Competitive Programming Notes

Raul Almeida

Contents

1 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

1.1 Graph

1.1.1 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

1.1.2 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

1.1.3 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: $\delta^+(v) = \delta^-(v) \forall v \in V$

1.1.4 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)

1.1.5 Kahn's Topological Sort

Particular order (alphabetical)

1.1.6 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)

1.1.7 Lowest Common Ancestor

Binary lift to binary search the LCA or Euler Path

1.1.8 Max Cardinality Bipartite Matching

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

1.1.9 Prim's Algorithm

Take smallest edge that leads to vertex v

1.1.10 Tarjan

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

1.2 Math

1.2.1 Floyd

Slow and fast (tortoise and hare)

1.3 Paradigm

1.3.1 Coordinate Compression

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

1.3.2 128 Bit Integers

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

1.4 String

1.4.1 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"