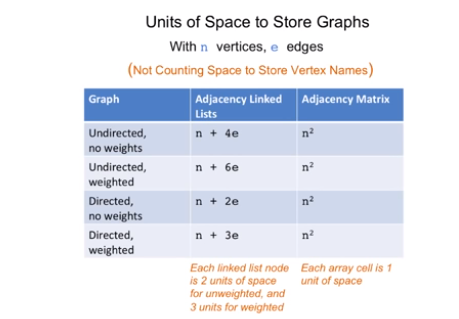
* Graphs consists of nodes(vertices (V)) and edges(E).
* Graphs that is edges can be directed (arcs) or undirected.
* **Cut** of a graph is a partition of V into two non empty sets A and B. There can be maximum ((2^n) – 2) cuts in a graph with n vertices.
* The **crossing edges** in an undirected graph are those which have one point in A and other in B.
* **Min Cut** is a cut with minimum number of crossing edges. A graph can at most have n(n-1)/2 minimum cuts.
* **Minimum number of edges** a graph can have are **n-1** and **maximum number of edges** a graph can have are **n(n-1)/2**.
* Let n be the number of vertices and m be the number of edges.
* A **sparse graph** is one if m is closer to **O(n)** that is lower bound and **dense graph** is if m is closer to upper bound or **O(n^2).**
* Adjacency Matrix
  + Let the matrix be denoted by ‘A’. Then the adjacency matrix is one which is denoted as 0-1 matrix where Aij is 1 if there is an edge between the i and j.
  + The storage space for the above matrix is n^2.
  + Not so suitable for sparse graphs due to their storage space.
* Adjacency List
  + Array of all the vertices O(n).
  + Array of all the edges O(m).
  + Each edge points towards its end points O(m).
  + Each vertex points to edges incident on it O(m).
  + Hence, total storage space required is (n + 4m).
  + Good for sparse graphs and very good for graph search.



* Compute the min cut.
* Applications:
  + One of the applications is finding the bottlenecks or weakness in a graph.
  + In cold war US and soviet were interested in this problem to find the most efficient way to disrupt the network.
  + Community detection on social network.
* Random Contraction Algorithm
  + Steps:
    - For more than 2 vertices:
      * Pick an edge randomly(equal probability).
      * Merge the nodes of that edge.
    - The two vertices at the end represent the cut.
  + May or may not give the minimum cut.
  + The **probability of success** is **2/n(n-1)**, which is very low.
  + The probability can be boosted by **repeated trials**.
  + The **running time** with tweaking can be brought to **O(n^2)**.
* Generic Graph Search
  + Find everything from a given start vertex.
  + Shouldn’t explore anything twice.
  + Running Time should be O(m+n).
* Breath First Search
  + Explore nodes in layers.
  + Shortest path can be done by this method.
  + We use **queue data structure** to perform BFS.
  + Can be used to check connectivity of graphs.
  + Linear running time.
* Depth First Search
  + Explores graph aggressively and backtracking only when needed.
  + Better for solving mazes.
  + Also computes topological ordering of a directed acyclic graph.
  + Helps commute strongly connected components of directed graphs.
  + Use **stacks instead of queues** to implement DFS.
  + Linear running time.
* Topological Ordering
  + It is a ordering of directed graph G such that:
    - The node number at vertex of the tail is greater than that at the head.
  + A Directed cycle will never result in a topological ordering.
  + Best way to get a topological ordering is by DFS and can be done in O(m+n).
* Kosaraju’s Two Pass algorithm:
  + Steps:
    - Let G(rev) = G with all connections reversed.
    - Run DFS – loop on G(rev).
    - Run DFS-loop on G.
  + To find strongly connected components in a graph in O(m+n).
  + We use DFS cause it have linear running time.
* **Problem Definition:**
  + The minimum number of edges required to connect all the nodes.
  + No cycles are allowed.
* **Assumptions:**
  + Input graph G is connected.
  + Edge costs are distinct.
* **Prim’s Algorithms:**
  + It always spits out a minimum spanning tree.
  + It is a greedy approach.
  + Select any vertex as the starting point.
  + Consider the cheapest edge including that starting point.
  + Merge the two vertices into one and then continue the above step till all the vertices are connected.