



National Institute of Technology Tiruchirappalli

# IcyPeasy

Ashwanth Kannan, Rohit Meena, Satish Prajapati

2024-12-18

- 1 Contest
- 2 Mathematics
- 3 Data structures
- 4 Numerical
- 5 Number theory
- 6 Combinatorial
- 7 Graph
- 8 Geometry
- 9 Strings
- 10 Various

# Contest (1)

template.cpp73 lines

```
// #pragma GCC optimize("O3")
// #pragma GCC optimize("unroll-loops")

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

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

template <class T> using ordered_set = tree<T, null_type, less<
    T>, rb_tree_tag, tree_order_statistics_node_update>;

#define ll long long
#define ull unsigned long long
#define lld long double
#define pii pair<int,int>
#define pll pair<ll,ll>

#define fastio() ios_base::sync_with_stdio(false);cin.tie(NULL)
;
#define rep(i,a,n) for (int i = a; i < n; i++)
#define vi vector<int>
#define nline "\n"
#define inf (ll)1e18
#define iinf (int)2e9
#define eb emplace_back
#define vb vector<bool>
#define vll vector<ll>
#define vvl vector<vll>
#define vp11 vector<pll>
#define vvi vector<vector<int>>
#define vvb vector<vector<bool>>
#define vc vector<char>
#define vvc vector<vector<char>>
#define pb push_back
#define pf push_front
#define ppb pop_back
```

1#define ppf pop\_front
2#define mp make\_pair
3#define fs first
4#define sc second
5#define PI 3.141592653589793238462
6#define set\_bits \_\_builtin\_popcountll
7#define sz(x) ((int)(x).size())
8#define all(x) (x).begin(), (x).end()
9#define rall(x) (x).rbegin(), (x).rend()

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

.bashrc3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
    -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps =<
```

.vimrc6 lines

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on | im jk <esc> | im kj <esc> | no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \l tr -d '[:space:]' \
    \l md5sum \l cut -c-6
```

hash.sh3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum |cut -c-6
```

troubleshoot.txt52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
```

Are you clearing all data structures between test cases?  
Can your algorithm handle the whole range of input?  
Read the full problem statement again.  
Do you handle all corner cases correctly?  
Have you understood the problem correctly?  
Any uninitialized variables?  
Any overflows?  
Confusing N and M, i and j, etc.?  
Are you sure your algorithm works?  
What special cases have you not thought of?  
Are you sure the STL functions you use work as you think?  
Add some assertions, maybe resubmit.  
Create some testcases to run your algorithm on.  
Go through the algorithm for a simple case.  
Go through this list again.  
Explain your algorithm to a teammate.  
Ask the teammate to look at your code.  
Go for a small walk, e.g. to the toilet.  
Is your output format correct? (including whitespace)  
Rewrite your solution from the start or let a teammate do it.

Runtime error:  
Have you tested all corner cases locally?  
Any uninitialized variables?  
Are you reading or writing outside the range of any vector?  
Any assertions that might fail?  
Any possible division by 0? (mod 0 for example)  
Any possible infinite recursion?  
Invalidated pointers or iterators?  
Are you using too much memory?  
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:  
Do you have any possible infinite loops?  
What is the complexity of your algorithm?  
Are you copying a lot of unnecessary data? (References)  
How big is the input and output? (consider scanf)  
Avoid vector, map. (use arrays/unordered\_map)  
What do your teammates think about your algorithm?

Memory limit exceeded:  
What is the max amount of memory your algorithm should need?  
Are you clearing all data structures between test cases?

## Mathematics (2)

### 2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by  $x = -b/2a$ .

$$\begin{matrix} ax + by = e & x = \frac{ed - bf}{ad - bc} \\ cx + dy = f & \Rightarrow y = \frac{af - ec}{ad - bc} \end{matrix}$$

In general, given an equation  $Ax = b$ , the solution to a variable  $x_i$  is given by

$$x_i = \frac{\det A'_i}{\det A}$$

where  $A'_i$  is  $A$  with the  $i$ 'th column replaced by  $b$ .

2.2 Recurrences

If  $a_n = c_1a_{n-1} + \cdots + c_k a_{n-k}$ , and  $r_1, \dots, r_k$  are distinct roots of  $x^k - c_1x^{k-1} - \cdots - c_k$ , there are  $d_1, \dots, d_k$  s.t.

$$a_n = d_1r_1^n + \cdots + d_kr_k^n.$$

Non-distinct roots  $r$  become polynomial factors, e.g.  $a_n = (d_1n + d_2)r^n$ .

2.3 Trigonometry

$$\sin(v+w) = \sin v \cos w + \cos v \sin w$$
$$\cos(v+w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$

$$(V+W)\tan(v-w)/2 = (V-W)\tan(v+w)/2$$

where  $V, W$  are lengths of sides opposite angles  $v, w$ .

$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

where  $r = \sqrt{a^2 + b^2}, \phi = \operatorname{atan2}(b, a)$ .

2.4 Geometry

2.4.1 Triangles

Side lengths:  $a, b, c$

Semiperimeter:  $p = \frac{a+b+c}{2}$

Area:  $A = \sqrt{p(p-a)(p-b)(p-c)}$

Circumradius:  $R = \frac{abc}{4A}$

Inradius:  $r = \frac{A}{p}$

Length of median (divides triangle into two equal-area triangles):  
 $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

$$s_a = \sqrt{bc \left[ 1 - \left( \frac{a}{b+c} \right)^2 \right]}$$

Law of sines:  $\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$

Law of cosines:  $a^2 = b^2 + c^2 - 2bc \cos \alpha$

Law of tangents:  $\frac{a+b}{a-b} = \frac{\tan \frac{\alpha+\beta}{2}}{\tan \frac{\alpha-\beta}{2}}$

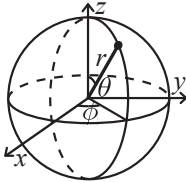
2.4.2 Quadrilaterals

With side lengths  $a, b, c, d$ , diagonals  $e, f$ , diagonals angle  $\theta$ , area  $A$  and magic flux  $F = b^2 + d^2 - a^2 - c^2$ :

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is  $180^\circ$ ,  $ef = ac + bd$ , and  $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$ .

2.4.3 Spherical coordinates



$$x = r \sin \theta \cos \phi \qquad r = \sqrt{x^2 + y^2 + z^2}$$
$$y = r \sin \theta \sin \phi \qquad \theta = \operatorname{acos}(z/\sqrt{x^2 + y^2 + z^2})$$
$$z = r \cos \theta \qquad \phi = \operatorname{atan2}(y, x)$$

2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx} \tan x = 1 + \tan^2 x \qquad \frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$
$$\int \tan ax = -\frac{\ln |\cos ax|}{a} \qquad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x) \qquad \int x e^{ax} dx = \frac{e^{ax}}{a^2} (ax - 1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.6 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, (-\infty < x < \infty)$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots, (-1 < x \leq 1)$$
$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \cdots, (-1 \leq x \leq 1)$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, (-\infty < x < \infty)$$

2.8 Probability theory

Let  $X$  be a discrete random variable with probability  $p_X(x)$  of assuming the value  $x$ . It will then have an expected value (mean)  $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$  and variance  $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$  where  $\sigma$  is the standard deviation. If  $X$  is instead continuous it will have a probability density function  $f_X(x)$  and the sums above will instead be integrals with  $p_X(x)$  replaced by  $f_X(x)$ .

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent  $X$  and  $Y$ ,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

2.8.1 Discrete distributions  
Binomial distribution

The number of successes in  $n$  independent yes/no experiments, each which yields success with probability  $p$  is  $\operatorname{Bin}(n, p)$ ,  $n = 1, 2, \dots$ ,  $0 \leq p \leq 1$ .

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \sigma^2 = np(1-p)$$

$\operatorname{Bin}(n, p)$  is approximately  $\operatorname{Po}(np)$  for small  $p$ .

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability  $p$  is  $\operatorname{Fs}(p)$ ,  $0 \leq p \leq 1$ .

$$p(k) = p(1-p)^{k-1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time  $t$  if these events occur with a known average rate  $\kappa$  and independently of the time since the last event is  $\text{Po}(\lambda)$ ,  $\lambda = t\kappa$ .

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$
$$\mu = \lambda, \sigma^2 = \lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between  $a$  and  $b$  and 0 elsewhere it is  $\text{U}(a, b)$ ,  $a < b$ .

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$
$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is  $\text{Exp}(\lambda)$ ,  $\lambda > 0$ .

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean  $\mu$  and variance  $\sigma^2$  are well described by  $\mathcal{N}(\mu, \sigma^2)$ ,  $\sigma > 0$ .

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If  $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$  and  $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$  then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

2.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let  $X_1, X_2, \dots$  be a sequence of random variables generated by the Markov process. Then there is a transition matrix  $\mathbf{P} = (p_{ij})$ , with  $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$ , and  $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$  is the probability distribution for  $X_n$  (i.e.,  $p_i^{(n)} = \Pr(X_n = i)$ ), where  $\mathbf{p}^{(0)}$  is the initial distribution.

$\pi$  is a stationary distribution if  $\pi = \pi \mathbf{P}$ . If the Markov chain is *irreducible* (it is possible to get to any state from any state), then  $\pi_i = \frac{1}{\mathbb{E}(T_i)}$  where  $\mathbb{E}(T_i)$  is the expected time between two visits in state  $i$ .  $\pi_j/\pi_i$  is the expected number of visits in state  $j$  between two visits in state  $i$ .

OrderStatisticTree HashMap InfoTag SegmentTree

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors,  $\pi_i$  is proportional to node  $i$ 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1).  $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$ .

A Markov chain is an A-chain if the states can be partitioned into two sets  $\mathbf{A}$  and  $\mathbf{G}$ , such that all states in  $\mathbf{A}$  are absorbing ( $p_{ii} = 1$ ), and all states in  $\mathbf{G}$  leads to an absorbing state in  $\mathbf{A}$ . The probability for absorption in state  $i \in \mathbf{A}$ , when the initial state is  $j$ , is  $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$ . The expected time until absorption, when the initial state is  $i$ , is  $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$ .

Data structures (3)

OrderStatisticTree.h

**Description:** findbyorder finding the n'th element, and orderOfKey finding the index of an element. To get a map, change null\_type.  
**Time:**  $\mathcal{O}(\log N)$

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

template<class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;

void example() {
    Tree<int> t, t2; t.insert(8);
    auto it = t.insert(10).first;
    assert(it == t.lower_bound(9));
    assert(t.order_of_key(10) == 1);
    assert(t.order_of_key(11) == 2);
    assert(*t.find_by_order(0) == 8);
    t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

HashMap.h

**Description:** Hash map with mostly the same API as unordered\_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
    const uint64_t C = 11(4e18 * acos(0)) | 71;
    ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};

// usage
int main() {
    __gnu_pbds::gp_hash_table<ll, int, chash> h({},{},{},{},{1<<16}>);
    h[2] = 3;
}
```

InfoTag.h

**Description:** Info and Tag structures used for segment tree. Sample for min segment tree.

```
struct Tag {
    int add = 0;
    void apply(const Tag &t) & {
    }
    bool operator==(const Tag &t) const {
        return add == t.add;
    }
};

struct Info {
    int mn = 1e9;
    void apply(const Tag &t) & {
    }
};

Info operator+(const Info &l, const Info &r) {
    return {min(l.mn, r.mn)};
}

// pred (find first function)
auto predicate = [&](const Info &info) {
    if(info.val >= k) return true;
    k -= info.val;
    return false;
};
```

SegmentTree.h

**Description:** Zero-indexed tree. Bounds are inclusive to the left and exclusive to the right.  
**Usage:** SegmentTree<Info> segtree  
**Time:**  $\mathcal{O}(\log N)$

```
template<class Info>
struct SegmentTree {
    int n;
    vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(vector<Info>(n_, v_));
    }
    template<class T>
    void init(vector<T> init_) {
        n = init_.size();
        info.assign(4 << __lg(n), Info());
        function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = {init_[l]};
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
        }
    }
};
```

```
        return;
    }
    int m = (l + r) / 2;
    if (x < m) modify(2 * p, l, m, x, v);
    else modify(2 * p + 1, m, r, x, v);
    pull(p);
}

void modify(int p, const Info &v) {
    modify(1, 0, n, p, v);
}

Info rangeQuery(int p, int l, int r, int x, int y) {
    if (l >= y || r <= x) return Info();
    if (l >= x && r <= y) return info[p];
    int m = (l + r) / 2;
    return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p +
        1, m, r, x, y);
}

Info rangeQuery(int l, int r) {
    return rangeQuery(1, 0, n, l, r);
}

template<class F>
int findFirst(int p, int l, int r, int x, int y, F pred) {
    if (l >= y || r <= x || !pred(info[p])) return -1;
    if (r - l == 1) return l;

    int m = (l + r) / 2;
    int res = findFirst(2 * p, l, m, x, y, pred);
    if (res == -1) {
        res = findFirst(2 * p + 1, m, r, x, y, pred);
    }
    return res;
}

template<class F>
int findFirst(int l, int r, F pred) {
    return findFirst(1, 0, n, l, r, pred);
}
};
```

LazySegmentTree.h

**Description:** Lazy Segment Tree with range updates and queries  
**Usage:** LazySegmentTree<Info,Tag> segtree;  
**Time:**  $\mathcal{O}(\log N)$ .

9ad845, 90 lines

```
template<class Info, class Tag>
struct LazySegmentTree {
    int n;
    vector<Info> info;
    vector<Tag> tag;
    LazySegmentTree() : n(0) {}
    LazySegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    LazySegmentTree(vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(vector<Info>(n_, v_));
    }
    template<class T>
    void init(vector<T> init_) {
        n = init_.size();
        info.assign(4 << __lg(n), Info());
        tag.assign(4 << __lg(n), Tag());
        function<void(int, int, int)> build = [&](int p, int l, int
            r) {
                if (r - l == 1) {
                    info[p] = {init_[l]};
                    return;
                }
                int m = (l + r) / 2;
                build(2 * p, l, m, x, y, pred);
                build(2 * p + 1, m, r, x, y, pred);
                pull(p);
            };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        if (tag[p] == Tag()) return;
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = Tag();
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) return Info();
        if (l >= x && r <= y) return info[p];
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p +
            1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x) return -1;
        if (l >= x && r <= y && !pred(info[p])) return -1;
        if (r - l == 1) return l;
        int m = (l + r) / 2;
        push(p);
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
};
```

```
    }
    int m = (l + r) / 2;
    build(2 * p, l, m);
    build(2 * p + 1, m, r);
    pull(p);
};

build(1, 0, n);

void pull(int p) {
    info[p] = info[2 * p] + info[2 * p + 1];
}

void apply(int p, const Tag &v) {
    info[p].apply(v);
    tag[p].apply(v);
}

void push(int p) {
    if (tag[p] == Tag()) return;
    apply(2 * p, tag[p]);
    apply(2 * p + 1, tag[p]);
    tag[p] = Tag();
}

Info rangeQuery(int p, int l, int r, int x, int y) {
    if (l >= y || r <= x) return Info();
    if (l >= x && r <= y) return info[p];

    int m = (l + r) / 2;
    push(p);
    return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p +
        1, m, r, x, y);
}

Info rangeQuery(int l, int r) {
    return rangeQuery(1, 0, n, l, r);
}

void rangeApply(int p, int l, int r, int x, int y, const Tag
    &v) {
    if (l >= y || r <= x) return;
    if (l >= x && r <= y) {
        apply(p, v);
        return;
    }
    int m = (l + r) / 2;
    push(p);
    rangeApply(2 * p, l, m, x, y, v);
    rangeApply(2 * p + 1, m, r, x, y, v);
    pull(p);
}

void rangeApply(int l, int r, const Tag &v) {
    return rangeApply(1, 0, n, l, r, v);
}

template<class F>
int findFirst(int p, int l, int r, int x, int y, F &&pred) {
    if (l >= y || r <= x) return -1;
    if (l >= x && r <= y && !pred(info[p])) return -1;
    if (r - l == 1) return l;
    int m = (l + r) / 2;
    push(p);
    int res = findFirst(2 * p, l, m, x, y, pred);
    if (res == -1) {
        res = findFirst(2 * p + 1, m, r, x, y, pred);
    }
    return res;
}

template<class F>
int findFirst(int l, int r, F &&pred) {
    return findFirst(1, 0, n, l, r, pred);
}
};
```

PersistentSegmentTree.h

**Description:** Persistent SegmentTree with point updates and range queries, [L,R)  
**Usage:** node \*head = build(arr , 0 , N);  
**Time:**  $\mathcal{O}(\log N)$ .

b4459a, 38 lines

```
ll f(ll x , ll y) {
    return (x + y);
}

struct node {
    node *l, *r;
    ll val;
    node(ll val) : l(nullptr), r(nullptr), val(val) {}
    node(node *l, node *r) : l(l), r(r), val(lll) {
        if (l) val = f(val , l->val);
        if (r) val = f(val , r->val);
    }
};

node* build(ll a[], ll lx, ll rx) {
    if (rx - lx == 1)
        return new node(a[lx]);
    ll m = (lx + rx) / 2;
    return new node(build(a, lx, m), build(a, m, rx));
}

ll range_calc(node* v, ll lx, ll rx, ll l, ll r) {
    if (r <= lx || rx <= l)
        return lll;
    if (l <= lx && rx <= r)
        return v->val;
    ll m = (lx + rx) / 2;
    return f(range_calc(v->l, lx, m, l, r) , range_calc(v->r, m
        , rx, l, r));
}

node* update(node* v, ll lx, ll rx, ll pos, ll new_val) {
    if (rx - lx == 1)
        return new node(new_val);
    ll m = (lx + rx) / 2;
    if (pos < m)
        return new node(update(v->l, lx, m, pos, new_val), v->r
            );
    else
        return new node(v->l, update(v->r, m, rx, pos, new_val)
            );
}
```

SparseTable.h

**Description:** Sparse Table finding sum in range. Both L,R inclusive  
**Usage:** st god(N);  
god.build();  
god.qry(L,R) in  $\mathcal{O}(\log N)$  god.qry-i(L,R) in  $\mathcal{O}(1)$   
**Time:**  $\mathcal{O}(|V| \log |V| + Q)$

2dfa9e, 39 lines

```
const ll N=200000;
const ll M = 21;
ll tab[N+1][M+1],L[N+1],a[N];

struct st{
    ll n;
    st(ll _n){
        n=_n;
        for(ll i=2;i<=n;i++) L[i]=L[i/2]+1;
    }
    ll f(ll x,ll y){
        return (x+y);
    }
    void build(){
        for(ll i=0;i<n;i++) tab[i][0]=a[i];
    }
};
```

```
for(1l j=1;j<=M;j++){
    for(1l i=0;i<n;i++){
        if(i+(1<<j)-1<n)
            tab[i][j]=f(tab[i][j-1],tab[i+(1<<(j-1))][j-1]);
    }
}
}
1l qry(1l l,1l r){
    1l len=r-l+1;
    1l idx=1;
    1l tot=0; // initialize neutral
    for(1l j=M;j>=0;j--){
        if(len&(1l<<j)){
            tot=f(tot,tab[idx][j]);
            idx+=(1<<j);
        }
    }
    return tot;
}
1l qry_i(1l l,1l r){
    1l lg=L[r-l+1];
    return f(tab[1][lg],tab[r-(1<<lg)+1][lg]);
}
};
```

MoQueries.h

**Description:** Answer offline interval or tree path queries  
**Usage:** Create vector<query> and pass to the mo fn  
**Time:**  $\mathcal{O}(N\sqrt{Q})$

```
void remove(int idx){}
void add(int idx){}
1l get_answer(){}
1l BLOCK_SIZE = 700;

struct Query {
    1l l, r, idx;
    bool operator<(Query other) const
    {
        if (l / BLOCK_SIZE != other.l / BLOCK_SIZE)
            return make_pair(l,r) < make_pair(other.l , other.r);
        return (l / BLOCK_SIZE & 1) ? (r < other.r) : (r >
            other.r);
    }
};

v1 mo_s_algorithm(vector<Query> queries) {

    vector<1l> answers(queries.size());
    sort(queries.begin(), queries.end());

    int cur_l = 0;
    int cur_r = -1;

    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
        }
    }
}
```

```
cur_r--;
}
answers[q.idx] = get_answer();
}
return answers;
}
```

DSURollback.h

**Description:** Disjoint-set data structure with undo.  
**Time:**  $\mathcal{O}(\log N)$

```
struct DSUwithRollback {
    vector<int> par, size;
    int c;
    stack<array<int,4>> st;
    void init(int n){
        c = n;
        par.resize(n + 1);
        size.assign(n + 1, 1);
        iota(par.begin(), par.end(), 0);
    }
    int leader(int u){
        return u == par[u] ? u : leader(par[u]);
    }
    bool same(int u,int v){
        return leader(u) == leader(v);
    }
    int get_size(int u){
        return size[leader(u)];
    }
    int count(){
        return c;
    }
    bool merge(int u,int v){
        u = leader(u);
        v = leader(v);
        if(u == v) return false;
        if(size[u] < size[v]) swap(u,v);
        st.push({u, v, size[u], size[v]});
        c--;
        par[v] = u;
        size[u] += size[v];
        return true;
    }
    void rollback(){
        if(st.empty()) return;
        auto [u, v, su, sv] = st.top();
        st.pop();
        c++;
        par[v] = v;
        size[u] = su;
        size[v] = sv;
    }
};
```

DynamicConnectivity.h

**Description:** Dynamic Connectivity (DSU rollback example)  
**Time:**  $\mathcal{O}(N\log^2(N))$

```
template<class Info>
struct DynamicConnectivity{
    int Q;
    vector<vector<Info>> seg;
    void init(int _q){
        Q = _q;
        seg.assign(4 * _q, {});
    }
    void update(int p, int l, int r, int ql, int qr, const Info &
        v){
```

```
if(ql >= r || qr <= l) return;
if(ql <= l && r <= qr){
    seg[p].push_back(v);
    return;
}
int m = (l + r) >> 1;
update(p << 1, l, m, ql, qr, v);
update(p << 1 | 1, m, r, ql, qr, v);
}
void dfs(int p, int l, int r, int L, int R){
    for(auto &v : seg[p]) v.add();
    if(l + 1 == r){
        // print answer
        if(l>=L && l<=R){
            cout<<dsu.count()<<" ";
        }
        for(int i = sz(seg[p]) - 1; i >= 0; i--){seg[p][i].remove
            ()};
        return;
    }
    int m = (l + r) >> 1;
    dfs(p << 1, l, m, L, R);
    dfs(p << 1 | 1, m, r, L, R);
    for(int i = sz(seg[p]) - 1; i >= 0; i--){seg[p][i].remove()
        ;}
}

void add(const Info &v, int l, int r){
    update(l, 0, Q, l, r, v);
}
void get(int L, int R){
    dfs(1, 0, Q, L, R);
    cout<<nl;
}
};
struct Info{
    int u, v, added = 0;
    void add(){added = dsu.merge(u, v);}
    void remove(){if(added) dsu.rollback();}
```

Treap.h

**Description:** A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.  
**Time:**  $\mathcal{O}(\log N)$

```
struct Treap{
    int data, priority, subtreeSize;
    long long sum;
    bool rev;
    Treap *left, *right;
    Treap(int d);
};
int size(Treap *t){
    return t ? t->subtreeSize : 0;
}
long long getSum(Treap *t){
    return t ? t->sum : 0;
}
void pull(Treap *t){
    if(!t) return;
    t->subtreeSize = 1 + size(t->left) + size(t->right);
    t->sum = t->data + getSum(t->left) + getSum(t->right);
}
Treap::Treap(int d){
    this->data = d;
    priority = rand(0, INT_MAX);
    subtreeSize = 1;
    sum = d;
}
```

```
    rev = false;
    left = right = NULL;
    pull(this);
}

void push(Treap *t){
    if(!t) return;
    if(t->rev){
        swap(t->left, t->right);
        if(t->left) t->left->rev ^= 1;
        if(t->right) t->right->rev ^= 1;
        t->rev = 0;
    }
    pull(t);
}

pair<Treap*, Treap*> split(Treap *t, int k){
    if(!t) return {NULL, NULL};
    push(t);
    if(size(t->left) >= k){
        auto [lTreap, rTreap] = split(t->left, k);
        t->left = rTreap;
        pull(t);
        return {lTreap, t};
    } else {
        k = k - size(t->left) - 1;
        auto [lTreap, rTreap] = split(t->right, k);
        t->right = lTreap;
        pull(t);
        return {t, rTreap};
    }
}

return {NULL, NULL};
}

Treap* merge(Treap *l, Treap *r){
    if(!l) return r;
    if(!r) return l;
    push(l);push(r);
    if(l->priority < r->priority){
        l->right = merge(l->right, r);
        pull(l);
        return l;
    } else{
        r->left = merge(l, r->left);
        pull(r);
        return r;
    }
}
```

## Numerical (4)

### 4.1 Matrices

#### Determinant.h

**Description:** Calculates determinant of a matrix. Destroys the matrix.  
**Time:**  $\mathcal{O}(N^3)$

```
bd5cec, 15 lines

double det(vector<vector<double>>& a) {
    int n = sz(a); double res = 1;
    rep(i,0,n) {
        int b = i;
        rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
        if (i != b) swap(a[i], a[b]), res *= -1;
        res *= a[i][i];
        if (res == 0) return 0;
        rep(j,i+1,n) {
            double v = a[j][i] / a[i][i];
            if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
        }
    }
}
```

```
    return res;
}
```

#### IntDeterminant.h

**Description:** Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.  
**Time:**  $\mathcal{O}(N^3)$

```
3313dc, 18 lines

const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }
    return (ans + mod) % mod;
}
```

#### SolveLinear.h

**Description:** Solves  $A * x = b$ . If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in  $A$  and  $b$  is lost.  
**Time:**  $\mathcal{O}(n^2m)$

```
44c9ab, 38 lines

typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
    int n = sz(A), m = sz(x), rank = 0, br, bc;
    if (n) assert(sz(A[0]) == m);
    vi col(m); iota(all(col), 0);

    rep(i,0,n) {
        double v, bv = 0;
        rep(r,i,n) rep(c,i,m)
            if ((v = fabs(A[r][c])) > bv)
                br = r, bc = c, bv = v;
        if (bv <= eps) {
            rep(j,i,n) if (fabs(b[j]) > eps) return -1;
            break;
        }
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) swap(A[j][i], A[j][bc]);
        bv = 1/A[i][i];
        rep(j,i+1,n) {
            double fac = A[j][i] * bv;
            b[j] -= fac * b[i];
            rep(k,i+1,m) A[j][k] -= fac*A[i][k];
        }
        rank++;
    }

    x.assign(m, 0);
    for (int i = rank; i--;) {
        b[i] /= A[i][i];
        x[col[i]] = b[i];
        rep(j,0,i) b[j] -= A[j][i] * b[i];
    }
    return rank; // (multiple solutions if rank < m)
}
```

```

}

SolveLinear2.h
Description: To get all uniquely determined values of  $x$  back from SolveLinear, make the following changes:

"SolveLinear.h"                                08e495, 7 lines

rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
    rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
    x[col[i]] = b[i] / A[i][i];
fail:; }

SolveLinearBinary.h
Description: Solves  $Ax = b$  over  $\mathbb{F}_2$ . If there are multiple solutions, one is returned arbitrarily. Returns rank, or -1 if no solutions. Destroys  $A$  and  $b$ .  
Time:  $\mathcal{O}(n^2m)$ 
```

```
fa2d7a, 34 lines

typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);
    rep(i,0,n) {
        for (br=i; br<n; ++br) if (A[br].any()) break;
        if (br == n) {
            rep(j,i,n) if(b[j]) return -1;
            break;
        }
        int bc = (int)A[br]._Find_next(i-1);
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) if (A[j][i] != A[j][bc]) {
            A[j].flip(i); A[j].flip(bc);
        }
        rep(j,i+1,n) if (A[j][i]) {
            b[j] ^= b[i];
            A[j] ^= A[i];
        }
        rank++;
    }

    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
        x[col[i]] = 1;
        rep(j,0,i) b[j] ^= A[j][i];
    }
    return rank; // (multiple solutions if rank < m)
}
```

#### MatrixInverse.h

**Description:** Invert matrix  $A$ . Returns rank; result is stored in  $A$  unless singular ( $\text{rank} < n$ ). Can easily be extended to prime moduli; for prime powers, repeatedly set  $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$  where  $A^{-1}$  starts as the inverse of  $A$  mod  $p$ , and  $k$  is doubled in each step.  
**Time:**  $\mathcal{O}(n^3)$

```
ebffff, 35 lines

int matInv(vector<vector<double>>& A) {
    int n = sz(A); vi col(n);
    vector<vector<double>> tmp(n, vector<double>(n));
    rep(i,0,n) tmp[i][i] = 1, col[i] = i;

    rep(i,0,n) {
        int r = i, c = i;
        rep(j,i,n) rep(k,i,n)
```

```
        if (fabs(A[j][k]) > fabs(A[r][c]))
            r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
        swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
        double f = A[j][i] / v;
        A[j][i] = 0;
        rep(k,i+1,n) A[j][k] -= f*A[i][k];
        rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
}

for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
}

rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
return n;
}
```

## 4.2 Fourier transforms

**FastFourierTransform.h**  
**Description:**  $\text{fft}(a)$  computes  $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$  for all  $k$ .  $N$  must be a power of 2. Useful for convolution:  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by  $n$ , reverse(start+1, end), FFT back. Rounding is safe if  $(\sum a^2 + \sum b^2) \log_2 N < 9 \cdot 10^{14}$  (in practice  $10^{16}$ ; higher for random inputs). Otherwise, use NTT/FFTMod.  
**Time:**  $\mathcal{O}(N \log N)$  with  $N = |A| + |B|$  ( $\sim 1s$  for  $N = 2^{22}$ )

|  |                  |
|--|------------------|
| <pre>typedef complex&lt;double&gt; C; typedef vector&lt;double&gt; vd; void fft(vector&lt;C&gt;&amp; a) {     int n = sz(a), L = 31 - __builtin_clz(n);     static vector&lt;complex&lt;long double&gt;&gt; R(2, 1);     static vector&lt;C&gt; rt(2, 1); // (^ 10% faster if double)     for (static int k = 2; k &lt; n; k *= 2) {         R.resize(n); rt.resize(n);         auto x = polar(1.0L, acos(-1.0L) / k);         rep(i,k,2*k) rt[i] = R[i] = i&amp;1 ? R[i/2] * x : R[i/2];     }     vi rev(n);     rep(i,0,n) rev[i] = (rev[i / 2]   (i &amp; 1) &lt;&lt; L) / 2;     rep(i,0,n) if (i &lt; rev[i]) swap(a[i], a[rev[i]]);     for (int k = 1; k &lt; n; k *= 2)         for (int i = 0; i &lt; n; i += 2 * k) rep(j,0,k) {             C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)             a[i + j + k] = a[i + j] - z;             a[i + j] += z;         } }  vd conv(const vd&amp; a, const vd&amp; b) {     if (a.empty()    b.empty()) return {};     vd res(sz(a) + sz(b) - 1);     int L = 32 - __builtin_clz(sz(res)), n = 1 &lt;&lt; L;     vector&lt;C&gt; in(n), out(n);     copy(all(a), begin(in));     rep(i,0,sz(b)) in[i].imag(b[i]);     fft(in);     for (C&amp; x : in) x *= x;     rep(i,0,n) out[i] = in[-i &amp; (n - 1)] - conj(in[i]); }</pre> | 00ced6, 35 lines |
|--|------------------|

```
fft(out);
rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
return res;
}
```

**NumberTheoreticTransform.h**  
**Description:**  $\text{ntt}(a)$  computes  $\hat{f}(k) = \sum_x a[x]g^{xk}$  for all  $k$ , where  $g = \text{root}^{(\text{mod}-1)/N}$ .  $N$  must be a power of 2. Useful for convolution modulo specific nice primes of the form  $2^ab + 1$ , where the convolution result has size at most  $2^a$ . For arbitrary modulo, see FFTMod.  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x-i]$ . For manual convolution: NTT the inputs, multiply pointwise, divide by  $n$ , reverse(start+1, end), NTT back. Inputs must be in  $[0, \text{mod})$ .  
**Time:**  $\mathcal{O}(N \log N)$

|   |                  |
|---|------------------|
| <pre>"../number-theory/ModPow.h" const ll mod = (119 &lt;&lt; 23) + 1, root = 62; // = 998244353 // For p &lt; 2^30 there is also e.g. 5 &lt;&lt; 25, 7 &lt;&lt; 26, 479 &lt;&lt; 21 // and 483 &lt;&lt; 21 (same root). The last two are &gt; 10^9.  void ntt(vll &amp;a) {     int n = sz(a), L = 31 - __builtin_clz(n);     static vll rt(2, 1);     for (static int k = 2, s = 2; k &lt; n; k *= 2, s++) {         rt.resize(n);         ll z[] = {1, modpow(root, mod &gt;&gt; s)};         rep(i,k,2*k) rt[i] = rt[i / 2] * z[i &amp; 1] % mod;     }     vi rev(n);     rep(i,0,n) rev[i] = (rev[i / 2]   (i &amp; 1) &lt;&lt; L) / 2;     rep(i,0,n) if (i &lt; rev[i]) swap(a[i], a[rev[i]]);     for (int k = 1; k &lt; n; k *= 2)         for (int i = 0; i &lt; n; i += 2 * k) rep(j,0,k) {             ll z = rt[j + k] * a[i + j + k] % mod, &amp;ai = a[i + j];             a[i + j + k] = ai - z + (z &gt; ai ? mod : 0);             ai += (ai + z &gt;= mod ? z - mod : z);         } }  vll conv(const vll &amp;a, const vll &amp;b) {     if (a.empty()    b.empty()) return {};     int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),         n = 1 &lt;&lt; B;     int inv = modpow(n, mod - 2);     vll L(a), R(b), out(n);     L.resize(n), R.resize(n);     ntt(L), ntt(R);     rep(i,0,n)         out[-i &amp; (n - 1)] = (ll)L[i] * R[i] % mod * inv % mod;     ntt(out);     return {out.begin(), out.begin() + s}; }</pre> | af8353, 35 lines |
|---|------------------|

## Number theory (5)

### 5.1 Modular arithmetic

|  |                 |
|--|-----------------|
| <pre>ModPow.h  const ll mod = 1000000007; // faster if const  ll modpow(ll b, ll e) {     ll ans = 1;     for (; e; b = b * b % mod, e /= 2)         if (e &amp; 1) ans = ans * b % mod;     return ans; }</pre> | b83e45, 8 lines |
|--|-----------------|

## 5.2 Primes

**linearSieve.h**  
**Description:** Finds all primes in  $\mathcal{O}(N)$ , computes any multiplicative function

|  |                  |
|--|------------------|
| <pre>#define maxsz 10000005 int lp[maxsz]; vi pr;  void sieve() {     for(int i = 0 ; i &lt; maxsz ; i++) lp[i] = i;     lp[0] = 0;     lp[1] = 0;     for(int i = 2 ; i &lt; maxsz ; i++)     {         if(lp[i] == i) {             pr.pb(i);         }         for(int j = 0 ; (i*pr[j] &lt; maxsz) ; j++)         {             lp[i*pr[j]] = pr[j];             if((i%pr[j]) == 0) {                 break;             }         }     } }</pre> | ec3c00, 23 lines |
|--|------------------|

## 5.3 Divisibility

**euclid.h**  
**Description:** Finds two integers  $x$  and  $y$ , such that  $ax + by = \text{gcd}(a, b)$ . If you just need gcd, use the built in `__gcd` instead. If  $a$  and  $b$  are coprime, then  $x$  is the inverse of  $a \pmod b$ .

|   |                  |
|---|------------------|
| <pre>ll gcd(ll a, ll b, ll&amp; x, ll&amp; y) {     if (b == 0) {         x = 1;         y = 0;         return a;     }     ll x1, y1;     ll d = gcd(b, a % b, x1, y1);     x = y1;     y = x1 - y1 * (a / b);     return d; }</pre> | 31d390, 12 lines |
|---|------------------|

**linearDiophantine.h**  
**Description:** Finds all solutions to two integers  $x$  and  $y$ , such that  $ax + by = c$ .

|  |                  |
|--|------------------|
| <pre>bool find_any_solution(ll a, ll b, ll c, ll &amp;x0, ll &amp;y0, ll &amp;g) {     g = gcd(abs(a), abs(b), x0, y0);     if (c % g) {         return false;     }     x0 *= c / g;     y0 *= c / g;     if (a &lt; 0) x0 = -x0;     if (b &lt; 0) y0 = -y0;     return true; }  void shift_solution(ll &amp; x, ll &amp; y, ll a, ll b, ll cnt) {     x += cnt * b;     y -= cnt * a; }</pre> | 6dc8eb, 61 lines |
|--|------------------|



```
11 find_all_solutions(11 a, 11 b, 11 c, 11 minx, 11 maxx, 11
    miny, 11 maxy) {

    11 x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    11 sign_a = a > 0 ? +1 : -1;
    11 sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    11 lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    11 rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    11 lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    11 rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    11 lx = max(lx1, lx2);
    11 rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

**CRT.h**  
**Description:** Chinese Remainder Theorem.  
crt(a, m, b, n) computes  $x$  such that  $x \equiv a \pmod m, x \equiv b \pmod n$ . If  $|a| < m$  and  $|b| < n$ ,  $x$  will obey  $0 \leq x < \text{lcm}(m, n)$ . Assumes  $mn < 2^{62}$ .  
**Time:**  $\log(n)$

"euclid.h" 35ac10, 7 lines

```
11 crt(11 a, 11 m, 11 b, 11 n) {
    if (n > m) swap(a, b), swap(m, n);
    11 x, y, g = gcd(m, n, x, y);
    assert((a - b) % g == 0); // else no solution
    x = (b - a) % n * x % n / g * m + a;
    return x % m;
```

3.1.1 Bézout's identity

For  $a \neq 0, b \neq 0$ , then  $d = \gcd(a, b)$  is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If  $(x, y)$  is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

**phiFunction.h**  
**Description:** Euler's  $\phi$  function is defined as  $\phi(n) := \#$  of positive integers  $\leq n$  that are coprime with  $n$ .  $\phi(1) = 1, p$  prime  $\Rightarrow \phi(p^k) = (p - 1)p^{k-1}, m, n$  coprime  $\Rightarrow \phi(mn) = \phi(m)\phi(n)$ . If  $n = p_1^{k_1}p_2^{k_2}...p_r^{k_r}$  then  $\phi(n) = (p_1 - 1)p_1^{k_1-1}...(p_r - 1)p_r^{k_r-1}$ .  $\phi(n) = n \cdot \prod_{p|n} (1 - 1/p)$ .  
 $\sum_{d|n} \phi(d) = n, \sum_{1 \leq k \leq n, \gcd(k, n) = 1} k = n\phi(n)/2, n > 1$   
**Euler's thm:**  $a, n$  coprime  $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$ .  
**Fermat's little thm:**  $p$  prime  $\Rightarrow a^{p-1} \equiv 1 \pmod p \ \forall a$ .

cf7d6d, 8 lines

```
const int LIM = 50000000;
int phi[LIM];

void calculatePhi() {
    rep(i, 0, LIM) phi[i] = i&1 ? i : i/2;
    for (int i = 3; i < LIM; i += 2) if (phi[i] == i)
        for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

5.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \quad b = k \cdot (2mn), \quad c = k \cdot (m^2 + n^2),$$

with  $m > n > 0, k > 0, m \perp n$ , and either  $m$  or  $n$  even.

5.5 Primes

$p = 962592769$  is such that  $2^{21} \mid p - 1$ , which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power  $p^a$ , except for  $p = 2, a > 2$ , and there are  $\phi(\phi(p^a))$  many. For  $p = 2, a > 2$ , the group  $\mathbb{Z}_{2^a}^\times$  is instead isomorphic to  $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$ .

5.6 Estimates

$$\sum_{d|n} d = O(n \log \log n).$$

The number of divisors of  $n$  is at most around 100 for  $n < 5e4$ , 500 for  $n < 1e7$ , 2000 for  $n < 1e10$ , 200 000 for  $n < 1e19$ .

5.7 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d)g(n/d)$$

Other useful formulas/forms:

$$\sum_{d|n} \mu(d) = [n = 1] \text{ (very useful)}$$

$$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n)g(d)$$

$$g(n) = \sum_{1 \leq m \leq n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \leq m \leq n} \mu(m)g(\lfloor \frac{n}{m} \rfloor)$$

Combinatorial (6)

6.1 Templates

**power.h**  
**Description:** A Power B  
**Time:**  $\mathcal{O}(\log(n))$

0ac6cf, 19 lines

```
11 mod = 1e9 + 7;
int sz = 1000005;
11 fact[1000005];
11 ifact[1000005];

11 power(11 x, 11 n) {
    if (n==0) return 1;
    x = x%mod;
    if (x==0) return 0;
    11 pow = 1;
    while (n) {
        if (n & 1) pow = (pow*x)%mod;
        n = n >> 1;
        x = (x*x)%mod;
    }
    return pow;
}

11 inv_mod(11 x) { return power(x , mod - 2)%mod; }
```

**ncr.h**  
**Description:** ncr, precomputation of factorials  
**Time:**  $\mathcal{O}(n)$

7e37cb, 15 lines

```
void factorial() {
    fact[0] = fact[1] = 1;
    ifact[0] = ifact[1] = 1;
    for (int i = 2 ; i < sz ; i++)
        fact[i] = (fact[i-1]*i)%mod;
    ifact[sz-1] = inv_mod(fact[sz-1]);
    for (int i = sz-2 ; i > 0 ; i--)
        ifact[i] = (ifact[i+1]*(i+1))%mod;
}

11 ncr(11 n , 11 r) {
    if (n<r || r<0) return 0;
    if (r == 0) return 1;
    return (((fact[n]*ifact[n-r])%mod)*ifact[r])%mod;
}
```

6.2 Permutations

6.2.1 Cycles

Let  $g_S(n)$  be the number of  $n$ -permutations whose cycle lengths all belong to the set  $S$ . Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp \left( \sum_{n \in S} \frac{x^n}{n} \right)$$

6.2.2 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rfloor$$

6.2.3 Burnside’s lemma

Given a group  $G$  of symmetries and a set  $X$ , the number of elements of  $X$  up to symmetry equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where  $X^g$  are the elements fixed by  $g$  ( $g.x = x$ ).

If  $f(n)$  counts “configurations” (of some sort) of length  $n$ , we can ignore rotational symmetry using  $G = \mathbb{Z}_n$  to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k) \phi(n/k).$$

6.3 Partitions and subsets

6.3.1 Partition function

Number of ways of writing  $n$  as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1, \quad p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k - 1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

|        |   |   |   |   |   |   |    |    |    |    |     |            |            |
|--------|---|---|---|---|---|---|----|----|----|----|-----|------------|------------|
| $n$    | 0 | 1 | 2 | 3 | 4 | 5 | 6  | 7  | 8  | 9  | 20  | 50         | 100        |
| $p(n)$ | 1 | 1 | 2 | 3 | 5 | 7 | 11 | 15 | 22 | 30 | 627 | $\sim 2e5$ | $\sim 2e8$ |

6.3.2 Lucas’ Theorem

Let  $n, m$  be non-negative integers and  $p$  a prime. Write  $n = n_k p^k + \dots + n_1 p + n_0$  and  $m = m_k p^k + \dots + m_1 p + m_0$ . Then  $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod p$ .

6.3.3 Binomials

multinomial.h

**Description:** Computes  $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \dots k_n!}$ . a0a312, 5 lines

```
ll multinomial(vi& v) {
    ll c = 1, m = v.empty() ? 1 : v[0];
    rep(i, 1, sz(v)) rep(j, 0, v[i]) c = c * ++m / (j+1);
    return c;
}
```

6.4 General purpose numbers

6.4.1 Bernoulli numbers

EGF of Bernoulli numbers is  $B(t) = \frac{t}{e^t - 1}$  (FFT-able).  
 $B[0, \dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$

Sums of powers:

$$\sum_{i=1}^n n^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\begin{aligned} \sum_{i=m}^\infty f(i) &= \int_m^\infty f(x) dx - \sum_{k=1}^\infty \frac{B_k}{k!} f^{(k-1)}(m) \\ &\approx \int_m^\infty f(x) dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m)) \end{aligned}$$

6.4.2 Stirling numbers of the first kind

Number of permutations on  $n$  items with  $k$  cycles.

$$\begin{aligned} c(n, k) &= c(n-1, k-1) + (n-1)c(n-1, k), \quad c(0, 0) = 1 \\ \sum_{k=0}^n c(n, k) x^k &= x(x+1) \dots (x+n-1) \end{aligned}$$

$$\begin{aligned} c(8, k) &= 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1 \\ c(n, 2) &= 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \dots \end{aligned}$$

6.4.3 Eulerian numbers

Number of permutations  $\pi \in S_n$  in which exactly  $k$  elements are greater than the previous element.  $k$   $j$ :s s.t.  $\pi(j) > \pi(j+1)$ ,  $k+1$   $j$ :s s.t.  $\pi(j) \geq j$ ,  $k$   $j$ :s s.t.  $\pi(j) > j$ .

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$$

6.4.4 Stirling numbers of the second kind

Partitions of  $n$  distinct elements into exactly  $k$  groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k)$$

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$

6.4.5 Bell numbers

Total number of partitions of  $n$  distinct elements.  $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$  For  $p$  prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod p$$

6.4.6 Labeled unrooted trees

# on  $n$  vertices:  $n^{n-2}$   
# on  $k$  existing trees of size  $n_i$ :  $n_1 n_2 \dots n_k n^{k-2}$   
# with degrees  $d_i$ :  $(n-2)! / ((d_1-1)! \dots (d_n-1)!)$

6.4.7 Catalan numbers

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \quad C_{n+1} = \sum C_i C_{n-i}$$

$$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$$

- sub-diagonal monotone paths in an  $n \times n$  grid.
- strings with  $n$  pairs of parenthesis, correctly nested.
- binary trees with with  $n+1$  leaves (0 or 2 children).
- ordered trees with  $n+1$  vertices.
- ways a convex polygon with  $n+2$  sides can be cut into triangles by connecting vertices with straight lines.
- permutations of  $[n]$  with no 3-term increasing subseq.

Graph (7)

7.1 Network flow

MinCostMaxFlow.h  
**Description:** Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.  
**Time:**  $\mathcal{O}(FE \log(V))$  where F is max flow.  $\mathcal{O}(VE)$  for setpi. 58385b, 78 lines

```
#include <bits/extc++.h>
const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
    struct edge {
        int from, to, rev;
        ll cap, cost, flow;
    };
    int N;
    vector<vector<edge>> ed;
    vi seen;
    vector<ll> dist, pi;
    vector<edge*> par;

    MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}

    void addEdge(int from, int to, ll cap, ll cost) {
        if (from == to) return;
        ed[from].push_back(edge{ from, to, sz(ed[to]), cap, cost, 0 });
        ed[to].push_back(edge{ to, from, sz(ed[from])-1, 0, -cost, 0 });
    }

    void path(int s) {
        fill(all(seen), 0);
        fill(all(dist), INF);
        dist[s] = 0; ll di;

        __gnu_pbds::priority_queue<pair<ll, int>> q;
        vector<decltype(q)::point_iterator> its(N);
        q.push({ 0, s });

        while (!q.empty()) {
            s = q.top().second; q.pop();
            seen[s] = 1; di = dist[s] + pi[s];
            for (edge& e : ed[s]) if (!seen[e.to]) {
                ll val = di - pi[e.to] + e.cost;
                if (e.cap - e.flow > 0 && val < dist[e.to]) {
```

```
dist[e.to] = val;
par[e.to] = &e;
if (its[e.to] == q.end())
    its[e.to] = q.push({ -dist[e.to], e.to });
else
    q.modify(its[e.to], { -dist[e.to], e.to });
}
}
rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
}

pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
        ll fl = INF;
        for (edge* x = par[t]; x; x = par[x->from])
            fl = min(fl, x->cap - x->flow);

        totflow += fl;
        for (edge* x = par[t]; x; x = par[x->from]) {
            x->flow += fl;
            ed[x->to][x->rev].flow -= fl;
        }
        rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
        return {totflow, totcost/2};
    }

    // If some costs can be negative, call this before maxflow:
    void setpi(int s) { // (otherwise, leave this out)
        fill(all(pi), INF); pi[s] = 0;
        int it = N, ch = 1; ll v;
        while (ch-- && it--)
            rep(i,0,N) if (pi[i] != INF)
                for (edge& e : ed[i]) if (e.cap)
                    if ((v = pi[i] + e.cost) < pi[e.to])
                        pi[e.to] = v, ch = 1;
        assert(it >= 0); // negative cost cycle
    }
};
```

Dinic.h  
Description: Flow algorithm with complexity  $O(VE\log U)$  where  $U = \max|\text{cap}|$ .  $O(\min(E^{1/2}, V^{2/3})E)$  if  $U = 1$ ;  $O(\sqrt{V}E)$  for bipartite matching.

fb1d5e, 81 lines

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }
};
```

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

bool bfs() {
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int id : adj[v]) {
            if (edges[id].cap == edges[id].flow)
                continue;
            if (level[edges[id].u] != -1)
                continue;
            level[edges[id].u] = level[v] + 1;
            q.push(edges[id].u);
        }
    }
    return level[t] != -1;
}

long long dfs(int v, long long pushed) {
    if (pushed == 0)
        return 0;
    if (v == t)
        return pushed;
    for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
        int id = adj[v][cid];
        int u = edges[id].u;
        if (level[v] + 1 != level[u])
            continue;
        long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
        if (tr == 0)
            continue;
        edges[id].flow += tr;
        edges[id ^ 1].flow -= tr;
        return tr;
    }
    return 0;
}

long long flow() {
    long long f = 0;
    while (true) {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        q.push(s);
        if (!bfs())
            break;
        fill(ptr.begin(), ptr.end(), 0);
        while (long long pushed = dfs(s, flow_inf)) {
            f += pushed;
        }
    }
    return f;
};
```

7.2 Matching

Kuhn.h  
Description:  $O(N^2M)$  maximal edge matching  
// maximal matching only bipartite  
// first group n, second group k

e89f65, 32 lines

```
// graph g[firstGrp].pushback(secondGrp)

int n, k;
vector<int> g[200005];
vector<bool> used;
vector<int> mt;

bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

// inside main function
mt.assign(k, -1);
for (int v = 0; v < n; ++v) {
    used.assign(n, false);
    try_kuhn(v);
}

for (int i = 0; i < k; ++i)
    if (mt[i] != -1)
        printf("%d %d\n", mt[i] + 1, i + 1);

Hungarian.h
Description:  $O(N^3)$  Task assignment minimum cost
vector<ll> u (n+1), v (m+1), p (m+1), way (m+1);
for (ll i=1; i<=n; ++i) {
    p[0] = i;
    ll j0 = 0;
    vector<ll> minv (m+1, inf);
    vector<bool> used (m+1, false);
    do {
        used[j0] = true;
        ll i0 = p[j0], delta = inf, j1;
        for (ll j=1; j<=m; ++j)
            if (!used[j]) {
                ll cur = A[i0][j]-u[i0]-v[j];
                if (cur < minv[j])
                    minv[j] = cur, way[j] = j0;
                if (minv[j] < delta)
                    delta = minv[j], j1 = j;
            }
        for (ll j=0; j<=m; ++j)
            if (used[j])
                u[p[j]] += delta, v[j] -= delta;
            else
                minv[j] -= delta;
        j0 = j1;
    } while (p[j0] != 0);
    do {
        ll j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    } while (j0);
}

ll cost = -v[0];
vector<ll> ans (n+1);
for (ll j=1; j<=m; ++j)
    ans[p[j]] = j;
```

667e79, 35 lines

7.3 DFS algorithms

SCC.h

**Description:** Finds strongly connected components in a directed graph. If vertices  $u, v$  belong to the same component, we can reach  $u$  from  $v$  and vice versa.  
**Usage:** scc(graph, [&](vi& v) { ... }) visits all components in reverse topological order. comp[i] holds the component index of a node (a component only has edges to components with lower index). ncomps will contain the number of components.  
**Time:**  $\mathcal{O}(E + V)$

5a2d60, 61 lines

```
vector<bool> visited;
// keeps track of which vertices are already visited
// runs depth first search starting at vertex v.
// each visited vertex is appended to the output vector when
// dfs leaves it.
void dfs(int v, vector<vector<int>> const& adj, vector<int> &
output) {
    visited[v] = true;
    for (auto u : adj[v])
        if (!visited[u])
            dfs(u, adj, output);
    output.push_back(v);
}

// input: adj — adjacency list of G
// output: components — the strongly connected components in G
// output: adj_cond — adjacency list of G^SCC (by root
//          vertices)
void strongly_connected_components(vector<vector<int>> const&
adj,
                                vector<vector<int>> &
                                components,
                                vector<vector<int>> &adj_cond
                                ) {

    int n = adj.size();
    components.clear(), adj_cond.clear();

    vector<int> order;
    // will be a sorted list of G's vertices by exit time

    visited.assign(n, false);

    // first series of depth first searches
    for (int i = 0; i < n; i++)
        if (!visited[i])
            dfs(i, adj, order);

    // create adjacency list of G^T
    vector<vector<int>> adj_rev(n);
    for (int v = 0; v < n; v++)
        for (int u : adj[v])
            adj_rev[u].push_back(v);

    visited.assign(n, false);
    reverse(order.begin(), order.end());

    vector<int> roots(n, 0);
    // gives the root vertex of a vertex's SCC

    // second series of depth first searches
    for (auto v : order)
        if (!visited[v]) {
            std::vector<int> component;
            dfs(v, adj_rev, component);
            components.push_back(component);
            int root = *min_element(begin(component), end(
                component));
            for (auto u : component)
```

```
                roots[u] = root;
        }

    // add edges to condensation graph
    adj_cond.assign(n, {});
    for (int v = 0; v < n; v++)
        for (auto u : adj[v])
            if (roots[v] != roots[u])
                adj_cond[roots[v]].push_back(roots[u]);
}
```

2sat.h

**Description:** Calculates a valid assignment to boolean variables  $a, b, c, \dots$  to a 2-SAT problem, so that an expression of the type  $(a||b)\&\&(a||c)\&\&(d||b)\&\&\dots$  becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions ( $\sim x$ ).  
**Usage:** TwoSat ts(number of boolean variables);  
ts.either(0, ~3); // Var 0 is true or var 3 is false  
ts.setValue(2); // Var 2 is true  
ts.atMostOne({0, ~1, 2}); //  $\leq 1$  of vars 0, ~1 and 2 are true  
ts.solve(); // Returns true iff it is solvable  
ts.values[0..N-1] holds the assigned values to the vars  
**Time:**  $\mathcal{O}(N + E)$ , where  $N$  is the number of boolean variables, and  $E$  is the number of clauses.

9ee997, 65 lines

```
// 0-based indexing
struct TwoSatSolver {
    int n_vars;
    int n_vertices;
    vector<vector<int>> adj, adj_t;
    vector<bool> used;
    vector<int> order, comp;
    vector<bool> assignment;

    TwoSatSolver(int _n_vars) : n_vars(_n_vars), n_vertices(2 *
        n_vars), adj(n_vertices), adj_t(n_vertices), used(
            n_vertices), order(), comp(n_vertices, -1), assignment
            (n_vars) {
        order.reserve(n_vertices);
    }
    void dfs1(int v) {
        used[v] = true;
        for (int u : adj[v]) {
            if (!used[u])
                dfs1(u);
        }
        order.push_back(v);
    }

    void dfs2(int v, int cl) {
        comp[v] = cl;
        for (int u : adj_t[v]) {
            if (comp[u] == -1)
                dfs2(u, cl);
        }
    }

    bool solve_2SAT() {
        order.clear();
        used.assign(n_vertices, false);
        for (int i = 0; i < n_vertices; ++i) {
            if (!used[i])
                dfs1(i);
        }

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

```
                dfs2(v, j++);
        }

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

    void add_disjunction(int a, bool na, int b, bool nb) {
        // na and nb signify whether a and b are to be negated
        a = 2 * a ^ na;
        b = 2 * b ^ nb;
        int neg_a = a ^ 1;
        int neg_b = b ^ 1;
        adj[neg_a].push_back(b);
        adj[neg_b].push_back(a);
        adj_t[b].push_back(neg_a);
        adj_t[a].push_back(neg_b);
    }
};
```

7.4 Trees

LCA.h

**Description:** Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.  
**Time:**  $\mathcal{O}(N \log N + Q)$

3afd94, 56 lines

```
//nodes are 1-based indexing
//initialize all parents to 0.

#define maxsz 200005
#define logmax 18

ll parent[maxsz];
ll level[maxsz];
ll memo[maxsz][logmax];

void preprocess(ll n)
{
    for(ll i = 0 ; i < logmax ; i ++){
        for(ll j = 0 ; j <= n ; j ++){
            if(i == 0)
            {
                memo[j][i] = parent[j];
            }
            else{
                memo[j][i] = memo[memo[j][i-1]][i-1];
            }
        }
    }
}

int lca(ll u , ll v)
{
    if(level[u] > level[v]) {
        swap(u , v);
    }

    for(int i = logmax-1 ; i>=0 ; i--){
        if(level[v] - (1ll << i) >= level[u])
        {
            v = memo[v][i];
        }
```

```
    }
}

for(int i = logmax-1 ; i >= 0 ; i --)
{
    if(memo[u][i] != memo[v][i])
    {
        u = memo[u][i];
        v = memo[v][i];
    }
}

if(u != v)
{
    u = memo[u][0];
    v = memo[v][0];
}
return u;
}
```

**HLD.h**  
**Description:** Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most  $\log(n)$  light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS\_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.  
**Time:**  $\mathcal{O}((\log N)^2)$

4f19f5, 119 lines

```
struct HLD {
    int n;
    vector<int> siz, top, dep, par, in, out, seq;
    vector<vector<int>>> adj;
    int timer;

    HLD() {}
    HLD(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        par.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        timer = -1;
        adj.assign(n, {});
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void work(int root = 0) {
        top[root] = root;
        dep[root] = 0;
        par[root] = -1;
        dfs1(root);
        dfs2(root);
    }

    void dfs1(int u) {
        if (par[u] != -1) {
            adj[u].erase(find(adj[u].begin(), adj[u].end(), par[u]));
        }

        siz[u] = 1;
```

```
        for (auto &v : adj[u]) {
            par[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                swap(v, adj[u][0]);
            }
        }
    }

    void dfs2(int u) {
        in[u] = ++timer;
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = timer;
    }

    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = par[top[u]];
            } else {
                v = par[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }

    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }

    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }

        int d = dep[u] - k;

        while (dep[top[u]] > d) {
            u = par[top[u]];
        }

        return seq[in[u] - dep[u] + d];
    }

    bool isAncestor(int u, int v) {
        return in[u] <= in[v] && in[v] <= out[u];
    }

    int rootedParent(int u, int v) {
        swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncestor(u, v)) {
            return par[u];
        }
        auto it = upper_bound(adj[u].begin(), adj[u].end(), v, [&] (
            int x, int y) {
                return in[x] < in[y];
            }) - 1;
        return *it;
    }

    int rootedSize(int u, int v) {
        if (u == v) {
```

```
            return n;
        }
        if (!isAncestor(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }

    int rootedLca(int a, int b, int c) {
        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
    }
};
```

**centroidDecomposition.h**  
**Description:** Centroid Decomposition. Perform  $\mathcal{O}(N)$  algorithms like DFS only on *is\_removed* = *False*  
**Time:**  $\mathcal{O}(V \log V)$

1e2d42, 35 lines

```
vector<int> adjlst[100005];
bool is_removed[100005];
int subtree_size[100005];

int get_subtree_size(int node, int parent = -1) {
    subtree_size[node] = 1;
    for (int child : adjlst[node]) {
        if (child == parent || is_removed[child]) { continue; }
        subtree_size[node] += get_subtree_size(child, node);
    }
    return subtree_size[node];
}

int get_centroid(int node, int tree_size, int parent = -1) {
    for (int child : adjlst[node]) {
        if (child == parent || is_removed[child]) { continue; }
        if (subtree_size[child] * 2 > tree_size) {
            return get_centroid(child, tree_size, node);
        }
    }
    return node;
}

void build_centroid_decomp(int node = 0) {
    int centroid = get_centroid(node, get_subtree_size(node));

    // calculate

    is_removed[centroid] = true;

    for (int child : adjlst[centroid]) {
        if (is_removed[child]) { continue; }
        build_centroid_decomp(child);
    }
}
```

**LinkCutTree.h**  
**Description:** Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.  
**Time:** All operations take amortized  $\mathcal{O}(\log N)$ .

0fb462, 90 lines

```
struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
        // (+ update sum of subtree elements etc. if wanted)
    }
}
```

```

void pushFlip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
}
int up() { return p ? p->c[1] == this : -1; }
void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
    if ((y->p == p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
        x->c[h] = y->c[h ^ 1];
        y->c[h ^ 1] = x;
    }
    z->c[i ^ 1] = this;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
}
void splay() {
    for (pushFlip(); p; ) {
        if (p->p) p->p->pushFlip();
        p->pushFlip(); pushFlip();
        int c1 = up(), c2 = p->up();
        if (c2 == -1) p->rot(c1, 2);
        else p->p->rot(c2, c1 != c2);
    }
}
Node* first() {
    pushFlip();
    return c[0] ? c[0]->first() : (splay(), this);
}
};

struct LinkCut {
    vector<Node> node;
    LinkCut(int N) : node(N) {}

    void link(int u, int v) { // add an edge (u, v)
        assert(!connected(u, v));
        makeRoot(&node[u]);
        node[u].pp = &node[v];
    }

    void cut(int u, int v) { // remove an edge (u, v)
        Node *x = &node[u], *top = &node[v];
        makeRoot(top); x->splay();
        assert(top == (x->pp ? x->c[0]));
        if (x->pp) x->pp = 0;
        else {
            x->c[0] = top->p = 0;
            x->fix();
        }
    }

    bool connected(int u, int v) { // are u, v in the same tree?
        Node* nu = access(&node[u])->first();
        return nu == access(&node[v])->first();
    }

    void makeRoot(Node* u) {
        access(u);
        u->splay();
        if (u->c[0]) {
            u->c[0]->p = 0;
            u->c[0]->flip ^= 1;
            u->c[0]->pp = u;
            u->c[0] = 0;
            u->fix();
        }
    }
}

```

```

}
Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
        pp->splay(); u->pp = 0;
        if (pp->c[1]) {
            pp->c[1]->p = 0; pp->c[1]->pp = pp; }
        pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
}
};

```

## 7.5 Math

### 7.5.1 Number of Spanning Trees

Create an  $N \times N$  matrix  $\text{mat}$ , and for each edge  $a \rightarrow b \in G$ , do  $\text{mat}[a][b]--$ ,  $\text{mat}[b][b]++$  (and  $\text{mat}[b][a]--$ ,  $\text{mat}[a][a]++$  if  $G$  is undirected). Remove the  $i$ th row and column and take the determinant; this yields the number of directed spanning trees rooted at  $i$  (if  $G$  is undirected, remove any row/column).

### 7.5.2 Erdős–Gallai theorem

A simple graph with node degrees  $d_1 \geq \dots \geq d_n$  exists iff  $d_1 + \dots + d_n$  is even and for every  $k = 1 \dots n$ ,

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

## Geometry (8)

### 8.1 Geometric primitives

Point.h

**Description:** Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

695f29, 68 lines

```

#define point complex<lld>
#define X real()
#define Y imag()
#define PI 3.141592653589793238462

```

```

lld dot(point x, point y){ // accurate
    return (conj(x) * y).X ;
}

```

```

lld cross(point x, point y){ // accurate
    return (conj(x) * y).Y ;
}

```

```

point rotate(point x, lld angle, point p = point(0, 0)){ //
    precision
    // rotate point x w.r.t. point p with 'angle' Rad. similar
    scaling
    // default rotation is done w.r.t. origin (0, 0)
    counterclockwise
    return (x - p) * polar((lld)1.0, angle) + p ;
}

```

```

lld angle(point v, point w) { // precision
    lld cosTheta = dot(v,w) / abs(v) / abs(w);
    return acos(max((lld)-1.0, min((lld)1.0, cosTheta)));
}

```

```

}

lld orient(point a, point b, point c) { // accurate
    // orient(a, b, c) = ab X ac
    // positive if C is on left side of ab (counterclockwise of
    // ab)
    return cross(b-a, c-a);
}

```

```

bool isConvex(vector<point> p) { // accurate
    // check if polygon is convex, points in the order of
    // indices
    bool hasPos=false, hasNeg=false;
    for (int i=0, n=p.size(); i<n; i++) {
        lld o = orient(p[i], p[(i+1)%n], p[(i+2)%n]);
        if (o > 0) hasPos = true;
        if (o < 0) hasNeg = true;
    }
    return !(hasPos && hasNeg);
}

```

```

bool half(point p) { // true if angle is in [0, PI) false if [
    PI, 2*PI)
    assert(p.X != 0 || p.Y != 0); // the argument of (0,0) is
    undefined
    return p.Y > 0 || (p.Y == 0 && p.X > 0);
}

```

```

void polarSort(vector<point> &v) { // accurate
    // sorts point in increasing order of their angle from X axis
    counter clockwise
    sort(v.begin(), v.end(), [&](point x, point y){
        return make_tuple(!half(x), 0) < make_tuple(!half(y),
            cross(x, y));
    });
}

```

```

point intersectionLine(point a, point b, point p, point q) { //
    precision
    // finds intersection of infinite lines AB and PQ
    lld c1 = cross(p - a, b - a), c2 = cross(q - a, b - a);
    return (c1 * q - c2 * p) / (c1 - c2); // undefined if
    parallel
}

```

```

point project(point p, point v){ // precision
    // Project p onto vector v (line ov)
    return v * dot(p, v) / norm(v);
}

```

```

point reflect(point p, point a, point b){ // precision
    // reflect point p across line passing through ab
    return a + conj((p - a) / (b - a)) * (b - a);
}

```

### lineDistance.h

**Description:**

Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b.  $a==b$  gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

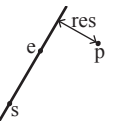
"Point.h"

f6bf6b, 4 lines

```

template<class P>
double lineDist(const P& a, const P& b, const P& p) {
    return (double) (b-a).cross(p-a) / (b-a).dist();
}

```



## 8.2 Useful Geometric Function

### usefulGeometricFunctions.h

Description:

Useful functions for geometry

"Point.h", "Line.h" a605c1, 67 lines

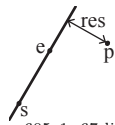
```
bool inDisk(point a, point b, point p) { // check if p lies
    inside circle with Daimeter AB
    return dot(a-p, b-p) <= 0; // accurate
}
bool onSegment(point a, point b, point p) { // check if P lies
    on AB
    return orient(a,b,p) == 0 && inDisk(a,b,p); // accurate
}
```

```
point properIntersection(point a, point b, point c, point d) {
    // precision
    // used to return the only single point of intersection btw
    AB and CD
    lld oa = orient(c,d,a),
    ob = orient(c,d,b),
    oc = orient(a,b,c),
    od = orient(a,b,d);
    // Proper intersection exists iff opposite signs
    if (oa*ob < 0 && oc*od < 0) { // accurate
        point ans = (a*ob - b*oa) / (ob-oa);
        return ans;
    }
    return point(-2e9, -2e9);
}
point intersectionLineSegment(point a, point b, point c, point
    d) {
    // finds any point of intersection between AB and CD
    // returns {-2e9, -2e9} if no point is found !!
    if (onSegment(c,d,a)) return a;
    if (onSegment(c,d,b)) return b;
    if (onSegment(a,b,c)) return c;
    if (onSegment(a,b,d)) return d;
    return properIntersection(a,b,c,d);
}
```

```
lld areaTriangle(point a, point b, point c) {
    return abs(cross(b-a, c-a)) / 2.0;
}
lld areaPolygon(vector<point> &p) { // Risky!! careful sums can
    overflow
    ll area = 0;
    for (int i = 0, n = p.size(); i < n; i++) {
        ll x = cross(p[i], p[(i+1)%n]);
        area += x;
    }

    return abs((lld)area / 2.0);
}
```

```
bool above(point a, point p) { // accurate
    // true if P at least as high as A (blue part)
    return p.Y >= a.Y;
}
bool crossesRay(point a, point p, point q) { // accurate
    // check if [PQ] crosses ray from A
    // casts a ray towards right if points are in counter-
    clockwise
```



```
// otherwise casts ray towards left
return (above(a,q) - above(a,p)) * orient(a,p,q) > 0;
}
int inPolygon(vector<point> &p, point a) { // accurate
    // -1 if pt(a) -> inside polygon, 0 if lies on a side, 1
    otherwise
    int numCrossings = 0;
    for (int i = 0, n = p.size(); i < n; i++) {
        if (onSegment(p[i], p[(i+1)%n], a))
            return 0;
        numCrossings += crossesRay(a, p[i], p[(i+1)%n]);
    }

    return (numCrossings & 1 ? -1 : 1); // inside if odd number
    of crossings
}
```

## 8.3 Polygons

### ConvexHull.h

Description:  
Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.  
Time:  $\mathcal{O}(n \log n)$

"Point.h" 5685ae, 45 lines

```
vector<point> ConvexHull(vector<point> &p){ // accurate
    // returns the convex Hull of some set of points
    ll n = p.size();
    if(n <= 2) return p;

    vector<point> up, down; // stores ans for top and bottom
    HULLS
    vector<point> active; // acts as stack
    map<pll, ll> vis; // helps avoid taking duplicates

    sort(all(p), [&](point x, point y){
        return x.X != y.X ? x.X < y.X : x.Y < y.Y;
    });

    active.push_back(p[0]);
    active.push_back(p[1]);

    for(ll i = 2; i < n; i++){
        while(active.size() > 1 and cross(active[active.size() -
            1] - active[active.size() - 2], p[i] - active[active
            .size() - 2]) > 0){
            active.pop_back();
        }
        active.push_back(p[i]);
    }

    up = active;
    active.clear();

    for(auto i: up) vis[{i.X, i.Y}] = 1;

    active.push_back(up[up.size() - 2]);
    active.push_back(up.back());

    for(ll i = n - 2; i > -1; i--){
        while(active.size() > 1 and cross(active[active.size() -
            1] - active[active.size() - 2], p[i] - active[active
            .size() - 2]) > 0){
            active.pop_back();
        }
        active.push_back(p[i]);
    }
}
```



```
down = active;

for(auto i: down) {
    if(not vis[{i.X, i.Y}]) up.push_back(i);
}
return up;
}
```

## 8.4 Misc. Point Set Problems

### ClosestPair.h

Description: Finds the closest pair of points.

Time:  $\mathcal{O}(n \log n)$

"Point.h" 42dcd2, 42 lines

```
ll ClosestPairDist(vector<point> &p, ll l, ll r){ // accurate
    // returns the square of the two closest points in range [l,
    r]
    if(l >= r) return 8e18;
    ll mid = (l + r) / 2;

    ll DisL = ClosestPairDist(p, l, mid);
    ll DisR = ClosestPairDist(p, mid + 1, r);

    ll allowD = min(DisL, DisR);

    vector<point> candidates;
    for(auto i = l; i <= r; i++){
        point v = p[i];
        ll d = v.X - p[mid].X;
        if(d * d <= allowD) candidates.push_back(v);
    }

    sort(all(candidates), [&](point x, point y){
        return x.Y != y.Y ? x.Y < y.Y : x.X < y.X;
    });

    for(ll i = 0; i < sz(candidates); i++){ // this won't be n
        ^2
        for(ll j = i + 1; j < sz(candidates); j++){
            ll dx = candidates[i].X - candidates[j].X;
            ll dy = candidates[i].Y - candidates[j].Y;

            if(dy * dy >= allowD) break; // this limits to maximum
            7 iterations
            allowD = min(allowD, dx * dx + dy * dy);
        }

        return allowD;
    }
}
ll ClosestPairDist(vector<point> &a){
    // returns the square of the two closest points
    sort(all(a), [&](point x, point y){
        return x.X == y.X ? x.Y < y.Y : x.X < y.X;
    });

    return ClosestPairDist(a, 0, a.size() - 1);
}
```

## Strings (9)

### KMP.h

Description:  $pi[x]$  computes the length of the longest prefix of  $s$  that ends at  $x$ , other than  $s[0..x]$  itself (abacaba -> 0010123). Can be used to find all occurrences of a string.

Time:  $\mathcal{O}(n)$

01ac10, 80 lines

```
template <typename T>
```

```
vector<int> lps(const T &s, int n = 1){
    n = (int)s.size();
    vector<int> lps(n);
    int j = 0;
    for(int i = 1;i < n;i++){
        while(j > 0 && s[i] != s[j]) j = lps[j-1];
        if(s[i] == s[j]) j++;
        lps[i] = j;
    }
    return lps;
}
```

```
template <typename T>
vector<int> kmp(const T &s, const T &p){
    int n = (int)s.size(), m = (int)p.size();
    vector<int> lps = lps(p);
    vector<int> ans;
    int j = 0;
    for(int i = 0;i < n;i++){
        while(j > 0 && s[i] != p[j]) j = lps[j-1];
        if(s[i] == p[j]) j++;
        if(j == m){
            ans.push_back(i-j+1);
            j = lps[j-1];
        }
    }
    return ans;
}
```

// Prefix Automation

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

```
void compute_automaton(string s, vector<vector<int>>& aut) {
    s += '#';
    int n = s.size();
    vector<int> pi = prefix_function(s);
    aut.assign(n, vector<int>(26));
    for (int i = 0; i < n; i++) {
        for (int c = 0; c < 26; c++) {
            if (i > 0 && 'a' + c != s[i])
                aut[i][c] = aut[pi[i-1]][c];
            else
                aut[i][c] = i + ('a' + c == s[i]);
        }
    }
}
```

```
// search fn for automaton
int search(string& arr,string& pattern) {
```

```
    vector<vector<int>> aut;
    compute_automaton(pattern,aut);
    int cnt = 0;
    int m = (int)pattern.size();
    int n = (int)arr.size();
    pattern += "#";
```

```
    int j = 0,i = 0;
    for(int i = 0 ; i < n ; i ++) {
        j = aut[j][arr[i] - 'a'];
        if(j == m) cnt++;
    }
    return cnt;
}
```

Zfunc.h

**Description:** z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)  
**Time:**  $\mathcal{O}(n)$

```
template <typename T>
vector<int> z(const T &s, int n = 1){
    n = (int)s.size();
    vector<int> z(n);
    int L = 0, R = 0;
    for(int i = 1;i < n;i++){
        if(i < R) z[i] = min(R-i,z[i-L]);
        while(i + z[i] < n && s[z[i]] == s[i+z[i]]) z[i]++;
        if(i + z[i] > R){
            L = i;
            R = i+z[i];
        }
    }
    return z;
}
```

Manacher.h

**Description:** For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).  
**Time:**  $\mathcal{O}(N)$

// Returns {even, odd} each value denotes how much left/right u can go from this point

```
array<vi, 2> manacher(const string& s) {
    int n = sz(s);
    array<vi,2> p = {vi(n+1), vi(n)};
    rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
        int t = r-i+!z;
        if (i<r) p[z][i] = min(t, p[z][l+t]);
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L>=1 && R+1<n && s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    }
    return p;
}
```

```
pii getLongestPal(const array<vi, 2> &p){
    int idx = 0, len = 1;
    for(int i = 0;i<sz(p[1]);i++){
        if(2*p[0][i] > len) len = 2*p[0][i], idx = i - p[0][i];
        if(2*p[1][i] + 1 > len) len = 2*p[1][i] + 1, idx = i - p[1][i];
    }
    return {idx, len};
}
bool isPal(int l, int r,const array<vi, 2> &p){
    int len = r-l+1;
    return p[len%2][l+len/2] >= len/2;
}
```

MinRotation.h

**Description:** Finds the lexicographically smallest rotation of a string.  
**Usage:** rotate(v.begin(), v.begin()+minRotation(v), v.end());  
**Time:**  $\mathcal{O}(N)$

```
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(k,0,N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
        if (s[a+k] > s[b+k]) {a = b; break;}
    }
    return a;
}
```

SuffixArray.h

**Description:** Builds suffix array for a string. sa[i] is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size n + 1, and sa[0] = n. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0. The input string must not contain any zero bytes.  
**Time:**  $\mathcal{O}(n \log n)$

```
struct sufarr {
    string s;
    vector<int>lcp,order,rank;
    int n;
    sufarr(string _s) {
        s=_s +'$';
        n=s.length();
    }
    void build() {
        order.resize(n);
        rank.resize(n);
        {
            vector<pair<int,int>>temp;
            for(int i=0;i<n;i++){
                temp.push_back({s[i]-'a',i});
            }
            sort(temp.begin(),temp.end());
            for(int i =0;i<n;i++){
                order[i]=temp[i].second;
            }
            rank[order[0]]=0;
            for(int i=1;i<n;i++){
                rank[order[i]]=rank[order[i-1]]+(temp[i].first!=temp[i-1].first);
            }
        }
    }
}
```

```
int k=0;
vector<int>order_t(n,0),rank_t(n,0);
while((1<<k)<n) {
    for(int i =0;i<n;i++){
        (order[i]==(1<<k)-n)%n;
    }
    vector<int>cnt(n,0),pos(n,0);
    for(auto &c:rank)
        cnt[c]++;
    for(int i=1;i<n;i++){
        pos[i]=pos[i-1]+cnt[i-1];
    }
    for(int i=0;i<n;i++){
        order_t[pos[rank[order[i]]]++]=order[i];
        order=order_t;
    }
    for(int i=1;i<n;i++){
        pair<int,int>old_val={rank[order[i-1]],rank[(order[i-1]+(1<<k))%n]};
        pair<int,int>new_val={rank[order[i]],rank[(order[i]+(1<<k))%n]};
        rank_t[order[i]]=rank_t[order[i-1]]+(old_val!=new_val);
    }
}
```



```
    }
    rank=rank_t;
    k++;
}
}

void build_lcp(){
    lcp.resize(n,0);
    int k=0;
    for(int i=0;i<n-1;i++){
        int pos=rank[i];
        int j=order[pos-1];
        while(s[i+k]==s[j+k]) k++;
        lcp[pos]=k;
        k=max(k-1,(int)0);
    }
}

};
```

RabinKarp.h

**Description:** calcpow() preprocessing powers calchash() preprocess hash for a string hashval(l,r) compute hash for a substring  
**Time:**  $\mathcal{O}(n)$

898dc4, 43 lines

```
pll p = {31,53};
ll mod = 1e9+7;
pll powerval[SZ];

void calcpow()
{
    powerval[0] = {1,1};
    for(int i = 1 ; i < SZ ; i ++){
        powerval[i].first = (p.first*powerval[i-1].first)%mod;
        powerval[i].second = (p.second*powerval[i-1].second)%mod;
    }
}

vpll calchash(string &s)
{
    int n = s.length();
    vpll hash_array(n);
    for(int i = 0 ; i < n ; i ++){
        hash_array[i] = {(s[0] - 'a' + 1) , (s[0] - 'a' + 1)};
        for(int i = 1 ; i < n ; i ++){
            hash_array[i].first = ((p.first*hash_array[i-1].first) + (s[i] - 'a' + 1))%mod;
            hash_array[i].second = ((p.second*hash_array[i-1].second) + (s[i] - 'a' + 1))%mod;
        }
    }
    return hash_array;
}

pll hashval(ll l , ll r , vpll &hash_array)
{
    ll len = r - l + 1;
    int n = hash_array.size();
    if(len <= 0 || l < 0 || r >= n || r<l) return {0,0};
    pll ans = hash_array[r];
    if(l >= 1){
        ans.first -= (hash_array[l-1].first*powerval[len].first)%mod;
        ans.second -= (hash_array[l-1].second*powerval[len].second)%mod;
    }
    if(ans.first<0) ans.first += mod;
    if(ans.second<0) ans.second += mod;
}
```

```
    return ans;
}
```

Hashing.h

**Description:** Self-explanatory methods for string hashing.

2d2a67, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
typedef uint64_t ull;
struct H {
    ull x; H(ull x=0) : x(x) {}
    H operator+(H o) { return x + o.x + (x + o.x < x); }
    H operator-(H o) { return *this + ~o.x; }
    H operator*(H o) { auto m = (__uint128_t)x * o.x;
        return H((ull)m) + (ull)(m >> 64); }
    ull get() const { return x + !~x; }
    bool operator==(H o) const { return get() == o.get(); }
    bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (1ll)1e11+3; // (order ~ 3e9; random also ok)

struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
        pw[0] = 1;
        rep(i,0,sz(str))
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    }
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
};

vector<H> getHashes(string& str, int length) {
    if (sz(str) < length) return {};
    H h = 0, pw = 1;
    rep(i,0,length)
        h = h * C + str[i], pw = pw * C;
    vector<H> ret = {h};
    rep(i,length,sz(str)) {
        ret.push_back(h = h * C + str[i] - pw * str[i-length]);
    }
    return ret;
}
```

```
H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

Various (10)

10.1 Tries

characterTrie.h

**Description:** character trie

**Time:**  $\mathcal{O}(|S|)$

71371e, 52 lines

```
struct Trie {
    static const int ALPHA = 26; // Number of alphabets
    static const char c = 'a'; // Starting alphabet
    struct node {
        node* child[ALPHA];
        int start, stop;
        node() {
            for (int i = 0; i < ALPHA; i++) child[i] = NULL;
            start = stop = 0;
        }
    };
};
```

```
    }
} *root;

Trie() {
    root = new node();
}

void insert(string s) { // Insert string into trie
    node* cur = root;
    for (int i = 0; i < sz(s); i++) {
        if (!cur->child[s[i] - c]) cur->child[s[i] - c] = new
            node();
        cur->start++;
        cur = cur->child[s[i] - c];
    }
    cur->start++;
    cur->stop++;
}

bool search(string s) { // Search if a string is
    // present or not
    node* cur = root;
    for (int i = 0; i < sz(s); i++) {
        if (!cur->child[s[i] - c]) return false;
        cur = cur->child[s[i] - c];
    }
    return cur->stop;
}

int count(string s) { // Count how many words have
    // prefix=s
    node* cur = root;
    for (int i = 0; i < sz(s); i++) {
        if (!cur->child[s[i] - c]) return 0;
        cur = cur->child[s[i] - c];
    }
    return cur->start;
}

void del(string s) { // Delete a word from trie
    if (!search(s)) return;
    node* cur = root;
    for (int i = 0; i < sz(s); i++) {
        cur->start--;
        cur = cur->child[s[i] - c];
    }
    cur->start--;
    cur->stop--;
}
};
```

binaryTrie.h

**Description:** binary trie

**Time:**  $\mathcal{O}(|S|)$

956536, 83 lines

```
struct BinTrie {
    static const int B = 31; // change if LL
    struct node {
        node *nxt[2];
        int sz;
        node() {
            nxt[0] = nxt[1] = NULL;
            sz = 0;
        }
    } *root;
    BinTrie() {
        root = new node();
    }
    void insert(int val) {
        node *cur = root;
        cur->sz++;
        for (int i = B - 1; i >= 0; i--) {
            int b = val >> i & 1;
        }
    }
};
```

```

    if (cur->nxt[b] == NULL)
        cur->nxt[b] = new node();
    cur = cur->nxt[b];
    cur->sz++;
}
}
bool search(int val) {
    node *cur = root;
    for (int i = B - 1; i >= 0; i--) {
        int b = val >> i & 1;
        if (cur->nxt[b] == NULL) return false;
        cur = cur->nxt[b];
    }
    return true;
}
int query(int x, int k) { // number of values s.t. val ^ x <
    k
    node *cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
        if (cur == NULL) break;
        int b1 = x >> i & 1, b2 = k >> i & 1;
        if (b2 == 1) {
            if (cur->nxt[b1])
                ans += cur->nxt[b1]->sz;
            cur = cur->nxt[!b1];
        } else cur = cur->nxt[b1];
    }
    return ans;
}
int get_max(int x) { // returns maximum of val ^ x
    node *cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
        int k = x >> i & 1;
        if (cur->nxt[!k]) cur = cur->nxt[!k], ans <= 1, ans++;
        else cur = cur->nxt[k], ans <= 1;
    }
    return ans;
}
int get_min(int x) { // returns minimum of val ^ x
    node *cur = root;
    int ans = 0;
    for (int i = B - 1; i >= 0; i--) {
        int k = x >> i & 1;
        if (cur->nxt[k]) cur = cur->nxt[k], ans <= 1;
        else cur = cur->nxt[!k], ans <= 1, ans++;
    }
    return ans;
}
void del(int val) {
    if (!search(val)) return;
    node *cur = root;
    cur->sz--;
    for (int i = B - 1; i >= 0; i--) {
        int b = val >> i & 1;
        if (cur->nxt[b] == NULL) return;
        else if (cur->nxt[b]->sz == 1) {
            cur->nxt[b] = NULL;
            return;
        }
        cur = cur->nxt[b];
        cur->sz--;
    }
}
};

```

## 10.2 Misc. algorithms

### TernarySearch.h

**Description:** Find the smallest  $i$  in  $[a, b]$  that maximizes  $f(i)$ , assuming that  $f(a) < \dots < f(i) \geq \dots \geq f(b)$ . To reverse which of the sides allows non-strict inequalities, change the  $<$  marked with (A) to  $<=$ , and reverse the loop at (B). To minimize  $f$ , change it to  $>$ , also at (B).

**Usage:** int ind = ternSearch(0, n-1, [&](int i){return a[i];});

**Time:**  $\mathcal{O}(\log(b-a))$

9155b4, 11 lines

```

template<class F>
int ternSearch(int a, int b, F f) {
    assert(a <= b);
    while (b - a >= 5) {
        int mid = (a + b) / 2;
        if (f(mid) < f(mid+1)) a = mid; // (A)
        else b = mid+1;
    }
    rep(i, a+1, b+1) if (f(a) < f(i)) a = i; // (B)
    return a;
}

```

### LIS.h

**Description:** Compute indices for the longest increasing subsequence.

**Time:**  $\mathcal{O}(N \log N)$

2932a0, 17 lines

```

template<class I> vi lis(const vector<I>& S) {
    if (S.empty()) return {};
    vi prev(sz(S));
    typedef pair<I, int> p;
    vector<p> res;
    rep(i, 0, sz(S)) {
        // change 0 → i for longest non-decreasing subsequence
        auto it = lower_bound(all(res), p{S[i], 0});
        if (it == res.end()) res.emplace_back(), it = res.end()-1;
        *it = {S[i], i};
        prev[i] = it == res.begin() ? 0 : (it-1)->second;
    }
    int L = sz(res), cur = res.back().second;
    vi ans(L);
    while (L--) ans[L] = cur, cur = prev[cur];
    return ans;
}

```

### FastKnapsack.h

**Description:** Given  $N$  non-negative integer weights  $w$  and a non-negative target  $t$ , computes the maximum  $S \leq t$  such that  $S$  is the sum of some subset of the weights.

**Time:**  $\mathcal{O}(N \max(w_i))$

b20ccc, 16 lines

```

int knapsack(vi w, int t) {
    int a = 0, b = 0, x;
    while (b < sz(w) && a + w[b] <= t) a += w[b++];
    if (b == sz(w)) return a;
    int m = *max_element(all(w));
    vi u, v(2*m, -1);
    v[a+m-t] = b;
    rep(i, b, sz(w)) {
        u = v;
        rep(x, 0, m) v[x+w[i]] = max(v[x+w[i]], u[x]);
        for (x = 2*m; --x > m;) rep(j, max(0, u[x]), v[x])
            v[x-w[j]] = max(v[x-w[j]], j);
    }
    for (a = t; v[a+m-t] < 0; a--);
    return a;
}

```

### xorBasis.h

**Description:** xor basis

**Time:**  $\mathcal{O}(30)$

25b147, 56 lines

```

const int M = 31;
int basis[M];
int cnt;
void init() {
    memset(basis, 0, sizeof basis);
    cnt = 0;
}
bool insertVector(int val) {
    for (int j = M - 1; j >= 0; j--) {
        if (!(val & (1 << j))) continue;
        if (basis[j] == 0) {
            basis[j] = val;
            cnt++;
            return true;
        }
        val ^= basis[j];
    }
    return false;
}
int max_ele() {
    int ans = 0;
    for (int j = M - 1; j >= 0; j--) {
        if (ans & (1 << j)) continue;
        ans ^= basis[j];
    }
    return ans;
}
int kth_ele(int k) {
    int ans=0;
    int rem=cnt;
    for(int j=M-1;j>=0;j--) {
        if (!basis[j]) continue;
        rem--;
        if (ans & (1 << j)) {
            if ((1 << rem) >= k) {
                ans ^= basis[j];
            }else{
                k--(1<<rem);
            }
        } else {
            if ((1 << rem) < k) {
                ans ^= basis[j];
                k -= (1 << rem);
            }
        }
    }
    return ans;
}
bool is_in_space(int x) {
    for (int j = M - 1; j >= 0; j--) {
        if (!(x & (1 << j))) continue;
        if (!basis[j]) return false;
        x ^= basis[j];
    }
    return true;
}
}

```

## 10.3 Dynamic programming

### KnuthDP.h

**Description:** When doing DP on intervals:  $a[i][j] = \min_{i < k < j} (a[i][k] + a[k][j]) + f(i, j)$ , where the (minimal) optimal  $k$  increases with both  $i$  and  $j$ , one can solve intervals in increasing order of length, and search  $k = p[i][j]$  for  $a[i][j]$  only between  $p[i][j - 1]$  and  $p[i + 1][j]$ . This is known as Knuth DP. Sufficient criteria for this are if  $f(b, c) \leq f(a, d)$  and  $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$  for all  $a \leq b \leq c \leq d$ . Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.  
**Time:**  $\mathcal{O}(N^2)$

DivideAndConquerDP.h

**Description:** Given  $a[i] = \min_{lo(i) \leq k < hi(i)} (f(i, k))$  where the (minimal) optimal  $k$  increases with  $i$ , computes  $a[i]$  for  $i = L..R - 1$ .  
**Time:**  $\mathcal{O}((N + (hi - lo)) \log N)$

```
struct DP { // Modify at will:
    int lo(int ind) { return 0; }
    int hi(int ind) { return ind; }
    ll f(int ind, int k) { return dp[ind][k]; }
    void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

    void rec(int L, int R, int LO, int HI) {
        if (L >= R) return;
        int mid = (L + R) >> 1;
        pair<ll, int> best(LLONG_MAX, LO);
        rep(k, max(LO, lo(mid)), min(HI, hi(mid)))
            best = min(best, make_pair(f(mid, k), k));
        store(mid, best.second, best.first);
        rec(L, mid, LO, best.second+1);
        rec(mid+1, R, best.second, HI);
    }
    void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

convexHullDP.h

**Description:** convex hull trick DP  
**Time:**  $\mathcal{O}(N^2)$

```
// Container where you can add lines of the form kx+m, and
// query maximum values at points // x.

struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};
```

```
}
};

sosDP.h
Description: SOS DP
Time: O(N^2)

const int LOG = 22;
//pull contribution from its subsets
void forward1(vl &dp){
    for(int bit = 0; bit < LOG; bit++){
        for(int i = 0; i < MAXN; i++){
            if(i&(1<<bit)) dp[i] += dp[i^(1<<bit)];
        }
    }
    void backward1(vl &dp){
        for(int bit = 0; bit < LOG; bit++){
            for(int i = MAXN-1; i >= 0; i--){
                if(i&(1<<bit)) dp[i] -= dp[i^(1<<bit)];
            }
        }
    }
    //pull contribution from its supersets
    void forward2(vl &dp){
        for(int bit = 0;bit < LOG;bit++){
            for(int i = MAXN-1; i >= 0; i--){
                if(i&(1<<bit)) dp[i^(1<<bit)] += dp[i];
            }
        }
    }
    void backward2(vl &dp){
        for(int bit = 0;bit < LOG;bit++){
            for(int i = 0; i < MAXN; i++){
                if(i&(1<<bit)) dp[i^(1<<bit)] -= dp[i];
            }
        }
    }
};
```

10.4 Debugging tricks

- signal(SIGSEGV, [](int) { \_Exit(0); }); converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). \_GLIBCXX\_DEBUG failures generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- feenableexcept(29); kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

10.5 Optimization tricks

\_\_builtin\_ia32\_ldmxcsr(40896); disables denormals (which make floats 20x slower near their minimum value).

10.5.1 Bit hacks

- x & -x is the least bit in x.
- for (int x = m; x; ) { --x &= m; ... } loops over all subset masks of m (except m itself).
- c = x&-x, r = x+c; (((r^x) >> 2)/c) | r is the next number after x with the same number of bits set.
- rep(b,0,K) rep(i,0,(1 << K)) if (i & 1 << b) D[i] += D[i^(1 << b)]; computes all sums of subsets.

10.5.2 Pragmas

- #pragma GCC optimize ("Ofast") will make GCC auto-vectorize loops and optimizes floating points better.

- #pragma GCC target ("avx2") can double performance of vectorized code, but causes crashes on old machines.
- #pragma GCC optimize ("trapv") kills the program on integer overflows (but is really slow).

FastMod.h

**Description:** Compute  $a\%b$  about 5 times faster than usual, where  $b$  is constant but not known at compile time. Returns a value congruent to  $a$  (mod  $b$ ) in the range  $[0, 2b)$ .

```
typedef unsigned long long ull;
struct FastMod {
    ull b, m;
    FastMod(ull b) : b(b), m(-1ULL / b) {}
    ull reduce(ull a) { // a % b + (0 or b)
        return a - (ull)((__uint128_t(m) * a) >> 64) * b;
    }
};
```

FastInput.h

**Description:** Read an integer from stdin. Usage requires your program to pipe in input from file.  
**Usage:** ./a.out < input.txt  
**Time:** About 5x as fast as cin/scanf.

```
inline char gc() { // like getchar()
    static char buf[1 << 16];
    static size_t bc, be;
    if (bc >= be) {
        buf[0] = 0, bc = 0;
        be = fread(buf, 1, sizeof(buf), stdin);
    }
    return buf[bc++]; // returns 0 on EOF
}

int readInt() {
    int a, c;
    while ((a = gc()) < 40);
    if (a == '-') return -readInt();
    while ((c = gc()) >= 48) a = a * 10 + c - 48;
    return a - 48;
}
```

BumpAllocator.h

**Description:** When you need to dynamically allocate many objects and don't care about freeing them. "new X" otherwise has an overhead of something like 0.05us + 16 bytes per allocation.

```
// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
    static size_t i = sizeof buf;
    assert(s < i);
    return (void*)&buf[i -= s];
}
void operator delete(void*) {}
```

SmallPtr.h

**Description:** A 32-bit pointer that points into BumpAllocator memory.  
"BumpAllocator.h"  
template<class T> struct ptr {
 unsigned ind;
 ptr(T\* p = 0) : ind(p ? unsigned((char\*)p - buf) : 0) {
 assert(ind < sizeof buf);
 }
 T& operator\*() const { return \*(T\*)(buf + ind); }
 T\* operator->() const { return &\*this; }

```
T& operator[](int a) const { return (&*this)[a]; }
explicit operator bool() const { return ind; }
};
```

BumpAllocatorSTL.h

Description: BumpAllocator for STL containers.

Usage: vector<vector<int, small<int>>> ed(N); bb66d4, 14 lines

```
char buf[450 << 20] alignas(16);
size_t buf_ind = sizeof buf;

template<class T> struct small {
    typedef T value_type;
    small() {}
    template<class U> small(const U&) {}
    T* allocate(size_t n) {
        buf_ind -= n * sizeof(T);
        buf_ind &= 0 - alignof(T);
        return (T*)(buf + buf_ind);
    }
    void deallocate(T*, size_t) {}
};
```

SIMD.h

Description: Cheat sheet of SSE/AVX intrinsics, for doing arithmetic on several numbers at once. Can provide a constant factor improvement of about 4, orthogonal to loop unrolling. Operations follow the pattern "\_mm(256)?\_name\_(si(128|256)|epi(8|16|32|64)|pd|ps)". Not all are described here; grep for \_mm\_ in /usr/lib/gcc/\*/4.9/include/ for more. If AVX is unsupported, try 128-bit operations, "emmintrin.h" and #define \_SSE\_ and \_MMX\_ before including it. For aligned memory use \_mm\_malloc(size, 32) or int buf[N] alignas(32), but prefer loadu/storeu. 551b82, 43 lines

```
#pragma GCC target ("avx2") // or sse4.1
#include "emmintrin.h"
```

```
typedef __m256i mi;
#define L(x) _mm256_loadu_si256((mi*)&(x))
```

```
// High-level/specific methods:
// load(u)?_si256, store(u)?_si256, setzero_si256, _mm_malloc
// blendv_(epi8|ps|pd) (z?y:x), movemask_epi8 (hibits of bytes)
// i32gather_epi32(addr, x, 4): map addr[] over 32-b parts of x
// sad_epu8: sum of absolute differences of u8, outputs 4xi64
// maddubs_epi16: dot product of unsigned i7's, outputs 16xi15
// madd_epi16: dot product of signed i16's, outputs 8xi32
// extractf128_si256(, i) (256->128), cvtsi128_si32 (128->lo32)
// permute2f128_si256(x,x,1) swaps 128-bit lanes
// shuffle_epi32(x, 3*64+2*16+1*4+0) == x for each lane
// shuffle_epi8(x, y) takes a vector instead of an mm
```

```
// Methods that work with most data types (append e.g. _epi32):
// set1, blend (i8?x:y), add, adds (sat.), mullo, sub, and/or,
// andnot, abs, min, max, sign(1,x), cmp(gt|eq), unpack(lo|hi)
```

```
int sumi32(mi m) { union {int v[8]; mi m;} u; u.m = m;
    int ret = 0; rep(i,0,8) ret += u.v[i]; return ret; }
mi zero() { return _mm256_setzero_si256(); }
mi one() { return _mm256_set1_epi32(-1); }
bool all_zero(mi m) { return _mm256_testz_si256(m, m); }
bool all_one(mi m) { return _mm256_testc_si256(m, one()); }
```

```
ll example_filteredDotProduct(int n, short* a, short* b) {
    int i = 0; ll r = 0;
    mi zero = _mm256_setzero_si256(), acc = zero;
    while (i + 16 <= n) {
        mi va = L(a[i]), vb = L(b[i]); i += 16;
        va = _mm256_and_si256(_mm256_cmpgt_epi16(vb, va), va);
```

```
        mi vp = _mm256_madd_epi16(va, vb);
        acc = _mm256_add_epi64(_mm256_unpacklo_epi32(vp, zero),
            _mm256_add_epi64(acc, _mm256_unpackhi_epi32(vp, zero)));
    }
    union {ll v[4]; mi m;} u; u.m = acc; rep(i,0,4) r += u.v[i];
    for (;i<n;++i) if (a[i] < b[i]) r += a[i]*b[i]; // <- equiv
    return r;
}
```

# Techniques (A)

|  |           |
|--|-----------|
| techniques.txt                                     | 159 lines |
| Recursion  |           |
| Divide and conquer                                 |           |
| Finding interesting points in N log N              |           |
| Algorithm analysis                                 |           |
| Master theorem                                     |           |
| Amortized time complexity                          |           |
| Greedy algorithm                                   |           |
| Scheduling   |           |
| Max contiguous subvector sum                       |           |
| Invariants   |           |
| Huffman encoding                                   |           |
| Graph theory                                       |           |
| Dynamic graphs (extra book-keeping)                |           |
| Breadth first search                               |           |
| Depth first search                                 |           |
| * Normal trees / DFS trees                         |           |
| Dijkstra's algorithm                               |           |
| MST: Prim's algorithm                              |           |
| Bellman-Ford                                       |           |
| Konig's theorem and vertex cover                   |           |
| Min-cost max flow                                  |           |
| Lovasz toggle                                      |           |
| Matrix tree theorem                                |           |
| Maximal matching, general graphs                   |           |
| Hopcroft-Karp                                      |           |
| Hall's marriage theorem                            |           |
| Graphical sequences                                |           |
| Floyd-Warshall                                     |           |
| Euler cycles                                       |           |
| Flow networks                                      |           |
| * Augmenting paths                                 |           |
| * Edmonds-Karp                                     |           |
| Bipartite matching                                 |           |
| Min. path cover                                    |           |
| Topological sorting                                |           |
| Strongly connected components                      |           |
| 2-SAT  |           |
| Cut vertices, cut-edges and biconnected components |           |
| Edge coloring                                      |           |
| * Trees  |           |
| Vertex coloring                                    |           |
| * Bipartite graphs (=> trees)                      |           |
| * 3^n (special case of set cover)                  |           |
| Diameter and centroid                              |           |
| K'th shortest path                                 |           |
| Shortest cycle                                     |           |
| Dynamic programming                                |           |
| Knapsack   |           |
| Coin change  |           |
| Longest common subsequence                         |           |
| Longest increasing subsequence                     |           |
| Number of paths in a dag                           |           |
| Shortest path in a dag                             |           |
| Dynprog over intervals                             |           |
| Dynprog over subsets                               |           |
| Dynprog over probabilities                         |           |
| Dynprog over trees                                 |           |
| 3^n set cover                                      |           |
| Divide and conquer                                 |           |
| Knuth optimization                                 |           |
| Convex hull optimizations                          |           |
| RMQ (sparse table a.k.a 2^k-jumps)                 |           |
| Bitonic cycle                                      |           |
| Log partitioning (loop over most restricted)       |           |
| Combinatorics                                      |           |

|  |
|--|
| Computation of binomial coefficients         |
| Pigeon-hole principle                        |
| Inclusion/exclusion                          |
| Catalan number                               |
| Pick's theorem                               |
| Number theory                                |
| Integer parts                                |
| Divisibility                                 |
| Euclidean algorithm                          |
| Modular arithmetic                           |
| * Modular multiplication                     |
| * Modular inverses                           |
| * Modular exponentiation by squaring         |
| Chinese remainder theorem                    |
| Fermat's little theorem                      |
| Euler's theorem                              |
| Phi function                                 |
| Frobenius number                             |
| Quadratic reciprocity                        |
| Pollard-Rho                                  |
| Miller-Rabin                                 |
| Hensel lifting                               |
| Vieta root jumping                           |
| Game theory                                  |
| Combinatorial games                          |
| Game trees                                   |
| Mini-max                                     |
| Nim  |
| Games on graphs                              |
| Games on graphs with loops                   |
| Grundy numbers                               |
| Bipartite games without repetition           |
| General games without repetition             |
| Alpha-beta pruning                           |
| Probability theory                           |
| Optimization                                 |
| Binary search                                |
| Ternary search                               |
| Unimodality and convex functions             |
| Binary search on derivative                  |
| Numerical methods                            |
| Numeric integration                          |
| Newton's method                              |
| Root-finding with binary/ternary search      |
| Golden section search                        |
| Matrices                                     |
| Gaussian elimination                         |
| Exponentiation by squaring                   |
| Sorting                                      |
| Radix sort                                   |
| Geometry                                     |
| Coordinates and vectors                      |
| * Cross product                              |
| * Scalar product                             |
| Convex hull                                  |
| Polygon cut                                  |
| Closest pair                                 |
| Coordinate-compression                       |
| Quadtrees                                    |
| KD-trees                                     |
| All segment-segment intersection             |
| Sweeping                                     |
| Discretization (convert to events and sweep) |
| Angle sweeping                               |
| Line sweeping                                |
| Discrete second derivatives                  |
| Strings                                      |
| Longest common substring                     |
| Palindrome subsequences                      |

|   |
|---|
| Knuth-Morris-Pratt                                    |
| Tries   |
| Rolling polynomial hashes                             |
| Suffix array  |
| Suffix tree   |
| Aho-Corasick  |
| Manacher's algorithm                                  |
| Letter position lists                                 |
| Combinatorial search                                  |
| Meet in the middle                                    |
| Brute-force with pruning                              |
| Best-first (A*)                                       |
| Bidirectional search                                  |
| Iterative deepening DFS / A*                          |
| Data structures                                       |
| LCA (2^k-jumps in trees in general)                   |
| Pull/push-technique on trees                          |
| Heavy-light decomposition                             |
| Centroid decomposition                                |
| Lazy propagation                                      |
| Self-balancing trees                                  |
| Convex hull trick (wcipeg.com/wiki/Convex_hull_trick) |
| Monotone queues / monotone stacks / sliding queues    |
| Sliding queue using 2 stacks                          |
| Persistent segment tree                               |