) máximo em um ponto X.
Para retornar o minimo deve-se inserir o negativo da função e pegar o negativo do resultado.
Está pronta para usar função linear do tipo F(x) = mx + b.
Funciona para funções com a seguinte propriedade, sejam duas funções f(x) e g(x), uma vez que f(x) ganha/perde de g(x), f(x) vai continuar ganhando/perdendo de g(x), ou seja f(x) e g(x) se intersectam apenas uma vez.

- Complexidade de consulta: $O(log(N))$
- Complexidade de update: $O(log(N))$

Cortesia do BRUTE-UDESC.

```
typedef long long ll;

const ll MAXN = 1e5+5, INF = 1e18+9;

struct Line {
    ll a, b = -INF;
    ll operator()(ll x) {
        return a * x + b;
    }
} tree[4 * MAXN];

int le(int n) { return 2*n+1; }
int ri(int n) { return 2*n+2; }

void insert(Line line, int n=0, int l=0, int r=MAXN) {
    int mid = (l + r) / 2;
    bool bl = line(l) < tree[n](l);
    bool bm = line(mid) < tree[n](mid);
    if(!bm) swap(tree[n], line);
    if(l == r) return;
    if(bl != bm) insert(line, le(n), l, mid);
    else insert(line, ri(n), mid+1, r);
}

ll query(int x, int n=0, int l=0, int r=MAXN) {
    if(l == r) return tree[n](x);
    int mid = (l + r) / 2;
    if(x < mid) return max(tree[n](x), query(x, le(n),
        l, mid));
    else return max(tree[n](x), query(x, ri(n), mid+1,
        r));
}
```

3.8 LiChao Tree Sparse

O mesmo que a superior, no entanto suporta consultas com |x| <= 1e18.

- Complexidade de consulta: $O(log(\text{tamanho do intervalo}))$
- Complexidade de update: $O(log(\text{tamanho do intervalo}))$

Cortesia do BRUTE-UDESC.

```
typedef long long ll;

const ll MAXN = 1e5+5, INF = 1e18+9, MAXR=1e18;

struct Line {
    ll a, b = -INF;
    __int128 operator()(ll x) {
        return (__int128) a * x + b;
    }
} tree[4 * MAXN];
int idx = 0, L[4 * MAXN], R[4 * MAXN];

int le(int n) {
    if (!L[n]) L[n] = ++idx;
    return L[n];
}
int ri(int n) {
    if (!R[n]) R[n] = ++idx;
    return R[n];
}

void insert(Line line, int n=0, ll l=-MAXR, ll r=MAXR)
    {
    ll mid = (l + r) / 2;
    bool bl = line(l) < tree[n](l);
    bool bm = line(mid) < tree[n](mid);
    if(!bm) swap(tree[n], line);
    if(l == r) return;
    if(bl != bm) insert(line, le(n), l, mid);
    else insert(line, ri(n), mid+1, r);
}

__int128 query(int x, int n=0, ll l=-MAXR, ll r=MAXR) {
    if(l == r) return tree[n](x);
    ll mid = (l + r) / 2;
    if(x < mid) return max(tree[n](x), query(x, le(n),
        l, mid));
    else return max(tree[n](x), query(x, ri(n), mid+1,
        r));
}
```

3.9 Max-Queue

Cortesia do Macacário do ITA.

```
class MaxQueue {
    list<ii> q, l;
    int cnt = 0;
public:
    MaxQueue() : cnt(0) {}
    void push(int x) {
        ii cur = ii(x, cnt++);
        while(!l.empty() && l.back() <= cur) l.pop_back
            ();
        q.push_back(cur);
        l.push_back(cur);
    }
    int front() { return q.front().first; }
    void pop() {
        if (q.front().second == l.front().second) l.
            pop_front();
        q.pop_front();
    }
    int max() { return l.front().first; }
    int size() { return q.size(); }
};
```

3.10 Operation Queue

Fila que armazena o resultado do operatório dos itens.

- Complexidade de tempo (Push): $O(1)$
- Complexidade de tempo (Pop): $O(1)$

Cortesia do BRUTE-UDESC.

```
template <typename T>
struct op_queue{
    stack<pair<T, T>> s1, s2;
    T result;
    T op(T a, T b){
        return a; // TODO: op to compare
        // min(a, b);
        // gcd(a, b);
        // lca(a, b);
    }
    T get(){
        if (s1.empty() || s2.empty())
            return result = s1.empty() ?
                s2.top().second : s1.top().second;
        else
            return result =
                op(s1.top().second, s2.top().second);
    }
    void add(T element){
        result = s1.empty() ?
            element : op(element, s1.top().second);
        s1.push({element, result});
    }
    void remove(){
        if (s2.empty()) {
            while (!s1.empty()) {
                T elem = s1.top().first;
                s1.pop();
                T result = s2.empty() ?
                    elem : op(elem, s2.top().second);
                s2.push({elem, result});
            }
        }
        T remove_elem = s2.top().first;
        s2.pop();
    }
};
```

3.11 Operation Stack

Pilha que armazena o resultado do operatório dos itens.

- Complexidade de tempo (Push): $O(1)$
- Complexidade de tempo (Pop): $O(1)$

Cortesia do BRUTE-UDESC.

```
template <typename T>
struct op_stack{
    stack<pair<T, T>> st;
    T result;
    T op(T a, T b){
        return a; // TODO: op to compare
        // min(a, b);
        // gcd(a, b);
        // lca(a, b);
    }
    T get(){
        return result = st.top().second;
    }
    void add(T element){
        result = st.empty() ?
            element : op(element, st.top().second);
        st.push({element, result});
    }
    void remove(){
        T removed_element = st.top().first;
        st.pop();
    }
};
```

3.12 Ordered Set

Pode ser usado como um set normal, a principal diferença são duas novas operações possíveis:
`find_by_order(x)`: retorna o item na posição x.
`order_of_key(k)`: retorna o número de elementos menores que k. (o índice de k)

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/trie_policy.hpp>
using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> ordered_set;
```

3.13 Segment Tree

Árvore dos segmentos em 1D, construtor para construção com array em $O(n)$. Queries e updates em $O(\log n)$, memória $O(n)$. Indexado em 0. O update substitui o valor no local, não executa *comp*. Cortesia do Macacário do ITA.

```
const int neutral = 0;
#define comp(a, b) ((a)+(b))

class SegmentTree {
    vector<int> a;
    int n;
public:
    SegmentTree(int* st, int* en) {
        int sz = int(en-st);
        for (n = 1; n < sz; n <<= 1);
        a.assign(n << 1, neutral);
        for(int i=0; i<sz; i++) a[i+n] = st[i];
        for(int i=n+sz-1; i>1; i--)
            a[i>>1] = comp(a[i>>1], a[i]);
    }
    void update(int i, int x) {
        a[i += n] = x; //substitui
        for (i >>= 1; i; i >>= 1)
            a[i] = comp(a[i<<1], a[1+(i<<1)]);
    }
    int query(int l, int r) {
        int ans = neutral;
        for (l+=n, r+=n+1; l<r; l>>=1, r>>=1) {
            if (l & 1) ans = comp(ans, a[l++]);
            if (r & 1) ans = comp(ans, a[--r]);
        }
        return ans;
    }
};
```

3.14 Segment Tree com Lazy Propagation

Idem acima. O update soma um valor em todos os pontos no intervalo $[a,b]$, mas pode ser modificado para aplicar uma função linear. Cortesia do Macacário do ITA.

```
const int neutral = 0; //comp(x, neutral) = x
#define comp(a, b) ((a)+(b))

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

3.15 2D Segment Tree

Árvore de Segmentos 2D $O(q \log^2 n)$ em tempo e memória. Suporta operações as *point-update* e *range-query* de adição. O elemento neutro é definido como 0 pelo padrão do compilador. *st* de nós-y possui a raíz da árvore-x, o de nós-x, o valor.

```
int rs[MAXN], ls[MAXN], st[MAXN], cnt = 0;

class SegmentTree2D {
    int sizex, sizey, v, root;
    int x, y, ix, jx, iy, jy;
    void updatex(int p, int lx, int rx) {
        if (x < lx || rx < x) return;
        st[p] += v;
        if (lx == rx) return;
        if (!rs[p]) rs[p] = ++cnt, ls[p] = ++cnt;
        int mx = (lx + rx) / 2;
        updatex(ls[p], lx, mx);
        updatex(rs[p], mx + 1, rx);
    }
    void updatey(int p, int ly, int ry) {
        if (y < ly || ry < y) return;
        if (!st[p]) st[p] = ++cnt;
        updatex(st[p], 0, sizex);
        if (ly == ry) return;
        if (!rs[p]) rs[p] = ++cnt, ls[p] = ++cnt;
        int my = (ly + ry) / 2;
        updatey(ls[p], ly, my);
        updatey(rs[p], my + 1, ry);
    }
    int queryx(int p, int lx, int rx) {
        if (!p || jx < lx || ix > rx) return 0;
        if (ix <= lx && rx <= jx) return st[p];
        int mx = (lx + rx) / 2;
        return queryx(ls[p], lx, mx) +
            queryx(rs[p], mx + 1, rx);
    }
    int queryy(int p, int ly, int ry) {
        if (!p || jy < ly || iy > ry) return 0;
        if (iy <= ly && ry <= jy) return queryx(st[p],
            0, sizex);
        int my = (ly + ry) / 2;
        return queryy(ls[p], ly, my) +
            queryy(rs[p], my + 1, ry);
    }
public:
    SegmentTree2D(int nx, int ny) : sizex(nx), sizey(ny
        ) {
        root = ++cnt;
    }
    void update(int _x, int _y, int _v) {
        x = _x; y = _y; v = _v;
        updatey(root, 0, sizey);
    }
    int query(int _ix, int _jx, int _iy, int _jy) {
        ix = _ix; jx = _jx; iy = _iy; jy = _jy;
        return queryy(root, 0, sizey);
    }
};
```

3.16 SQRT Decomposition

Exemplo usado no BEE 1399 - Transformador de Matriz, com uma SQRT Decomposition de Ordered Sets.

```
#include <cstdio>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#define MAX 312345
#define SQ 550
using namespace std;
using namespace __gnu_pbds;

typedef long long ll;
typedef pair<ll, ll> pll;
typedef tree<pll, null_type, less<pll>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;

ordered_set b[SQ];

int n, m, u;
ll a[MAX];

int query(int l, int r, int v) {
  int ans = 0;
  for (int i = l/SQ+1; i < r/SQ; i++)
    ans += b[i].order_of_key({ v, 0 });

  if (l/SQ != r/SQ) {
    for (int i = l; i < (l/SQ+1)*SQ; i++) ans += a[i] <
        v;
    for (int i = r/SQ*SQ; i <= r; i++) ans += a[i] < v;
  } else {
    for (int i = l; i <= r; i++) ans += a[i] < v;
  }

  return ans;
}

void update(int p, int k, int l, int r) {
  ll val = 1LL*u*k/(r-l+1);
  b[p/SQ].erase({ a[p], p });
  a[p] = val;
  b[p/SQ].insert({ a[p], p });
}

int main() {
  scanf("%d %d %d", &n, &m, &u);
  for (int i = 1; i <= n; i++) {
    scanf("%lld", &a[i]);
    b[i/SQ].insert({ a[i], i });
  }
}
```

3.17 Sparse Table

Responde consultas de maneira eficiente em um conjunto de dados estáticos.
Realiza um pré-processamento para diminuir o tempo de cada consulta.

- Complexidade de tempo (Pré-processamento): $O(N * log(N))$
- Complexidade de tempo (Consulta para operações sem sobreposição amigável): $O(N * log(N))$
- Complexidade de tempo (Consulta para operações com sobreposição amigável): $O(1)$
- Complexidade de espaço: $O(N * log(N))$

Cortesia do BRUTE-UDESC.

```
struct SparseTable {
    int n, e;
    vector <vi> st;
    SparseTable(vi &v) : n(v.size()), e(floor(log2(n)))
        {
        st.assign(e+1, vi(n));
        for (int i = 0; i < n; i++) st[0][i] = v[i];
        for (int i = 1; i <= e; i++) {
            for (int j = 0; j + (1 << i) <= n; j++) {
                st[i][j] = min(st[i-1][j],
                    st[i-1][j+(1 << (i-1))]);
            }
        }
    }
    // O(NLog(N)) Query for non overlab friendly
        operations
    int logquery(int l, int r) {
        int res = 1123456789;
        for (int i = e; i >= 0; i--) {
            if ((1 << i) <= r-l+1) {
                res = min(res, st[i][l]);
                l += 1 << i;
            }
        }
        return res;
    }
    // O(1) Query for overlab friendly operations
    // ex: max(), min(), gcd(), f(x, y) = x
    int query(int l, int r) {
        int i = ilogb(r-l+1);
        return min(st[i][l], st[i][r - (1 << i) + 1]);
    }
};
```

3.18 Disjoint Sparse Table

Resolve Query de range para qualquer operação associativa (i.e., `(a^b)^c = a^(b^c)`) em $O(1)$.

- Complexidade de tempo (Pré-processamento): $O(N * log(N))$

Cortesia do BRUTE-UDESC.

```
struct dst{
    const int neutral = 1;
    #define comp(a, b) (a | b)
    vector<vector<int>> t;
    dst(vi v){
        int n, k, sz = v.size();
        for(n = 1, k = 0; n < sz; n <<= 1, k++);
            t.assign(k, vector<int>(n));
        for(int i = 0; i < n; i++)
            t[0][i] = i < sz ? v[i] : neutral;
        for(int j = 0, len = 1; j <= k; j++, len <<= 1)
            {
            for(int s = len; s < n; s += (len << 1)) {
                t[j][s] = v[s]; t[j][s-1] = v[s-1];
                for(int i = 1; i < len; i++) {
                    t[j][s+i] = comp(t[j][s+i-1], v[s+i
                        ]);
                    t[j][s-1-i] = comp(v[s-1-i], t[j][s
                        -i]);
                }
            }
        }
    }
    int query(int l, int r) {
        if(l == r) return t[0][r];
        int i = 31 - __builtin_clz(l^r);
        return comp(t[i][l], t[i][r]);
    }
};
```

Capítulo 4

Paradigmas

4.1 Merge Sort

Algoritmo $O(n \log n)$ para ordenar o vetor em $[a,b]$. inv conta o número de inversões do bubble-sort nesse trecho.

```
#define MAXN 100009
long long inv;
int aux[MAXN];

void mergesort(int arr[], int l, int r) {
    if (l == r) return;
    int m = (l + r) / 2;
    mergesort(arr, l, m);
    mergesort(arr, m+1, r);
    int i = l, j = m + 1, k = l;
    while(i <= m && j <= r) {
        if (arr[i] > arr[j]) {
            aux[k++] = arr[j++];
            inv += j - k;
        }
        else aux[k++] = arr[i++];
    }
    while(i <= m) aux[k++] = arr[i++];
    for(i = l; i < k; i++) arr[i] = aux[i];
}
```

4.2 Quick Sort

Algoritmo *Expected* $O(n \log n)$ para ordenar o vetor em $[a,b]$. É o mais rápido conhecido.

```
void quicksort(int* arr, int l, int r) {
    if (l >= r) return;
    int mid = l + (r - l) / 2;
    int pivot = arr[mid];
    swap(arr[mid],arr[l]);
    int i = l + 1, j = r;
    while (i <= j) {
        while(i <= j && arr[i] <= pivot) i++;
        while(i <= j && arr[j] > pivot) j--;
        if (i < j) swap(arr[i], arr[j]);
    }
    swap(arr[i-1], arr[l]);
    quicksort(arr, l, i-2);
    quicksort(arr, i, r);
}
```

4.3 Longest Increasing Subsequence (LIS)

$O(n \log n)$. Ao final de cada iteração i, o k-ésimo elemento (1-indexed) de s é o menor elemento que tem uma subsequência crescente de tamanho k terminando nele.

```
int lis(int arr[], int n) {
    multiset<int> s;
    multiset<int>::iterator it;
    for(int i = 0; i < n; i++) {
        s.insert(arr[i]);
        it = s.upper_bound(arr[i]); //non-decreasing
        //it = ++s.lower_bound(arr[i]); //strictly
            increasing
        if (it != s.end()) s.erase(it);
    }
    return s.size();
}
```

4.4 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 knapsack(int W, vi wt, vi v, int n){
    int ks[n+1][W+1];
    for(int i = 0; i <= n; i++){
        for(int w = 0; w <= W; w++){
            if(i == 0 || w == 0) ks[i][w] = 0;
            else if(wt[i-1] > w){
                ks[i][w] = ks[i-1][w];
            }else{
                ks[i][w] = max(v[i-1] +
                ks[i-1][w-wt[i-1]], ks[i-1][w]);
            }
        }
    }
    int res = ks[n][W];
    // Soma dos pesos:
    int soma_pesos = 0;
    int w = W;
    for(int i = n; i > 0; i--){
        if(res <= 0) break;
        if(res == ks[i-1][w]) continue;
        soma_pesos += wt[i-1];
        res -= v[i-1];
        w -= wt[i-1];
    }
}
```

4.5 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];
}
```

4.6 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];
}
```

4.7 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;
}
```

4.8 Maximum subarray sum

Algoritmo de Kadane

```
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);
}
```

4.9 Maximum circular subarray sum

```
int maxCircularSum(){
    int max_kadane = kadane();
    int max_wrap = 0;
    for(int i = 0; i < n; i++){
        max_wrap += v[i];
        v[i] = -v[i];
    }
    max_wrap += kadane();
    return max(max_wrap, max_kadane);
}
```

4.10 All Submasks

Cortesia do BRUTE-UDESC.
Percorre todas as submáscaras de uma máscara de tamanho N.

- Complexidade de tempo: $O(3^N)$.

```
int mask;
for(int sub = mask; sub; sub = (sub-1) & mask) {
    // seu código aqui
}
```

4.11 Busca Binária Paralela

Cortesia do BRUTE-UDESC.
Faz a busca binária para múltiplas consultas quando a busca binária é muito pesada.

- Complexidade de tempo: $O((N+Q)log(N)*O(F))$, onde N é o tamanho do espaço de busca, Q é o número de consultas e $O(F)$, o custo de avaliação da função.

```
namespace parallel_binary_search{
    typedef tuple<int, int, long long, long long> query; //{value, id, l, r}
    vector<query> queries[1123456]; // pode ser um mapa se for muito esparso
    long long ans[1123456]; // definir pro tamanho das queries
    long long l, r, mid;
    int id = 0;
    void set_lim_search(long long n){
        l = 0;
        r = n;
        mid = (l+r)/2;
    }
    void add_query(long long v){
        queries[mid].push_back({v, id++, l, r});
    }
    void advance_search(long long v){
        // advance search
    }
    bool satisfies(long long mid, int v, long long l, long long r){
        // implement the evaluation
    }
    bool get_ans(){
        // implement the get ans
    }
    void parallel_binary_search(long long l, long long r){

        bool go = 1;
        while(go){
            go = 0;
            int i = 0; // outra logica se for usar um mapa
            for(auto& vec: queries){
                advance_search(i++);
                for(auto q: vec){
                    auto [v, id, l, r] = q;
                    if(l > r) continue;
                    go = 1;
                    // return while satisfies
                    if(satisfies(i, v, l, r)){
                        ans[i] = get_ans();
                        long long mid = (i+l)/2;
                        queries[mid] = query(v, id, l, i-1);
                    }else{
                        long long mid = (i+r)/2;
                        queries[mid] = query(v, id, i+1, r);
                    }
                }
                vec.clear();
            }
        }
    }
} // namespace name
```

4.12 Busca Ternária

Encontra um ponto ótimo em uma função que pode ser separada em duas funções estritamente monotônicas (e.g. parábolas).

- Complexidade de tempo: $O(log(N) * O(eval))$. Onde N é o tamanho do espaço de busca e $O(eval)$ o custo de avaliação da função.

Cortesia do BRUTE-UDESC.

```
double eval(double mid){
    // implement the evaluation
}

double ternary_search(double l, double r){
    int k = 100;
    while(k--){
        double step  = (l+r)/3;
        double mid_1 = l + step;
        double mid_2 = r - step;

        // minimizing. To maximize use >= to compare
        if(eval(mid_1) <= eval(mid_2)) r = mid_2;
        else l = mid_1;
    }
    return l;
}
```

4.13 Busca Ternária Discreta

Encontra um ponto ótimo em uma função que pode ser separada em duas funções estritamente monotônicas (e.g. parábolas). Versão para espaços discretos.

- Complexidade de tempo: $O(log(N) * O(eval))$. Onde N é o tamanho do espaço de busca e $O(eval)$ o custo de avaliação da função.

Cortesia do BRUTE-UDESC.

```
long long eval(long long mid){
    // implement the evaluation
}

long long discrete_ternary_search(long long l, long
    long r){
    long long ans = -1;
    r--; // to not space r
    while(l<=r){
        long long mid = (l+r)/2;

        // minimizing. To maximize use >= to compare
        if(eval(mid) <= eval(mid+1)){
            ans = mid;
            r = mid-1;
        }else l = mid+1;
    }
    return ans;
}
```

4.14 Convex Hull Trick

Otimização de DP onde se mantém as retas que formam um Convex Hull em uma estrutura que permite consultar qual o melhor valor para um determinado x.
Só funciona quando as retas são monotônicas. Caso não forem, usar LiChao Tree para guardar as retas.
Complexidade de tempo:

- Inserir reta: $O(1)$ amortizado
- Consultar x: $O(log(N))$
- Consultar x quando x tem crescimento monotônico: $O(1)$

Cortesia do BRUTE-UDESC.

```
const ll INF = 1e18+18;
bool op(ll a, ll b) {
    return a >= b;  // either >= or <=
}
struct line {
    ll a, b;
    ll get(ll x) {
        return a * x + b;
    }
    ll intersect(line l) {
        return (l.b - b + a - l.a) / (a - l.a); //
            rounds up for integer only
    }
};
deque<pair<line, ll>> fila;
void add_line(ll a, ll b) {
    line nova = {a, b};
    if (!fila.empty() && fila.back().first.a == a &&
        fila.back().first.b == b) return;
    while (!fila.empty() && op(fila.back().second, nova
        .intersect(fila.back().first)))
        fila.pop_back();
    ll x = fila.empty()? -INF:nova.intersect(fila.back
        ().first);
    fila.emplace_back(nova, x);
}
ll get_binary_search(ll x) {
    int esq = 0, dir = fila.size()-1, r = -1;
    while (esq <= dir) {
        int mid = (esq + dir)/2;
        if (op(x, fila[mid].second)) {
            esq = mid+1;
            r = mid;
        } else dir = mid-1;
    }
    return fila[r].first.get(x);
}
// O(1), use only when QUERIES are monotonic!
ll get(ll x) {
    while (fila.size() >= 2 && op(x, fila[1].second))
        fila.pop_front();
    return fila.front().first.get(x);
}
```

4.15 Digit DP

Calcula a soma de todos os números entre A e B MOD 1000000007.

```
const int MOD = 1e9+7;

// 20 -> maximo de 18 digitos para um número de 64 bits
// 180 -> maior soma dos digitos 9*18 = 162 = ~ 200
// 2 -> tight 1 ou 0
ll dp[20][180][2];

void getDigits(ll x, vector<int> &digit){
    while(x){
        digit.push_back(x%10);
        x /= 10;
    }
}

ll digitSum(int idx, int sum, int tight,
vector<int> &digit){
    if(idx == -1) return sum % MOD;
    if(dp[idx][sum][tight] != -1 && tight != 1)
        return dp[idx][sum][tight];
    ll ret = 0;
    int k = (tight) ? digit[idx] : 9;
    for(int i = 0; i <= k; i++){
        int newTight = (!tight) ? 0 : (digit[idx] == i)
            ;
        // ret += digitSum(idx-1, sum+i, newTight,
            digit);
        ret = (ret + digitSum(idx-1, sum+i, newTight,
            digit)) % MOD;
    }
    if(!tight)
        dp[idx][sum][tight] = ret;
    return ret;
}

ll rangeDigitSum(ll a, ll b){
    memset(dp, -1, sizeof dp);
    vector<int> digitA;
    getDigits(a-1, digitA);
    // finding sum of digits from 1 to a-1 which is
        passed as digitA
    ll ans1 = digitSum(digitA.size()-1, 0, 1, digitA);
    vector<int> digitB;
    getDigits(b, digitB);
    ll ans2 = digitSum(digitB.size()-1, 0, 1, digitB);
    return ans2 - ans1;
}
```

4.16 Divide and Conquer

Otimização para DP de prefixo quando se pretende separar o vetor em K subgrupos.
É preciso fazer a função `query(i, j)` que computa o custo do subgrupo $[i, j]$.

- Complexidade de tempo: $O(n*k*log(n)*O(query))$

Cortesia do BRUTE-UDESC.

```
namespace DC{
    vi dp_before, dp_cur;
    void compute(int l, int r, int optl, int optr) {
        if (l > r) return;
        int mid = (l + r) >> 1;
        pair<ll, int> best = {0, -1}; // {INF, -1} se quiser minimizar
        for (int i = optl; i <= min(mid, optr); i++) {
            best = max(best, {(i ? dp_before[i - 1] : 0) + query(i, mid), i}); // min() se quiser minimizar
        }
        dp_cur[mid] = best.first;
        int opt = best.second;
        compute(l, mid - 1, optl, opt);
        compute(mid + 1, r, opt, optr);
    }

    ll solve(int n, int k) {
        dp_before.assign(n+5, 0);
        dp_cur.assign(n+5, 0);
        for (int i = 0; i < n; i++)
            dp_before[i] = query(0, i);
        for (int i = 1; i < k; i++) {
            compute(0, n - 1, 0, n - 1);
            dp_before = dp_cur;
        }
        return dp_before[n - 1];
    }
};
```

4.17 Divide and Conquer com Query on demand

Usado para evitar queries pesadas ou o custo de pré-processamento.
É preciso fazer as funções da estrutura **janela**, eles adicionam e removem itens um a um como uma janela flutuante.

- Complexidade de tempo: $O(n*k*log(n)*O(\text{update da janela}))$

Cortesia do BRUTE-UDESC.

```
namespace DC{
    // Eh preciso definir a forma de calcular o range
    struct range{
        vi freq;
        ll sum = 0;
        int l = 0, r = -1;
        // Mover o 'l' do range para a esquerda
        void back_l(int v){
            sum += freq[v];
            freq[v]++;
            l--;
        }
         // Mover o 'r' do range para a direita
        void advance_r(int v){
            sum += freq[v];
            freq[v]++;
            r++;
        }
        // Mover o 'l' do range para a direita
        void advance_l(int v){
            freq[v]--;
            sum -= freq[v];
            l++;
        }
        // Mover o 'r' do range para a esquerda
        void back_r(int v){
            freq[v]--;
            sum -= freq[v];
            r--;
        }
        void clear(int n){ // Limpar range
            l = 0; r = -1; sum = 0;
            freq.assign(n+5, 0);
        }
    }s;

    vi dp_before, dp_cur;
    void compute(int l, int r, int optl, int optr) {
        if (l > r) return;
        int mid = (l + r) >> 1;
        pair<ll, int> best = {0, -1};
         // {INF, -1} se quiser minimizar

        while(s.l < optl) s.advance_l(v[s.l]);
        while(s.l > optl) s.back_l(v[s.l-1]);
        while(s.r < mid) s.advance_r(v[s.r+1]);
        while(s.r > mid) s.back_r(v[s.r]);

        vi removed;
        for (int i = optl; i <= min(mid, optr); i++) {
            // min() se quiser minimizar
            best = min(best, {(i ? dp_before[i - 1] :
                0) + s.sum, i});
            removed.push_back(v[s.l]);
            s.advance_l(v[s.l]);
        }
        for (int rem: removed)  s.back_l(v[s.l-1]);

        dp_cur[mid] = best.first;
        int opt = best.second;
        compute(l, mid - 1, optl, opt);
        compute(mid + 1, r, opt, optr);
    }

    ll solve(int n, int k) {
        dp_before.assign(n, 0);
        dp_cur.assign(n, 0);
        s.clear(n);
        for (int i = 0; i < n; i++){
            s.advance_r(v[i]);
            dp_before[i] = s.sum;
        }
        for (int i = 1; i < k; i++) {
            s.clear(n);
            compute(0, n - 1, 0, n - 1);
            dp_before = dp_cur;
        }
        return dp_before[n-1];
    }
};
```

4.18 Exponenciação de Matriz

Otimização para DP de prefixo quando o valor atual está em função dos últimos K valores já calculados.

- Complexidade de tempo: $O(log(n)k^3)$

É preciso mapear a DP para uma exponenciação de matriz. Cortesia do BRUTE-UDESC.

```
ll dp[100];
mat T;

#define MOD 1000000007

mat mult(mat a, mat b) {
    mat res(a.size(), vi(b[0].size()));
    for (int i = 0; i < a.size(); i++) {
        for (int j = 0; j < b[0].size(); j++) {
            for (int k = 0; k < b.size(); k++) {
                res[i][j] += a[i][k] * b[k][j] % MOD;
                res[i][j] %= MOD;
            }
        }
    }
    return res;
}

mat exp_mod(mat b, ll exp){
    mat res(b.size(), vi(b.size()));
    for(int i = 0; i < b.size(); i++) res[i][i] = 1;
    while(exp){
        if(exp & 1) res = mult(res, b);
        b = mult(b, b);
        exp /= 2;
    }
    return res;
}

// MUDA MUITO DE ACORDO COM O PROBLEMA
// LEIA COMO FAZER O MAPEAMENTO NO README
ll solve(ll exp, ll dim){
    if(exp < dim) return dp[exp];

    T.assign(dim, vi(dim));
    // TO DO: Preencher a Matriz que vai ser
        exponenciada
    // T[0][1] = 1;
    // T[1][0] = 1;
    // T[1][1] = 1;

    mat prod = exp_mod(T, exp);

    mat vec; vec.assign(dim, vi(1));
    // Valores iniciais
    for(int i = 0; i < dim; i++) vec[i][0] = dp[i];

    mat ans = mult(prod, vec);
    return ans[0][0];
}
```

4.19 Mo

Resolve Queries Complicadas Offline de forma rápida.
É preciso manter uma estrutura que adicione e remova elementos nas extremeidades de um range (tipo janela).

- Complexidade de tempo (Query offline): $O(N * sqrt(N))$

Cortesia do BRUTE-UDESC.

```
typedef pair<int, int> ii;
int block_sz; // Better if 'const';

namespace mo{
    struct query {
        int l, r, idx;
        bool operator<(query q) const {
            int _l  = l/block_sz;
            int _ql = q.l/block_sz;
            return ii(_l, (_l&1? -r: r)) < ii(_ql, (_ql
                &1? -q.r: q.r));
        }
    };
    vector<query> queries;

    void build(int n){
        block_sz = (int) sqrt(n);
        // TODO: initialize data structure
    }
    inline void add_query(int l, int r){
        queries.push_back({l, r, (int) queries.size()})
            ;
    }
    inline void remove(int idx){
        // TODO: remove value at idx from data
            structure
    }
    inline void add(int idx){
        // TODO: add value at idx from data structure
    }
    inline int get_answer(){
        // TODO: extract the current answer of the data
            structure
        return 0;
    }

    vector<int> run() {
        vector<int> answers(queries.size());
        sort(queries.begin(), queries.end());
        int L = 0;
        int R = -1;
        for (query q : queries) {
            while (L > q.l) add(--L);
            while (R < q.r) add(++R);
            while (L < q.l) remove(L++);
            while (R > q.r) remove(R--);
            answers[q.idx] = get_answer();
        }
        return answers;
    }

};
```

4.20 Mo com Update

Resolve Queries Complicadas Offline de forma rápida.
Permite que existam **UPDATES PONTUAIS**!
É preciso manter uma estrutura que adicione e remova elementos nas extremeidades de um range (tipo janela).

- Complexidade de tempo: $O(Q * N^(2/3))$

Cortesia do BRUTE-UDESC.

```
typedef pair <int, int> ii;
typedef tuple <int, int, int> iii;
int block_sz; // Better if 'const';
vector<int> vec;
namespace mo{
    struct query {
        int l, r, t, idx;
        bool operator<(query q) const {
            int _l  = l/block_sz;
            int _r  = r/block_sz;
            int _ql = q.l/block_sz;
            int _qr = q.r/block_sz;
            return iii(_l, (_l&1?-_r:_r), (_r&1?t:-t)) < iii(_ql, (_ql&1?-_qr:_qr), (_qr&1?q.t:-q.t));
        }
    };
    vector<query> queries;
    vector<ii> updates;

    void build(int n){
        block_sz = pow(1.4142*n, 2.0/3);
        // TODO: initialize data structure
    }
    inline void add_query(int l, int r){
        queries.push_back({l, r, (int) updates.size(), (int) queries.size()});
    }
    inline void add_update(int x, int v){
        updates.push_back({x, v});
    }
    inline void remove(int idx){
        // TODO: remove value at idx from data structure
    }
    inline void add(int idx){
        // TODO: add value at idx from data structure
    }
    inline void update(int l, int r, int t){
        auto& [x, v] = updates[t];
        if(l <= x &&  x <= r) remove(x);
        swap(vec[x], v);
        if(l <= x &&  x <= r) add(x);
    }
    inline int get_answer(){
        // TODO: extract the current answer from the data structure
        return 0;
    }

    vector<int> run() {
        vector<int> answers(queries.size());
        sort(queries.begin(), queries.end());
        int L = 0;
        int R = -1;
        int T = 0;
        for (query q : queries) {
            while(T < q.t) update(L, R, T++);
            while(T > q.t) update(L, R, --T);
            while (L > q.l) add(--L);
            while (R < q.r) add(++R);
            while (L < q.l) remove(L++);
            while (R > q.r) remove(R--);
            answers[q.idx] = get_answer();
        }
        return answers;
    }
};
```

4.21 Otimização de Dois Ponteiros

Reduz a complexidade de $O(n^2k)$ para $O(nk)$ de PD's da seguinte forma (e outras variantes):

$$dp[i][j] = 1 + \min_{1\le k\le i}(\max(dp[k-1][j-1], dp[i-k][j])),\ caso\ base:\ dp[0][j], dp[i][0] \tag{4.1}$$

- $A[i][j] = k$ ótimo que minimiza $dp[i][j]$.
- É necessário que $dp[i][j]$ seja crescente em i: $dp[i][j] \le dp[i+1][j]$.
- Este exemplo é o problema dos ovos e dos prédios.

```
#include <algorithm>
using namespace std;
#define MAXN 1009
#define MAXK 19
#define INF (1<<30)

int dp[MAXN][MAXK], A[MAXN][MAXK], N, K;

void twopointer() {
    for(int i=0; i<=N; i++) dp[i][0] = INF;
    for(int j=0; j<=K; j++) dp[0][j] = 0, A[0][j] = 1;
    dp[0][0] = 0;
    for(int i=1; i<=N; i++) {
        for(int j=1; j<=K; j++) {
            dp[i][j] = INF;
            for(int k=A[i-1][j]; k<=i; k++) {
                int cur = 1 + max(dp[k-1][j-1], dp[i-k
                    ][j]);
                if (dp[i][j] > cur) {
                    dp[i][j] = cur;
                    A[i][j] = k;
                }
                if (dp[k-1][j-1] > dp[i-k][j]) break;
            }
        }
    }
}
```

4.22 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 2-SAT

Resolve problema do 2-SAT.

- Complexidade de tempo (caso médio): $O(N+M)$

N é o número de variáveis e M é o número de cláusulas. A configuração da solução fica guardada no vetor *assignment*.
Em relação ao sinal, tanto faz se 0 liga ou desliga, apenas siga o mesmo padrão.
Cortesia do BRUTE-UDESC.

```
struct sat2{
    int n;
    vector<vector<int>> g, gt;
    vector<bool> used;
    vector<int> order, comp;
    vector<bool> assignment;

    //number of variables
    sat2(int _n) {
        n = 2*(_n+5);
        g.assign(n, vector<int>());
        gt.assign(n, vector<int>());
    }
    void add_edge(int v, int u, bool v_sign, bool
        u_sign){
        g[2*v + v_sign].push_back(2*u + !u_sign);
        g[2*u + u_sign].push_back(2*v + !v_sign);
        gt[2*u + !u_sign].push_back(2*v + v_sign);
        gt[2*v + !v_sign].push_back(2*u + u_sign);
    }
    void dfs1(int v) {
        used[v] = true;
        for (int u : g[v]) if (!used[u])
            dfs1(u);
        order.push_back(v);
    }
    void dfs2(int v, int cl) {
        comp[v] = cl;
        for (int u : gt[v]) if (comp[u] == -1)
            dfs2(u, cl);
    }
    bool solve(){
        order.clear();
        used.assign(n, false);
        for (int i = 0; i < n; ++i) if (!used[i])
            dfs1(i);

        comp.assign(n, -1);
        for (int i = 0, j = 0; i < n; ++i) {
            int v = order[n - i - 1];
            if (comp[v] == -1) dfs2(v, j++);
        }

        assignment.assign(n / 2, false);
        for (int i = 0; i < n; i += 2) {
            if (comp[i] == comp[i + 1]) return false;
            assignment[i / 2] = comp[i] > comp[i + 1];
        }
        return true;
    }
};
```

5.2 BFS

Para DFS, substituir queue por `stack`, e `.front()` por `.top()`.

```
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];
}
```

5.3 DFS recursiva (com Flood Fill)

Exemplo de uso no BEE 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;
}
```

5.4 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));
            }
        }
    }
}
```

5.5 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;
}
```

5.6 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;
}
```

5.7 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;
```

5.8 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
```

5.9 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);
}
```

5.10 Pontos de Articulação e Pontes (grafo não-dirigido)

```
#include <cstdio>
#include <vector>
#define UNVISITED -1
using namespace std;

int C, V;
int bridgeCount, dfsNumberCounter, rootChildren,
    dfsRoot;
vector<vector<int> > adj;
vector<int> articulation_vertex;
vector<int> dfs_parent, dfs_num, dfs_low;

void articulationPointAndBridge(int u){
    dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
    for(auto v : adj[u]){
        if(dfs_num[v] == UNVISITED){
            dfs_parent[v] = u;
            if(u == dfsRoot) rootChildren++;
            articulationPointAndBridge(v);
            if(dfs_low[v] >= dfs_num[u]){
                articulation_vertex[u] = true;
            }
            if(dfs_low[v] > dfs_num[u])
                // bridgeCount++;
                    // Edge is a bridge

                dfs_low[u] = min(dfs_low[u], dfs_low[v]);
        }else if(v != dfs_parent[u]) dfs_low[u] = min(
            dfs_low[u], dfs_num[v]);
    }
}

int main(){
    dfsNumberCounter = 0; dfs_num.assign(V, UNVISITED);
    dfs_low.assign(V, 0);
    dfs_parent.assign(V, 0); articulation_vertex.assign
        (V, 0);
    for(int i = 0; i < V; i++){
        if(dfs_num[i] == UNVISITED){
            dfsRoot = i; rootChildren = 0;
                articulationPointAndBridge(i);
            articulation_vertex[dfsRoot] = (
                rootChildren > 1);
        }
    }
    return 0;
}
```

5.11 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 = (int)(2e9); // 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;
}
```

5.12 Componentes Fortemente Conexos: Algoritmo de Tarjan

Descobre o número de componentes fortemente conexos em um grafo direcionado.

```
#define UNVISITED -1
typedef vector<int> vi;
int N, dfsNumberCounter, numSCC;
vi num,
, 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;
        }
    }
}
```

5.13 Componentes Fortemente Conexos: Algoritmo de Kosaraju

Cortesia do Macacário do ITA.

```
#define MAXN 100009

vector<int> adjList[MAXN], revAdjList[MAXN], ts;
bool vis[MAXN];
int comp[MAXN], parent = 0, numSCC;

void revdfs(int u) {
    vis[u] = true;
    for(int i = 0, v; i < (int)revAdjList[u].size(); i
        ++) {
        v = revAdjList[u][i];
        if(!vis[v]) revdfs(v);
    }
    ts.push_back(u);
}

void dfs(int u) {
    vis[u] = true; comp[u] = parent;
    for(int i = 0, v; i < (int)adjList[u].size(); i++)
        {
        v = adjList[u][i];
        if(!vis[v]) dfs(v);
    }
}

void kosaraju(int n) {
    memset(&vis, false, sizeof vis);
    for(int i = 0; i < n; i++)  {
        if(!vis[i]) revdfs(i);
    }
    memset(&vis, false, sizeof vis);
    numSCC = 0;
    for(int i = n-1; i >= 0; i--) {
        if(!vis[ts[i]]) {
            parent = ts[i];
            dfs(ts[i]);
            numSCC++;
        }
    }
}
```

5.14 Inverse Graph

Resolve problemas em que se deseja encontrar as componentes conexas quando são dadas as arestas que não pertencem ao grafo.

- Complexidade de tempo: $O(NlogN + NlogM)$

Cortesia do BRUTE-UDESC.

```
#include <bits/stdc++.h>
using namespace std;

set<int> nodes;
vector<set<int>> adj;

void bfs(int s) {
    queue<int> f;
    f.push(s);
    nodes.erase(s);
    set<int> aux;
    while (!f.empty()) {
        int x = f.front();
        f.pop();
        for (int y : nodes) {
            if (adj[x].count(y) == 0) {
                aux.insert(y);
            }
        }
        for (int y : aux) {
            f.push(y); nodes.erase(y);
        }
        aux.clear();
    }
}
```

5.15 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 = 1; 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);
}
```

5.16 Emparelhamento Máximo em Grafos Bipartidos

Exemplo no BEE 1056 - Fatores e Múltiplos

```
int A[MAX], B[MAX];
bool graph[MAX][MAX];

bool bipartiteMatch(int u, vector<bool> &visited,
    vector<int> &assign){
    for(int v = 0; v < M; v++){
        if(graph[u][v] && !visited[v]){
            visited[v] = true;
            if(assign[v] < 0 || bipartiteMatch(assign[v
                ], visited, assign)){
                assign[v] = u;
                return true;
            }
        }
    }
    return false;
}

// Encontra o emparelhamento máximo no grafo
int maxMatch(){
    vector<int> assign(M, -1);
    int count = 0;
    for(int u = 0; u < N; u++){
        vector<bool> visited(M, false);
        if(bipartiteMatch(u, visited, assign)) count++;
    }
    return count;
}

// Gera a matriz de adjacências onde graph[i][j] é 1 se
     j é multiplo de i
void geraGrafo(){
    memset(graph, 0, sizeof graph);
    for(int i = 0; i < N; i++){
        for(int j = 0; j < M; j++){
            if(A[i] == 0){
                // 0 é multiplo de 0
                if(B[j] == 0) graph[i][j] = true;
            }else{
                // quando 0 não é o denominador,
                    verifica-se a divisão
                if(B[j] % A[i] == 0)
                    graph[i][j] = true;
            }
        }
    }
}
```

5.17 Dinic - Max Flow

Exemplo no problema Gasolina, Maratona de Programação 2018

```
const int MAXV = 1123;

int A[MAXV]; // capacidade dos postos
int B[MAXV]; // capacidade das refinarias

typedef pair<int, pair<int, int> > edge;
typedef long long ll;

#define MAXN 21234
#define MAXM 51234

ll INF = 1e15;

edge edges[MAXN];

int ned, first[MAXN], work[MAXN];
ll cap[MAXM];
int to[MAXM], nxt[MAXM], dist[MAXM];

void init(){
    memset(first, -1, sizeof first);
    ned = 0;
}

void add(int u, int v, ll f){
    to[ned] = v, cap[ned] = f;
    nxt[ned] = first[u];
    first[u] = ned++;
    to[ned] = u, cap[ned] = 0;
    nxt[ned] = first[v];
    first[v] = ned++;
}

int dfs(int u, ll f, int s, int t){
    if(u == t) return f;
    int v, df;
    for(int &e = work[u]; e != -1; e = nxt[e]){
        v = to[e];
        if(dist[v] == dist[u] + 1 && cap[e] > 0){
            df = dfs(v, min(f, cap[e]), s, t);
            if(df > 0){
                cap[e] -= df;
                cap[e^1] += df;
                return df;
            }

        }
    }
    return 0;
}

bool bfs(int s, int t){
    int u, v;
    memset(&dist, -1, sizeof dist);
    dist[s] = 0;
    queue<int> q; q.push(s);
    while(!q.empty()){
        u = q.front(); q.pop();
        for(int e = first[u]; e != -1; e = nxt[e]){
            v = to[e];
            if(dist[v] < 0 && cap[e] > 0){
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
    return dist[t] >= 0;
}

ll dinic(int s, int t){
    ll result = 0, f;
    while(bfs(s, t)){
        memcpy(work, first, sizeof work);
        while(f = dfs(s, INF, s, t)) result += f;
    }
    return result;
}

int p, r, c;
long long total = 0;

bool binarySearch(int k){
    int S = p+r, T = S+1;
    int N = p+r+2;
    init();
    for(int i = 0; i < p; i++)
        add(i, T, A[i]); // posto -> sink
    for(int i = 0; i < r; i++)
        add(S, p+i, B[i]); // source -> refinaria
    for(int i = 0; i <= k; i++)
        add(p+edges[i].second.second, edges[i].second.
            first, (ll)1e10);
    return dinic(S, T) == total;
}

int main(){
    scanf("%d %d %d", &p, &r, &c);
    for(int i = 0; i < p; i++){
        scanf("%d", &A[i]);
        total += A[i];
    }
    for(int i = 0; i < r; i++)
        scanf("%d", &B[i]);
    int u, v, w;
    for(int i = 0; i < c; i++){
        scanf("%d %d %d", &u, &v, &w);
        u--, v--;
        edges[i] = {w, {u, v}};
    }
    sort(edges, edges+c);
    int l = 0, r = c-1;
    int ans = -1;
    while(l <= r){
        int mid = (l+r)/2;
        if(binarySearch(mid)){
            ans = edges[mid].first;
            r = mid-1;
        }else{
            l = mid+1;
        }
    }
    printf("%d\n", ans);
    return 0;
}
```

5.18 Edmonds–Karp (Fluxo)

```
int res[MAX_V][MAX_V], mf, f, s, t;
vector<vector<int>> adj;
vector<int> p;

void augment(int v, int minEdge) {
    if (v == s) {
        f = minEdge;
        return;
    } else if (p[v] != -1) {
        augment(p[v], min(minEdge, res[p[v]][v]));
        res[p[v]][v] -= f;
        res[v][p[v]] += f;
    }
}

void EdmondsKarp() {
    mf = 0;
    while (1) {
        f = 0;
        bitset<MAX_V> visited;
        visited.set(s);
        queue<int> q; q.push(s);
        p.assign(MAX_V, -1);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            if (u == t)
                break;
            for(auto v : adj[u]){
                if (res[u][v] > 0 && !visited.test(v))
                    {
                    visited.set(v); q.push(v); p[v] = u
                        ;
                }
            }
        }
        augment(t, INF);
        if (f == 0)
            break;
        mf += f;
    }
}
```

5.19 Min Cost Max Flow

Computa o fluxo máximo com custo mínimo.

- Complexidade de tempo: $O(V^2 * E^2)$

Cortesia do BRUTE-UDESC.

```
struct MinCostMaxFlow {
    int n, s, t, m = 0;
    ll maxflow = 0, mincost = 0;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;

    MinCostMaxFlow(int n, int s, int t) : n(n), s(s), t
        (t) {
        adj.resize(n);
    }

    void add_edge(int u, int v, ll cap, ll cost) {
        edges.emplace_back(u, v, cap, cost);
        edges.emplace_back(v, u, 0, -cost);
        adj[u].push_back(m);
        adj[v].push_back(m + 1);
        m += 2;
    }

    bool spfa() {
        vector <int> pego(n, -1);
        vector <ll> dis(n, INF);
        vector <bool> inq(n, false);
        queue <int> fila;
        fila.push(s);
        dis[s] = 0;
        inq[s] = 1;
        while (!fila.empty()) {
            int u = fila.front();
            fila.pop();
            inq[u] = false;
            for (int id : adj[u]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                int v = edges[id].v;
                if (dis[v] > dis[u] + edges[id].cost) {
                    dis[v] = dis[u] + edges[id].cost;
                    pego[v] = id;
                    if (!inq[v]) {
                        inq[v] = true;
                        fila.push(v);
                    }
                }
            }
        }

        if (pego[t] == -1) return 0;
        ll f = INF;
        for (int id = pego[t]; id != -1; id = pego[
            edges[id].u]) {
            f = min(f, edges[id].cap - edges[id].flow);
            mincost += edges[id].cost;
        }
        for (int id = pego[t]; id != -1; id = pego[
            edges[id].u]) {
            edges[id].flow += f;
            edges[id ^ 1].flow -= f;
        }
        maxflow += f;
        return 1;
    }

    ll flow() {
        while (spfa());
        return maxflow;
    }
};
```

5.20 Checa se um grafo é bipartido

```
bool ehBipartido(){
    for(int i = 0; i < N; i++) color[i] = -1;
    int s = 0;
    queue<int> q;
    q.push(s);
    color[s] = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (auto v : LG[u]) {
            if(color[v] == -1){
                color[v] = 1 - color[u];
                q.push(v);
            }else if(color[v] == color[u]) return false
                ;
        }
    }
    return true;
}
```

5.21 Tree Isomorphism

Colocar uma árvore no vetor A como lista de adjacências e outra no vetor B.

```
const int MAX = 30100;

vector<int> A[MAX], B[MAX];
vector<int> NA[MAX], NB[MAX];
int n;

bool comp(const vector<int>& a, const vector<int>& b){
    if(a.size() != b.size()) return a.size() < b.size()
        ;
    for(int i = 0; i < a.size(); i++){
        if(a[i] != b[i]) return a[i] < b[i];
    }
    return false;
}

bool eq(const vector<int>& a, const vector<int>& b){
    if(a.size() != b.size()) return false;
    for(int i = 0; i < a.size(); i++){
        if(a[i] != b[i]) return false;
    }
    return true;
}
bool treeIsomorphism(){
    memset(NA, 0, sizeof NA);
    memset(NB, 0, sizeof NB);
    for(int i = 1; i <= n; i++){
        for(int j = 0; j < A[i].size(); j++){
            NA[i].push_back(A[A[i][j]].size());
        }
        sort(NA[i].begin(), NA[i].end());

        for(int j = 0; j < B[i].size(); j++){
            NB[i].push_back(B[B[i][j]].size());
        }
        sort(NB[i].begin(), NB[i].end());
    }
    sort(NA+1, NA+n+1, comp);
    sort(NB+1, NB+n+1, comp);
    bool equals = true;
    for(int i = 1; i <= n; i++){
        equals &= eq(NA[i], NB[i]);
    }
    return equals;
}
```

5.22 Binary Lifting (sem LCA)

Usa uma sparse table para calcular o k-ésimo ancestral de u.
Pode ser usada com o algoritmo de EulerTour para calcular o LCA (versão na próxima página).
Complexidade de tempo:

- Pré-processamento: $O(N * log(N))$
- Consulta do k-ésimo ancestral de u: $O(log(N))$
- LCA: $O(log(N))$

Complexidade de espaço: $O(N * log(N))$
Cortesia do BRUTE-UDESC.

```
namespace st {
    int n, me;
    vector <vector<int>> st;
    void bl_dfs(int u, int p) {
        st[u][0] = p;
        for (int i = 1; i <= me; i++)
            st[u][i] = st[st[u][i-1]][i-1];
        for (int v : adj[u])
            if (v != p)
                bl_dfs(v, u);
    }
    void build(int _n, int root=0) {
        n = _n;
        me = floor(log2(n));
        st.assign(n, vector<int>(me+1, 0));
        bl_dfs(root, root);
    }
    int ancestor(int u, int k) { // k-th ancestor of u
        for (int i = me; i >= 0; i--)
            if ((1 << i) & k)
                u = st[u][i];
        return u;
    }
}
```

5.23 Binary Lifting (com LCA)

Cortesia do BRUTE-UDESC.

```
namespace st {
    int n, me, timer;
    vector <int> tin, tout;
    vector <vector <int>> st;
    void et_dfs(int u, int p) {
        tin[u] = ++timer;
        st[u][0] = p;
        for (int i = 1; i <= me; i++) {
            st[u][i] = st[st[u][i-1]][i-1];
        }
        for (int v : adj[u]) if (v != p) {
            et_dfs(v, u);
        }
        tout[u] = ++timer;
    }
    void build(int _n, int root=0) {
        n = _n;
        tin.assign(n, 0);
        tout.assign(n, 0);
        timer = 0;
        me = floor(log2(n));
        st.assign(n, vector <int>(me+1, 0));
        et_dfs(root, root);
    }
    bool is_ancestor(int u, int v) {
        return tin[u] <= tin[v] && tout[u] >= tout[v];}
    int lca(int u, int v) {
        if (is_ancestor(u, v)) return u;
        if (is_ancestor(v, u)) return v;
        for (int i = me; i >= 0; i--)
            if (!is_ancestor(st[u][i], v))
                u = st[u][i];
        return st[u][0];
    }
    int ancestor(int u, int k) { // k-th ancestor of u
        for (int i = me; i >= 0; i--)
            if ((1 << i) & k)
                u = st[u][i];
        return u;
    }
}
```

5.24 Graph Center

Encontra o centro e o diâmetro de um grafo.

- Complexidade de tempo: $O(N)$

Cortesia do BRUTE-UDESC.

```
const int INF = 1e9+9;

vector<vector<int>> adj;

struct GraphCenter{
    int n, diam = 0;
    vector<int> centros, dist, pai;
    int bfs(int s){
        queue<int> q; q.push(s);
        dist.assign(n+5, INF);
        pai.assign(n+5, -1);
        dist[s] = 0;
        int maxidist = 0, maxinode = 0;
        while(!q.empty()){
            int u = q.front(); q.pop();
            if(dist[u] >= maxidist)
                maxidist = dist[u], maxinode = u;
            for(int v : adj[u]){
                if (dist[u] + 1 < dist[v]){
                    dist[v] = dist[u] + 1;
                    pai[v] = u;
                    q.push(v);
                }
            }
        }
        diam = max(diam, maxidist);
        return maxinode;
    }
    GraphCenter(int st=0) : n(adj.size()) {
        int d1 = bfs(st);
        int d2 = bfs(d1);
        vector<int> path;
        for (int u = d2; u != -1; u = pai[u])
            path.push_back(u);
        int len = path.size();
        if(len%2 == 1)
            centros.push_back(path[len / 2]);
        else{
            centros.push_back(path[len/2]);
            centros.push_back(path[len/2 - 1]);
        }
    }
};
```

5.25 Heavy-Light Decomposition

Técnica usada para otimizar a execução de operações em árvores.

- Pré-processamento: $O(N)$
- Range Query/Update: $O(log(N)) * O$(complexidade de query da estrutura)
- Point Query/Update: O(complexidade de query da estrutura)
- LCA: $O(log(N))$
- Subtree Query: O(complexidade de query da estrutura)
- Complexidade de espaço: $O(N)$

Cortesia do BRUTE-UDESC.

```
namespace hld {
    const int MAX = 2e5+5;
    int t, sz[MAX], pos[MAX], pai[MAX], head[MAX];
    bool e = 0;
    ll merge(ll a, ll b) {
        return max(a, b); } // how to merge paths
    void dfs_sz(int u, int p=-1) {
        sz[u] = 1;
        for (int &v : adj[u]) if (v != p) {
            dfs_sz(v, u);
            sz[u] += sz[v];
            if (sz[v] > sz[adj[u][0]] || adj[u][0] == p
                )
                swap(v, adj[u][0]);
        }
    }
    void dfs_hld(int u, int p=-1) {
        pos[u] = t++;
        for (int v : adj[u]) if (v != p) {
            pai[v] = u;
            head[v] = (v == adj[u][0] ? head[u] : v);
            dfs_hld(v, u);
        }
    }
    void build(int root) {
        dfs_sz(root);
        t = 0;
        pai[root] = root;
        head[root] = root;
        dfs_hld(root);
    }
    void build(int root, vector<ll>& v) {
        build(root);
        vector<ll> aux(v.size());
        for (int i = 0; i < (int)v.size(); i++)
            aux[pos[i]] = v[i];
        seg::build(aux);
    }
    // use this if weighted edges
    void build(int root, vector<i3>& edges) {
        build(root);
        e = 1;
        vector<ll> aux(edges.size()+1);
        for (auto [u, v, w]: edges) {
            if (pos[u] > pos[v]) swap(u, v);
            aux[pos[v]] = w;
        }
        seg::build(aux);
    }
    ll query(int u, int v) {
        if (pos[u] > pos[v]) swap(u, v);
        if (head[u] == head[v])
            return seg::query(pos[u]+e, pos[v]);
        else {
            ll qv = seg::query(pos[head[v]], pos[v]);
            ll qu = query(u, pai[head[v]]);
            return merge(qu, qv);
        }
    }
    void update(int u, int v, ll k) {
        if (pos[u] > pos[v]) swap(u, v);
        if (head[u] == head[v])
            seg::update(pos[u]+e, pos[v], k);
        else {
            seg::update(pos[head[v]], pos[v], k);
            update(u, pai[head[v]], k);
        }
    }
    int lca(int u, int v) {
        if (pos[u] > pos[v]) swap(u, v);
        return (head[u] == head[v] ?
            u : lca(u, pai[head[v]]));
    }
    ll query_subtree(int u) {
        return seg::query(pos[u], pos[u]+sz[u]-1);
    }
}
```

Capítulo 6

Matemática

6.1 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;
}
```

6.2 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;
}
```

6.3 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;
}
```

6.4 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;
}
```

6.5 Teorema de Lucas e paridade de coeficientes binomiais

O Teorema de Lucas é uma forma de descobrir se o valor de um coeficiente binomial $\binom{n}{p}$ é par ou ímpar. Sendo $(p)_2$ a representação de p na base 2, dados dois inteiros p e q, dizemos que $(p)_2 \subseteq (q)_2$ se ao compararmos dois a dois os bits de $(p)_2$ e $(q)_2$ começando a partir do bit mais à direita temos que cada bit que compõe $(p)_2$ é menor ou igual ao correspondente bit que compõe $(q)_2$. **Teorema de Lucas:** Sejam n e k dois inteiros, $0 \le k \le n$. Então $\binom{n}{k}$ é ímpar se e somente se $(k)_2 \subseteq (n)_2$.

```
bool ehImpar(int n, int k){
    for(int i = 0; i < 64; i++){
        if(((k & (1 << i)) > (n & (1 << i)))) return false;
    }
    return true;
}
```

6.6 Aritmética Modular

MDC, MMC, euclides extendido, inverso modular $a^{-1}(mod m)$, divisão modular $(a/b)(mod m)$, exponenciação modular $a^b(mod m)$, solução inteira da equação de Diophantine $ax+by=c$. $modMul$ calcula $(a*b)\%m$ sem overflow. Triângulo de Pascal até 10^6. Todos os inversos modulares em relação a um primo p em $O(p)$. Resolve em $O(n\log n)$ o sistema $x \equiv a[i](mod\ p[i])$, $0 \le i < n$, $gcd(a[i], a[j]) = 1$ para todo $i \neq j$.

```
template <typename T>
T gcd(T a, T b) {
    return b == 0 ? a : gcd(b, a % b);
}

template <typename T>
T lcm(T a, T b) {
    return a * (b / gcd(a, b));
}

template <typename T>
T extGcd(T a, T b, T& x, T& y) {
    if (b == 0) {
        x = 1; y = 0; return a;
    }
    else {
        T g = extGcd(b, a % b, y, x);
        y -= a / b * x; return g;
    }
}

template <typename T>
T modInv(T a, T m) {
    T x, y;
    extGcd(a, m, x, y);
    return (x % m + m) % m;
}

template <typename T>
T modDiv(T a, T b, T m) {
    return ((a % m) * modInv(b, m)) % m;
}

template<typename T>
T modMul(T a, T b, T m) {
    T x = 0, y = a % m;
    while (b > 0) {
        if (b % 2 == 1) x = (x + y) % m;
        y = (y * 2) % m; b /= 2;
    }
    return x % m;
}

template<typename T>
T modExp(T a, T b, T m) {
    if (b == 0) return (T)1;
    T c = modExp(a, b / 2, m);
    c = (c * c) % m;
    if (b % 2 != 0) c = (c*a) % m;
    return c;
}

template<typename T>
void diophantine(T a, T b, T c, T& x, T& y) {
    T d = extGcd(a, b, x, y);
    x *= c / d; y *= c / d;
}

#define MAXN 1000009
typedef long long ll;

ll fat[MAXN];
void preprocessfat(ll m) {
    fat[0] = 1;
    for(ll i=1; i<MAXN; i++)
        fat[i] = (i*fat[i-1])%m;
}

template<typename T>
T pascal(int n, int k, T m) {
    return modDiv(fat[n], (fat[k]*fat[n-k])%m, m);
}

template<typename T>
void allInv(T inv[], T p) {
    inv[1] = 1;
    for (int i = 2; i < p; i++)
        inv[i] = (p - (p/i)*inv[p%i]%p)%p;
}

template<typename T>
T chinesert(T* a, T* p, int n, T m) {
    T P = 1;
    for(int i=0; i<n; i++) P = (P * p[i]) % m;
    T x = 0, pp;
    for(int i=0; i<n; i++) {
        pp = modDiv(P, p[i], m);
        x = (x + (((a[i] * pp) % m) * modInv(pp, p[i]))
            ) % m;
    }
    return x;
}
```

6.7 Teorema Chinês dos Restos generalizado

```
ll crt(ll rem[], ll mod[], int n) {
    if (n == 0) return 0;
    ll ans = rem[0], m = mod[0];
    for (int i = 1; i < n; i++) {
        ll x, y;
        ll g = extGcd(mod[i], m, x, y);
        if ((ans - rem[i])%g != 0) return -1;
        ans = ans + 1ll*(rem[i]-ans)*(m/g)*y;
        m = (mod[i]/g)*(m/g)*g;
        ans = (ans%m + m)%m;
    }
    return ans;
}
```

6.8 Números primos

Diversas operações com números primos. Crivo de Eratóstenes, número de divisores, totiente de Euler e número de diferentes fatores primos. *isPrimeSieve* funciona em $O(\sqrt{n}/\log n)$ se os fatores estiverem em *primes*.

```
#define MAXN 10000009
ll sievesize, numDiffPF[MAXN];
bitset<MAXN> bs;
vector<ll> primes;

void sieve(ll n) {
    sievesize = n + 1;
    bs.set();
    bs[0] = bs[1] = 0;
    for (ll i = 2; i <= sievesize; i++) {
        if (bs[i]) {
            for (ll j = i * i; j <= (ll)sievesize; j +=
                 i) bs[j] = 0;
            primes.push_back(i);
        }
    }
}
bool isPrimeSieve(ll N) {
    if (N <= (ll)sievesize) return bs[N];
    for (int i = 0; i < (int)primes.size() && primes[i
        ]*primes[i] <= N; i++)
        if (N % primes[i] == 0) return false;
    return true;
}
bool isPrime(ll N) {
    if (N < 0) return isPrime(-N);
    for(ll i=2; i*i <= N; i++) {
        if (N % i == 0) return false;
    }
    return true;
}
bool isPrimeFast(ll n) {
    if (n < 0) n = -n;
    if (n < 5 || n % 2 == 0 || n % 3 == 0)
        return (n == 2 || n == 3);
    ll maxP = sqrt(n) + 2;
    for (ll p = 5; p < maxP; p += 6) {
        if (p < n && n % p == 0) return false;
        if (p+2 < n && n % (p+2) == 0) return false;
    }
    return true;
}
vector<ll> primeFactors(ll N) {
    vector<int> factors;
    ll PF_idx = 0, PF = primes[PF_idx];
    while (PF * PF <= N) {
        while (N % PF == 0) {
            N /= PF;
            factors.push_back(PF);
        }
        PF = primes[++PF_idx];
    }
    if (N != 1) factors.push_back(N);
    return factors;
}
ll numDiv(ll N) {
    ll i = 0, p = primes[i], ans = 1;
    while (p * p <= N) {
        ll power = 0;
        while (N % p == 0) { N /= p; power++; }
        ans *= (power + 1);
        p = primes[++i];
    }
    if (N != 1) ans *= 2;
    return ans;
}
ll eulerPhi(ll N) {
    ll i = 0, p = primes[i], ans = N;
    while (p * p <= N) {
        if (N % p == 0) ans -= ans / p;
        while (N % p == 0) N /= p;
        p = primes[++i];
    }
    if (N != 1) ans -= ans / N;
    return ans;
}
void numDiffPf() {
    memset(numDiffPF, 0, sizeof numDiffPF);
    for (int i = 2; i < MAXN; i++)
        if (numDiffPF[i] == 0)
            for (int j = i; j < MAXN; j += i)
                numDiffPF[j]++;
}
```

6.9 Fórmula de Legendre

Dados um inteiro n e um primo p, calcula o expoente da maior potência de p que divide $n!$ em $O(\log n)$.

```
ll legendre(ll n, ll p) {
    int ans = 0;
    ll prod = p;
    while(prod <= n) {
        ans += n/prod;
        prod *= p;
    }
    return ans;
}
```

6.10 Soma de MDC

Pode-se usar o crivo de Eratóstenes para computar o $\phi(x)$ (totiente de Euler) e $F(x)$ para todo x de 1 a n em $O(n \log n)$, ou a fatoração para computar $F(n)$ em $O(\sqrt{n}/\log n)$. F é definida como:

$$F(n) = \sum_{i=1}^{n} gcd(i,n) = \sum_{d|n} d\phi(\frac{n}{d}) = \sum_{d|n} n\frac{\phi(d)}{d} \tag{6.1}$$

```
#define MAXN 200009

typedef unsigned long long ll;
int phi[MAXN], sievesize;
ll f[MAXN];

void gcdsieve(int n) {
    sievesize = n+1;
    for(int i=0; i <= sievesize; i++)
        phi[i] = f[i] = 0;
    for(int i = 1; i <= sievesize; i++) {
        phi[i] += i;
        for(int j = i; j <= sievesize; j += i) {
            if (j > i) phi[j] -= phi[i];
            f[j] += j / i * phi[i];
        }
    }
}

bitset<MAXN> bs;
vector<ll> primes;
void sieve(ll n) { ... }

ll F(ll N) {
    ll i = 0, p = primes[i], ans = 1;
    while (p * p <= N) {
        if (N % p == 0) {
            int e = 0;
            ll prod = 1;
            while (N % p == 0) {
                N /= p; e++; prod *= p;
            }
            prod /= p;
            ans *= prod * ((p-1)*e + p);
        }
        p = primes[++i];
    }
    ans *= 2*N-1;
    return ans;
}
```

6.11 Crivo linear e funções multiplicativas

Implementação alternativa do crivo de Eratóstenes em $O(n)$ e que computa funções multiplicativas: funções f tal que $f(p^k) = g(p,k), p$ primo e $f(pq) = f(p)f(q), gcd(p,q) = 1$. $f(1) = 1$ sempre.

Algumas funções multiplicativas comuns:

- Função constante: $I(p^k) = 1$;
- Função identidade: $Id(p^k) = p^k$;
- Função potência: $Id_a(p^k) = p^{ap}$;
- Função unidade: $\epsilon(p^k) = [k = 1]$;
- Função divisores de grau $a \geq 0$: $\sigma_a(p^k) = \sum_{i=0}^{k} p^{ai}$, $\sigma_a(n) = \sum_{d|n} d^a$;
- Função de Möbius: $\mu(p^k) = [k = 0] - [k = 1]$;
- Função totiente de Euler: $\phi(p^k) = p^k - p^{k-1}$.

```
vector<int> primes;
bitset<MAXN> bs;
int f[MAXN], pw[MAXN];

int g(int p, int k) {
    if (p == 1) return 1;
    return (k==0) - (k==1);
    //int q = 1;
    //for(int i = 1; i < k; i++) q *= p;
    //return q*(p-1);
}

void sieve(int n) {
    bs.set(); bs[0] = bs[1] = 0;
    primes.clear(); f[1] = g(1, 1);
    for (int i = 2; i <= n; i++) {
        if (bs[i]) {
            primes.push_back(i);
            f[i] = g(i, 1); pw[i] = 1;
        }
        for (int j = 0; j < primes.size() && i*1ll*
            primes[j] <= n; j++) {
            bs[i * primes[j]] = 0;
            if (i % primes[j] == 0) {
                int pwr = 1;
                for(int k = 0; k < pw[i]; k++) pwr *=
                    primes[j];
                f[i * primes[j]] = f[i / pwr] * g(
                    primes[j], pw[i]+1);
                pw[i * primes[j]] = pw[i] + 1;
                break;
            } else {
                f[i * primes[j]] = f[i] * f[primes[j]];
                pw[i * primes[j]] = 1;
            }
        }
    }
}
```

6.12 Inversão de Möbius

Seja $\mu(n)$ a função de Möbius, cujos valores até n podem ser calculados em $O(n)$ com o crivo linear. Sejam duas funções aritméticas f e g, então a fórmula da inversão de Möbius afirma que:

$$f(n) = \sum_{d|n} g(d) \rightarrow g(n) = \sum_{d|n} f(d)\mu(\frac{n}{d}) \tag{6.2}$$

Propriedades importantes:

- $\forall n, 1 = \sum_{d|n} \epsilon(d) \rightarrow \epsilon(n) = [n = 1] = \sum_{d|n} \mu(d)$;
- Para funções não aritméticas definidas em $[1, \inf)$: $f(x) = \sum_{i=1}^{x} g(\frac{x}{i}) \rightarrow g(x) = \sum_{i=1}^{x} \mu(i) f(\frac{x}{i})$;
- Inversão com truncamento: $f(n) = \sum_{i=1}^{n} g(\lfloor \frac{n}{i} \rfloor) \rightarrow g(n) = \sum_{i=1}^{n} \mu(i) f(\lfloor \frac{n}{i} \rfloor)$;
- Inversão multiplicativa: $f(n) = \prod_{d|n} g(d) \rightarrow g(n) = \prod_{d|n} f(d)^{\mu(\frac{n}{d})}$;
- Número de pares coprimos em $[1, n]$: $\sum_{i=1}^{n} \sum_{j=1}^{n} [gcd(i, j) = 1] = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k|gcd(i,j)} \mu(k) =$
  $= \sum_{k=1}^{n} \mu(k) \sum_{i=1}^{n} [k|i] \sum_{j=1}^{n} [k|j] = \sum_{k=1}^{n} \mu(k) \lfloor \frac{n}{k} \rfloor^2$;
- Função de Möbius em conjuntos parcialmente ordenados: $\mu(s, s) = 1, \mu(s, u) = -\sum_{s \leq t < u} \mu(s, t)$;
- Inversão de Möbius em conjuntos parcialmente ordenados: $f(t) = \sum_{s \leq t} g(s) \rightarrow g(t) = \sum_{s \leq t} \mu(s, t) f(s)$.

6.13 Números de Catalan

Números de Catalan podem ser computados pelas fórmulas:

$$Cat(0) = 1 \quad Cat(n) = \frac{4n-2}{n+1} Cat(n-1) = \sum_{i=0}^{n-1} Cat(i)Cat(n-1-i) = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} \tag{6.3}$$

- $Cat(n)$ = número de arvores binárias completas de $n+1$ folhas ou $2n+1$ elementos;
- $Cat(n)$ = número de combinações válidas para n pares de parêntesis;
- $Cat(n)$ = número de formas que o parentesiamento de $n+1$ elementos pode ser feito;
- $Cat(n)$ = número de triangulações de um polígono convexo de $n+2$ lados; e
- $Cat(n)$ = número de caminhos monotônicos discretos para ir de $(0, 0)$ a (n, n).
- Generalização: número de caminhos para ir de $(0, 0)$ a (x, y) que não cruzam $y - x \geq T = \binom{x+y}{y} - \binom{x+y}{y-T}$.

6.14 Números de Stirling de primeira espécie

Números de Stirling de primeira espécie $s(n, m), n \geq m$ podem ser calculados pela recursão $s(n, m) = s(n-1, m-1) - (n-1)s(n-1, m)$, com $s(n, n) = 1$ e $s(n, 0) = 0, n > 0$

- $s(n, m)$ = coeficiente de x^m em $P(x) = x(x+1)\cdots(x+n-1)$. Usar FFT pra computar $s(n, k), 0 \leq k \leq n$ em $O(n \log^2 n)$. Ver código abaixo (m = índice do módulo na NTT).
- $|s(n, m)|$ = número de permutações de tamanho n com exatamente m ciclos ou número de formas de alocar n pessoas em m mesas circulares.

```
void compute(vector<ll> & res, int l, int r, int m) {
    if (l == r) {
        res = vector<ll>({r, 1LL});
        return;
    }
    vector<ll> a, b;
    compute(a, l, (l + r) / 2, m);
    compute(b, (l + r) / 2 + 1, r, m);
    convolution(a, b, res, m);
}
```

```
void stirling_first(vector<ll> & res, int n, int m) {
    if (n == 0) {
        res = vector<ll>({1LL});
        return;
    }
    compute(res, 0, n-1, m);
    while(res.back() == 0) res.pop_back();
}
```

6.15 Números de Stirling de segunda espécie

Números de Stirling de segunda espécie $S(n,m), n \geq m$ podem ser calculados pela recursão: $S(n,m) = S(n-1,m-1) + mS(n-1,m)$, com $S(n,n) = 1$ e $S(n,0) = 0, n > 0$, ou pelo princípio da inclusão-exclusão:

$$S(n,m) = \frac{1}{m!}\sum_{i=0}^{m}(-1)^i\binom{m}{i}(n-i)^n \tag{6.4}$$

- $S(n,m)$ = número de formas de alocar n objetos em exatamente m conjuntos não vazios.
- $S(n,m)m!$ = número de funções sobrejetoras de um conjunto de n elementos em um de m elementos.

6.16 Identidades de soma de binômio

Identidade de Vandermonde, identidade da meia de natal (ou taco de hockey) e teorema multinomial.

$$(x+y)^n = \sum_{i=0}^{n}\binom{n}{i}x^iy^{n-i}, \qquad \binom{m+n}{k} = \sum_{i=0}^{k}\binom{m}{i}\binom{n}{k-i}, \qquad \binom{k+n}{k-1} = \sum_{i=0}^{k-1}\binom{n+i}{i} \tag{6.5}$$

$$(x_1+x_2+\cdots+x_m)^n = \sum_{k_1+k_2+\cdots+k_m=n}\binom{n}{k_1,k_2,\cdots,k_m}\prod_{t=1}^{m}x_t^{k_t} \tag{6.6}$$

6.17 Lemma de Burnside e Teorema da Enumeração de Pólya

Seja X um conjunto e G um conjunto de transformações de elementos de X em outros elementos de X. Para cada transformação $g \in G$, tem-se $I(g)$ elementos $x \in X$ tal que $g(x) = x$. O subconjunto máximo X/G de X é tal que se $x,y \in X/G$, não existe $g \in G$ tal que $g(x) = y$. Outra forma de se pensar é que cada elemento $x \in X$ é uma *representação* e está associada a um único *objeto*. Um *objeto* pode ter várias *representações* associadas a si. G é o conjunto de transformações transitivas e invariantes, ou seja, para todo $g \in G$, se dois elementos x e y estão associados a um mesmo objeto, então $g(x)$ e $g(y)$ também estão. $|X/G|$ representa o número de classes de equivalências de G, o número de *objetos* distintos. O lemma de Burnside é dado pela fórmula à esquerda. Caso G seja um conjunto de permutações, seja $C(g)$ o número de ciclos de uma permutação g e k o número de possíveis elementos para preencher a array que representa um elemento $x \in X$. O teorema da enumeração de Pólya é dado à direita. CUIDADO: G deve ser transitivo, ou seja, se $f,g \in G$, então $f \circ g \in G$.

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}I(g) \qquad |X/G| = \frac{1}{|G|}\sum_{g\in G}k^{C(g)} \tag{6.7}$$

6.18 Teste de Primalidade de Miller-Rabin

$O(k\log^2 n)$. Probabilístico, mas provado correto para $n < 2^{64}$ com $k = 9$.

```
template<typename T>
T modMulExp(T a, T b, T m) {
    if (b == 0) return (T)1;
    T c = modMulExp(a, b / 2, m);
    c = modMul(c, c, m);
    if (b % 2 != 0) c = modMul(c, a, m);
    return c;
}

bool miller(long long n) {
    const int pn = 9;
    const int p[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    for (int i = 0; i < pn; i++)
        if (n % p[i] == 0) return n == p[i];
    if (n < p[pn - 1]) return false;
    long long s = 0, t = n - 1;
    while (~t & 1) t >>= 1, ++s;
    for (int i = 0; i < pn; ++i) {
        long long pt = modMulExp((long long)p[i], t, n)
            ;
        if (pt == 1) continue;
        bool ok = false;
        for (int j = 0; j < s && !ok; j++) {
            if (pt == n - 1) ok = true;
            pt = modMul(pt, pt, n);
        }
        if (!ok) return false;
    }
    return true;
}
```

6.19 Algoritmo de Pollard-Rho

Retorna um fator de n, usar para $n > 9 \times 10^{13}$.

```
template<typename T>
T pollard(T n) {
    int i = 0, k = 2, d;
    T x = 3, y = 3;
    while (++i) {
        x = (modMul(x, x, n) + n - 1) % n;
        d = gcd(abs(y - x), n);
        if (d != 1 && d != n) return d;
        if (i == k) y = x, k *= 2;
    }
}
```

6.20 Baby-Step Giant-Step para Logaritmo Discreto

Resolve a equação $a^x \equiv b \mod m$ em $O(\sqrt{m}\log m)$. Retorna -1 se não há solução.

```
template <typename T>
T baby (T a, T b, T m) {
    a %= m; b %= m;
    T n = (T) sqrt (m + .0) + 1, an = 1;
    for (T i=0; i<n; ++i) an = (an * a) % m;
    map<T,T> vals;
    for (T i=1, cur=an; i<=n; ++i) {
        if (!vals.count(cur)) vals[cur] = i;
        cur = (cur * an) % m;
    }
    for (T i=0, cur=b; i<=n; ++i) {
        if (vals.count(cur)) {
            T ans = vals[cur] * n - i;
            if (ans < m) return ans;
        }
        cur = (cur * a) % m;
    }
    return -1;
}
```

6.21 Jogo de Nim e teorema de Sprague-Grundy

- Jogo de nim: dois jogadores navegam por um DAG (normalmente modelado a partir das regras de um jogo), em cada jogada um escolhe por qual aresta andar. Cada nó é um estado do jogo. Existe um estado final que é uma posição perdedora (se os jogadores estiverem nele, o jogador atual perde).
- Equivalente de nim g: o jogador atual tem estratégia vencedora se e somente se o equivalente de nim g_u do estado atual u é diferente de zero, ou seja, $g_u \neq 0$ (Teorema de Bouton).
- Teorema de Sprague-Grundy: em um estado u, $g_u = 0$ se for o estado final, $g_u = mex_v(g_v) \forall v$ alcançável por u. $mex(a_1, a_2, ..., a_n)$ é o primeiro número maior que ou igual a zero que não aparece no conjunto $\{a_1, a_2, ..., a_n\}$.
- Jogo de nim em paralelo: se $u_1, u_2, ..., u_n$ forem os estados atuais de cada jogo jogado em paralelo, o equivalente de nim geral é o xor de todos os individuais ($g_{u_1} \oplus g_{u_2} \oplus ... \oplus g_{u_n}$).
- Misère nim: é igual ao jogo de nim, mas o estado final é uma posição vencedora. Caso exista algum estado u com $g_u > 1$, então usa-se o equivalente de nim. Caso contrário, usa-se a paridade do número de estados mais um.
- Método de Steve Halim: é muito comum usar o método para imprimir todos os possíveis $mex[i]$ no código, pois o grafo normalmente é descrito pelas regras de um jogo e é, portanto, constante.

```
#include <set>
using namespace std;
#define MAXN 10009

void stevehalim() {
    for(int n=1; n<MAXN; n++) {
        if (n <= 2) { mex[n] = 0; continue; }
        set<int> jog;
        for(int i=3; i<=n-2; i++) {
            jog.insert(mex[i-1]^mex[n-i]);
        }
        int cnt = 0;
        while(jog.count(cnt)) cnt++;
        mex[n] = cnt;
    }
    //for(int n=1; n<MAXN; n++)
    //  printf("%d,", mex[n]);
}
```

6.22 Triplas Pitagóricas

Todas as triplas pitagóricas (a, b, c), $a^2 + b^2 = c^2$ podem ser geradas a partir das equações ($k = 1$ gera triplas primitivas):

$$a = k(m^2 - n^2), \quad b = 2kmn, \quad c = k(m^2 + n^2) \tag{6.8}$$

6.23 Matrizes

```
typedef vector< vector< double > > matrix;

matrix operator +(matrix a, matrix b) {
    int n = (int)a.size();
    int m = (int)a[0].size();
    matrix c;
    c.resize(n);
    for(int i=0; i<n; i++) {
        c[i].resize(m);
        for(int j=0; j<m; j++)
            c[i][j] = a[i][j] + b[i][j];
    }
    return c;
}

matrix operator *(matrix a, matrix b) {
    int n = (int)a.size();
    int m = (int)b.size();
    int p = (int)b[0].size();
    matrix c(n, vector<double>(p));
    vector<double> col(m);
    for (int j = 0; j < p; j++) {
        for (int k = 0; k < m; k++)
            col[k] = b[k][j]; //cache friendly
        for (int i = 0; i < n; i++) {
            double s = 0;
            for (int k = 0; k < m; k++)
                s += a[i][k] * col[k];
            c[i][j] = s;
        }
    }
    return c;
}

matrix operator *(double k, matrix a) {
    int n = (int)a.size();
    int m = (int)a[0].size();
    for(int i=0; i<n; i++) for(int j=0; j<m; j++)
        a[i][j] *= k;
    return a;
}

matrix operator -(matrix a, matrix b) {
    return a + ((-1.0) * b);
}
```

6.24 Exponenciação de matrizes e Fibonacci

Calcula o n-ésimo termo de fibonacci em tempo $O(\log n)$. Calcula uma matriz $m \times m$ elevado a n em $O(m^3 \log n)$.

```
matrix matrixExp(matrix a, int n) {
    if (n == 0) return id(a.size());
    matrix c = matrixExp(a, n/2);
    c = c*c;
    if (n%2 != 0) c = c*a;
    return c;
}

matrix fibo() {
    matrix c(2, vector<double>(2, 1));
    c[1][1] = 0;
    return c;
}

double fibo(int n) {
    matrix f = matrixExp(fibo(), n);
    return f[0][1];
}
```

6.25 Sistemas Lineares: Determinante e Eliminação de Gauss

$gauss(A,B)$ retorna se o sistema $Ax = B$ possui solução e executa a eliminação de Gauss em A e B. $det(A)$ computa o determinante em $O(n^3)$ por Eliminação de Gauss.

```
void switchLines(matrix & a, int i, int j) {
    int m = (int)a[i].size();
    for(int k = 0; k < m; k++)
        swap(a[i][k], a[j][k]);
}

void lineSumTo(matrix & a, int i, int j, double c) {
    int m = (int)a[0].size();
    for(int k = 0; k < m; k++) a[j][k] += c*a[i][k];
}

bool gauss(matrix & a, matrix & b, int & switches) {
    switches = 0;
    int n = (int)a.size();
    int m = (int)a[0].size();
    for(int i = 0, l; i < min(n, m); i++) {
        l = i;
        while(l < n && fabs(a[l][i]) < EPS) l++;
        if (l == n) return false;
        switchLines(a, i, l);
        switchLines(b, i, l);
        switches++;
        for(int j=0; j<n; j++) {
            if (i == j) continue;
            double p = -a[j][i] / a[i][i];
            lineSumTo(a, i, j, p);
            lineSumTo(b, i, j, p);
        }
    }
    return true;
}

double det(matrix a) {
    int n = a.size();
    matrix b(n);
    for(int i=0; i<n; i++) b[i].resize(1);
    int sw = 0;
    if (gauss(a, b, sw)) {
        double ans = 1;
        for(int i=0; i<n; i++) ans *= a[i][i];
        return sw % 2 == 0 ? ans : -ans;
    }
    return 0.0;
}
```

6.26 Multiplicação de matriz esparsa

Multiplica duas matrizes em $O(n^2 m)$, onde m é o mínimo do número médio de números não nulos em cada linha e coluna.

```
vector< vector<int> > adjA, adjB;

matrix sparsemult(matrix a, matrix b) {
    int n = (int)a.size();
    //assert(a[0].size() == b.size());
    int m = (int)b.size();
    int p = (int)b[0].size();
    adjA.resize(n);
    for(int i=0; i<n; i++) {
        adjA[i].clear();
        for(int k=0; k<m; k++)
            if (fabs(a[i][k]) > EPS)
                adjA[i].push_back(k);
    }
    adjB.resize(p);
    for(int j=0; j<p; j++) {
        adjB[j].clear();
        for(int k=0; k<m; k++)
            if (fabs(b[k][j]) > EPS)
                adjB[j].push_back(k);
    }
    matrix c;
    c.resize(n);
    for(int i=0; i<n; i++) {
        c[i].assign(p, 0);
        for(int j=0; j<p; j++)
            for(int u=0, v=0, k; u<(int)adjA[i].size()
                && v<(int)adjB[j].size();) {
                if (adjA[i][u] > adjB[j][v]) v++;
                else if (adjA[i][u] < adjB[j][v]) u++;
                else {
                    k = adjA[i][u];
                    c[i][j] += a[i][k]*b[k][j];
                    u++; v++;
                }
            }
    }
    return c;
}
```

6.27 Método de Gauss-Seidel

Resolve o sistema iterativamente em $O(n^2 \log PREC^{-1})$. É necessário que a diagonal principal seja dominante.

```
matrix gaussSeidel(matrix & a, matrix & b, double PREC)
     {
    int n = (int)a.size();
    matrix x = b, xp = b;
    double error;
    do {
        error = 0.0;
        for(int i=0; i<n; i++) {
            xp[i][0] = b[i][0];
            for(int j=0; j<n; j++) {
                if (i < j) xp[i][0] -= a[i][j]*xp[j
                    ][0];
                if (i > j) xp[i][0] -= a[i][j]*x[j][0];
            }
            xp[i][0] /= a[i][i];
            error = max(error, fabs(xp[i][0]-x[i][0]));
        }
        x = xp;
    } while(error > PREC);
    return xp;
}
```

6.28 XOR-SAT

Executa a eliminação gaussiana com xor sobre o sistema $Ax = b$ em $O(nm^2/64)$.

```
#define MAXN 2009

vector<int> B;
vector< bitset<MAXN> > A;
bitset<MAXN> x;

bool check() {
    int n = A.size(), m = MAXN;
    for(int i = 0; i < n; i++) {
        int acum = 0;
        for(int j = 0; j < m; j++) {
            if (A[i][j]) acum ^= x[j];
        }
        if (acum != B[i]) return false;
    }
    return true;
}
bool gaussxor() {
    int cnt = 0, n = A.size(), m = MAXN;
    bitset<MAXN> vis; vis.set();
    for(int j = m-1, i; j >= 0; j--) {
        for(i = cnt; i < n; i++) {
            if (A[i][j]) break;
        }
        if (i == n) continue;
        swap(A[i], A[cnt]); swap(B[i], B[cnt]);
        i = cnt++; vis[j] = 0;
        for(int k = 0; k < n; k++) {
            if (i == k || !A[k][j]) continue;
            A[k] ^= A[i]; B[k] ^= B[i];
        }
    }
    x = vis;
    for(int i = 0; i < n; i++) {
        int acum = 0;
        for(int j = 0; j < m; j++) {
            if (!A[i][j]) continue;
            if (!vis[j]) {
                vis[j] = 1;
                x[j] = acum^B[i];
            }
            acum ^= x[j];
        }
        if (acum != B[i]) return false;
    }
    return true;
}
```

6.29 Fast Fourier Transform (FFT)

Usar em caso de double. Em caso de inteiro converter com $int(a[i].real() + 0.5)$. Usar struct caso precise ser rápido.

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

typedef complex<double> base;

void fft(vector<base> &a, bool invert) {
    int n = (int)a.size();
    for(int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for(; j >= bit; bit >>= 1) j -= bit;
        j += bit;
        if (i < j) swap(a[i], a[j]);
    }
    for(int len = 2; len <= n; len <<= 1) {
        double ang = 2*acos(-1.0)/len * (invert ? -1 :
            1);
        base wlen(cos(ang), sin(ang));
        for(int i = 0; i < n; i += len) {
            base w(1);
            for(int j = 0; j < len/2; j++) {
                base u = a[i+j], v = a[i+j+len/2] * w;
                a[i + j] = u + v;
                a[i + j + len/2] = u - v;
                w *= wlen;
            }
        }
    }
    for (int i = 0; invert && i < n; i++) a[i] /= n;
}

void convolution(vector<base> a, vector<base> b, vector
    <base> &res) {
    int n = 1;
    while(n < max(a.size(), b.size())) n <<= 1;
    n <<= 1;
    a.resize(n), b.resize(n);
    fft(a, false); fft(b, false);
    res.resize(n);
    for(int i=0; i<n; ++i) res[i] = a[i]*b[i];
    fft(res, true);
}
```

6.30 Number Theoretic Transform (NTT)

Usar long long. Cuidado com overflow. m é o primo selecionado. O resultado é calculado $mod[m]$.

```
template <typename T>
T extGcd(T a, T b, T& x, T& y) { ... }

template <typename T>
T modInv(T a, T m) { ... }

const ll mod[3] = {1004535809LL, 1092616193LL,
    998244353LL};
const ll root[3] = {12289LL, 23747LL, 15311432LL};
const ll root_1[3] = {313564925LL, 642907570LL,
    469870224LL};
const ll root_pw[3] = {1LL<<21, 1LL<<21, 1LL<<23};

void ntt(vector<ll> & a, bool invert, int m) {
    ll n = (ll)a.size();
    for(ll i = 1, j = 0; i < n; i++) {
        ll bit = n >> 1;
        for (; j >= bit; bit >>= 1) j -= bit;
        j += bit;
        if (i < j) swap(a[i], a[j]);
    }
    for(ll len = 2, wlen; len <= n; len <<= 1) {
        wlen = invert ? root_1[m] : root[m];
        for (ll i = len; i < root_pw[m]; i <<= 1)
            wlen = (wlen * wlen % mod[m]);
        for(ll i = 0; i < n; i += len) {
            for(ll j = 0, w = 1; j < len/2; j++) {
                ll u = a[i+j],  v = a[i+j+len/2] * w %
                    mod[m];
                a[i+j] = (u+v < mod[m] ? u+v : u+v-mod[
                    m]);
                a[i+j+len/2] = (u-v >= 0 ? u-v : u-v+
                    mod[m]);
                w = w * wlen % mod[m];
            }
        }
    }
    if (invert) {
        ll nrev = modInv(n, mod[m]);
        for (ll i=0; i<n; ++i)
            a[i] = a[i] * nrev % mod[m];
    }
}

void convolution(vector<ll> a, vector<ll> b, vector<ll>
     & res, int m) {
    ll n = 1;
    while(n < max (a.size(), b.size())) n <<= 1;
    n <<= 1;
    a.resize(n), b.resize(n);
    ntt(a, false, m); ntt(b, false, m);
    res.resize(n);
    for(int i=0; i<n; ++i)
        res[i] = (a[i]*b[i])%mod[m];
    ntt(res, true, m);
}
```

6.31 Convolução circular

Utiliza FFT/NTT para computar em $O(n \log n)$: $res[i] = \sum_{j=0}^{n-1} a[j]b[(i-j+n)\%n]$

```
void circularConv(vector<base> &a, vector<base> &b,
    vector<base> &res) { // fft
    //assert(a.size() == b.size());
    int n = a.size();
    convolution(a, b, res);
    for(int i = n; i < (int)res.size(); i++)
        res[i%n] += res[i];
    res.resize(n);
}
```

6.32 Números complexos

```
struct base { // faster than complex<double>
    double x, y;
    base() : x(0), y(0) {}
    base(double a, double b=0) : x(a), y(b) {}
    base operator/=(double k) { x/=k; y/=k; return (*
        this); }
    base operator*(base a) const { return base(x*a.x -
        y*a.y, x*a.y + y*a.x); }
    base operator*=(base a) {
        double tx = x*a.x - y*a.y;
        double ty = x*a.y + y*a.x;
        x = tx; y = ty;
        return (*this);
    }
    base operator=(double a) { x=a; y=0; return (*this)
        ; }
    base operator+(base a) const { return base(x+a.x, y
        +a.y); }
    base operator+=(base a) { x+=a.x; y+=a.y; return (*
        this); }
    base operator-(base a) const { return base(x-a.x, y
        -a.y); }
    base operator-=(base a) { x-=a.x; y-=a.y; return (*
        this); }
    double& real() { return x; }
    double& imag() { return y; }
};
```

6.33 Divisão de polinômios

Divisão e resto polinômios em $O(n \log n)$, em que n é o grau do dividendo. $inverse(A(x), t)$ computa os t primeiros termos da série infinita $1/A(x)$ em $O(n \log n)$.

```
void inverse(vector<base> &a, int t) {
    if (t == 1) {
        a.resize(1); a[0].real() = 1.0/a[0].real();
        return;
    }
    vector<base> a0 = a;
    inverse(a0, (t/2) + (t%2));
    convolution(a0, a, a);
    a0.resize(t), a.resize(t);
    a[0] -= base(1.0);
    convolution(a0, a, a);
    a0.resize(t), a.resize(t);
    for(int i = 0; i < t; i++) a[i] = a0[i] - a[i];
}
void divide(vector<base> &a, vector<base> &b, vector<
    base> &d) {
    int n = a.size(), m = b.size();
    if (n < m) {
        d.clear(); d.push_back(0);
        return;
    }
    vector<base> ar = a, br = b;
    reverse(ar.begin(), ar.end());
    reverse(br.begin(), br.end());
    inverse(br, n - m + 1);
    convolution(ar, br, d);
    d.resize(n - m + 1);
    reverse(d.begin(), d.end());
}
void remainder(vector<base> &a, vector<base> &b, vector
    <base> &r) {
    int n = a.size(), m = b.size();
    if (n < m) { r = a; return; }
    vector<base> d, aux;
    divide(a, b, d);
    r = a;
    convolution(d, b, aux);
    r.resize(m-1); aux.resize(m-1);
    for(int i = 0; i < m-1; i++) r[i] -= aux[i];
}
```

6.34 Evaluação em múltiplos pontos

$roots(x[], n)$ computa o polinômio $A(x) = \prod_{i=0}^{n-1}(x - x[i])$ em $O(n \log^2 n)$. $multievaluate(A(x), x, y, n)$ calcula $y[i] = A(x[i]) \forall 0 \leq i < n$ em $O(n \log^2 n)$.

```
#define MAXN 10000

vector<base> rt[4*MAXN];
void roots(int u, double x[], int i, int j) {
    if (i == j) {
        rt[u].resize(2); rt[u][0] = -x[i];
        rt[u][1] = 1.0; return;
    }
    int m = (i + j) / 2;
    roots(2*u, x, i, m); roots(2*u+1, x, m+1, j);
    convolution(rt[2*u], rt[2*u+1], rt[u]);
    rt[u].resize(j-i+2);
}
void roots(vector<base> &a, double x[], int n) {
    roots(1, x, 0, n-1); a = rt[1];
}
vector<base> et[4*MAXN];
void multievaluate(int u, double x[], double y[], int i
    , int j) {
    if (i == j) {
        y[i] = 0;
        double p = 1;
        for(int k = 0; k < (int)et[u].size(); k++)
            y[i] += et[u][k].real()*p, p *= x[i];
        return;
    }
    remainder(et[u], rt[2*u], et[2*u]);
    remainder(et[u], rt[2*u+1], et[2*u+1]);
    int m = (i + j) / 2;
    multievaluate(2*u, x, y, i, m);
    multievaluate(2*u+1, x, y, m+1, j);
}
void multievaluate(vector<base> &a, double x[], double
    y[], int n) {
    roots(1, x, 0, n-1); et[1] = a;
    multievaluate(1, x, y, 0, n-1);
}
```

6.35 Interpolação de polinômios

Calcula o polinômio de grau n que passa pelos pontos $(x[i], y[i]), 0 \leq i < n$ em $O(n \log^3 n)$. Também pode-se resolver em $O(n^2 \log n)$ com o polinômio interpolador de Lagrange $p(x) = \sum_{i=0}^{n-1} \prod_{j \ni} \frac{x - x[j]}{x[i] - x[j]}$ ou em $O(n^3)$ com eliminassão gaussiana.

```
vector<double> y[MAXN];
double ya0[MAXN], yp[MAXN];
void interpolate(vector<base> &a, int u, double x[],
     int n) {
    if (n == 1) {
        a.resize(1); a[0] = y[u][0];
        return;
    }
    y[2*u] = y[u]; y[2*u].resize(n/2);
    vector<base> a0, a1, p;
    interpolate(a0, 2*u, x, n/2);
    roots(p, x, n/2);
    int m = n-(n/2);
    multievaluate(a0, x+(n/2), ya0, m);
    multievaluate(p,  x+(n/2), yp, m);
    y[2*u+1].resize(m);
    for(int i = 0; i < m; i++)
        y[2*u+1][i] = (y[u][n/2 + i] - ya0[i]) / yp[i];
    interpolate(a1, 2*u+1, x+(n/2), m);
    convolution(p, a1, a);
    int sz = max(a.size(), a0.size());
    a.resize(sz);
    for(int i = 0; i < sz && i < n/2; i++)
        a[i] += a0[i];
    a.resize(n);
}
void interpolate(vector<base> &a, double x[], double ry
    [], int n) {
    y[1].resize(n);
    for(int i = 0; i < n; i++) y[1][i] = ry[i];
    interpolate(a, 1, x, n);
}
```

6.36 Fast Walsh–Hadamard Transform

Computa a convolução com XOR: o termo $a[i]b[j]$ é somado em $c[i \oplus j]$, OR: o termo $a[i]b[j]$ é somado em $c[i|j]$, AND: o termo $a[i]b[j]$ é somado em $c[i\&j]$ em $O(n \log n)$. Em caso de inteiro converter com $f\,a[i] + 0.5$.

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

//int mat[2][2] = {{1, 1}, {1, -1}}, inv[2][2] = {{1,
    1}, {1, -1}}; //xor
int mat[2][2] = {{1, 1}, {1, 0}}, inv[2][2] = {{0, 1},
    {1, -1}}; //or
//int mat[2][2] = {{0, 1}, {1, 1}}, inv[2][2] = {{-1,
    1}, {1, 0}}; //and

void fwht(vector<double> & a, bool invert) {
    int n = (int)a.size();
    double u, v;
    for (int len = 1; 2 * len <= n; len <<= 1) {
        for (int i = 0; i < n; i += 2 * len) {
            for (int j = 0; j < len; j++) {
                u = a[i + j];
                v = a[i + len + j];
                if (!invert) {
                    a[i + j] = mat[0][0]*u + mat[0][1]*
                        v;
                    a[i + len + j] = mat[1][0]*u + mat
                        [1][1]*v;
                }
                else {
                    a[i + j] = inv[0][0]*u + inv[0][1]*
                        v;
                    a[i + len + j] = inv[1][0]*u + inv
                        [1][1]*v;
                }
            }
        }
    }
    //for (int i=0; invert && i<n; ++i) a[i] /= n; //
        xor
}

void convolution(vector<double> a, vector<double> b,
    vector<double> & res) {
    int n = 1;
    while(n < max(a.size(), b.size())) n <<= 1;
    a.resize(n), b.resize(n);
    fwht(a, false); fwht(b, false);
    res.resize(n);
    for(int i=0; i<n; ++i) res[i] = a[i]*b[i];
    fwht(res, true);
}
```

6.37 Convolução com CRT

Utiliza o teorema chinês dos restos e duas NTT's para calcular a resposta módulo $mod[0] * mod[1] = 1,097,572,091,361,755,137$. Este número é normalmente grande o suficiente para calcular os valores exatos se as arrays originais tiverem cada elemento menor que aproximadamente 10^6 e $n \leq 2^{20}$. Implementação do teorema chinês dos restos por cortesia do IME.

```
template<typename T>
T modMul(T a, T b, T m) { ... }

//convolution mod 1,097,572,091,361,755,137
void modConv(vector<ll> a, vector<ll> b, vector<ll> &
    res) {
    vector<ll> r0, r1;
    convolution(a, b, r0, 0);
    convolution(a, b, r1, 1);
    ll x, y, s, r, p = mod[0]*mod[1];
    extGcd(mod[0], mod[1], r, s);
    res.resize(r0.size());
    for(int i=0; i<(int)res.size(); i++) {
        res[i] = (modMul((s*mod[1]+p)%p, r0[i], p)
        + modMul((r*mod[0]+p)%p, r1[i], p) + p) % p;
    }
}
```

6.38 Convolução com Decomposição SQRT

Se os números forem menores que aproximadamente 10^6, separa a primeira metade de bits da segunda em cada array e executa 4 FFT's com números menores que aproximadamente 10^3. Isso permite a FFT complexa com double ter precisão suficiente pra calcular de forma exata. Depois basta juntar.

```
#include <cmath>
#define MOD 1000003LL
#define SMOD 1024LL // ~ sqrt(MOD)
typedef long long ll;

void sqrtConv(vector<ll> a, vector<ll> b, vector<ll> &
    c) {
    vector<base> ca[2], cb[2], cc[2][2];
    ca[0].resize(a.size());
    ca[1].resize(a.size());
    for(int i=0; i<(int)a.size(); i++) {
        ca[0][i] = base(a[i] % SMOD, 0);
        ca[1][i] = base(a[i] / SMOD, 0);
    }
    cb[0].resize(b.size());
    cb[1].resize(b.size());
    for(int i=0; i<(int)b.size(); i++) {
        cb[0][i] = base(b[i] % SMOD, 0);
        cb[1][i] = base(b[i] / SMOD, 0);
    }
    for(int l=0; l<2; l++) for(int r=0; r<2; r++)
        convolution(ca[l], cb[r], cc[l][r]);
    c.resize(cc[0][0].size());
    for(int i=0; i<(int)c.size(); i++) {
        c[i] =
        (((ll)round(cc[1][1][i].real()))%MOD*(SMOD*SMOD
            )%MOD)%MOD +
        ((ll)round(cc[0][1][i].real()))%MOD*SMOD%MOD +
        ((ll)round(cc[1][0][i].real()))%MOD*SMOD%MOD +
        ((ll)round(cc[0][0][i].real()))%MOD;
        c[i] %= MOD;
    }
}
```

6.39 Integração pela regra de Simpson

Integração por interporlação quadrática. Erro: $\frac{h^4}{180}(b-a)max_{x\in[a,b]}|f^{(4)}(x)|$, $h=(b-a)/n$.

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

double simpson(double a, double b, int n = 1e6) {
    double h = (b - a) / n, s = f(a) + f(b);
    for (int i = 1; i < n; i += 2) s += 4*f(a+h*i);
    for (int i = 2; i < n; i += 2) s += 2*f(a+h*i);
    return s*h/3;
}
```

6.40 Código de Gray

Converte para o cógido de gray, ida $O(1)$ e volta $O(\log n)$. Útil para gerar números consecutivos que diferem por 1 bit. Caso se fixe o número de bits 1 do código, eles saem em ordem a diferirem por uma posição.

```
int gray(int n) { return n ^ (n >> 1); }

int rev_gray(int g) {
    int n = 0;
    for (; g; g>>=1) n ^= g;
    return n;
}
```

6.41 BigInteger em Java

```
import java.util.Scanner;
import java.math.BigInteger;

public final class Main { /* UVa 10925 - Krakovia */
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int caseNo = 1;
        while (true) {
            int N = sc.nextInt(), F = sc.nextInt();
            if (N == 0 && F == 0) break;
            BigInteger sum = BigInteger.ZERO;
            for (int i = 0; i < N; i++) {
                BigInteger V = sc.nextBigInteger();
                sum = sum.add(V);
            }
            System.out.println("Bill #" +
                (caseNo++) + " costs " + sum +
            ": each friend should pay " +
            sum.divide(BigInteger.valueOf(F)));
            System.out.println();
        }
    }
}
```

6.42 Bignum em C++

print imprime o número. *fix* remove os zeros à frente. *str2bignum* converte de string para para bignum. *int2bignum* gera um bignum a partir de um inteiro menor que a base. *bignum2int* só funciona se não der overflow. A divisão por inteiros só funciona para inteiros menores que a base. Soma, subtração, shift left e shift right em $O(n)$, multiplicação em $O(n^2)$. Divisão e resto em uma única operação, é lenta para bases muito grandes. A subtração só funciona para $a \geq b$.

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

typedef vector<int> bignum;
const int base = 1000*1000*1000;

void print(bignum & a) {
    printf("%d", a.empty() ? 0 : a.back());
    for (int i=(int)a.size()-2; i>=0; --i) {
        printf("%09d", a[i]);
    }
}
void fix(bignum & a) {
    while (a.size() > 1u && a.back() == 0)
        a.pop_back();
}
bool comp(bignum a, bignum b) {
    fix(a); fix(b);
    if (a.size() != b.size()) return a.size() < b.size
        ();
    for(int i=(int)a.size()-1; i>=0; i--) {
        if (a[i] != b[i]) return a[i] < b[i];
    }
    return false;
}
void str2bignum(char* s, bignum & a) {
    a.clear();
    for (int i=(int)strlen(s); i>0; i-=9) {
        s[i] = 0;
        a.push_back(atoi(i>=9 ? s+i-9 : s));
    }
    fix(a);
}
void int2bignum(int n, bignum & a) {
    a.clear();
    if (n == 0) a.push_back(0);
    for(; n > 0; n /= base)
        a.push_back(n%base);
}
int bignum2int(bignum & a) {
    int ans = 0, p=1;
    for(int i=0; i<(int)a.size(); i++) {
        ans += a[i]*p; p *= base;
    }
    return ans;
}
void sum(bignum & a, bignum & b, bignum & c) {
    int carry = 0, n = max(a.size(), b.size());
    c.resize(n);
    for (int i=0, ai, bi; i<n; i++) {
        ai = i < (int)a.size() ? a[i] : 0;
        bi = i < (int)b.size() ? b[i] : 0;
        c[i] = carry + ai + bi;
        carry = c[i] / base;
        c[i] %= base;
    }
    if (carry > 0) c.push_back(carry);
    fix(c);
}
void subtract(bignum & a, bignum & b, bignum & c) {
    int carry = 0, n = max(a.size(), b.size());
    c.resize(n);
    for (int i=0, ai, bi; i<n; i++) {
        ai = i < (int)a.size() ? a[i] : 0;
        bi = i < (int)b.size() ? b[i] : 0;
        c[i] = carry + ai - bi;
        carry = c[i] < 0 ? 1 : 0;
        if (c[i] < 0) c[i] += base;
    }
    fix(c);
}
void shiftL(bignum & a, int b, bignum & c) {
    c.resize((int)a.size() + b);
    for(int i=(int)c.size()-1; i>=0; i--) {
        if(i>=b) c[i] = a[i-b];
        else c[i] = 0;
    }
    fix(c);
}
void shiftR(bignum & a, int b, bignum & c) {
    if (((int)a.size()) <= b) {
        c.clear(); c.push_back(0);
        return;
    }
    c.resize((int)a.size() - b);
    for(int i=0; i<(int)c.size(); i++)
        c[i] = a[i+b];
    fix(c);
}
void multiply(int a, bignum & b, bignum & c) {
    int carry = 0, bi;
    c.resize(b.size());
    for (int i=0; i<(int)b.size() || carry; i++) {
        if (i == (int)b.size()) c.push_back(0);
        bi = i < (int)b.size() ? b[i] : 0;
        long long cur = carry + a * 1ll * bi;
        c[i] = int(cur % base);
        carry = int(cur / base);
    }
    fix(c);
}
void multiply(bignum a, bignum b, bignum & c) {
    int n = a.size()+b.size();
    long long carry = 0, acum;
    c.resize(n);
    for (int k=0; k<n || carry; k++) {
        if (k == n) c.push_back(0);
        acum = carry; carry = 0;
        for (int i=0, j=k; i <= k && i<(int)b.size(); i
            ++, j--) {
            if (j >= (int)b.size()) continue;
            acum += a[i] * 1ll * b[j];
            carry += acum / base;
            acum %= base;
        }
        c[k] = acum;
    }
    fix(c);
}
void divide(bignum & a, int b, bignum & c) {
    int carry = 0;
    c.resize(a.size());
    for (int i=(int)a.size()-1; i>=0; --i) {
        long long cur = a[i] + carry * 1ll * base;
        c[i] = int (cur / b);
        carry = int (cur % b);
    }
    fix(a);
}
```

6.43 A ruína do Apostador

Seja o seguinde jogo: dois jogadores 1 e 2 com n_1 e n_2 moedas, respectivamente jogam um jogo tal que, a cada rodada, existem probabilidade p do jogador 2 transferir uma moeda para 1 (1 ganha a rodada) e $q = 1 - p$ do contrário (2 ganha a rodada). O jogo acaba quando um dos jogadores fica sem moeda (perdedor). A probabilidade de cada um vencer é:

$$P_1 = \frac{n_1}{n_1 + n_2}, P_2 = \frac{n_2}{n_1 + n_2}, p = q \qquad P_1 = \frac{1 - (\frac{q}{p})^{n_1}}{1 - (\frac{q}{p})^{n_1+n_2}}, P_2 = \frac{1 - (\frac{p}{q})^{n_2}}{1 - (\frac{p}{q})^{n_1+n_2}}, p \neq q \tag{6.9}$$

6.44 Teoremas e Fórmulas

- **Desarranjo**: o número $der(n)$ de permutações de n elementos em que nenhum dos elementos fica na posição original é dado por: $der(n) = (n-1)(der(n-1) + der(n-2))$, onde $der(0) = 1$ e $der(1) = 0$.
- **Fórmula de Euler para poliedros convexos**: $V - E + F = 2$, onde F é o número de faces.
- **Círculo de Moser**: o número de peças em que um círculo pode ser divido por cordas ligadas a n pontos tais que não se tem 3 cordas internamente concorrentes é dada por: $g(n) = \binom{n}{4} + \binom{n}{2} + 1 = \frac{1}{24}(n^4 - 6n^3 + 23n^2 - 18n + 24)$.
- **Teorema de Pick**: se I é o número de pontos inteiros dentro de um polígono, A a área do polígono e b o número de pontos inteiros na borda, então $A = i + b/2 - 1$.
- **Teorema de Zeckendorf**: qualquer inteiro positivo pode ser representado pela soma de números de Fibonacci que não inclua dois números consecutivos. Para achar essa soma, usar o algoritmo guloso, sempre procurando o maior número de fibonacci menor que o número.
- **Teorema de Wilson**: um número n é primo se e somente se $(n-1)! \equiv -1 \mod n$
- **Teorema de Euler**: se a e b forem coprimos entre si, então $a^{\phi(b)} \equiv 1 \mod b$ ou $a^n \equiv a^{n\%\phi(b)} \mod b$.
- **Teorema Pequeno de Fermat**: se p é um número primo, então, para qualquer inteiro a, $a^p - a$ é múltiplo de p. Ou seja, $a^p \equiv a \mod p$ ou $a^{p-1} \equiv 1 \mod p$.
- **Último Teorema de Fermat**: para qualquer inteiro $n > 2$, a equação $x^n + y^n = z^n$ não possui soluções inteiras.
- **Teorema de Lagrange**: qualquer inteiro positivo pode ser escrito como a soma de 4 quadrados perfeitos.
- **Conjectura de Goldbach**: qualquer par $n > 2$ pode ser escrito como a soma de dois primos (testada até 4×10^{18}).
- **Conjectura dos primos gêmeos**: existem infinitos primos p tal que $p + 2$ também é primo.
- **Conjectura de Legendre**: para todo n inteiro positivo, existe um primo entre n^2 e $(n+1)^2$.

Capítulo 7

Strings

7.1 Biblioteca <ctype.h>

Funções de classificação de caracteres:
isalnum(int c) // Verifica se o caractere é alfanumérico
isalpha(int c) // Verifica se o caractere é alfabético
isblank(int c) // Verifica se o caractere é um espaço ou tab
iscntrl(int c) // Verifica se é um caractere de controle
isdigit(int c) // Verifica se é um dígito decimal (0-9)
isgraph(int c) // Verifica se tem representação gráfica
islower(int c) // Verifica se é uma letra minúscula
isprint(int c) // Verifica se o caractere é imprimível
ispunct(int c) // Verifica se é um sinal de pontuação
isspace(int c) // Verifica se o caractere é um espaço
isupper(int c) // Verifica se é uma letra maiúscula
isxdigit(int c) // Verifica se é hexadecimal (0-9, a-f, A-F)

Funções de conversão de caracteres:
tolower(int c) // Converte uma letra para minúscula
toupper(int c) // Converte uma letra para maiúscula

7.2 Verificar se uma letra é vogal ou consoante

```
#include <ctype.h> // tolower(), isalpha()

bool isVowel(char ch) {
    char vowel[6] = "aeiou";
    for (int j = 0; vowel[j]; j++)
        if (vowel[j] == tolower(ch))
            return true;
    return false;
}

bool isConsonant(char ch) {
    return isalpha(ch) && !isVowel(ch);
}
```

7.3 Converter string C++ em char[]

Mudar *LEN* conforme o tamanho máximo da string.

```
#include<string.h> // strcpy()
#define LEN 112345

// exemplo de uso dentro do programa principal
int main(){
    // [...]

    char cs[LEN];
    string str = "exemplo";
    // Copia a string str para o array de char cs
    strcpy(cs, str.c_str());

    // [...]
    return 0;
}
```

7.4 Dividir string em tokens

Converter a string em char[ ] antes, se necessário.

```
vector<string> tokenize(const char *str){
    vector<string> tokens;
    map<string, int> freq;
    // mudar delimitadores se precisar
    for (p = strtok(str, "␣."); p; p = strtok(NULL, "␣.
        ")) {
        tokens.push_back(p);
        // casting from C string to C++ string is
            automatic
        freq[p]++;
    }
    // sort(tokens.begin(), tokens.end());
    return tokens;
}
```

7.5 Distância de Levenshtein (Edit Distance)

```
int editDistance(string s1, string s2, 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(s1[i-1] == s2[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];
}
```

7.6 Longest Common Subsequence (LCS)

```
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];
}
```

7.7 Knuth–Morris–Pratt

Pré-processa a substring P, permitindo buscas mais eficientes na string T.
Complexidade: $O(m+n)$, onde m é o tamanho de P e n é o tamanho de T.

```
#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];
        }
    }
}
```

7.8 Patricia Tree (ou Patricia Trie)

Implementação PB-DS, extremamente curta e confusa:

- **Criar**: `patricia_tree pat;`
- **Inserir**: `pat.insert("sei la");`
- **Remover**: `pat.erase("sei la");`
- **Verificar existência**: `pat.find("sei la") != pat.end();`
- **Pegar palavras que começam com um prefixo**: `auto match = pat.prefix_range("sei");`
- **Percorrer *match***: `for(auto it = match.first; it != match.second; ++it);`
- **Pegar menor elemento lexicográfico *maior ou igual* ao prefixo**: `*pat.lower_bound("sei");`
- **Pegar menor elemento lexicográfico *maior* ao prefixo**: `*pat.upper_bound("sei");`

Todas as operações em $O(|S|)$. Não aceita elementos repetidos. Cortesia do BRUTE-UDESC.

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/trie_policy.hpp>

using namespace __gnu_pbds;
typedef trie< string, null_type, trie_string_access_traits<>, pat_trie_tag, trie_prefix_search_node_update>
    patricia_tree;
```

7.9 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

```
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;
    }
};
```

7.10 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.

```
#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;
}
```

7.11 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 $O(n)$. $z[0] = 0$.

```
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;
    }
}
```

7.12 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 $str[i+j] = str[i-j]$, $0 \le j \le L[i]$. Pra calcular os palíndromos pares, basta adicionar '|' entre todos os caracateres e calcular o maior valor de L da string.

```
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]; }
    }
}
```

7.13 Hashing

Hashing para testar igualdade de duas strings. A função *range(i, j)* retorna o hash da substring nesse range. Pode ser necessário usar pares de hash para evitar colisões. Cortesia do BRUTE-UDESC.

- Complexidade de tempo (Construção): $O(N)$
- Complexidade de tempo (Consulta de range): $O(1)$

```
struct hashing{
    const long long LIM = 1000006;
    long long p, m;
    vector<long long> pw, hsh;
    hashing(long long _p, long long _m) :
    p(_p), m(_m) {
        pw.resize(LIM);
        hsh.resize(LIM);
        pw[0] = 1;
        for(int i = 1; i < LIM; i++)
            pw[i] = (pw[i-1] * p) % m;
    }
    void set_string(string& s) {
        hsh[0] = s[0];
        for (int i = 1; i < s.size(); i++)
            hsh[i] = (hsh[i-1] * p + s[i]) % m;
    }
    long long range(int esq, int dir) {
        long long ans = hsh[dir];
        if(esq > 0) ans = (ans - (hsh[esq-1] *
            pw[dir-esq+1] % m) + m) % m;
        return ans;
    }
};
```

7.14 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;
    }
};
```

7.15 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 $last, link(last), link(link(last)), \cdots$. Cortesia do Macacário do ITA.

```
#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;
    }
};
```

7.16 Suffix Array e Longest Common Prefix

computeSA computa a Suffix Array em $O(n \log n)$. *computeLCP* computa o Longest 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]];
    }
};
```

7.17 Trie

Estrutura que guarda informações indexadas por palavra.
Útil encontrar todos os prefixos inseridos anteriormente de uma palavra específica.

- Complexidade de tempo (Update): $O(|S|)$
- Complexidade de tempo (Consulta de palavra): $O(|S|)$

Cortesia do BRUTE-UDESC.

```
struct trie{
    map<char, int> trie[100005];
    int value[100005];
    int n_nodes = 0;
    void insert(string &s, int v){
        int id = 0;
        for (char c: s){
          if(!trie[id].count(c)) trie[id][c] = ++n_nodes;
          id = trie[id][c];
        }
        value[id] = v;
    }
    int get_value(string &s){
        int id = 0;
        for (char c: s){
          if(!trie[id].count(c)) return -1;
          id = trie[id][c];
        }
        return value[id];
    }
};
```

7.18 Palindromic Tree

Palindromic Tree em $O(n)$. $root1$ é a raíz de tamanho -1 (palíndromos ímpares). $root2$ é a raíz de tamanho 0 (palíndromos pares). $len[u]$ é o tamanho do palíndromo no nó u. $nxt[u][c]$ aponta para um nó que representa um palíndromo de tamanho $len[u]+2$ e começa e acaba com c. $link[u]$ aponta pro maior sufixo menor que o palíndromo de u que também é palíndromo. $extend(c)$ adiciona c ao final da string em $O(1)$ amortizado e retorna quantos palíndromos novos não distintos foram adicionados. O número de palíndromos distintos é o número de nós não triviais. Ao resetar cnt, é necessário limpar nxt. Cortesia do Macacário do ITA.

```
#define MAXS 1000009
#define ALFA 26

int nxt[MAXS][ALFA], len[MAXS];
int link[MAXS], num[MAXS];
int cnt = 0; //limpar nxt se me resetar

class PalindromicTree {
private:
    string s;
    int suff;
    char reduce(char c) { return c-'a'; }
    int getlink(int u, int pos) {
        for(; true; u = link[u]) {
            int st = pos - 1 - len[u];
            if (st >= 0 && s[st] == s[pos])
                return u;
        }
    }
public:
    int root1, root2;
    PalindromicTree() {
        root1 = ++cnt; root2 = ++cnt;
        suff = root2;
        len[root1] = -1; link[root1] = root1;
        len[root2] = 0; link[root2] = root1;
    }
    int extend(char c) {
        int pos = s.size(); s.push_back(c);
        c = reduce(c);
        int u = getlink(suff, pos);
        if (nxt[u][c]) {
            suff = nxt[u][c];
            return 0;
        }
        int v = suff = ++cnt;
        len[v] = len[u] + 2;
        nxt[u][c] = v;
        if (len[v] == 1) {
            link[v] = root2;
            return num[v] = 1;
        }
        u = getlink(link[u], pos);
        link[v] = nxt[u][c];
        return num[v] = 1 + num[link[v]];
    }
};
```

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.

```
#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 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);
    }
    vector< pair<point,point> > getTangentSegs(circle
        other) {
        vector<pair<point, point> > ans;
        double d = dist(other.c, c);
        double dr = abs(r - other.r), sr = r + other.r;
        if (dr >= d) return ans;
        double u = acos(dr / d);
        point dc1 = ((other.c - c).normalized())*r;
        point dc2 = ((other.c - c).normalized())*other.
            r;
        ans.push_back(make_pair(c + rotate(dc1, u),
            other.c + rotate(dc2, u)));
        ans.push_back(make_pair(c + rotate(dc1, -u),
            other.c + rotate(dc2, -u)));
        if (sr >= d) return ans;
        double v = acos(sr / d);
        dc2 = ((c - other.c).normalized()) * other.r;
        ans.push_back({c + rotate(dc1, v), other.c +
            rotate(dc2, v)});
        ans.push_back({c + rotate(dc1, -v), other.c +
            rotate(dc2, -v)});
        return ans;
    }
    pair<point, point> getIntersectionPoints(circle
        other){
        assert(intersects(other));
        double d = dist(c, other.c);
        double u = acos((other.r*other.r + d*d - r*r) /
            (2*other.r*d));
        point dc = ((other.c - c).normalized()) * r;
        return make_pair(c + rotate(dc, u), c + rotate(
            dc, -u));
    }
};

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);
}
```

8.3 Grande Círculo

Dado o raio da Terra e as coordenadas em latitude e longitude de dois pontos p e q, retorna o ângulo pOq. Retorna a distância mínima de viagem pela superfície.

```
double gcTheta(double pLat, double pLong, double qLat,
    double qLong) {
    pLat *= PI / 180.0; pLong *= PI / 180.0; // convert
        degree to radian
    qLat *= PI / 180.0; qLong *= PI / 180.0;
    return acos(cos(pLat)*cos(pLong)*cos(qLat)*cos(
        qLong) +
        cos(pLat)*sin(pLong)*cos(qLat)*sin(qLong) +
            sin(pLat)*sin(qLat));
}

double gcDistance(double pLat, double pLong, double
    qLat, double qLong, double radius) {
    return radius*gcTheta(pLat, pLong, qLat, qLong);
}
```

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

```
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;
    int n = P.size();
    for (int i = 0; i < n; i++) result += dist(P[i], P
        [(i+1)%n]);
    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 (P[i] == p || between(P[i], p, P[(i+1)%n]))
            return true;
        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.

```
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 insideConvex(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 Soma de Minkowski

Determina o polígono que contorna a soma de Minkowski de duas regiões delimitadas por polígonos regulares. A soma de Minkowski de dois conjuntos de pontos A e B é o conjunto $C = \{c \in R^2 | c = a + b, a \in A, b \in B\}$. Algumas aplicações interessantes:

- Para verificar se A e B possuem intersecção, basta verificar se $(0,0) \in minkowski(A, -B)$.
- $(1/n) * minkowski(A_1, A_2, ..., A_n)$ representa todos os baricentros possíveis de pontos em $A_1, A_2, ..., A_n$.

```
polygon minkowski(polygon & A, polygon & B) {
    polygon P; point v1, v2;
    int n1 = A.size(), n2 = B.size();
    P.push_back(A[0]+B[0]);
    for(int i = 0, j = 0; i < n1 || j < n2;) {
        v1 = A[(i+1)%n1]-A[i%n1];
        v2 = B[(j+1)%n2]-B[j%n2];
        if (j == n2 || cross(v1, v2) > EPS) {
            P.push_back(P.back() + v1); i++;
        }
        else if (i == n1 || cross(v1, v2) < -EPS) {
            P.push_back(P.back() + v2); j++;
        }
        else {
            P.push_back(P.back() + (v1+v2));
            i++; j++;
        }
    }
    P.pop_back();
    return P;
}
```

8.9 Comparador polar

Função para inteiros: *polarCmp* para $x, y \approx 10^9$.

```
bool polarCmp(point a, point b) {
    if (b.y*a.y > 0) return cross(a, b) > 0;
    else if (b.y == 0 && b.x > 0) return false;
    else if (a.y == 0 && a.x > 0) return true;
    else return b.y < a.y;
}
```

8.10 Triangulação de Delaunay

Execução $O(n \log^2 n)$ com alto overhead. $\approx 2.5s$ para $n = 10^5$. adj é a lista de adjacência dos nós, tri são os triângulos.

```
#include <set>
#include <algorithm>
#define MAXN 200309

struct truple {
    int a, b, c;
    truple(int _a, int _b, int _c) :
        a(_a), b(_b), c(_c) {
        if (a > b) swap(a, b);
        if (a > c) swap(a, c);
        if (b > c) swap(b, c);
    }
};
bool operator < (truple x, truple y) {
    if (x.a != y.a) return x.a < y.a;
    if (x.b != y.b) return x.b < y.b;
    if (x.c != y.c) return x.c < y.c;
    return false;
}
namespace Delaunay {
vector< set<int> > adj;
vector<point> P;
set<truple> tri;
void add_edge(int u, int v, int w = -1) {
    adj[u].insert(v); adj[v].insert(u);
    if (w >= 0) tri.insert(truple(u, v, w));
}
void del_edge(int u, int v) {
    adj[u].erase(v); adj[v].erase(u);
}
void brute(int l, int r) {
    if (r-l == 2 && collinear(P[l],P[l+1],P[r])) {
        add_edge(l, l+1); add_edge(l+1, r);
        return;
    }
    for(int u = l; u <= r; u++)
        for(int v = u+1; v <= r; v++)
            add_edge(u, v);
    if (r-l == 2) tri.insert(truple(l,l+1,r));
}
double theta[MAXN];
bool comp(int u, int v) {
    return theta[u] < theta[v];
}
vector< vector<int> > g;
bool right; point base;
void computeG(int u, int r = -1) {
    if (!g[u].empty()) return;
    if (r >= 0) adj[u].erase(r);
    g[u] = vector<int>(adj[u].begin(), adj[u].end());
    if (r >= 0) adj[u].insert(r);
    for(int i = 0; i < int(g[u].size()); i++) {
        double co = inner(base, P[g[u][i]]-P[u]);
        double si = cross(base, P[g[u][i]]-P[u]);
        theta[g[u][i]] = atan2(si, co);
    }
    sort(g[u].begin(), g[u].end(), comp);
    if (right) reverse(g[u].begin(), g[u].end());
}
int getNext(int u, int a, int b) {
    right = (u == b);
    base = (right ? P[b]-P[a] : P[a]-P[b]);
    computeG(u, a + b - u);
    int ans = -1, w;
    while(!g[u].empty()) {
        int j = g[u].size() - 1;
        ans = g[u][j];
        if (right && cross(base, P[ans]-P[b]) > -EPS)
            return -1;
        if (!right && cross(base, P[ans]-P[a]) < EPS)
            return -1;
        circle c = circumcircle(P[a],P[b],P[ans]);
        if (g[u].size() > 1u && dist(c.c, P[w = g[u][j
            -1]]) < c.r - EPS) {
            del_edge(u, ans); g[u].pop_back();
            tri.erase(truple(u, w, ans));
        }
        else break;
    }
    return ans;
}
int moveEdge(int & u, int a, int b) {
    right = (b == u); int v = a+b-u;
    for(set<int>::iterator it = adj[u].begin(); it !=
        adj[u].end(); it++) {
        int nu = *it;
        double cr = cross(P[u]-P[v], P[nu]-P[v]);
        if (right && cr > EPS) return u = nu;
        if (!right && cr < -EPS) return u = nu;
        if (fabs(cr) < EPS && dist(P[nu], P[v]) < dist(
            P[u], P[v])) return u = nu;
    }
    return -1;
}
void delaunay(int l, int r) {
    if (r - l < 3) { brute(l, r); return; }
    int mid = (r + l) / 2;
    delaunay(l, mid); delaunay(mid + 1, r);
    int u = l, v = r, nu, nv;
    double du, dv;
    do {
        nu = moveEdge(u, u, v);
        nv = moveEdge(v, u, v);
    } while (nu != -1 || nv != -1);
    g[u].clear(); g[v].clear();
    add_edge(u, v);
    while(true) {
        nu = getNext(u, u, v); nv = getNext(v, u, v);
        if (nu == -1 && nv == -1) break;
        if (nu != -1 && nv != -1) {
            point nor = point(P[u].y-P[v].y, P[v].x-P[u
                ].x);
            circle cu = circumcircle(P[u],P[v],P[nu]);
            circle cv = circumcircle(P[u],P[v],P[nv]);
            du = inner(cu.c-P[u], nor);
            dv = inner(cv.c-P[v], nor);
        }
        else du = 1, dv = 0;
        if (nu == -1 || du < dv) {
            add_edge(u, nv, v); g[v = nv].clear();
        }
        else if (nv == -1 || du >= dv) {
            add_edge(nu, v, u); g[u = nu].clear();
        }
    }
}
vector< set<int> > compute(vector<point> & Q) {
    sort(Q.begin(), Q.end());
    int n = (P = Q).size();
    adj.clear(); adj.assign(n, set<int>());
    g.clear(); g.assign(n, vector<int>());
    tri.clear();
    delaunay(0, n-1);
    return adj;
}
set<truple> getTriangles() { return tri; }
};
```

8.11 Intersecção de polígonos

Intersecção de dois polígons convexos em $O(nm\log(n+m))$.

```
polygon intersect(polygon & A, polygon & B) {
    polygon P;
    int n = A.size(), m = B.size();
    for (int i = 0; i < n; i++) {
        if (inPolygon(B, A[i])) P.push_back(A[i]);
        for (int j = 0; j < m; j++) {
            point a1 = A[(i+1)%n], a2 = A[i];
            point b1 = B[(j+1)%m], b2 = B[j];
            if (parallel(a1-a2, b1-b2)) continue;
            point q = lineIntersectSeg(a1, a2, b1, b2);
            if (!between(a1, q, a2)) continue;
            if (!between(b1, q, b2)) continue;
            P.push_back(q);
        }
    }
    for (int i = 0; i < m; i++){
        if (inPolygon(A, B[i])) P.push_back(B[i]);
    }
    set<point> inuse; //Remove duplicates
    int sz = 0;
    for (int i = 0; i < (int)P.size(); ++i) {
        if (inuse.count(P[i])) continue;
        inuse.insert(P[i]);
        P[sz++] = P[i];
    }
    P.resize(sz);
    if (!P.empty()) {
        pivot = P[0];
        sort(P.begin(), P.end(), angleCmp);
    }
    return P;
}
```

8.12 Minimum Enclosing Circle

Computa o círculo de raio mínimo que contém um conjunto de pontos. Complexidade: *expected* $O(n)$.

```
circle minimumCircle(vector<point> p) {
    random_shuffle(p.begin(), p.end());
    circle C = circle(p[0], 0.0);
    for(int i = 0; i < (int)p.size(); i++) {
        if (C.contains(p[i])) continue;
        C = circle(p[i], 0.0);
        for(int j = 0; j < i; j++) {
            if (C.contains(p[j])) continue;
            C = circle((p[j] + p[i])*0.5, 0.5*dist(p[j
                ], p[i]));
            for(int k = 0; k < j; k++) {
                if (C.contains(p[k])) continue;
                C = circumcircle(p[j], p[i], p[k]);
            }
        }
    }
    return C;
}
```

8.13 Intersecção de semi-planos

Computa a intersecção de semi-planos em $O(n\log n)$.

```
typedef pair<point, point> halfplane;

point dir(halfplane h) {
    return h.second - h.first;
}
bool belongs(halfplane h, point a) {
    return cross(dir(h), a - h.first) > EPS;
}
bool hpcomp(halfplane ha, halfplane hb) {
    point a = dir(ha), b = dir(hb);
    if (parallel(a, b) && inner(a, b) > EPS)
        return cross(b, ha.first - hb.first) < -EPS;
    if (b.y*a.y > EPS) return cross(a, b) > EPS;
    else if (fabs(b.y) < EPS && b.x > EPS) return false
        ;
    else if (fabs(a.y) < EPS && a.x > EPS) return true;
    else return b.y < a.y;
}
polygon intersect(vector<halfplane> H, double W = INF)
    {
    H.push_back(halfplane(point(-W,-W),point(W,-W)));
    H.push_back(halfplane(point(W,-W),point(W,W)));
    H.push_back(halfplane(point(W,W),point(-W,W)));
    H.push_back(halfplane(point(-W,W),point(-W,-W)));
    sort(H.begin(), H.end(), hpcomp);
    int i = 0;
    while(parallel(dir(H[0]), dir(H[i]))) i++;
    deque<point> P;
    deque<halfplane> S;
    S.push_back(H[i-1]);
    for(; i < (int)H.size(); i++) {
        while(!P.empty() && !belongs(H[i], P.back()))
            P.pop_back(), S.pop_back();
        point df = dir(S.front()), di = dir(H[i]);
        if (P.empty() && cross(df, di) < EPS)
            return polygon();
        P.push_back(lineIntersectSeg(H[i].first, H[i].
            second,
            S.back().first, S.back().second));
        S.push_back(H[i]);
    }
    while(!belongs(S.back(), P.front()) ||
        !belongs(S.front(), P.back())) {
        while(!belongs(S.back(), P.front()))
            P.pop_front(), S.pop_front();
        while(!belongs(S.front(), P.back()))
            P.pop_back(), S.pop_back();
    }
    P.push_back(lineIntersectSeg(S.front().first,
        S.front().second, S.back().first, S.back().
            second));
    return polygon(P.begin(), P.end());
}
```

8.14 Ponto 3D

```
#include <cstdio>
#include <cmath>
#define EPS 1e-9

struct point {
    double x, y, z;
    point() { x = y = z = 0.0; }
    point(double _x, double _y, double _z) : x(_x), y(
        _y), z(_z) {}
    double norm() { return sqrt(x*x + y*y + z*z); }
    point normalized() {
        return point(x,y,z)*(1.0/norm());
    }
    bool operator < (point other) const {
        if (fabs(x - other.x) > EPS)
            return x < other.x;
        else if (fabs(y - other.y) > EPS)
            return y < other.y;
        else return z < other.z;
    }
    bool operator == (point other) const {
        return (fabs(x - other.x) < EPS && fabs(y -
            other.y) < EPS && fabs(z - other.z) < EPS);
    }
    point operator +(point other) const{
        return point(x + other.x, y + other.y, z +
            other.z);
    }
    point operator -(point other) const{
        return point(x - other.x, y - other.y, z -
            other.z);
    }
    point operator *(double k) const{
        return point(x*k, y*k, z*k);
    }
};
double dist(point p1, point p2) {
    return (p1-p2).norm();
}
double inner(point p1, point p2) {
    return p1.x*p2.x + p1.y*p2.y + p1.z*p2.z;
}
point cross(point p1, point p2) {
    point ans;
    ans.x = p1.y*p2.z - p1.z*p2.y;
    ans.y = p1.z*p2.x - p1.x*p2.z;
    ans.z = p1.x*p2.y - p1.y*p2.x;
    return ans;
}
bool collinear(point p, point q, point r) {
    return cross(p-q, r-p).norm() < EPS;
}
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));
}
```

8.15 Triângulo 3D

```
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();
    }
    double rCircumCircle() {
        return dist(a,b)*dist(b,c)*dist(c,a)/(4.0*area
            ());
    }
    point normalVector() {
        return cross(y-x, z-x).normalized();
    }
    int isInside(point p) {
        point n = normalVector();
        double u = proj(cross(b-a,p-a), n).normalized()
            *proj(cross(b-a,c-a), n).normalized();
        double v = proj(cross(c-b,p-b), n).normalized()
            *proj(cross(c-b,a-b), n).normalized();
        double w = proj(cross(a-c,p-c), n).normalized()
            *proj(cross(a-c,b-c), n).normalized();
        if (u > 0.0 && v > 0.0 && w > 0.0) return 0;
        else if (u < 0.0 || v < 0.0 || w < 0.0) return
            2;
        else return 1;
    } //0 = inside/ 1 = border/ 2 = outside
    int isProjInside(point p) {
        return isInside(p + proj(a-p, normalVector()));
    } //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 isProjInsideTriangle(point a, point b, point c,
    point p) {
    return triangle(a,b,c).isProjInside(p);
} //0 = inside/ 1 = border/ 2 = outside
```

8.16 Linha 3D

Desta vez a linha é implementada com um ponto de referência e um vetor base. $distVector$ é um vetor que é perpendicular a ambas as linhas e tem como comprimento a distância entre elas. $distVector$ é a "ponte"de a a b de menor caminho entre as duas linhas. $distVectorBasePoint$ é o ponto da linha a de onde sai o $distVector$. $distVectorEndPoint$ é o ponto na linha b onde chega a ponte.

```
struct line{
    point r;
    point v;
    line(point _r, point _v) {
        v = _v; r = _r;
    }
    bool operator == (line other) const{
        return fabs(cross(r-other.r, v).norm()) < EPS
            && fabs(cross(r-other.r, other.v).norm()) <
             EPS;
    }
};

point distVector(line l, point p) {
    point dr = p - l.r;
    return dr - proj(dr, l.v);
}

point distVectorBasePoint(line l, point p) {
    return proj(p - l.r, l.v) + l.r;
}

point distVectorEndPoint(line l, point p) {
    return p;
}

point distVector(line a, line b) {
    point dr = b.r - a.r;
    point n = cross(a.v, b.v);
    if (n.norm() < EPS) {
        return dr - proj(dr, a.v);
    }
    else return proj(dr, n);
}

double dist(line a, line b) {
    return distVector(a, b).norm();
}

point distVectorBasePoint(line a, line b) {
    if (cross(a.v, b.v).norm() < EPS) return a.r;
    point d = distVector(a, b);
    double lambda;
    if (fabs(b.v.x*a.v.y - a.v.x*b.v.y) > EPS)
        lambda = (b.v.x*(b.r.y-a.r.y-d.y) - b.v.y*(b.r.
            x-a.r.x-d.x))/(b.v.x*a.v.y - a.v.x*b.v.y);
    else if (fabs(b.v.x*a.v.z - a.v.x*b.v.z) > EPS)
        lambda = (b.v.x*(b.r.z-a.r.z-d.z) - b.v.z*(b.r.
            x-a.r.x-d.x))/(b.v.x*a.v.z - a.v.x*b.v.z);
    else if (fabs(b.v.z*a.v.y - a.v.z*b.v.y) > EPS)
        lambda = (b.v.z*(b.r.y-a.r.y-d.y) - b.v.y*(b.r.
            z-a.r.z-d.z))/(b.v.z*a.v.y - a.v.z*b.v.y);
    return a.r + (a.v*lambda);
}

point distVectorEndPoint(line a, line b) {
    return distVectorBasePoint(a, b) + distVector(a, b)
        ;
}
```

8.17 Geometria Analítica

Pontos de intersecção de dois círculos:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} = dist(c_1, c_2) \tag{8.1}$$

$$l = \frac{r_1^2 - r_2^2 + d^2}{2d} \tag{8.2}$$

$$h = \sqrt{r_1^2 - l^2} \tag{8.3}$$

$$x = \frac{l}{d}(x_2 - x_1) \pm \frac{h}{d}(y_2 - y_1) + x_1 \tag{8.4}$$

$$y = \frac{l}{d}(y_2 - y_1) \mp \frac{h}{d}(x_2 - x_1) + y_1 \tag{8.5}$$

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

bool circleCircle(circle c1, circle c2, pair<point,
    point> & out) {
    double d = dist(c2.c, c1.c);
    double co = (d*d + c1.r*c1.r - c2.r*c2.r)/(2*d*c1.r
        );
    if (fabs(co) > 1.0) return false;
    double alpha = acos(co);
    point rad = (c2.c-c1.c)*(1.0/d*c1.r);
    out = {c1.c + rotate(rad, -alpha), c1.c + rotate(
        rad, alpha)};
    return true;
}
```

A equação da reta que passa pelos pontos (x_1, y_1) e (x_2, y_2) é dada por:

$$(y_2 - y_1)x + (x_1 - x_2)y + (y_1 x_2 - x_1 y_2) = 0 \tag{8.6}$$

Matrizes de rotação. Sentido positivo a partir da regra da mão direita e supondo $\vec{z} = \vec{x} \times \vec{y}$.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sen\theta \\ 0 & sen\theta & cos\theta \end{bmatrix} \tag{8.7}$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sen\theta \\ 0 & 1 & 0 \\ -sen\theta & 0 & cos\theta \end{bmatrix} \tag{8.8}$$

$$R_z(\theta) = \begin{bmatrix} cos\theta & -sen\theta & 0 \\ sen\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8.9}$$

Baricentro de um polígono de n lados. A = área com sinal. CUIDADO: Transladar polígono para a origem para não perder precisão senão toma WA! Depois transladar de volta.

$$A = \frac{1}{2}\sum_{i=0}^{n-1} p_i \times p_{i+1} \quad (8.10)$$

$$C_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(p_i \times p_{i+1}) \quad (8.11)$$

$$C_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(p_i \times p_{i+1}) \quad (8.12)$$

8.18 Coordenadas polares, cilíndricas e esféricas

Coordenadas polares:

$$x = rcos\phi \qquad y = rsen\phi \qquad dS = rdrd\phi \quad (8.13)$$

Coordenadas cilíndricas:

$$x = rcos\phi \qquad y = rsen\phi \qquad z = z \quad (8.14)$$

$$d\vec{\gamma} = dr\hat{r} + rd\phi\hat{\phi} + dz\hat{z} \qquad dV = rdrd\phi dz \quad (8.15)$$

$$d\vec{S}_r = rd\phi dz\hat{r} \qquad d\vec{S}_\phi = drdz\hat{\phi} \qquad d\vec{S}_z = rdrd\phi\hat{z} \quad (8.16)$$

$$\vec{\nabla} f = \frac{\partial f}{\partial r}\hat{r} + \frac{1}{r}\frac{\partial f}{\partial \phi}\hat{\theta} + \frac{\partial f}{\partial z}\hat{z} \quad (8.17)$$

$$\vec{\nabla}.\vec{F} = \frac{1}{r}\frac{\partial}{\partial r}(rF_r) + \frac{1}{r}\frac{\partial F_\phi}{\partial \phi} + \frac{\partial F_z}{\partial z} \quad (8.18)$$

$$\vec{\nabla}\times\vec{F} = (\frac{1}{r}\frac{\partial F_z}{\partial \phi} - \frac{\partial F_\phi}{\partial z})\hat{r} + (\frac{\partial F_r}{\partial z} - \frac{\partial F_z}{\partial r})\hat{\phi} + \frac{1}{r}(\frac{\partial}{\partial r}(rF_\phi) - \frac{\partial F_r}{\partial \phi})\hat{z} \quad (8.19)$$

Coordenadas esféricas:

$$x = rcos\phi sen\theta \qquad y = rsen\phi sen\theta \qquad z = rcos\theta \quad (8.20)$$

$$d\vec{\gamma} = dr\hat{r} + rd\theta\hat{\theta} + rsen\theta d\phi\hat{\phi} \quad (8.21)$$

$$d\vec{S}_r = r^2 sen\theta d\theta d\phi\hat{r} \quad d\vec{S}_\theta = rsen\theta d\phi dr\hat{\theta} \quad d\vec{S}_\phi = rdrd\theta\hat{\phi} \quad (8.22)$$

$$dV = r^2 sen\theta drd\theta d\phi \qquad d\Omega = \frac{dS_r}{r^2} = sen\theta d\theta d\phi \quad (8.23)$$

$$\vec{\nabla} f = \frac{\partial f}{\partial r}\hat{r} + \frac{1}{r}\frac{\partial f}{\partial \theta}\hat{\theta} + \frac{1}{rsen\theta}\frac{\partial f}{\partial \phi}\hat{\phi} \quad (8.24)$$

$$\vec{\nabla}.\vec{F} = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2F_r) + \frac{1}{rsen\theta}\frac{\partial}{\partial \theta}(sen\theta F_\theta) + \frac{1}{rsen\theta}\frac{\partial F_\phi}{\partial \phi} \quad (8.25)$$

$$\vec{\nabla}\times\vec{F} = \frac{1}{rsen\theta}(\frac{\partial}{\partial \theta}(F_\phi sen\theta) - \frac{\partial F_\theta}{\partial \phi})\hat{r} + \frac{1}{r}(\frac{1}{sen\theta}\frac{\partial F_r}{\partial \phi} - \frac{\partial}{\partial r}(rF_\phi))\hat{\theta} + \frac{1}{r}(\frac{\partial}{\partial r}(rF_\theta) - \frac{\partial F_r}{\partial \theta})\hat{\phi} \quad (8.26)$$

8.19 Cálculo Vetorial 2D

Curva regular no plano e comprimento do arco:

$$\vec{\gamma}(t),\ C^1,\ \vec{\gamma}'(t) \neq 0 \qquad L(\gamma) = \int_a^b \|\vec{\gamma}'(t)\| dt \quad (8.27)$$

Reta tangente e normal:

$$T : X = \vec{\gamma}(t_0) + \lambda.\vec{\gamma}'(t_0) \quad (8.28)$$

$$N : \{X \in R^2 :< X - \vec{\gamma}(t_0), \vec{\gamma}'(t_0) >= 0\} \quad (8.29)$$

Curva de orientação invertida:

$$\vec{\gamma}^{-}(t) = \vec{\gamma}(a + b - t) \quad (8.30)$$

Referencial de Frenet:

$$\vec{T}(t) = \frac{\vec{\gamma}'(t)}{\|\vec{\gamma}'(t)\|} \qquad \vec{N}(t) = (-T_y(t), T_x(t)) \quad (8.31)$$

Curvatura, raio e centro de curvatura:

$$K(t) = \frac{\vec{\gamma}''(t).\vec{N}(t)}{\|\vec{\gamma}'(t)\|^2} \quad (8.32)$$

$$R(t) = \frac{1}{|k(t)|} \qquad \vec{C}(t) = \vec{\gamma}(t) + \frac{\vec{N}(t)}{K(t)} \quad (8.33)$$

Equações de Frenet:

$$\vec{T}'(t) = K(s).\vec{N}(t) \qquad \vec{N}'(t) = -K(t).\vec{T}(t) \quad (8.34)$$

Teorema de Gauss no plano:

$$\int_{\partial S} \vec{F}.\vec{N} d\gamma = \int\int_S \vec{\nabla}.\vec{F} dxdy \quad (8.35)$$

Teorema de Green:

$$\int_{\partial\Omega} Pdx + Qdy = \int\int_\Omega (\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}) dxdy \quad (8.36)$$

8.20 Cálculo Vetorial 3D

Referencial de Frenet:

$$\vec{T}(t) = \frac{\vec{\gamma}'(t)}{\|\vec{\gamma}'(t)\|} \quad \vec{B}(t) = \frac{\vec{\gamma}'(t)\times\vec{\gamma}''(t)}{\|\vec{\gamma}'(t)\times\vec{\gamma}''(t)\|} \quad \vec{N}(t) = \vec{B}(t)\times\vec{T}(t) \tag{8.37}$$

Curvatura e torção:

$$\tau(t) = \frac{<\vec{\gamma}'(t)\times\vec{\gamma}''(t), \vec{\gamma}'''(t)>}{\|\vec{\gamma}'(t)\times\vec{\gamma}''(t)\|^2} \quad K(t) = \frac{\|\vec{\gamma}'(t)\times\vec{\gamma}''(t)\|}{\|\vec{\gamma}'(t)\|^3} \tag{8.38}$$

Plano normal a $\vec{\gamma}(t_0)$:

$$< X - \vec{\gamma}(t_0), T(t_0) >= 0 \tag{8.39}$$

Equações de Frenet:

$$\vec{T}'(t) = K(t).\vec{N}(t) \tag{8.40}$$

$$\vec{N}'(t) = -K(t).\vec{T}(t) - \tau(t).\vec{B}(t) \tag{8.41}$$

$$\vec{B}'(t) = -\tau(t).\vec{N}(t) \tag{8.42}$$

Integral de linha de um campo escalar:

$$\int_\gamma f d\gamma = \int_a^b f(\vec{\gamma}(t))\|\vec{\gamma}'(t)\|dt \tag{8.43}$$

Integral de linha de um campo vetorial:

$$\int_\gamma \vec{F}.d\vec{\gamma} = \int_a^b \vec{f}(\vec{\gamma}(t)).\vec{\gamma}'(t)dt \tag{8.44}$$

Operador nabla:

$$\vec{\nabla} = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}) \tag{8.45}$$

Campo gradiente:

$$\vec{F}(x,y,z) = \vec{\nabla} f(x,y,z) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z})(x,y,z) \tag{8.46}$$

Campo conservativo:

$$\vec{F}(x,y,z) = \vec{\nabla} f(x,y,z) \Leftrightarrow \int_\gamma \vec{F}.d\vec{\gamma} = 0 \tag{8.47}$$

Campo rotacional:

$$\vec{\nabla}\times\vec{F} = (\frac{\partial R}{\partial y} - \frac{\partial Q}{\partial z}, \frac{\partial P}{\partial z} - \frac{\partial R}{\partial x}, \frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}) \tag{8.48}$$

$\vec{F}$ é conservativo $\rightarrow \vec{\nabla}\times\vec{F} = 0$ Campo divergente:

$$\vec{\nabla}.\vec{F} = (\frac{\partial P}{\partial x}, \frac{\partial Q}{\partial y}, \frac{\partial R}{\partial z}) \tag{8.49}$$

- $\vec{F}$ é solenoidal quando $\vec{\nabla}.\vec{F} = 0$
- Existe $\vec{G}$ tal que $\vec{F} = \vec{\nabla}\times\vec{G} \Rightarrow \vec{\nabla}.\vec{F} = 0$
- $\vec{\nabla}.\vec{F} \neq 0 \Rightarrow$ não existe $\vec{G}$ tal que $\vec{F} = \vec{\nabla}\times\vec{G}$

Superfície parametrizada:

$$\vec{S}(u,v) = (x(u,v), y(u,v), z(u,v)) \tag{8.50}$$

$$\vec{S_u}(u,v) = (\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u}),\ \vec{S_v}(u,v) = (\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v}) \tag{8.51}$$

Vetor normal à superfície:

$$\vec{N}(u,v) = (\vec{S_u}\times\vec{S_v})(u,v) \neq \vec{0} \tag{8.52}$$

Superfície diferenciável:

$$\vec{N}(u,v) \neq \vec{0} \tag{8.53}$$

Plano tangente à superfície:

$$< (x,y,z) - \vec{S}(u_0,v_0), \vec{N}(u_0,v_0) >= 0 \tag{8.54}$$

Área da superfície:

$$A(S) = \int\int_U \|\vec{N}(u,v)\| du dv \tag{8.55}$$

Integral de superfície de um campo escalar:

$$\int\int_S f dS = \int\int_U f(\vec{S}(u,v))\|\vec{N}(u,v)\| du dv \tag{8.56}$$

Integral de superfície de um campo vetorial:

$$\int\int_S \vec{F}.d\vec{S} = \int\int_U \vec{F}(\vec{S}(u,v)).\vec{N}(u,v) du dv \tag{8.57}$$

Massa e centro de massa:

$$M = \int_\gamma \rho(t) d\vec{\gamma}, \qquad \overline{x}M = \int_\gamma \vec{x}(t)\rho(t)d\vec{\gamma} \tag{8.58}$$

$$M = \int\int_S \rho(u,v)ds \quad \overline{x}M = \int\int_S \vec{x}(u,v)\rho(u,v)ds \tag{8.59}$$

$$M = \int\int\int_V \rho(x,y,z)dv \quad \overline{x}M = \int\int_S \vec{x}(x,y,z)\rho(z,y,z)dv \tag{8.60}$$

Teorema de Pappus para a área, d = distância entre $\overline{x}$ e o eixo de rotação:

$$A(S) = 2\pi d L(C) \tag{8.61}$$

Teorema de Pappus para o volume, d = distância entre $\overline{x}$ e o eixo de rotação:

$$V(\Omega) = 2\pi d A(S) \tag{8.62}$$

Teorema de Gauss no espaço:

$$\int\int_{\partial V} \vec{F}.\vec{N} d\gamma = \int\int\int_V \vec{\nabla}.\vec{F} dx dy dz \tag{8.63}$$

Teorema de Stokes:

$$\int_{\partial S} \vec{F}.d\vec{\gamma} = \int\int_S \vec{\nabla}\times\vec{F}.\vec{N} ds \tag{8.64}$$

8.21 Problemas de precisão, soma estável e fórmula de bháskara

Para IEEE 754:

- Números são representados pelo seu valor representável mais próximo.
- Operações básicas $(+,-,*,/,\sqrt{})$ são feitas com precisão infinita e depois arredondadas.

tipo	bits	expoente	precisão binária	precisão decimal	ϵ
float	32	38	24	6	$2^{-24} \approx 5.96 \times 10^{-8}$
double	64	308	53	15	$2^{-53} \approx 1.11 \times 10^{-16}$
long double	80	19.728	64	18	$2^{-64} \approx 5.42 \times 10^{-20}$

Todas as operações somente com *int* podem ser feitas perfeitamente com *double*. Com *long long*, *long double*.

Sempre que possível, trabalhar com erro absoluto, e não relativo. O erro relativo pode disparar no caso de cancelamento catastrófico - subtração de dois números com mesma ordem de grandeza. Seja ϵ o erro relativo natural do tipo e M o valor máximo de uma variável. O erro absoluto é dado por $M\epsilon$. Para as operações entre números:

- Soma e subtração normal: O erro absoluto após n somas é $nM\epsilon$. Cada soma ou subtração de não inteiro adiciona erro relativo ϵ. O erro relativo da soma de um número obtido com k_1 somas com um de k_2 somas é $(k_1 + k_2 + 1)\epsilon$.
- Soma de números não negativos: O erro relativo da soma de um número obtido com k_1 somas de números não negativos com um de k_2 somas é $(max(k_1, k_2) + 1)\epsilon$. Usando um esquema de árvore (ou divisão e conquista) para realizar a soma de n elementos não negativos, pode-se obter erro de $2M\epsilon \log_2 n$. Ver struct $StableSum$ abaixo.
- Multiplicação: Seja d a dimensão da resposta da operação (número máximo de multiplicações em série). O erro absoluto de combinações de n operações $(+,-,*)$ é $M^d((1+\epsilon)^n - 1) \approx nM^d\epsilon$, se $n\epsilon << 1$.
- Radiciação: Uma imprecisão δ se torna uma imprecisão $\sqrt{\delta}$. Ou seja, um erro absoluto de $nM^d\epsilon$ se torna $\sqrt{nM^d\epsilon}$. Cuidado, por exemplo, se $\delta = 10^{-6}, \sqrt{\delta} = 10^{-3}$.
- Divisão: Não é possível estimar o erro da divisão. Se um número x possui a mesma ordem de grandeza que seu erro absoluto, então $1/x$ pode assumir qualquer valor positivo ou negativo ou não existir. Evitar fazer divisão por variáveis.

A struct $StableSum$ (soma estável) executa a soma de n números não negativos com erro absoluto $2M\epsilon \log_2 n$. $val()$ retorna a soma total. Semelhante à Fenwick Tree, funciona em $O(1)$ amortizado na forma estática. Caso os números sejam dinâmicos, usar a Fenwick Tree mesmo.

A função $quadRoots$ retorna quantas raízes uma equação quadrática tem e as bota em out. Ela evita o cancelamento catastrófico causado quando $|b| \approx \sqrt{b^2 - 4ac}$, ou seja, $ac \approx 0$, usando ambas as fórmulas de bháskara:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}} \tag{8.65}$$

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

struct StableSum {
    int cnt = 0;
    vector<double> v, pref(1, 0);
    void operator += (double a) { // a >= 0
        for (int s = ++cnt; s % 2 == 0; s >>= 1) {
            a += v.back();
            v.pop_back(), pref.pop_back();
        }
        v.push_back(a);
        pref.push_back(pref.back() + a);
    }
    double val() { return pref.back(); }
};

int quadRoots(double a, double b, double c, pair<double
    , double> &out) {
    if (fabs(a) < EPS) return 0;
    double delta = b*b - 4*a*c;
    if (delta < 0) return 0;
    double sum = (b >= 0) ? -b-sqrt(delta) : -b+sqrt(
        delta);
    out = {sum/(2*a), fabs(sum) > EPS ? 0 : (2*c)/sum};
    return 1 + (delta > 0);
}
```

Capítulo 9

Outros

9.1 Compressão de coordenadas

```
void compressaoCoordenadas(int *vet, int n){
    int aux[n];
    for(int i = 0; i < n; i++) aux[i] = vet[i];
    sort(aux, aux+n);
    for(int i = 0; i < n; i++) vet[i] = lower_bound(aux
        , aux+n, vet[i]) - aux + 1;
}
```

9.2 Contagem de intersecções entre linhas horizontais ou verticais

```
ll intersectionCount(int *l, int *r, int n){
    FenwickTree ft(n);
    ll count = 0;
    vector<pair<int, int> > lines;
    for(int i = 0; i < n; i++) lines.push_back({l[i], r
        [i]});
    sort(lines.begin(), lines.end());
    for(int i = 0; i < n; i++){
        count += (ll)i - ft.rsq(lines[i].second);
        ft.update(lines[i].second, 1);
    }
    return count;
}
```

9.3 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;
}
```

9.4 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;
}
```

9.5 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;
}
```

9.6 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;
}
```

9.7 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;
}
```

9.8 Calendário gregoriano

count retorna a diferença de dias entre a data e 01/01/0001. *weekday* retorna o dia da semana indexado em 0, iniciando no domingo. *leap* retorna se o ano é bissexto. *advance* anda a data em tantos dias. Todas as funções $O(1)$.

```
int mnt[13] = {0, 31, 59, 90, 120, 151, 181, 212, 243,
    273, 304, 334, 365};

struct Date {
    int d, m, y;
    Date() : d(1), m(1), y(1) {}
    Date(int d, int m, int y) : d(d), m(m), y(y) {}
    Date(int days) : d(1), m(1), y(1) { advance(days);
        }
    bool leap() { return (y%4 == 0 && y%100) || (y%400
        == 0); }
    int mdays() { return mnt[m] - mnt[m-1] + (m == 2)*
        leap(); }
    int ydays() { return 365 + leap(); }
    int count() { // dist to 01/01/01
        return (d-1) + mnt[m-1] + (m > 2)*leap() +
            365*(y-1) + (y-1)/4 - (y-1)/100 + (y-1)
                /400;
    }
    int weekday() { return (count()+1) % 7; }
    void advance(int days) {
        days += count();
        d = m = 1, y = 1 + days/366;
        days -= count();
        while(days >= ydays()) days -= ydays(), y++;
        while(days >= mdays()) days -= mdays(), m++;
        d += days;
    }
};
```

9.9 Iteração sobre polyominos

Itera sobre todas as possíveis figuras de polyominos em uma grade. Posições com $num[i][j] \neq 0$ são proibidas. Neste algoritmo em específico, calcula a soma máxima. Caso precise da forma, guardar a pilha de recursão. Número do polyominos:

tamanho	1	2	3	4	5	6	7	8	9	10	11	12
quantidade	1	2	6	19	63	216	760	2,725	9,910	36,446	135,268	505,861

```
#define MAXN 59
int N, M;
int num[MAXN][MAXN], qi[MAXN*MAXN], qj[MAXN*MAXN];
int field[MAXN][MAXN], par[MAXN][MAXN];
int di[4] = {0, 1, 0, -1};
int dj[4] = {-1, 0, 1, 0};
int cnt;

void assign(int i, int j, int k) {
    qi[k] = i; qj[k] = j; num[i][j] = k;
}

#define valid(i, j) (i >= 0 && i < N && j >= 0 && j < M
    )

int iterate(int k, int h) {
    if (h == 0) return 0;
    int i = qi[k], j = qj[k], ni, nj;
    for(int d = 0; d < 4; d++) {
        ni = i + di[d]; nj = j + dj[d];
        if (!valid(ni, nj)) continue;
        if (num[ni][nj] == 0) {
            par[ni][nj] = k;
            assign(ni, nj, ++cnt);
        }
    }
    int ans = 0, cur;
    for(int t = k+1; t <= cnt; t++) {
        cur = iterate(t, h-1);
        if (cur > ans) ans = cur;
    }
    for(int d = 0; d < 4; d++) {
        ni = i + di[d]; nj = j + dj[d];
        if (!valid(ni, nj)) continue;
        if (par[ni][nj] == k) {
            par[ni][nj] = 0; cnt--;
            assign(ni, nj, 0);
        }
    }
    return ans + field[i][j];
}
```

9.10 Simplex

Algoritmo para resolver problemas de programação linear em que se deseja maximizar/minimizar um custo de N variáveis, dado por $\sum_{i=1}^{n} a_i x_i$, respeitando M condições de restrição (≥, ≤, =). Tem complexidade esperada $O(N^2 M)$ e pior caso exponencial. Para utilizar este algoritmo, chame o método *init* com a função objetivo, *add constraint* para adicionar restrições e *solve* para computar a solução ótima.

```
#define MAXN 109
#define MAXM 10009
#define EPS 1e-9

#define MINIMIZE -1
#define MAXIMIZE +1
#define LESSEQ -1
#define EQUAL 0
#define GREATEQ 1
#define INFEASIBLE -1
#define UNBOUNDED 999

/***
1. m is the number of constraints indexed from 1 to m,
    and n is the number of variables indexed from 0 to
    n-1
2. ar[0] contains the objective function f, and ar[1]
    to ar[m] contains the constraints, ar[i][n] = lim_i
3. All xi >= 0
4. If xi <= 0, replace xi by r1 - r2 (Number of
    variables increases by one, -x, +r1, +r2)
5. solution_flag = INFEASIBLE if no solution is
    possible and UNBOUNDED if no finite solution is
    possible
6. After successful completion, val[] contains the
    values of x0, x1 .... xn for the optimal value
    returned
7. If ABS(X) <= M in constraints, Replace with X <= M
    and -X <= M
8. Fractional LP:

    max/min
        3x1 + 2x2 + 4x3 + 6
        -------------------
        3x1 + 3x2 + 2x3 + 5

        s.t. 2x1 + 3x2 + 5x3 >= 23
        x1, x2, x3 >= 0

    Replace with:

    max/min
        3y1 + 2y2 + 4y3 + 6t

        s.t. 3y1 + 3y2 + 2y3 + 5t = 1
        2y1 + 3y2 + 53 - 23t >= 0
        y1, y2, y3, t >= 0
***/

namespace lp {
double val[MAXN], ar[MAXM][MAXN];
int m, n, solution_flag, minmax_flag, basis[MAXM],
    index[MAXN];

void init(int nvars, double* f, int flag) {
    solution_flag = 0;
    ar[0][nvars] = 0.0;
    m = 0, n = nvars, minmax_flag = flag;
    for (int i = 0; i < n; i++)
        ar[0][i] = f[i] * minmax_flag;
}

void add_constraint(double* C, double lim, int cmp) {
    m++, cmp *= -1;
    if (cmp == EQUAL) {
        for (int i = 0; i < n; i++) ar[m][i] = C[i];
        ar[m++][n] = lim;
        for (int i = 0; i < n; i++) ar[m][i] = -C[i];
        ar[m][n] = -lim;
    }
    else {
        for (int i = 0; i < n; i++) ar[m][i] = C[i] *
            cmp;
        ar[m][n] = lim * cmp;
    }
}

void init() {
    for (int i = 0; i <= m; i++) basis[i] = -i;
    for (int j = 0; j <= n; j++)
        ar[0][j] = -ar[0][j], index[j] = j, val[j] = 0;
}

inline void pivot(int m, int n, int a, int b) {
    for (int i = 0; i <= m; i++)
        for (int j = 0; i != a && j <= n; j++)
            if (j != b) ar[i][j] -= (ar[i][b] * ar[a][j
                ]) / ar[a][b];
    for (int j = 0; j <= n; j++)
        if (j != b) ar[a][j] /= ar[a][b];
    for (int i = 0; i <= m; i++)
        if (i != a) ar[i][b] = -ar[i][b] / ar[a][b];
    ar[a][b] = 1.0 / ar[a][b];
    swap(basis[a], index[b]);
}

inline double solve() {
    init();
    int i, j, k, l;
    while (true) {
        for (i = 1, k = 1; i <= m; i++)
            if ((ar[i][n] < ar[k][n]) || (ar[i][n] ==
                ar[k][n] && basis[i] < basis[k] && (
                rand() & 1))) k = i;
        if (ar[k][n] >= -EPS) break;
        for (j = 0, l = 0; j < n; j++)
            if ((ar[k][j] < (ar[k][l] - EPS)) || (ar[k
                ][j] < (ar[k][l] - EPS) && index[i] <
                index[j] && (rand() & 1))) l = j;
        if (ar[k][l] >= -EPS) {
            solution_flag = INFEASIBLE;
            return -1.0;
        }
        pivot(m, n, k, l);
    }
    while (true) {
        for (j = 0, l = 0; j < n; j++)
            if ((ar[0][j] < ar[0][l]) || (ar[0][j] ==
                ar[0][l] && index[j] < index[l] && (
                rand() & 1))) l = j;
        if (ar[0][l] > -EPS) break;
        for (i = 1, k = 0; i <= m; i++)
            if (ar[i][l] > EPS && (!k || ar[i][n] / ar[
                i][l] < ar[k][n] / ar[k][l] - EPS || (
                ar[i][n] / ar[i][l] < ar[k][n] / ar[k][
                l] + EPS && basis[i] < basis[k]))) k =
                i;
        if (ar[k][l] <= EPS) {
            solution_flag = UNBOUNDED;
            return -999.0;
```

```
        }
        pivot(m, n, k, l);
    }
    for (i = 1; i <= m; i++)
        if (basis[i] >= 0) val[basis[i]] = ar[i][n];
    solution_flag = 1;
    return (ar[0][n] * minmax_flag);
}
}
```

9.11 Quadrado Mágico Ímpar

Gera uma matriz quadrática $n \times n$ em $O(n^2)$, n ímpar, tal que a soma dos elementos ao longo de todas as linhas, de todas as colunas e das duas diagonais é a mesma. Os elementos vão de 1 a n^2. A matriz é indexada em 0.

```
#define MAXN 1009
int mat[MAXN][MAXN], n; //0-indexed

void magicsquare() {
    int i=n-1, j=n/2;
    memset(&mat, 0, sizeof mat);
    for(int k = 1; k <= n*n; k++) {
        mat[i][j] = k;
        if (mat[(i+1)%n][(j+1)%n] > 0) {
            i = (i-1+n)%n;
        }
        else {
            i = (i+1)%n;
            j = (j+1)%n;
        }
    }
}
```

9.12 Ciclos em sequências: Algoritmo de Floyd

```
ii floydCycleFinding(int x0) {
    // 1st part: finding k*start, hares speed is 2x
        tortoises
    int tortoise = f(x0), hare = f(f(x0)); // f(x0) is
        the node next to x0
    while (tortoise != hare) { tortoise = f(tortoise);
        hare = f(f(hare)); }
    // 2nd part: finding start, hare and tortoise move
        at the same speed
    int start = 0; hare = x0;
    while (tortoise != hare) { tortoise = f(tortoise);
        hare = f(hare); start++;}
    // 3rd part: finding period, hare moves, tortoise
        stays
    int period = 1; hare = f(tortoise);
    while (tortoise != hare) { hare = f(hare); period
        ++; }
    return ii(start, period);
}
```

9.13 Expressão Parentética para Polonesa

Dada uma expressão matemática na forma parentética, converte para a forma polonesa e retorna o tamanho da string na forma polonesa. Ex.: $(2*4/a \wedge b)/(2*c) \rightarrow 24*ab \wedge /2c*/$.

```
inline bool isOp(char c) {
    return c=='+' || c=='-' || c=='*' || c=='/' || c=='
        ^';
}

inline bool isCarac(char c) {
    return (c>='a' && c<='z') || (c>='A' && c<='Z') ||
        (c>='0' && c<='9');
}

int paren2polish(char* paren, char* polish) {
    map<char, int> prec;
    prec['('] = 0;
    prec['+'] = prec['-'] = 1;
    prec['*'] = prec['/'] = 2;
    prec['^'] = 3;
    int len = 0;
    stack<char> op;
    for (int i = 0; paren[i]; i++) {
        if (isOp(paren[i])) {
            while (!op.empty() && prec[op.top()] >=
                prec[paren[i]]) {
                polish[len++] = op.top(); op.pop();
            }
            op.push(paren[i]);
        }
        else if (paren[i]=='(') op.push('(');
        else if (paren[i]==')') {
            for (; op.top()!='('; op.pop())
                polish[len++] = op.top();
            op.pop();
        }
        else if (isCarac(paren[i]))
            polish[len++] = paren[i];
    }
    for(; !op.empty(); op.pop())
        polish[len++] = op.top();
    polish[len] = 0;
    return len;
}
```

9.14 Problema de Josephus

Em um círculo de n jogadores (0-indexed) em que eles se matam a cada k em ordem crescente. O índice do sobrevivente pode ser calculado pela recursão: $f(n,k) = (f(n-1,k)+k)\%n, f(1,k) = 0$.

9.15 Problema do histograma

Algoritmo $O(n)$ para resolver o problema do retângulo máximo em um histograma.

```
#define MAXN 100009

ll histogram(ll vet[], int n) {
    stack<ll> s;
    ll ans = 0, tp, cur;
    int i = 0;
    while(i < n || !s.empty()) {
        if (i < n && (s.empty() || vet[s.top()] <= vet[
            i])) s.push(i++);
        else {
            tp = s.top();
            s.pop();
            cur = vet[tp] * (s.empty() ? i : i - s.top
                () - 1);
            if (ans < cur) ans = cur;
        }
    }
    return ans;
}
```

9.16 Problema do casamento estável

Resolve o problema do casamento estável em $O(n^2)$. m é o número de homens, n é o número de mulheres, $L[i]$ é a lista de preferências do i-ésimo homem (melhor primeiro), $R[i][j]$ é a nota que a mulher i dá ao homem j. $R2L[i] == j$ retorna se a mulher i está casada com o homem j, $L2R[i] == j$ retorna se o homem i está casado com a mulher j.

```
#define MAXN 1009
int m, n, p[MAXN];
int L[MAXN][MAXN], R[MAXN][MAXN];
int R2L[MAXN], L2R[MAXN];

void stableMarriage() {
    memset(R2L, -1, sizeof(R2L));
    memset(p, 0, sizeof(p));
    for(int i = 0, wom, hubby; i < m; i++) {
        for (int man = i; man >= 0;) {
            while(true) {
                wom = L[man][p[man]++];
                if( R2L[wom] < 0 || R[wom][man] > R[wom
                    ][R2L[wom]] ) break;
            }
            hubby = R2L[wom];
            R2L[L2R[man] = wom] = man;
            man = hubby;
        }
    }
}
```

9.17 Código de Huffman

Computa o custo do autômato de Huffman: dado um conjunto de elementos, montar uma árvore binária cujas folhas são os elementos minimizando: $cost = \sum_{i=0}^{n-1} a[i] \times depth[i]$.

```
ll huffman(ll* a, int n) {
    ll ans = 0, u, v;
    priority_queue<ll> pq;
    for(int i=0; i<n; i++) pq.push(-a[i]);
    while(pq.size() > 1) {
        u = -pq.top(); pq.pop();
        v = -pq.top(); pq.pop();
        pq.push(-u-v);
        ans += u + v;
    }
    return ans;
}
```

9.18 Problema do Cavalo

Como achar, em um tabuleiro, um caminho hamiltoniano com o cavalo? Backtrack usando como heurística a regra de Warsndorf: visite primeiro lugares com o menor número de saídas.

9.19 Intersecção de Matróides

Encontra o conjunto independente máximo na intersecção de dois matróides. Um exemplo é a maior Spanning Tree com arestas de cores diferentes. Complexidade até $O(N*M^2)$, na prática bem menor. O algoritmo procura sequências aumentantes, fazendo um grafo bipartido entre arestas dentro/fora do conjunto máximo, e BFS de Q (arestas fora de cores não usadas) a T (arestas fora que não formam ciclo).

```
#define MAXM 1009
vector<set<ii> > adjList;
vector<vector<int> > colors;
vector<bool> T, Q, inSet, usedColor;
ii edges[MAXM];
int c[MAXM], N, M, C;

class UnionFind { ... };

void findCycle(int s, int t, vector<int>& cycle) {
    vector<ii> prv;
    queue<int> q;
    prv.assign(N, ii(-1, -1));
    q.push(s); prv[s] = ii(s, -1);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        if (u == t) break;
        for (set<ii>::iterator it = adjList[u].begin();
             it != adjList[u].end(); it++) {
            int v = (*it).first;
            if (prv[v].first == -1) {
                prv[v] = ii(u, (*it).second);
                q.push(v);
            }
        }
    }
    while (t != s) {
        cycle.push_back(prv[t].second);
        t = prv[t].first;
    }
}

void greedyAugmentation(UnionFind &UF) {
    int e, u, v;
    for (int i = 0; i < M; i++) {
        u = edges[i].first, v = edges[i].second;
        if (!UF.isSameSet(u, v) && !usedColor[c[i]]) {
            UF.unionSet(u, v);
            adjList[u].insert(ii(v, i));
            adjList[v].insert(ii(u, i));
            usedColor[c[i]] = true;
            inSet[i] = true;
        }
    }
}

int bfs(vector<int>& prv) {
    queue<int> q;
    prv.assign(M, -1);
    for (int i = 0; i < M; i++) {
        if (Q[i]) { q.push(i); prv[i] = i; }
    }
    int u;
    while (!q.empty()) {
        u = q.front(); q.pop();
        if (T[u]) return u;
        // If in independent set, check others from
            same color
        if (inSet[u]) {
            for (int i = 0; i < colors[c[u]].size(); i
                ++) {
                int v = colors[c[u]][i];
                if (v != u && prv[v] == -1) {
                    prv[v] = u; q.push(v);
                }
            }
        } else { // If not independent, check cycles
            vector<int> cycle;
            findCycle(edges[u].first, edges[u].second,
                cycle);
            for (int i = 0; i < cycle.size(); i++) {
                if (prv[cycle[i]] == -1) {
                    prv[cycle[i]] = u; q.push(cycle[i])
                        ;
                }
            }
        }
    }
    return -1;
}

bool augment(UnionFind& UF) {
    Q.assign(M, false); T.assign(M, false);
    for (int i = 0; i < M; i++) {
        if (!UF.isSameSet(edges[i].first, edges[i].
            second)) { T[i] = true; }
        if (!usedColor[c[i]]) { Q[i] = true; }
    }

    vector<int> prv;
    int u = bfs(prv);
    if (u == -1) return false;
    UF.unionSet(edges[u].first, edges[u].second);
    while (true) {
        if(inSet[u]) {
            adjList[edges[u].first].erase(ii(edges[u].
                second, u));
            adjList[edges[u].second].erase(ii(edges[u].
                first, u));
        } else {
            adjList[edges[u].first].insert(ii(edges[u].
                second, u));
            adjList[edges[u].second].insert(ii(edges[u
                ].first, u));
        }

        inSet[u] = !inSet[u];
        if (u == prv[u]) break;
        u = prv[u];
    }
    usedColor[c[u]] = true;
    return true;
}

int maxIndependentSet() {
    inSet.assign(M, false); usedColor.assign(C, false);
    adjList.clear(); adjList.resize(N+5);

    UnionFind UF(N);
    greedyAugmentation(UF);
    while (augment(UF));

    int sz = 0;
    for (int i = 0; i < M; i++) if(inSet[i]) sz++;
    return sz;
}
```

Apêndice A

Fórmulas

Agradecimentos a Felipe Gazzoni Foschiera.

A.1 Progressão Aritmética

Fórmula do Termo Geral: $a_n = a_1 + (n-1) \times r$

Soma dos termos da PA: $S_n = \dfrac{n \times (a_1 + a_n)}{2}$

A.2 Progressão Geométrica

Fórmula do Termo Geral: $a_n = a_1 \times q^{n-1}$
Soma dos termos da PG: $S_n = \dfrac{a_1 \times (q^n - 1)}{q - 1}$

A.3 Número de áreas em um plano divididas por retas e suas intersecções

$A = N + I + 1$ onde N é o número de retas e I é o número de intersecções. Cada reta horizontal tem uma intersecção com uma reta vertical, então sempre existem ao menos $v \times h$ intersecções, onde v é o número de retas verticais e h horizontais.

A.4 Números Triangulares

Um número triangular é um número natural que pode ser representado na forma de um triângulo equilátero. O n-ésimo número triangular pode ser visto como o número de pontos de uma forma triangular com lado formado por n pontos, o que equivale à soma dos primeiros n números naturais.
Em geral, o n-ésimo número triangular é dado por: $T_n = \sum_{k=1}^{n} k = 1 + 2 + 3 + ... + (n-2) + (n-1) + n = \dfrac{n(n+1)}{2}$

A soma dos primeiros n números triangulares é o n-ésimo número tetraédrico, que tem como fórmula: $\dfrac{n(n+1)(n+2)}{6}$
Raízes triangulares e teste de identificação (número de linhas triangulares que podem ser formadas com n elementos) $n = \dfrac{\sqrt{8x+1} - 1}{2}$

A.5 Múltiplos positivos de k num intervalo

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

A.6 Número par ou ímpar de divisores

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

A.7 Número de quadrados perfeitos de A a B

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

A.8 Quadrados e retângulos em um Grid de N lados com K dimensões

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

A.9 Número de pares que podem ser formados combinando N elementos

$P = d\frac{n \times (n-1)}{2}$

A.10 Quadrado

$A = b \times h$
$p = 4 \times l$
$D = l\sqrt{2}$

A.11 Círculo

$A = \pi \times r^2$
$d = 2 \times r$
$2p = 2 \times \pi \times r$

A.12 Triângulo

Área: $A = \frac{b \times h}{2}$
Área do Triângulo Equilátero: $A = \frac{\sqrt{3}}{4}L^2$
Semi-perímetro: $p = \frac{a+b+c}{2}$
Fórmula de Heron para área: $A = \sqrt{p(p-a)(p-b)(p-c)}$

A.13 Cubo

Diagonal Lateral: $= a\sqrt{2}$
Diagonal do Cubo: $d = a\sqrt{3}$
Área Total: $A_t = 6a^2$
Área Lateral: $A_l = 4a^2$
Área da Base: $A_b = a^2$
Volume: $V = a^3$

A.14 Cilindro

Área da Base: $A_b = \pi r^2$
Área Lateral: $A_l = 2\pi rh$
Área Total: $A_t = 2A_b + A_l$
Volume: $V = A_b \times h$

A.15 Prisma

Área da Face: $A_f = a \times h$
Área Lateral: $A_l = n \times a$ onde n = número de lados e a = área da face
Área da Base: $A_b = 3 \times a^3 \times \sqrt{3/2}$
Área Total: $A_t = S_l + 2S_b$ Volume: $V = A_b \times h$

A.16 Pirâmide

Área Total: $A_t = A_l + A_b$
Volume: $V = 1/3 \times A_b \times h$

A.17 Cone

$r^2 + h^2 = g^2$ Área da Base: $A_b = \pi \times r^2$
Área Lateral: $A_l = \pi \times r \times g$
Área total: $A_t = \pi \times r \times (g + r)$
Volume: $V = 1/3 \times \pi \times r^2 \times h$

A.18 Paralelepípedo

Área da base: $A_b = ab$
Área total: $A_t = 2ab + 2bc + 2ac$
Volume: $V = abc$
Diagonais: $D = \sqrt{a^2 + b^2 + c^2}$

Um paralelepípedo possui 3 grupos de 4 arestas idênticas, cada uma representando comprimento (x), largura (y) e altura (z).

$a = xy$
$b = yz$
$c = xz$
$\sqrt{abc} = xyz$
$x = \frac{\sqrt{abc}}{b}$
$y = \frac{\sqrt{abc}}{c}$
$z = \frac{\sqrt{abc}}{a}$

A.19 Quadrado inscrito e circunscrito em circuferência

Inscrito: Lado: $l = r\sqrt{2}$ | Apótema: $a = \frac{r\sqrt{2}}{2}$
Circunscrito: Lado: $l = 2r$ | Apótema: $a = l$

A.20 Hexágono regular inscrito e circunscrito em circunferência

Inscrito: Lado: $l = r$ | Apótema: $a = \frac{r\sqrt{3}}{2}$
Circunscrito: $R^2 = (l/2)^2 + r^2$ | Lado: $l = 2\sqrt{R^2 - r^2}$ | Apótema: $a = r$

A.21 Triângulo equilátero inscrito e circunscrito em circunferência

Inscrito: Lado: $l = r\sqrt{3}$ | Apótema: $\frac{r}{2}$
Circunscrito: Lado: $l = 2\sqrt{R^2 - r^2}$ | Apótema: $a = r$

A.22 Raio do círculo inscrito e circunscrito num triângulo

Inscrito: $R = \frac{areaTriangulo}{semiPerimetro}$ Circunscrito: $R = \frac{abc}{4 \times areaTriangulo}$

A.23 Círculo dentro de outro

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

A.24 Quantidade de pontos inteiros embaixo de uma reta entre dois pontos

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