

Competitive Programming Notebook

Raul Almeida¹

¹Universidade Federal do Paraná

September 15, 2023

Contents

1 Theory	2	6 Paradigm	9
1.1 Relevant comparisons	2	6.1 Coordinate Compression	9
1.2 Prime counting function - $\pi(x)$	2	6.2 128 Bit Integers	9
1.3 Progressions	2	6.3 Binary Search	9
1.4 Series Identities	2	7 String	9
1.5 Binomial Identities	3	7.1 Rolling hash (linear)	9
1.6 Lucas' Theorem	3	7.2 Prefix Function (KMP)	9
1.7 Fermat Theorems	3	7.3 Suffix Array	9
1.8 Modulo @ exponent	3	8 Structure	10
1.9 Heron's Formula	3	8.1 Merge/Disjoint Union-Find	10
1.10 Some Primes	3	8.2 Bottom-Up Segment Tree	10
1.11 Catalan Numbers	3	8.3 Segment Tree	10
1.12 Binomial	3	9 Extra	11
1.13 Trigonometry	4	9.1 C++ structs	11
1.14 Multiples of gcd	4	9.2 cmp	11
1.15 Expected Value	4	9.3 Vim	11
1.16 Combination	4	9.4 Generator	11
1.17 Permutation	4	9.5 Makefile	11
2 Emergency	4	9.6 C++ Template	11
3 Geometry	4	9.7 Stress	11
3.1 Points	4		
3.2 Convex Hull (Monotone)	4		
4 Graph	5		
4.1 Prim MST	5		
4.2 Dijkstra SSSP	5		
4.3 Graph Check	5		
4.4 Articulations and Bridges	5		
4.5 Euler Tour	6		
4.6 Heavy-Light Decomposition	6		
4.7 Kahn's topological sort	6		
4.8 Max Cardinality Bipartite Matching	6		
4.9 LCA - Binary lifting	6		
4.10 Tarjan Strongly Connected Component	7		
4.11 LCA - Euler Path	7		
4.12 Kosaraju SCC	7		
4.13 Bellman-Ford SSSP	7		
4.14 Kruskal MST	7		
4.15 Edmond Karp MaxFlow	7		
4.16 Floyd Warshall APSP	8		
5 Math	8		
5.1 Sieve of Eratosthenes	8		
5.2 Prime Factors w/ Optimized Trial Divisions	8		
5.3 Extended Euclid for Linear Diophantines	8		
5.4 Floyd's algorithm cycle-finding	9		

n	not-TLE algorithm	Example
$\leq [10..11]$	$\mathcal{O}(n!)$, $\mathcal{O}(n^6)$	Enumerate permutations
$\leq [15..18]$	$\mathcal{O}(2^n n^2)$	TSP with DP
$\leq [18..22]$	$\mathcal{O}(2^n n)$	Bitmask DP
≤ 100	$\mathcal{O}(n^4)$	3D DP with $\mathcal{O}(n)$ loop
≤ 400	$\mathcal{O}(n^3)$	Floyd-Warshall
$\leq 2 \cdot 10^3$	$\mathcal{O}(n^2 \lg n)$	2 nested loops + tree query
$\leq 5 \cdot 10^4$	$\mathcal{O}(n^2)$	Bubble/Selection/Insertion Sort
$\leq 10^5$	$\mathcal{O}(n \lg^2 n) = \mathcal{O}((\lg n)(\lg n))$	Build suffix array
$\leq 10^6$	$\mathcal{O}(n \lg n)$	MergeSort, build SegTree
$\leq 10^7$	$\mathcal{O}(n \lg \lg n)$	Sieve, totient function
$\leq 10^8$	$\mathcal{O}(n)$, $\mathcal{O}(\lg n)$, $\mathcal{O}(1)$	Mathy solution often with IO bottleneck ($n \leq 10^9$)

10⁸ ops/second

1 Theory

1.1 Relevant comparisons

lg 10 (1E1)	2.3
lg 100 (1E1)	4.6
lg 1000 (1E2)	6.9
lg 10000 (1E3)	9.2
lg 100000 (1E4)	11.5
lg 1000000 (1E5)	13.8
lg 10000000 (1E6)	16.1
lg 100000000 (1E7)	18.4
lg 1000000000 (1E8)	20.7
lg 10000000000 (1E9)	23.0
lg 100000000000 (1E10)	25.3
lg 1000000000000 (1E11)	27.6
lg 10000000000000 (1E12)	29.9
2 ¹⁰	≈ 10 ³
2 ²⁰	≈ 10 ⁶

Sign	Type	Bits	Max	Digits
±	char	8	127	2
+	unsigned char	8	255	2
±	short	16	32 767	4
+	unsigned short	16	65 535	4
±	int/long	32	≈ 2 · 10 ⁹	9
+	unsigned int/long	32	≈ 4 · 10 ⁹	9
±	long long	64	≈ 9 · 10 ¹⁸	18
+	unsigned long long	64	≈ 18 · 10 ¹⁸	19
±	__int128	128	≈ 17 · 10 ³⁷	38
+	unsigned __int128	128	≈ 3 · 10 ³⁸	38

1.2 Prime counting function - pi(x)

Asymptotic to $\frac{x}{\log x}$ by the prime number theorem.

1.3 Progressions

$$a_n = a_k + r(n - k)$$

$$a_n = a_k q^{(n-k)}$$

- r , q : Ratio
- k : Known term

Algorithm	Time	Space
ArticBridges	$\mathcal{O}(V + E)$	$\mathcal{O}(V + E)$
Bellman-Ford	$\mathcal{O}(VE)$	$\mathcal{O}(V + E)$
Dijkstra	$\mathcal{O}((V + E) \log V)$	$\mathcal{O}(V^2)$
Edmond Karp	$\mathcal{O}(VE^2)$	$\mathcal{O}(V + E)$
Euler Tour	$\mathcal{O}(E^2)$	
Floyd Warshall	$\mathcal{O}(V^3 + E)$	$\mathcal{O}(V^2 + E)$
Graph Check	$\mathcal{O}(V + E)$	$\mathcal{O}(V + E)$
Kahn	$\mathcal{O}(VE)$	$\mathcal{O}(V + E)$
Kruskal	$\mathcal{O}(E \log V)$	$\mathcal{O}(V + E)$
LCA	$\mathcal{O}(N \log N)$	$\mathcal{O}(N \log N)$
MCBM	$\mathcal{O}(VE)$	
Prim	$\mathcal{O}(E \log V)$	$\mathcal{O}(V + E)$
Tarjan	$\mathcal{O}(V + E)$	$\mathcal{O}(V + E)$
Extended Euclid	$\mathcal{O}(\log \min(a, b))$	$\mathcal{O}(1)$
Floyd (cycle)	$\mathcal{O}(V)$	$\mathcal{O}(1)$
PrimeFac + OptTrialDiv	$\mathcal{O}(\pi(\sqrt{n}))$	$\mathcal{O}(n)$
Sieve of Eratosthenes	$\mathcal{O}(n \log \log n)$	$\mathcal{O}(n)$
Binary Search	$\mathcal{O}(\log N)$	
Coordinate Compression	$\mathcal{O}(N \log N)$	
KMP	$\mathcal{O}(N)$	
MUF	$\mathcal{O}(AM)$	$\mathcal{O}(N)$
Bottom-Up SegTree	$\mathcal{O}(\log N)$	$\mathcal{O}(N)$

x	10	10 ²	10 ³	10 ⁴
π(x)	4	25	168	1 229
x	10 ⁵	10 ⁶	10 ⁷	10 ⁸
π(x)	9 592	78 498	664 579	5 761 455

- n : Term you want

$$S_n = \frac{n(a_1 + a_n)}{2}$$

$$S_n = \frac{a_1(q^n - 1)}{q - 1}$$

1.4 Series Identities

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4} = \left(\sum_{i=1}^n i \right)^2$$

$$g_k(n) = \sum_{i=1}^n i^k = \frac{1}{k+1} \left(n^{k+1} + \sum_{j=1}^k \binom{k+1}{j+1} (-1)^{j+1} g_{k-j}(n) \right)$$

$$\sum_{i=0}^n ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1$$

$$\sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1$$

$$l + (l+1) + \dots + r = \frac{(l+r) \cdot (r-l+1)}{2}$$

1.5 Binomial Identities

$$\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$$

$$\binom{n-1}{k} - \binom{n-1}{k-1} = \frac{n-2k}{k} \binom{n}{k}$$

$$\binom{n}{h} \binom{n-h}{k} = \binom{n}{k} \binom{n-k}{h}$$

$$\binom{n}{k} = \frac{n+1-k}{k} \binom{n}{k-1}$$

$$\sum_{k=0}^n k \binom{n}{k} = n2^{n-1}$$

$$\sum_{k=0}^n k^2 \binom{n}{k} = (n+n^2)2^{n-2}$$

$$\sum_{j=0}^k \binom{m}{j} \binom{n-m}{k-j} = \binom{n}{k}$$

$$\sum_{j=0}^m \binom{m}{j}^2 = \binom{2m}{m}$$

$$\sum_{m=0}^n \binom{m}{j} \binom{n-m}{k-j} = \binom{n+1}{k+1}$$

$$\sum_{m=k}^n \binom{m}{k} = \binom{n+1}{k+1}$$

$$\sum_{r=0}^m \binom{n+r}{r} = \binom{n+m+1}{m}$$

$$\sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n-k}{k} = \text{Fib}(n+1)$$

$$(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k$$

$$(1+x)^n = \sum_{k=0}^n \binom{n}{k} x^k$$

$$2 \sum_{i=L}^R \binom{n}{i} - \binom{n}{L} - \binom{n}{R} = \sum_{i=L+1}^R \binom{n+1}{i}$$

1.6 Lucas' Theorem

$$\binom{n}{m} = \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$$

For prime p , n_i and m_i are coefficients of the representations of n and m in base p .

1.7 Fermat Theorems

p is prime

$$a^p \equiv a \pmod{p}$$

$$a^{p-1} \equiv 1 \pmod{p}$$

$$(a+b)^p \equiv a^p + b^p \pmod{p}$$

$$a^{-1} \equiv a^{p-2} \pmod{p}$$

1.8 Modulo @ exponent

For coprime a, m :

$$a^n \equiv a^{n \bmod \varphi(m)} \pmod{m}$$

Generally, if $n \geq \log_2 m$, then

$$a^n \equiv a^{\varphi(m) + [n \bmod \varphi(m)]} \pmod{m}$$

1.9 Heron's Formula

Area of a triangle ($s = \frac{a+b+c}{2}$)

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

1.10 Some Primes

- $10^6 + 69$
- $10^9 + 7$
- $10^9 + 9$
- $10^{18} - 11$
- $10^{18} + 3$
- $2^{61} - 1$
- 1000696969
- 998244353
- 999999937
- 1000000007
- 1000000009
- 1000000021
- 1000000033
- $10^{18} - 11$
- $10^{18} + 3$
- 2305843009213693951 = $2^{61} - 1$

1.11 Catalan Numbers

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, 18367353072152, 69533550916004, 263747951750360, 1002242216651368.

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^n \frac{n+k}{k}, n \geq 0.$$

C_n is:

- The number of valid parenthesis strings with n parentheses
- The number of complete binary trees with $n+1$ leaves
- How many times a $n+2$ -sided convex polygon can be cut in triangles connecting its vertices with straight lines

1.12 Binomial

X is the number of successes in a sequence of n independent experiments. $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$, and $E[X] = np$ and $Var(X) = np(1-p)$.

1.13 Trigonometry

$\sin^2 \theta + \cos^2 \theta = 1$, $\sin = \frac{opo}{hip}$, $\cos = \frac{adj}{hip}$, $\tan = \frac{opo}{adj}$. $\sin \theta = x \rightarrow \arcsin x = \theta$.

α degrees to x rd: $\alpha = \frac{180x}{\pi}$

1.14 Multiples of gcd

Multiples of $\gcd(A, B)$ that are $\in [0, A)$

Let $A, B > 0$, $g = \gcd(A, B)$, $A = ag$ and $B = bg$.

a integers $(0 \times B)\%A$, $(1 \times B)\%A$, $(2 \times B)\%A \dots ((a - 1) \times B)\%A$ correspond to each multiple of g between 0 and $A - 1$ (inclusive); note that they are all unique.

1.15 Expected Value

Avg value of event. For each event, add to the sum the probability of an event times the value of X in that event
 $\mathbb{E}(X) = \sum_{\omega \in \Omega} (P(\omega) \times X(\omega))$

Another way of looking at it:

$$\mathbb{E}(X) = \sum_{i=1}^M (i \times P(X = i))$$

Since in the expanded version of this sum $P(X = i)$ will appear i times, you're also calculating for each i the probability that $X \geq i$ ($P(x = M)$ will appear M times, once for each i ; $P(x = 1)$ will appear exactly once, for $i = 1$; and so on). So

$$\mathbb{E}(X) = \sum_{i=1}^M (i \times P(X = i)) = \sum_{i=1}^M P(X \geq i)$$

1.16 Combination

A combination ${}_nC_k = \binom{n}{k}$ (n chooses k) refers to selecting k objects from a collection of n where the order of choice doesn't matter.

Without repetition: can't choose an element twice.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

With repetition: elements may be chosen more than

$$\text{once. } \binom{n}{k} = \frac{(k+n-1)!}{k!(n-1)!}$$

1.17 Permutation

A permutation ${}_nP_k$ refers to selecting k objects from a collection of n where the order of choice matters.

With repetition: elements may be chosen more than once. ${}_nP_k = n^k$

Without repetition: can't choose an element twice.

$${}_nP_k = \frac{n!}{(n-k)!}$$

2 Emergency

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 (check the filename you're editing).

Wrong answer

Print your solution and debug output!

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? (use references)

How big is the input and output? (consider **scanf** and **printf**)

Avoid **vector**, **map**. (use **array/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?

3 Geometry

3.1 Points

```
1 using pt = complex<double>;
2 #define px real()
3 #define py imag()
4
5 double dot(pt a, pt b) { return (conj(a)*b).px; }
6 double cross(pt a, pt b) { return (conj(a)*b).py; }
7 pt vec(pt a, pt b) { return b-a; }
8 int sgn(double v) { return (v > -EPS) - (v < EPS); }
9 int seg_ornt(pt a, pt b, pt c) {
10     return sgn(cross(vec(a, b), vec(a, c)));
11 }
12 int ccw(pt a, pt b, pt c, bool col) {
13     int o = seg_ornt(a, b, c);
14     return (o == 1) || (o == 0 && col);
15 }
16 const double PI = acos(-1);
17 double angle(pt a, pt b, pt c) {
18     return abs(remainder(arg(a-b) - arg(c-b), 2.0*PI));
19 }
```

3.2 Convex Hull (Monotone)

```
1 vector<pt> convex_hull(vector<pt>& ps, bool col = false) {
2     int k = 0, n = ps.size(); vector<pt> ans (2*n);
3     sort(ps.begin(), ps.end(), [](pt a, pt b) {
4         return make_pair(a.px, a.py) < make_pair(b.px, b.py);
5     });
```

```

5  });
6  for (int i = 0; i < n; i++) {
7      while (k >= 2 && !ccw( /* lower hull */
8          ans[k-2], ans[k-1], ps[i], col)) { k--; }
9      ans[k++] = ps[i];
10 }
11 if (k == n) { ans.resize(n); return ans; }
12 for (int i = n-2, t = k+1; i >= 0; i--) {
13     while (k >= t && !ccw( /* upper hull */
14         ans[k-2], ans[k-1], ps[i], col)) { k--; }
15     ans[k++] = ps[i];
16 }
17 ans.resize(k-1); return ans;
18 }
19
20 // with answer as idx of points
21 using pti = pair<pt, int>;
22 #define fi first
23 #define se second
24 vector<int> convex_hull(vector<pti>& ps, bool col = false) {
25     int k = 0, n = ps.size(); vector<int> ans (2*n);
26     sort(ps.begin(), ps.end(), [](pti a, pti b) {
27         return make_pair(a.fi.px, a.fi.py) < make_pair(b.fi.px,
28             b.fi.py);
29 });
30 for (int i = 0; i < n; i++) {
31     while (k >= 2 && !ccw( /* lower hull */
32         ps[ans[k-2]].fi, ps[ans[k-1]].fi, ps[i].fi, col)) {
33         k--; }
34     ans[k++] = i;
35 }
36 if (k == n) {
37     ans.resize(n);
38     for (auto &i : ans) i = ps[i].second;
39     return ans; }
40 for (int i = n-2, t = k+1; i >= 0; i--) {
41     while (k >= t && !ccw( /* upper hull */
42         ps[ans[k-2]].fi, ps[ans[k-1]].fi, ps[i].fi, col)) {
43         k--; }
44     ans[k++] = i;
45 }
46 ans.resize(k-1);
47 for (auto &i : ans) i = ps[i].second;
48 return ans;
49 }

```

4 Graph

4.1 Prim MST

Time	Space
$\mathcal{O}(E \log V)$	$\mathcal{O}(V + E)$

```

1 vector<vector<pair<int, int>> adj(M), mst(M);
2 vector<bool> taken(M, false);
3 int cost = 0;
4 using iii = pair<int, pair<int, int>>;
5 priority_queue<iii, vector<iii>, greater<iii> pq;
6
7 void process(int v) {
8     taken[v] = true;
9     for (auto &[w, u]: adj[v])
10         if (!taken[u])
11             pq.push({w, {v, u}});
12 }
13
14 void run(int n) {
15     process(0);
16     while (!pq.empty()) {
17         int w = pq.top().first,
18             v = pq.top().second.first,
19             u = pq.top().second.second;
20         pq.pop();
21         if (!taken[u]) {
22             mst_cost += w;
23             mst[u].push_back({w, v});
24             mst[v].push_back({v, w});
25             process(u);

```

```

26     }
27 }
28 for (int v = 1; v < n; ++v)
29     if (!taken[v]) {
30         process(v);
31         run(n);
32     }
33 }

```

4.2 Dijkstra SSSP

Time	Space
$\mathcal{O}((V + E) \log V)$	$\mathcal{O}(V^2)$

```

1 vector<int> d(MAXN, oo);
2
3 void dijkstra(int s) {
4     priority_queue<wv, vector<wv>, greater<wv>> pq;
5     pq.emplace(d[s] = 0, s);
6     add(s, 0);
7     while (!pq.empty()) {
8         auto [w, v] = pq.top(); pq.pop();
9         if (w > dist[v]) continue;
10        for (auto [x, u] : g[v])
11            if (w+x < d[u])
12                pq.emplace(d[u]=w+x, u);
13    }
14 }

```

4.3 Graph Check

Time/Space	Usage
$\mathcal{O}(V + E)$	graphCheck(firstVertex, -1)

```

1 int UNVISITED = -1, EXPLORED = 0, VISITED = 1;
2 vector<vector<int>> adj(M);
3 vector<int> tin;
4
5 void graphCheck(int v, int p) { //vertex, parent
6     tin[v] = EXPLORED;
7     for (auto u: adj[v]) {
8         if (tin[u] == UNVISITED) { //tree edge
9             graphCheck(u, v);
10        } else if (tin[u] == EXPLORED) {
11            if (u == p)
12                ; //two way edge u <-> v
13            else
14                ; //back edge v -> u
15        } else if (tin[u] == VISITED) {
16            ; //forward/cross edge u-v
17        }
18    }
19     tin[v] = VISITED;
20 }

```

4.4 Articulations and Bridges

Time/Space	Usage
$\mathcal{O}(V + E)$	dfs(src, -1)

```

1
2 int tk = 0;
3 vector<int> tin(M, -1);
4 vector<vector<int>> adj(M);
5
6 void dfs(int v, int p) {
7     tin[v] = low[v] = tk++;
8     int children = 0;
9     for (auto u: adj[v]) {
10        if (u == p) continue;
11        else if (tin[u] == -1) {
12            ++children;
13            dfs(u, v);

```

```

14     if (low[u] >= tin[v] && p != v)
15         ; //articulation point
16     if (low[u] > tin[v])
17         ; //bridge u-v
18     low[v] = min(low[v], low[u]);
19 } else {
20     low[v] = min(low[v], tin[u]);
21 }
22 }
23 }

```

4.5 Euler Tour

Time	Usage
$\mathcal{O}(E^2)$	tour(cyc.begin(), start_vertex)

```

1 list<int> cyc;
2 vector<vector<int>> adj(M);
3 vector<vector<bool>> traversed(M, vector<bool>(M, false));
4
5 //euler tour (list for fast insertion)
6 void tour(list<int>::iterator i, int v) {
7     for (auto u: adj[v]) {
8         if (!traversed[v][u]) {
9             traversed[v][u] = true;
10            for (auto t: adj[u])
11                if (t == v && !traversed[u][t]) {
12                    traversed[u][t] = true;
13                    break;
14                }
15            tour(cyc.insert(i, v), u);
16        }
17    }
18 }

```

4.6 Heavy-Light Decomposition

Query	Setup	Update
$\mathcal{O}(\log^2 n)$	define oper(a,b) for query	rmq.upd(pos[x],v)

Queries on edges: assign values of edges to child node, then change pos[x] to pos[x]+1 in query (see !!!)

```

1 vector<int> g[MAXN];
2 int wg[MAXN], par[MAXN], h[MAXN]; // subtree
   size, father, height
3 void dfs1(int x) {
4     wg[x] = 1;
5     for (int y: g[x]) if (y != par[x]) {
6         par[y] = x; h[y] = h[x] + 1; dfs1(y);
7         wg[x] += wg[y];
8     }
9 }
10 int curpos, pos[MAXN], head[MAXN]; // head = representante
11 void hld(int x, int c) {
12     if (c < 0) c = x;
13     pos[x] = curpos++; head[x] = c;
14     int mx = -1;
15     for (int y: g[x]) if (y != par[x] && (mx < 0 || wg[mx] < wg[y])) mx = y;
16     if (mx >= 0) hld(mx, c);
17     for (int y: g[x]) if (y != mx && y != par[x]) hld(y, -1);
18 }
19 void hld_init() { par[0] = -1; h[0] = 0; dfs1(0); curpos = 0; hld(0, -1); }
20 int query(int x, int y, stree& rmq) {
21     int r = NEUT;
22     while (head[x] != head[y]) {
23         if (h[head[x]] > h[head[y]]) swap(x, y);
24         r = oper(r, rmq.query(pos[head[y]], pos[y] + 1));
25         y = par[head[y]];
26     }
27     if (h[x] > h[y]) swap(x, y); // now x is lca
28     r = oper(r, rmq.query(pos[x], pos[y] + 1)); // !!!
29     return r;
30 }

```

4.7 Kahn's topological sort

Time	Space
$\mathcal{O}(VE)$	$\mathcal{O}(V + E)$

```

1
2 vector<vector<int>> adj(M);
3 vector<int> sorted;
4
5 void kahn(int n) {
6     vector<int> indeg(n, 0);
7     vector<bool> valid(n, true);
8     priority_queue<int> pq;
9
10    for (int v = 0; v < n; ++v)
11        for (auto u: adj[v])
12            indeg[u]++;
13    for (int v = 0; v < n; ++v)
14        if (!indeg[v]) pq.push(v);
15
16    while (!pq.empty()) {
17        int v = pq.top(); pq.pop();
18        sorted.push_back(v);
19        valid[v] = false;
20        for (auto u: adj[v])
21            if (valid[u] && (--indeg[u]) == 0)
22                pq.push(u);
23    }
24 }

```

4.8 Max Cardinality Bipartite Matching

$\mathcal{O}(VE)$ time

```

1 vector<vector<int>> adj(M);
2 vector<int> match(M, -1);
3 vector<bool> visited(M);
4
5 bool augment(int left) { //match one on the left with one
   on the right
6     if (visited[left]) return false;
7     visited[left] = true;
8     for (auto right: adj[left])
9         if (match[right] == -1 || augment(match[right])) {
10             match[right] = left;
11             return true;
12         }
13     return false;
14 }
15
16 //usage
17 //(mcbm = V iff there's at least one way to completely
   match both sides)
18 int mcbm = 0; //number of matched vertices
19 match.assign(M, -1);
20 for (int v = 0; v < ls; ++v) { //ls = size of the left set
21     visited.assign(ls, false);
22     mcbm += augment(v);
23 }

```

4.9 LCA - Binary lifting

$\mathcal{O}(n \log n)$ time	$\mathcal{O}(n \log n)$ space
------------------------------	-------------------------------

```

1 int L = //log2(n)
2
3 void dfs(int v, int p) { // uso: dfs(raiz, raiz)
4     up[v][0] = p;
5     for (int l = 1; l <= L; ++l)
6         up[v][l] = up[up[v][l-1]][l-1];
7     for (int u: g[v])
8         if (u != p) dfs(u, v);
9 }
10
11 int lca(int a, int b) {
12     if (dep[b] >= dep[a]) { swap(a, b); }
13     int diff = dep[a] - dep[b];

```

```

14 for (int l = L; l >= 0; l--) if (diff & (1 << l))
15     a = up[a][l];
16 if (a == b) { return a; }
17 for (int l = L; l >= 0; l--) if (up[a][l] != up[b][l])
18     a = up[a][l], b = up[b][l];
19 return up[a][0];
20 }

```

4.10 Tarjan Strongly Connected Component

Time/Space	Usage
$\mathcal{O}(V + E)$	Tarjan(n, adj)

```

1
2 vector<int> tin(M, -1), low(M, -1);
3 vector<bool> vis(M);
4 vector<vector<int>> adj(M);
5 stack<int> S;
6 int tk = 0;
7
8 void dfs(int v) {
9     low[v] = tin[v] = tk++;
10    S.push(v);
11    vis[v] = true;
12    for (auto u: adj[v]) {
13        if (tin[u] == -1)
14            dfs(u);
15        if (vis[u])
16            low[v] = min(low[v], low[u]);
17    }
18    if (low[v] == tin[v])
19        while (true) {
20            int u = S.top(); S.pop(); vis[u] = false;
21            if (u == v) break;
22        }
23 }

```

4.11 LCA - Euler Path

$\mathcal{O}(n \log n)$ time	$\mathcal{O}(n)$ space
------------------------------	------------------------

```

1 using ii = pair<int, int>;
2 vector<int> idx(n);
3 int tk = 1;
4
5 void dfs(int v, int d) { // call with dfs(root, 0);
6     for (auto u: adj[v]) {
7         st.update(tk, {d, v});
8         tk++;
9         dfs(u, d+1);
10    }
11    idx[v] = tk;
12    st.update(tk, {d, v});
13    tk++;
14 }
15
16 int lca(int v, int u) {
17     int l = idx[v], r = idx[u];
18     return st.minquery(l, r).second; // .first is depth
19 }

```

4.12 Kosaraju SCC

Time/Space	Usage
$\mathcal{O}(V + E)$	kosaraju()

rep: representante do componente de cada vtx
scc: 2a dfs, processa os vtx do componente c

```

1 vector<int> S, rep(MAXN);
2
3 void dfs(int v) {
4     vis[v] = true;
5     for (int u: g[v])
6         if (!vis[u]) dfs(u);

```

```

7     S.push_back(v);
8 }
9
10 void scc(int v, int c) {
11     vis[v] = true;
12     rep[v] = c;
13     for (int u: g1[v])
14         if (!vis[u]) scc(u, c);
15 }
16
17 void kosaraju() {
18     for (int i = 0; i < n; ++i)
19         if (!vis[i]) dfs(i);
20     vis.assign(n, false);
21     reverse(S.begin(), S.end());
22     for (int v: order)
23         if (!vis[v]) scc(v, v);
24 }

```

4.13 Bellman-Ford SSSP

Time	Space
$\mathcal{O}(VE)$	$\mathcal{O}(V + E)$

```

1 const int inf = 0x3f3f3f3f;
2 vector<vector<pair<int, int>>> adj(M);
3 vector<int> dist(M, inf);
4
5 void bellmanFord(int n) {
6     for (int i = 0; i < n-1; ++i)
7         for (int v = 0; v < n; ++v)
8             for (auto &[u, w]: adj[v])
9                 if (dist[v] != inf)
10                     dist[u] = min(dist[u], dist[v]+w);
11 }
12
13 //check if there are negative cycles
14 bool cycle(int n) {
15     bool ans = false;
16     for (int v = 0; v < n; ++v)
17         for (auto &[u, w]: v)
18             ans |= dist[v] != inf && dist[u] > dist[v]+w;
19 }

```

4.14 Kruskal MST

Time	Space
$\mathcal{O}(E \log V)$	$\mathcal{O}(V + E)$

```

1 using edge = tuple<ll, int, int>; // peso, u, v
2 vector<edge> edges;
3 UnionFind muf;
4
5 pair<ll, vector<edge>> kruskal(int n) { // n = #vertices
6     vector<edge> mst;
7     ll cost = 0; sort(all(edges));
8     for (auto [w, u, v]: edges)
9         if (!muf.isSameSet(u, v)) {
10             mst.emplace_back(w, u, v);
11             cost += w;
12             muf.unionSet(u, v);
13         }
14     return {cost, mst};
15 }

```

4.15 Edmond Karp MaxFlow

Time	Space
$\mathcal{O}(VE^2)$	$\mathcal{O}(V + E)$

```

1
2 vector<vector<int>>> capacity(M, vector<int>(M, 0)), adj(M);
3 vector<pair<int, int>> mc; //mincut edges
4

```

```

5 int bfs(int s, int t, vi &par) {
6   fill(all(par), -1);
7   par[s] = -2;
8   queue<pair<int, int>> q; q.push({s, inf});
9   while (!q.empty()) {
10    int v = q.front().first,
11        flow = q.front().second;
12    q.pop();
13    for (auto u: adj[v])
14      if (par[u] == -1 && capacity[v][u]) {
15        par[u] = v;
16        int new_flow = min(flow, capacity[v][u]);
17        if (u == t) return new_flow;
18        q.push({u, new_flow});
19      }
20  }
21  return 0;
22 }
23
24 int maxflow(int s, int t) {
25   int flow = 0;
26   vi par(M);
27   int new_flow;
28   while ((new_flow = bfs(s, t, par))) {
29     flow += new_flow;
30     int v = t;
31     while (v != s) {
32       int p = par[v];
33       capacity[p][v] -= new_flow;
34       capacity[v][p] += new_flow;
35       v = p;
36     }
37   }
38   return flow;
39 }
40
41 void mincut(int s, int t) {
42   maxflow(s, t);
43   stack<int> st;
44   vector<bool> visited(n, false);
45   vector<pair<int, int>> ans;
46   st.push(s); // changed from 0 to s
47   while (!st.empty()) {
48     int v = st.top(); st.pop();
49     if (visited[v]) continue;
50     visited[v] = true;
51     for (auto u: adj[v])
52       if (capacity[v][u] > 0)
53         st.push(u);
54     else
55       ans.push_back({v, u});
56  }
57  mc.clear();
58  for (auto &[v, u] : ans)
59    if (!visited[u])
60      mc.push_back({v, u});
61 }

```

4.16 Floyd Warshall APSP

Time	Space	Usage
$\mathcal{O}(V^3 + E)$	$\mathcal{O}(V^2 + E)$	FloydWarshall(n, edges)

```

1 vector<vector<int>> w(MAXN, vector<int>(MAXN, oo));
2 void fw(int n) {
3   for (int m = 0; m < n; ++m)
4     for (int u = 0; u < n; ++u)
5       for (int v = 0; v < n; ++v)
6         if (max(w[u][m], w[m][v]) < oo)
7           w[u][v] = min(w[u][v], w[u][m] + w[m][v]);
8 }

```

5 Math

5.1 Sieve of Eratosthenes

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

```

1
2 bitset<11234567> pr;
3 vector<int> factors(M, 0);
4 vector<int> primes;
5
6 void sieve(int n) {
7   pr.set();
8   for (int i = 2; i*i <= n; ++i)
9     if (pr[i]) { //factors[i] == 0
10      primes.push_back(i);
11      for (int p = i*i; p <= n; p += i) {
12        pr[p] = false;
13        factors[p]++;
14      }
15    }
16 }
17
18 // O(1) for small n, O(sieve_size) else
19 bool isPrime(int n) {
20   int sieve_size = 11234567;
21   if (n <= sieve_size) return pr[n];
22   for (auto p: primes) // only works if n <= primes.back()^2
23     if (!(n%p)) return false;
24   return true;
25 }

```

5.2 Prime Factors w/ Optimized Trial Divisions

Time	Space
$\mathcal{O}(\pi(\sqrt{n}))$	$\mathcal{O}(n)$

```

1
2 vector<int> primes;
3 vector<pair<int, int>> factors;
4
5 void pf(int n) {
6   for (auto p: primes) {
7     if (p*p > n) break;
8     int i = 0;
9     while (!(n%p)) {
10      n /= p;
11      i++;
12    }
13    factors.push_back({p, i});
14  }
15  if (n != 1) factors.push_back({n, 1});
16 }

```

5.3 Extended Euclid for Linear Diophantines

Time	Usage for a,b
$\mathcal{O}(\log \min(a, b))$	int x, y; gcd(a, b, x, y);

```

1 int gcd(int a, int b, int& x, int& y) {
2   if (!b) {
3     x = 1;
4     y = 0;
5     return a;
6   }
7   int x1, y1;
8   int d = gcd(b, a % b, x1, y1);
9   x = y1;
10  y = x1 - y1 * (a / b);
11  return d;
12 }

```


5.4 Floyd's algorithm cycle-finding

$\mathcal{O}(V)$ time

```
1 int findCycle(int x) {
2     int a, b;
3     a = succ(x);
4     b = succ(succ(x));
5     while (a != b) {
6         a = succ(a);
7         b = succ(succ(b));
8     }
9     a = x;
10    while (a != b) {
11        a = succ(a);
12        b = succ(b);
13    }
14    int first = a; // first element in cycle
15    b = succ(a);
16    int length = 1;
17    while (a != b) {
18        b = succ(b);
19        length++;
20    }
21 }
```

6 Paradigm

6.1 Coordinate Compression

Normalize vector access; can also be done with map/set but high constant. $\mathcal{O}(n \log n)$ time

```
1 vector<int> v, vals, cv; // all same size, cv = compressed v
2 vals = v;
3 sort(all(vals));
4 vals.erase(unique(all(vals)), vals.end());
5 for (int i = 0; i < n; ++i)
6     cv[i] = lower_bound(all(vals), v[i]) - vals.begin();
```

6.2 128 Bit Integers

```
1 // cout, cerr, etc; may over/underflow
2 ostream& operator<<(ostream& out, __int128 x) {
3     if (x == 0) return out << 0;
4     string s; bool sig = x < 0; x = x < 0 ? -x : x;
5     while(x > 0) s += x % 10 + '0', x /= 10;
6     if (sig) s += '-';
7     reverse(s.begin(), s.end());
8     return out << s;
9 }
10 // cin, etc; may over/underflow
11 istream& operator>>(istream& in, __int128 x) {
12     char c, neg = 0; while(isspace(c = in.get()));
13     if(!isdigit(c)) neg = (c == '-') ? 1 : 0;
14     else x = c - '0';
15     while(isdigit(c = in.get())) x = (x << 3) + (x << 1) -
16         '0' + c;
17     x = neg ? -x : x; return in;
18 }
```

6.3 Binary Search

```
1 // std
2 int l = 0, r = n-1;
3 while (l <= r) {
4     int m = l+(r-l)/2;
5     if (array[m] == x) // found
6         if (array[m] > x) r = m-1;
7     else l = m+1;
8 }
9 // nice - binary steps
10 int k = 0;
11 for (int b = n/2; b > 0; b /= 2)
12     while (k+b < n && array[k+b] <= x)
```

```
13     k += b;
14 if (array[k] == x) // found
```

7 String

7.1 Rolling hash (linear)

$\mathcal{O}(n)$ time

Let $h_{i..j} = \text{hash}(s_{i..j})$.

$h_{i..j} \times p^i = h_{0..j} - h_{0..i-1}$. Instead of finding the multiplicative inverse of p^i , you can multiply this term by p^{n-i} (so every hash is compared multiplied by p^n).

```
1 ll hash(string const& s) {
2     const int p = 31; // ~alphabet size (31 for lowercase, 53
3         // for uppercase)
4     const int M = 1e9 + 9;
5     ll h = 0;
6     ll p_pow = 1; // precompute for performance
7     for (char c : s) {
8         h = (h + (c - 'a' + 1)*p_pow) % M;
9         p_pow = (p_pow * p) % M;
10    }
11    return h;
```

7.2 Prefix Function (KMP)

$\mathcal{O}(n)$ time

To find occurrences of s in t , use the string $s\%+t$, then look for $pi[i] = s.length()$ on the "t side"

```
1 vector<int> prefix(string s) {
2     int n = s.length();
3     vector<int> pi(n, 0); // can be optimized if you know max
4         // prefix length
5     for (int i = 1; i < n; ++i) {
6         int j = pi[i-1];
7         while (j > 0 && s[i] != s[j])
8             j = pi[j-1];
9         j++;
10        pi[i] = j;
11    }
12    return pi;
13 }
```

7.3 Suffix Array

Build	Query
$\mathcal{O}(n \log n)$	$\mathcal{O}(\log n)$

To find whether p is a substring of s (and where this occurrence starts), you can build the suffix array A of s . Since A is sorted, you can binary search for p as a prefix of all suffixes of s . **Complexity (besides construction):** $\mathcal{O}(|p| \log(|s|))$.

```
1 // sort p by the values in c (stable) ( $\mathcal{O}(|\text{alphabet}| + n)$ )
2 void count_sort(vector<int> &p, vector<int> &c) {
3     int n = p.size();
4     int alphabet = 256; // ascii range
5     vector<int> cnt(max(alphabet, n));
6     for (auto x : c)
7         cnt[x]++;
8
9     vector<int> pos(max(alphabet, n));
10    pos[0] = 0;
11    for (int i = 1; i < max(alphabet, n); ++i)
12        pos[i] = pos[i-1] + cnt[i-1];
13
14    vector<int> p_sorted(n);
15    for (auto x : p) {
16        p_sorted[pos[c[x]]++] = x;
17    }
```

```

18
19 p = p_sorted;
20 }
21
22 // build suffix array
23 vector<int> suffix_array(string s) {
24     s += "$";
25     int n = s.size();
26     // at k = 2^0, sort strings of length 1
27     vector<int> p(n), c(n); // suffix start position,
        equivalence class
28     for (int i = 0; i < n; ++i) {
29         p[i] = i;
30         c[i] = s[i];
31     }
32     // at first c is just a hack to sort p, it's not really
        equiv. class
33     count_sort(p, c);
34     // but then it is
35     c[p[0]] = 0;
36     for (int i = 1; i < n; ++i) {
37         c[p[i]] = c[p[i-1]];
38         if (s[p[i]] != s[p[i-1]])
39             c[p[i]]++;
40     }
41     int k = 1;
42     while (k < n) {
43         // transition from k to k+1
44         for (int i = 0; i < n; ++i)
45             p[i] = (p[i] - k + n) % n;
46         count_sort(p, c);
47         // recalculate equiv.
48         vector<int> c_upd(n);
49         c_upd[p[0]] = 0;
50         for (int i = 1; i < n; ++i) {
51             pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + k)%n]};
52             pair<int, int> curr = {c[p[i]], c[(p[i] + k)%n]};
53             c_upd[p[i]] = c_upd[p[i-1]];
54             if (curr != prev)
55                 c_upd[p[i]]++;
56         }
57         c = c_upd;
58         k <= 1;
59     }
60     return p;
61 }

```

8 Structure

8.1 Merge/Disjoint Union-Find

Time	Space	Usage
$\mathcal{O}(A \times n)$	$\mathcal{O}(n)$	muf(n)

```

1 struct muf {
2     int N;
3     vector<int> par, rk, count;
4
5     muf(int N) : N(N), par(N), rk(N, 0), count(N, 1) {
6         for (int i = 0; i < N; ++i)
7             par[i] = i;
8     }
9
10    int findSet(int i) {
11        return par[i] == i ? i : (par[i] = findSet(par[i]));
12    }
13
14    int unite(int a, int b) {
15        int x = findSet(a), y = findSet(b);
16        if (x != y)
17            count[x] = count[y] = (count[x]+count[y]);
18        if (rk[x] < rk[y])
19            par[x] = y;
20        else {
21            par[y] = x;
22            if (rk[x] == rk[y])

```

```

23         rk[x]++;
24     }
25     return count[x];
26 }
27
28 bool sameSet(int i, int j) {
29     return findSet(i) == findSet(j);
30 }
31 };

```

8.2 Bottom-Up Segment Tree

Build	Query	Update	Usage
$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	seg(n)

Uses less space than top-down $4n$ segtree ($2n$ here)

```

1 struct seg {
2     int n;
3     vector<int> t;
4
5     seg(vector<int> v) : n(v.size()), t(2*n) {
6         for (int i = 0; i < n; ++i)
7             upd(i, v[i]);
8     }
9     seg(int sz) : n(sz), t(2*n) {}
10
11    int query(int a, int b) {
12        a += n, b += n;
13        int ans = 0;
14        for (a += n, b += n; a <= b; ++a /= 2, --b /= 2) {
15            if (a%2 == 1) ans += t[a];
16            if (b%2 == 0) ans += t[b];
17        }
18        return ans;
19    }
20
21    void upd(int p, int x) {
22        t[p += n] = x;
23        while (p /= 2) t[p] = t[p<<1] + t[(p<<1)+1];
24    }
25 };

```

8.3 Segment Tree

Build	Query	Modify	Usage
$\mathcal{O}(n \log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	stree(n)

```

1 struct stree {
2     int n;
3     vector<int> st, v;
4
5     stree(vector<int> v) : n(v.size()), st(4*n), v(v) {
6         build(1, 0, n-1);
7     }
8
9     int left(int i) { return i<<1; }
10    int right(int i) { return (i<<1)+1; }
11
12    void build(int p, int pl, int pr) {
13        if (pl == pr) {
14            st[p] = v[pl];
15            return;
16        }
17        int m = (pl+pr)/2;
18        build(left(p), pl, m);
19        build(right(p), m+1, pr);
20        st[p] = min(st[left(p)], st[right(p)]);
21    }
22
23    int query(int p, int pl, int pr, int ql, int qr) {
24        // same params as update, except [ql..qr] is the query
        range
25        if (qr < pl || ql > pr) return inf;
26        if (ql <= pl && pr <= qr) return st[p];
27        int m = (pl+pr)/2;

```

```

28 int query_left = query(left(p), pl, m, ql, qr);
29 int query_right = query(right(p), m+1, pr, ql, qr);
30 return min(query_left, query_right);
31 }
32
33 int query(int ql, int qr) { return query(1, 0, n-1, ql,
    qr); }
34
35 void update(int p, int pl, int pr, int i, int x) {
36     // p = st idx, corresponds to range [pl..pr]
37     if (i < pl || i > pr) return;
38     if (pl == pr) {
39         st[p] = x;
40         return;
41     }
42     int m = (pl+pr)/2;
43     update(left(p), pl, m, i, x);
44     update(right(p), m+1, pr, i, x);
45     st[p] = min(st[left(p)], st[right(p)]);
46 }
47
48 void update(int i, int x) { update(1, 0, n-1, i, x); }
49 };

```

9 Extra

9.1 C++ structs

```

1 struct st {
2     vector<int> a;
3     vector<bool> b = vector<bool>(5); // default value
4     int i;
5     st(int _i) : a(_i), i(_i) {};
6     bool operator< (st& e) const { return i < e.i; }
7 };
8
9 st e = st(3); st f(3);
10
11 struct matrix {
12     vector<vector<int>> m;
13     matrix(int n) m(n, vector<int>(n)) {};
14     matrix operator * (const matrix &b) {
15         matrix c = matrix();
16         for (int i = 0; i < m.size(); ++i)
17             for (int j = 0; j < m.size(); ++j)
18                 for (int k = 0; k < m.size(); ++k)
19                     c.m[i][j] = c.m[i][j] + 1LL*m[i][k]*b.m[k][j];
20         return c;
21     }
22 };

```

9.2 cmp

```

1 // upper_bound: 1st > x, lower_bound: 1st >= x
2 // last <= x: up-1, first >= x: lo
3 priority_queue<int, vector<int>, greater<int>> pq;
4 struct {
5     bool operator()(const int& a, const int& b) const {
6         return a < b;
7     }
8 } cmp;
9 priority_queue<int, vector<int>, cmp> pq2;
10 sort(v.begin(), v.end(), cmp);

```

9.3 Vim

```

1 set et ts=2 sw=2 ai si cindent sta is tm=50 nu noeb sm "cul
2 sy on

```

9.4 Generator

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(int argc, char *argv[]) {
5     cin.tie(0); ios_base::sync_with_stdio(0);

```

```

6     if (argc < 2) {
7         cout << "usage: " << argv[0] << " <seed>\n";
8         exit(1);
9     }
10    srand(atoi(argv[1]));
11    // use rand() for random value
12 }

```

9.5 Makefile

```

1 # p3: pypy3 -m py_compile
2 CXX = g++
3 CXXFLAGS = -Wall -Wconversion -Wfatal-errors -g -O2
4 -std=gnu++20 -fsanitize=address,undefined -Wshadow
5 -fno-omit-frame-pointer -Wno-unused-result
6 -Wno-sign-compare -Wno-char-subscripts #-fuse-ld=gold

```

9.6 C++ Template

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define all(x) begin(x), end(x)
4 #define endl '\n'
5 using ll = long long;
6 using ii = pair<int, int>;
7 using vv = pair<ll, int>;
8
9 signed main() {
10    cin.tie(0)->sync_with_stdio(0);
11 }

```

9.7 Stress

```

1 for (( I=0; I < 5; I++ )); do
2     ./gen $I > z.in
3     ./brute < z.in > expected.txt
4     ./prog < z.in > output.txt
5     if diff -u expected.txt output.txt; then : ; else
6         echo "--> input (z.in):"; cat z.in
7         echo "--> expected output:"; cat expected.txt
8         echo "--> received output:"; cat output.txt
9         break
10    fi
11    echo -n .
12 done

```