1. Topological sorting
2. Minimal spanning Trees
3. Shortest path algorithms

**Topological sorting:**

* For directed acyclic graph, not possible otherwise.
* Linear ordering of vertices such that every directed edge uv, vertex u comes before v in the ordering.
* There can be more than one topological sorting for a graph
* Order vertices so that edges point from lower order to higher order

Applications:

* Build systems
* Advanced-packaging tool
* Task scheduling
* Pre-req problems

References:

* <https://www.geeksforgeeks.org/topological-sorting/>
* <https://www.youtube.com/watch?v=AfSk24UTFS8> from 42:00
* <https://www.youtube.com/watch?v=HyVI8-nHgEg>
* <https://www.youtube.com/watch?v=ddTC4Zovtbc>
* <https://m.blog.naver.com/PostView.nhn?blogId=ndb796&logNo=221236874984&proxyReferer=https%3A%2F%2Fwww.google.com%2F>

**Minimal spanning Trees:**

* Given undirected, connected graph a spanning tree of graph G is tree that spans G => include every vertex of G. and is a subgraph of G(every edge in the tree belongs to G)
* Cost of spanning tree = sum of weights of all edges.
  + MST is spanning tree with lowest cost among other spanning trees, there can be multiple MST.
* **Two famous algorithms for finding MST: Kruskal, Prim’s Algorithm.**

**Kruskal’s Algorithm:**

* Build spanning tree by adding edges one by one into a growing spanning tree.
* Follow greedy approach -> at each iteration it finds an edge with least weight.
* How to check if 2 vertices are connected or not?
  + Could use DFS
  + Disjoint sets: sets whose intersection is empty set => do not have elements in common.
* Most time-consuming job here is sorting edges depending on their weights.
* Time complexity of disjoint-set operation will be O(E log V).

Steps:

1. Sort edges in order of their weights (smallest -> largest)
2. Add them in lowest to higher order to the MST.
3. Only add edges that do not form a cycle.

**Prim’s Algorithm:**

* Similar to Kruskal in that it uses greedy approach to find minimum spanning tree. Grow spanning tree from starting position by adding **vertex**.
* Starting from any vertex, select another vertex that will give lowest weight edge.
* Time complexity is O((V+E) log V) since each vertex is inserted in priority queue only once and insertion in priority queue take logarithmic time.

Steps:

1. Maintain two disjoint sets of vertices. One containing vertices in growing spanning tree and other not in it.
2. Select cheapest vertex that is connected to growing spanning tree and not in growing spanning tree set, then add it to the set. Can be done using Priority Queues, insert vertices that are connected to growing spanning tree (not already in), into priority queue.
3. Check for cycles to do that mark nodes which have been already selected and insert only those nodes in Priority Queue that are not marked.

Questions:

* Only for weighted trees graphs?

Applications:

* Designs of networks
* Used in alg approximating travelling salesman problem, minimum-cost weighted perfect matching, etc…
* Cluster Analysis
* Handwriting recognition
* Image segmentation

References:

* <https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/>

**Shortest Path Algorithm:**

* Family of algorithms designed for solving shortest path problems.
* Shortest path problem => given 2 points A and B find shortest path between them.
* Two main types of shortest path algorithm: 1. single-source 2. All-pairs.
* Main types of algorithms: Bellman-Ford, Dijkstra, Topological sort, Floyd-Warshall, Johnson

Single-Source shortest path algorithm:

* Def’n: Given a graph G, with vertices V, edges E with weight function w(u,v) = Wu,v and a single source vertex s, return shortest paths from s to all other vertices in V.
* Bellman-Ford:
  + Solve single-source problem in general case, where edges can have negative weights and graph is directed. If graph is un-directed must make it directed by including two edges in each direction.
  + Has property that can detect negative weight cycles reachable from the source => no shortest path exist.
  + If no negative weight cycle, then Bellman-Ford returns weight of shortest path along with path itself.
* Dijkstra:
  + Uses Breadth First Search(Not a single source SPA) to solve single source problem.
  + Graph cannot have negative weight edges because of this Dijkstra improves runtime of Bellman-Ford.

All-pairs:

* Def’n: Given a graph G, with vertices V, edges E with weight function w(u,v) = Wu,v. Return shortest path from u to v for all (u,v) in V.
* Floyd-Warshall:
  + Use dynamic programming approach.
  + Can have negative weight edges.
  + Works well with dense graphs
* Johnson’s:
  + Works best for sparse graphs
  + Take advantage of the concept of reweighting and use Dijkstra’s algorithm on many vertices to find shortest path once it has finished reweighting the edges.

Applications:

* Maps ex: google map, kakao map, etc…
* Networks, operations, and logistic research

References:

* <https://brilliant.org/wiki/shortest-path-algorithms/>
* <https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e>