Competitive Programming Notes

Raul Almeida

Contents

1 Tables

n	not-TLE	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
$\leq 2 \cdot 10^3$	$\mathcal{O}(n^2 \log n)$	2 nested loops + tree query
$\leq 5 \cdot 10^4$	$\mathcal{O}(n^2)$	Bubble/Selection/Insertion Sort
$\leq 10^5$	$\mathcal{O}(n \log^2 n)$	Build suffix array
$\leq 10^6$	$\mathcal{O}(n \log n)$	Merge Sort
$\leq 10^7$	$\mathcal{O}(n \log \log n)$	Totient function
$\leq 10^8$	$\mathcal{O}(n)$	Mathy solution often with IO bottleneck ($n \leq 10^9$)

10^8 operations per second

Sign	Type	Bits	Max	Digits
$\pm$	`char`	8	127	2
$+$	`char`	8	255	2
$\pm$	`short`	16	32 767	4
$+$	`short`	16	65 535	4
$\pm$	`int/long`	32	$2 \cdot 10^9$	9
$+$	`int/long`	32	$4 \cdot 10^9$	9
$\pm$	`long long`	64	$9 \cdot 10^{18}$	18
$+$	`long long`	64	$18 \cdot 10^{18}$	19
$\pm$	`__int128`	128	$17 \cdot 10^{37}$	38
$+$	`__int128`	128	$3 \cdot 10^{38}$	38

2 Algorithms

Algorithm	Time	Space
Articulations and Bridges	$\mathcal{O}(V+E)$	$\mathcal{O}(V+E)$
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)	$\mathcal{O}(V)$	$\mathcal{O}(1)$
Prime Fac. w/ Opt. Trial Div.	$\mathcal{O}(\pi(\sqrt{n}))$	$\mathcal{O}(n)$
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	$\mathcal{O}(\log N)$	$\mathcal{O}(N)$

A: Ackermann function

2.1 Graph

2.1.1 Articulations and Bridges

If vertex v is an **articulation point** and you remove it, the connected component to which it belongs becomes disconnected

If edge u, v is a **bridge** and you remove it, you can't reach v from u

2.1.2 Edmond Karp MaxFlow

Ford-Fulkerson's method with BFS $\rightarrow$ $\mathcal{O}(VE)$ BFS calls, $\mathcal{O}(E)$ per BFS

Vertex weights: if vertex V has a weight, create V_{in} (receives all in-edges of V and has an edge to V_{out}) and V_{out}(receives an edge from V_{in} and has all out-edges of V); edge $\{V_{in}, V_{out}\}$ has the weight from V

MinCut: run EdmondKarp; $S - T$ sets are: all V that you can reach from the source with edges of positive residual capacity and all other V

MultiSource/MultiSink: create a super source with infinite capacity pointing to all sources, analogous for sinks

Max Cardinality Bipartite Matching: use capacity 1 on all edges and apply the multi-source and multi-sink strategies

2.1.3 Euler Tour

Find the closest neighbor that has a path back to the current vertex to build an euler tour

Euler path: visits each edge once

Tour/cycle/circuit euler path that starts and ends at same node

Undirected and has path: every vertex has even degree or two have odd degree

Undirected and has circuit: every vertex has even degree

Directed and has path: $\delta^+(v) - \delta^-(v) = 1$ for at most one v, $= -1$ for at most one v, $= 0$ for all other v

Directed and has circuit: $\delta^+(v) = \delta^-(v) \forall v \in V$

2.1.4 Floyd Warshall

Also works for SSSP (V <= 400)

Printing path: `p[i][j]` set to `i` (last node that appears before `j` on the path), then `p[i][j] = p[k][j]` on update.

Transitive Closure: weight is boolean (init as `1` if there's an edge), update with bitwise OR

Minimax/Maximin: `w[i][j]` will be `min(w[i][j], max(w[i][k], w[k][j]))`

Finding negative/cheapest cycle: init `w[i][i] = inf`; `run()`; any `w[i][i] != inf` is a cycle and the smallest is the cheapest; any `w[i][i] < 0` is negative cycle

This can also be used for finding SCCs (check with transitive closure)

2.1.5 Kahn's Topological Sort

Particular order (alphabetical)

2.1.6 Kruskal

Order edges by increasing weight, then use a MUF to know if each edge is useful (if it connects two previously disconnected vertices)

Min Span Subgraph: previously process fixed edges

Min Span Forest: count number of sets on the MUF

2nd Best MST: run kruskal; for each chosen edge, flag it as unavailable and run it without using that edge ($O(VE)$)

Minimax: max edge weight on the MST (maximin: min)

2.1.7 Lowest Common Ancestor

Binary lift to binary search the LCA or Euler Path

2.1.8 Max Cardinality Bipartite Matching

Jump from free to matched edges until you've used them all

2.1.9 Prim's Algorithm

Take smallest edge that leads to vertex v

2.1.10 Tarjan

A node can reach any other node in its own SCC (DFS + stack)

2.1.11 Kosaraju

Get topological sort of a graph and then run DFS on the transposed graph following this topological sort.

Let C and C' be two strongly connected components in the graph G. If there is an edge $\{C, C'\}$, then after computing `tout` and `tin` in a DFS, `tout[C] > tout[C']`. Proof:

- If `tin[C] < tin[C']`, C shows up first in the DFS, and since there is an edge to C', it will be in C's subtree in the DFS tree; so `tout[C] > tout[C']`
- If `tin[C] > tin[C']`, C' shows up first, but since it has no path to C, C will show up later and then have a greater `tout` value.

So when you do DFS on the transposed graph following topological sort, you will start on the root vertex (because it'll have the largest `tout` value), and only be able to visit the nodes in its SCC (because the edges that would lead to other SCCs doesn't exist in the transposed graph).

Remember that you have to "disable" the vertices in already processed SCCs.

2.2 Math

2.2.1 Floyd

Slow and fast (tortoise and hare)

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

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

2.3 Paradigm

2.3.1 Coordinate Compression

Normalize vector access; can also be done with map/set but high constant

2.3.2 128 Bit Integers

GCC extension; 2^{127} 10^{38}

2.4 String

2.4.1 Prefix Function (KMP)

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

3 Emergency

3.1 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).

3.2 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.

3.3 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).

3.4 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?

3.5 Memory limit exceeded

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