Competitive Programming Notebook

Raul Almeida[1]

[1]Universidade Federal do Paraná

October 25, 2023

Contents

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 ($n \leq 10^9$)

10^8 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}	$\approx 10^3$
2^{20}	$\approx 10^6$

Sign	Type	Bits	Max	Digits
$\pm$	char	8	127	2
$+$	unsigned char	8	255	2
$\pm$	short	16	32 767	4
$+$	unsigned short	16	65 535	4
$\pm$	int/long	32	$\approx 2 \cdot 10^9$	9
$+$	unsigned int/long	32	$\approx 4 \cdot 10^9$	9
$\pm$	long long	64	$\approx 9 \cdot 10^{18}$	18
$+$	unsigned long long	64	$\approx 18 \cdot 10^{18}$	19
$\pm$	__int128	128	$\approx 17 \cdot 10^{37}$	38
$+$	unsigned __int128	128	$\approx 3 \cdot 10^{38}$	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	$\mathcal{O}(VE)$	$\mathcal{O}(V + E)$
Dijksta	$\mathcal{O}((V + E) \log V)$	$\mathcal{O}(V^2)$
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
$\pi(x)$	4	25	168	1 229
x	10^5	10^6	10^7	10^8
$\pi(x)$	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

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

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

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

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

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

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

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

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

$$a^p = a \pmod{p}$$
$$a^{p-1} = 1 \pmod{p}$$
$$(a+b)^p = a^p + b^p \pmod{p}$$
$$a^{-1} = a^{p-2} \pmod{p}$$

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.

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

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 conecting 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

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

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 $A, B > 0$, $g = \mathrm{GCD}(A, B)$, $A = ag$ and $B = bg$.

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

1.15 Mod value range

$A < B \implies A\%B = A$, $A \geq B \implies A\%B \leq A/2$

1.16 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

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

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

With repetition: elements may be chosen more than once. $\binom{n}{k} = \frac{(k+n-1)!}{k!(n-1)!}$

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

1.19 90deg rot matrix

```
m[i][j] = m[N-j-1][i]
```

1.20 Max LCP in list of strings

Pra pegar o maior LCP numa lista de strings, basta ordenar ela e comparar cada string i com a $i - 1$ e a $i + 1$

1.21 Together Square

Let $f(N)$ be the largest square divisor of an integer N, and $P(x)$ whether x is a square number.

For integers i, j, $P(i \times j) \iff \frac{i \times j}{f(i) \times f(j)}$.

Since $\frac{i}{f(i)}$ is indivisible by a prime p twice or more, $P(\frac{i \times j}{f(i) \times f(j)}) \iff \frac{i}{f(i)} = \frac{j}{f(j)}$

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)

$\mathcal{O}(n \log n)$, ans is vector of point indexes

```
1  using pti = pair<pt, int>;
2  #define fi first
3  #define se second
4  vi convex_hull(vector<pti>& ps, bool col = false) {
5    int k = 0, n = ps.size(); vi ans (2*n);
6    sort(all(ps), [](pti a, pti b) {
7      return make_pair(a.fi.px, a.fi.py) < make_pair(b.fi.px,
           b.fi.py);
8    });
9    for (int i = 0; i < n; i++) {
10     while (k >= 2 && !ccw( /* lower hull */
11         ps[ans[k-2]].fi, ps[ans[k-1]].fi, ps[i].fi, col)) {
             k--; }
12     ans[k++] = i;
13   }
14   if (k == n) {
15     ans.resize(n);
16     for (auto &i : ans) i = ps[i].second;
17     return ans; }
18   for (int i = n-2, t = k+1; i >= 0; i--) {
19     while (k >= t && !ccw( /* upper hull */
20         ps[ans[k-2]].fi, ps[ans[k-1]].fi, ps[i].fi, col)) {
             k--; }
21     ans[k++] = i;
22   }
23   ans.resize(k-1);
24   for (auto &i : ans) i = ps[i].second;
25   return ans;
26 }
```

4 Graph

4.1 Prim MST

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

```
1  vi par(N, -1);
2  vector<ll> d(N, oo);
3  vector<vector<wv>> mst(N);
4
5  ll prim(int s) {
6    ll sum = 0;
7    priority_queue<wv, vector<wv>, greater<wv>> Q;
8    auto add = [&](int v, ll x, int p) {
9      if (x < d[v]) {
10       Q.emplace(d[v]=x, v);
11       par[v] = u;
```

```
12   }};
13   add(s, 0, s);
14   while (Q.size()) {
15     auto [w, v] = Q.top(); Q.pop();
16     if (vis[u]) continue;
17     vis[u] = true;
18     if (par[u] != -1) {
19       int p = par[v]; ll w = d[v];
20       mst[v].emplace_back(w, u);
21       mst[u].emplace_back(w, v);
22     }
23     sum += w;
24     for (auto [c, u] : g[v])
25       if (!vis[u])
26         add(u, c, v);
27   }
28   return sum;
29 }
```

4.2 Dijkstra SSSP

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

```
1  vi 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<vi> adj(M);
3  vi 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  vi tin(M, -1);
```

```
4 vector<vi> 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<vi> 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 FFEK MaxFlow

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

```
1
2 vector<vi> capacity(M, vi(M, 0)), adj(M);
3 vector<ii> mc; //mincut edges
4
5 int bfs(int s, int t, vi &par) {
6   fill(all(par), -1);
7   par[s] = -2;
8   queue<ii> 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<ii> 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 }
```

4.7 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 vi 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 }
```

4.8 Kahn's topological sort

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

```cpp
1
2 vector<vi> adj(M);
3 vi sorted;
4
5 void kahn(int n) {
6   vi 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]))
22        pq.push(u);
23  }
24 }
```

4.9 MCBM (Kunh Matching)

$\mathcal{O}(VE)$ time

Max Cardinality Bipartite Matching

g: arcos do lado esquerdo pro lado direito do grafo

```cpp
1 vi mat;
2 vector<bool> vis;
3
4 bool match(int v) {
5   if (vis[v]) return false;
6   vis[v] = true;
7   for (int u: g[v])
8     if (mat[u] < 0 || match(mat[u])) {
9       mat[u] = v;
10      return true;
11    }
12  return false;
13 }
14
15 mat.assign(n, -1);
16 int mcbm = 0; // num matched vertices
17 for (int i = 0; i < n; ++i) {
18   vis.assign(n, false);
19   mcbm += try_kuhn(i);
20 }
21 // match: mat[i] -> i (i é do lado direito)
```

4.10 LCA - Binary lifting

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

```cpp
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))
```

```cpp
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 }
```

4.11 Tarjan Strongly Connected Component

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

```cpp
1
2 vi tin(M, -1), low(M, -1);
3 vector<bool> vis(M);
4 vector<vi> 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 }
```

4.12 LCA - Euler Path

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

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

4.13 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

```cpp
1 vi 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: gi[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(all(S));
22    for (int v: order)
23      if (!vis[v]) scc(v, v);
24  }
```

4.14 Bellman-Ford SSSP

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

```
 1  const int inf = 0x3f3f3f3f;
 2  vector<vector<ii>> adj(M);
 3  vi 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  }
```

4.15 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  }
```

4.16 Floyd Warshall APSP

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

```
 1  vector<vi> w(MAXN, vi(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  vi factors(M, 0);
 4  vi 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
$\mathcal{O}(\pi(\sqrt{n}))$	$\mathcal{O}(n)$

`lcm` is the same size as `primes` and stores the exponent of each prime in the LCM of all factorized numbers.

Useful to calculate LCM mod some P. You can also use map

```
 1
 2  vi primes, lcm;
 3
 4  vector<ii> pf(int n) {
 5    vector<ii> factors;
 6    rep(i,0,primes.size()) {
 7      int p = primes[i];
 8      if (p*p > n) break;
 9      int e = 0;
10      while (!(n%p)) {
11        n /= p;
12        e++;
13      }
14      factors.push_back({p, e});
15      lcm[i] = max(lcm[i], e);
16    }
17    if (n != 1) {
18      factors.push_back({n, 1});
19      int ix = lower_bound(all(primes), n) - primes.begin();
20      lcm[ix] = max(lcm[ix], 1);
21    }
22    return factors;
23  }
```

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);`

```cpp
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

```cpp
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

```cpp
1  vi 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

```cpp
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(all(s));
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 == '-'), x = 0;
14   else x = c - '0';
```

```cpp
15   while(isdigit(c = in.get())) x = (x << 3) + (x << 1) -
         '0' + c;
16   x = neg ? -x : x; return in;
17 }
```

6.3 Prefix AND

$\mathcal{O}(\log N)$ bitwise and on array, $\mathcal{O}(N \log N)$ build

```cpp
1  vector<vi> ps(n, vi(32));
2  const int L = 32; // teto(log2(max(v[i])))
3
4  void build(vi &v) {
5    int n = v.size();
6    for (int i = 0; i < n; ++i) // build
7      for (int b = 0; b < L; ++b) {
8        if (i > 0) ps[i][b] = ps[i-1][b];
9        if (v[i] & (1ll << b)) ps[i][b]++;
10     }
11 }
12
13 int qry(int l, int r) {
14   int num = 0;
15   for (int b = 0; b < L; ++b)
16     // ligado em todos
17     if (ps[r][b] - (l ? ps[l-1][b] : 0) == r-l+1)
18       num |= (1ll << b);
19   return num;
20 }
```

6.4 Binary Search

```cpp
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 Problems

7.1 Tree Distances I

Given a tree of n nodes, determine for each node its maximum distance to another node

```cpp
1  vi d(maxn);
2
3  void dfs1(int v, int p=-1) { // d[v] = max dist below
4    d[v] = 0;
5    for (int u : g[v])
6      if (u != p) {
7        dfs1(u, v);
8        d[v] = max(d[v], 1+d[u]);
9      }
10 }
11
12 void add(int &a, int &b, int c) {
13   if (c > a)
14     tie(a, b) = make_pair(c, a);
15   else if (c > b)
16     b = c;
17 }
18
19 void dfs2(int v, int a, int p=-1) { // include dist above
20   // a,b = max and 2nd max subtrees
21   int b = 0;
22   for (int u : g[v])
23     if (u != p)
```

```
24       add(a, b, 1+d[u]);
25   for (int u : g[v])
26     if (u != p) {
27       int dist = (1+d[u]) == a ? b : a;
28       dfs2(u, dist+1, v);
29     }
30   d[v] = max(d[v], a);
31 }
32
33 signed main() {
34   dfs1(src);
35   dfs2(src, 0);
36   rep(i,0,n)
37     cout << d[i] << '\n';
38 }
```

7.2 Coin Combinations II

Given money system of n coins $c_i > 0$. How many distinct **ordered** ways to produce sum x using these coins? (use each one as many time as you want, order doesn't matter i.e. $2 + 2 + 5 = 2 + 5 + 2$)

```
1 int n, x;
2 cin >> n >> x;
3 vector<int> c(n);
4
5 for (auto &i: c)
6   cin >> i;
7
8 vector<int> dp(x+1);
9 const int M = 1000000007;
10
11 dp[0] = 1;
12 for (int k: c)
13   for (int j = k; j <= x; ++j)
14     dp[j] = (dp[j] + dp[j-k]) % M;
15
16 cout << dp[x] << '\n';
```

7.3 Kth smallest sum

Given an array A, sum at least one element, may use the same element multiple times. Find K-th smallest possible sum

$1 \le N \le 10, 1 \le K \le 2 \times 10^5, 1 \le A_i \le 10^9$

```
1 int n, k;
2 cin >> n >> k;
3 vector<int> a(n);
4 priority_queue<int, vector<int>, greater<int>> pq;
5 set<int> seen;
6 for (int &x : a) {
7   cin >> x;
8   if (seen.find(x) == seen.end()) {
9     pq.push(x);
10     seen.insert(x);
11   }
12 }
13 int cnt = 0;
14 while (pq.size()) {
15   int t = pq.top(); pq.pop();
16   if (++cnt == k) {
17     cout << t << '\n';
18     break;
19   }
20   for (int x : a)
21     if (seen.find(t+x) == seen.end()) {
22       pq.push(t + x);
23       seen.insert(t+x);
24     }
25 }
```

7.4 Art Gallery on Graph

Given graph with $N \le 2 \times 10^5$ vertices, $M \le 2 \times 10^5$ edges.

$K \le N$ security guards are on some vertices, each with a different stamina h_i.

Vertex v is guarded if there is a guard i with distance $\le h_i$
List all guarded vertices

```
1 priority_queue<pair<int, int>> pq;
2 vector<int> dist(n, -1);
3
4 for (int i = 0; i < k; ++i) {
5   int p, h;
6   cin >> p >> h;
7   --p;
8   pq.emplace(dist[p] = h, p);
9 }
10
11 while (pq.size()) {
12   auto [h, p] = pq.top(); pq.pop();
13   if (h != dist[p])
14     continue;
15   for (int &q : g[p])
16     if (h-1 > dist[q])
17       pq.emplace(dist[q] = h-1, q);
18 }
19
20 vector<int> ans;
21 for (int i = 0; i < n; ++i)
22   if (dist[i] >= 0)
23     ans.push_back(i+1);
```

7.5 Coin Combinations I

Given money system of n coins $c_i > 0$. How many ways to produce sum x using these coins? (use each one as many time as you want, order matters i.e. $2 + 2 + 5 \ne 2 + 5 + 2$)

```
1 int n, x;
2 cin >> n >> x;
3 vector<int> c(n);
4 for (auto &i : c)
5   cin >> i;
6 const int M = int(1e9)+7;
7 vector<int> ans(x+1);
8 ans[0] = 1;
9 for (int i = 1; i <= x; ++i)
10   for (int ci : c)
11     if (ci <= i)
12       ans[i] = (ans[i] + ans[i-ci]) % M;
13 cout << ans[x] << '\n';
```

8 String

8.1 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
        for uppercase)
3   const int M = 1e9 + 9;
4   ll h = 0;
5   ll p_pow = 1; // precompute for performance
6   for (char c : s) {
7     h = (h + (c - 'a' + 1)*p_pow) % M;
8     p_pow = (p_pow * p) % M;
9   }
10   return h;
11 }
```

8.2 Prefix Function (KMP)

$\mathcal{O}(n)$ time

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

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

8.3 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 ocurrence 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) (O(|alphabet| + n))
2  void count_sort(vi &p, vi &c) {
3    int n = p.size();
4    int alphabet = 256; // ascii range
5    vi cnt(max(alphabet, n));
6    for (auto x : c)
7      cnt[x]++;
8
9    vi 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   vi 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 vi suffix_array(string s) {
24   s += "$";
25   int n = s.size();
26   // at k = 2^0, sort strings of length 1
27   vi 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     vi c_upd(n);
49     c_upd[p[0]] = 0;
50     for (int i = 1; i < n; ++i) {
51       ii prev = {c[p[i-1]], c[(p[i-1] + k)%n]};
52       ii 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 }
```

8.4 Palindromic Tree

árvore de palíndromos + cada letra tem um custo e vc quer maximizar (pra tirar o custo só remover sigmac e cost)

Construção $\mathcal{O}(|S|\log\Sigma)$, S = sua string. Cada índice é um nó, e cada nó representa um palíndromo diferente.

to[i][c] é o vértice do palíndromo cic

link[i] é o vértice do maior sufixo palindrômico de i

len[i] é o tamanho do palíndromo i

s[i] é o primeiro caractere no palíndromo i (isso é altamente duvidável)

```
1  int sigmac[26];
2
3  const int maxn = 5e5, sigma = 26;
4
5  int s[maxn], len[maxn], link[maxn], to[maxn][sigma],
        cost[maxn];
6
7  int n, last, sz;
8
9  void init()
10 {
11     s[n++] = -1;
12     link[0] = 1;
13     len[1] = -1;
14     cost[0] = 0;
15     sz = 2;
16 }
17
18 int get_link(int v)
19 {
20     while(s[n - len[v] - 2] != s[n - 1]) v = link[v];
21     return v;
22 }
23
24 void add_letter(int c)
25 {
26     s[n++] = c;
27     last = get_link(last);
28     if(!to[last][c])
29     {
30         int _c = len[last] == -1 ? sigmac[c] : (sigmac[c]*2);
31         cost[sz] = cost[last] + _c;
32         len [sz] = len[last] + 2;
33         link[sz] = to[get_link(link[last])][c];
34         to[last][c] = sz++;
35     }
36     last = to[last][c];
37 }
38
39 signed main() {
40   cin.tie(0)->sync_with_stdio(0);
41   for (int i = 0; i < 26; ++i)
42     cin >> sigmac[i];
43   string _s;
44   cin >> _s;
45   init();
46   for (char c : _s)
47     add_letter(c-'a');
```

```
48    int ans = -1e18;
49    for (int i = 0; i < maxn; ++i)
50      if (len[i] > 0)
51        ans = max(ans, cost[i]);
52    cout << ans << '\n';
53 }
```

9   Structure

9.1   MUF with distance to root

`par[i].se` tem a distancia, nesse caso só precisa da paridade então usamos xor. Se quiser dist toda, usa soma (!!!)

`merge` retorna 1 se o componente for bipartido

```
1 struct MUF {
2   int n;
3   vector<ii> par;
4   vi rk, bip; // bipartite check
5
6   MUF(int n) : n(n), par(n), rk(n, 0), bip(n, 1) {
7     rep(i,0,n)
8       par[i] = {i,0};
9   }
10
11  ii find(int i) {
12    if (par[i].fi != i) {
13      int p = par[i].se;
14      par[i] = find(par[i].fi);
15      par[i].se ^= p; // !!!
16    }
17    return par[i];
18  }
19
20  int merge(int a, int b) {
21    int x, y;
22    tie(a, x) = find(a);
23    tie(b, y) = find(b);
24    if (a == b)
25      bip[a] &= x != y;
26    if (rk[a] < rk[b])
27      swap(a, b);
28    par[b] = {a, x^y^1}; // !!!
29    bip[a] &= bip[b];
30    rk[a] += rk[a] == rk[b];
31    return bip[a];
32  }
33
34  bool same(int i, int j) {
35    return find(i) == find(j);
36  }
37 };
```

9.2   Merge/Disjoint Union-Find

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

```
1 struct MUF {
2   int n;
3   vi par, rk, sz;
4
5   MUF(int n) : n(n), par(n), rk(n, 0), sz(n, 1) {
6     rep(i,0,n)
7       par[i] = i;
8   }
9
10  int find(int i) {
11    return par[i] == i ? i : (par[i] = find(par[i]));
12  }
13
14  int merge(int a, int b) {
15    a = find(a); b = find(b);
16    if (a != b)
17      sz[a] = sz[b] = (sz[x] + sz[y]);
```

```
18    if (rk[a] < rk[b])
19      swap(a, b);
20    par[b] = a;
21    rk[a] += rk[a] == rk[b];
22    return sz[x];
23  }
24
25  bool same(int i, int j) {
26    return find(i) == find(j);
27  }
28 };
```

9.3   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   vi t;
4
5   seg(vi 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    int ans = 0;
13    for (a += n, b += n; a <= b; ++a /= 2, --b /= 2) {
14      if (a%2 == 1) ans += t[a];
15      if (b%2 == 0) ans += t[b];
16    }
17    return ans;
18  }
19
20  void upd(int p, int x) {
21    t[p += n] = x;
22    while (p /= 2) t[p] = t[p<<1] + t[(p<<1)+1];
23  }
24 };
```

9.4   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   vi st, v;
4
5   stree(vi 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 };
```

10 Extra

10.1 C++ structs

```
1 struct st {
2   vi 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<vi> m;
13   matrix(int n) m(n, vi(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 };.
```

10.2 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(all(v), cmp);
```

10.3 Vim

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

10.4 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 }
```

10.5 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
```

10.6 C++ Template

```
1 #include <bits/stdc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 using namespace std;
4 using namespace __gnu_pbds;
5 #define fi first
6 #define se second
7 #define rep(i,a,b) for (int i = (a); i < (b); ++i)
8 #define all(x) begin(x), end(x)
9 #define endl '\n'
10 #define int long long
11 using vi = vector<int>;
12 using ll = long long;
13 using ii = pair<int, int>;
14 using wv = pair<ll, int>;
15 typedef tree<int,null_type,less<int>,rb_tree_tag,
16 tree_order_statistics_node_update> indexed_set;
17
18 signed main() {
19   cin.tie(0)->sync_with_stdio(0);
20 }
```

10.7 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
```