IF Tree edge then N1->n2 if n1 outer and n2 inner but not necessary otherwise.

Non tree edge forw back and cross edge( (n2) (n1) n1 to n2

DFS- visited if 0 then explore on that , so backtracks later first immediate . Connected comp, cycles, partition

BFS- dist and queue. Dequeu and check neighbor if dist is infi then update and enqueu so they can be expanded after. Shortest dist with no weights. Make into bipartite or constraint problem. With additional variables like color can be used.

DAG- post dec to increasing so left to right nodes all as no cycles so backward edge so this possible.

Sort in Linear time?

SCC – path u to v and v to u. graph broken into scc’s overall superDAG

Djikstra- weight based shortest src to sink . priority queue so heaps. Src dist 0 and pq works on this dist sort, so extract min then expand child and check for their update and decrease key. After this don’t put u again in queue.

Fibonacci heap O(n) build, Ologn extract min and decrease in O(1).

O(m+nlogn) running tie djikstra best case, Fibonacci heap else O(n+m)logn) n\*extractmin+m\*decrease

If -ve edge= infi loop djikstra, but if -eve in dag will still work as will end graph .

Dag shortest path- DFS- Topological sort – now from highest post or left to right go on updating child if dist update required. O(n+m)

Bellman outer loop i 1 to n-1 times, so every step I we look at most i edges. This is because initial dist[s]=0, then step 1 we look at immediate neighbor and update them. The next step we also include step=2 so two nodes so one more depth for all nodes with no depth as inifinity.O(nm) but for all pairs O(n2m)

Flyod warshall- O(V3) for all pairs based on dp, dij k = min( dijk-1 and dikk-1+dkjk-1 ) for k nfor I n for j n , initial dij set for only edges direct, with their weights.

For path, backtrack another variabale pi, for first cond pi is piijk-1 for second cond, piijk is pkjk-1

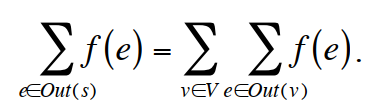
Prim- MST, from u like djikstra a bit only diff is we have key[v] as inifi and update it if key[v]>w(u,v), so only diff is we only look aat last edge here, so prim we build a graph one node and continue around it.

Kruskal works on disjoint sets, parent list for representing graph, then we sort edges in min for mst, then we look at edge if find and check for parent of u and v, if disjoint we union them(can be on rank) running time is 0(m+nlogn) logn for merge and nlogn for n merges max total as n nodes, m for m edges. We check for parent and disjoint as otherwise cycle form, so disjoint ensures only one connection/path.

Flow Single src ,sink

Capacity constraint <C(e), and conservation for every node is sum of f(e) in = sum of f(e) out

Flow leaving s is v(f) is what is received at t. also its what is trans ferred to other edges.

 so flow from s is whats is distributes on others, note this is for edges not same as conservation.

Define flow=0, make residual graph Gf, with forw with leftover, and backward for undo. Then find st path P, then b from this path P. Now in the initial flow update if forw edge in G initial f+b if backward edge f-b. To confirm if correct check for internal flow on nodes, and that f\* is less than = Ce

In ff every step atleast increase by b, and if all are integers then b>=1 so we have a solution. O(mC) as C is max C out of s and every step 1 increase min and each iteration we do O(m) work as update and traverse on edges only.

 Capacity of cut is sum of out capacity along the out edges. From a to B.

St cut is s in a t in B.

 This is true as all internal nodes net fllow is 0 but only for s we have, so fout and fin if we include internal edge become 00 and only v(f) or flow from s is left with us.

For any cut this is satisfied but capacity of cut is only out edge so if that is equal to v(f) we have min cut and max flow.

F-F algorithm can solve maximum matching problem in O(mn)

A set of paths in a graph G is edge-disjoint if their edges are disjoint, i.e., no  
two paths share an edge, though they may go through the same vertex