

**Text

Description automatically generated**Text, letter

Description automatically generated**3. Gale Shapley**

Time: O(n^2)

n is number of each type

What is Stable?

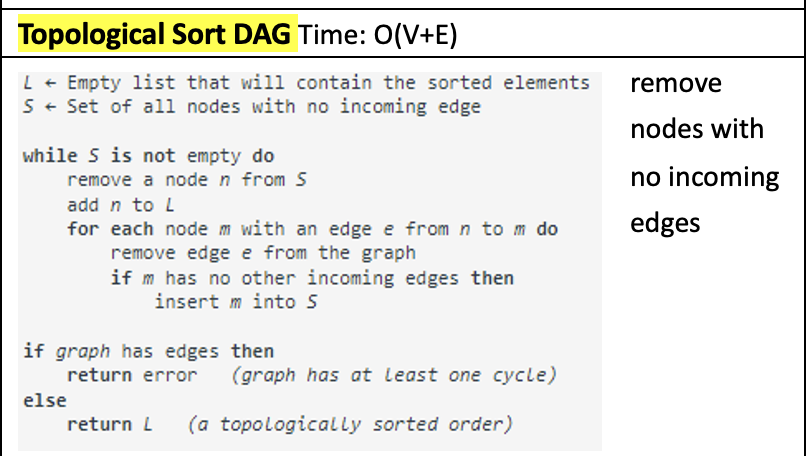
≠Roommate Problem

Room:

A:BCD; B:CAD

C:ABD; D:ABC

Ex: Iterate through all free men while there is any free man available.

**4.BFS & DFS**

**A\* Search** Time: depends on the heuristic, O(b^d)

BFS: Time: O(V+E) E->1 ~ V^2 ; Space: O(V)

can use on any graph

Pseudo BFS:

1. Create a queue Q

2. Mark the starting node as visited and add it to the queue Q

3. While Q is not empty:

a. Remove the first node from the queue Q

b. For each neighbor of the removed node:

i. If it has not been visited yet:

1. Mark it as visited

2. Add it to the queue Q

c. Do whatever processing you need to do with the removed node

DFS: Time: O(V+E); Space: O(V) non-recursion O(E)

Pseudo DFS:

1. Mark the starting node as visited

2. For each neighbor of the starting node:

a. If the neighbor has not been visited yet:

i. Recursively apply DFS to the neighbor

3. Do whatever processing you need to do with the starting node

Topological才需要无进入点，BFS可从任意点开始

BFS不代表一定能traversal所有点（DAG中）

**5.Adjacency Matrix & List**

Matrix: Complex & Space O(N^2) n=vertices

**Pro**: check time O(1); e = (n下面2) **Con:** space cost in many v little m

List: Complex:O(m) Space:O(m+n) m=edges

**Pro:** space efficient **Con:** e = (n下面2) check time O(n)

**6.Kruskal’s Algorithm & Prim’s Algorithm →Greedy**-**minimum covering graph**

**Mst:** nodes = v, edges = v-1

**Kruskal:** Time O(E logV or E logE) Space: O(V+E)

1.Sort all the edges in non-decreasing order of their weight.

2.Pick the smallest edge. Check if it forms a cycle with the spanning-tree formed so far. If the cycle is not formed, include this edge. Else, discard it.

3.Repeat step#2 until there are (V-1) edges in the spanning tree.

**Prim:** Time O(V^2) adj-list can reduce to O(E logV). Space O(V)

1.Create a set mstSet that keeps track of vertices already included in MST.

2.Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

3.While mstSet doesn’t include all vertices

1.Pick a vertex u which is not there in mstSet and has minimum key value.

2.Include u to mstSet.

3.Update the key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if the weight of edge u-v is less than the previous key value of v, update the key value as the weight of u-v

**Other:**

**P:** traverses one node more than one time; **K:** traverses one node only once.

**P:** local optimal 不一定是global optimal

**P and K both**: a weighted, undirected graph with non-negative edge weights or negative edge weights, only if not contain any negative-weight cycles **X**

But with negative edge, normally we choose K esp. when E >> V

Note: can be avoid by adding a sufficiently large constant to all edge weights.

**7. Dijkstra’s Algorithm & Bellman-Ford Algorithm →Greedy**-**shortest path**

**D:** Time-O(V^2)) **Heap**:(O(E logV)); Space-O(V)

**B:** Time-O(VE) Space:O(V)

**Other:**

**D** and **B** both work on weighted directed or undirected graphs.

**D** deal only with non-negative edge weights

**B** can handle graphs with negative edge weights (but not negative-weight cycles)

**Graphical user interface, text

Description automatically generatedB:** 直接列起始点到所有点的min count更容易

**8.DP**

Knapsack Problem

Right is item, below is weight

**9. Network Flow**

Augmenting path: An additional possible path in the residual graph from source to sink with a capacity of at least 1.

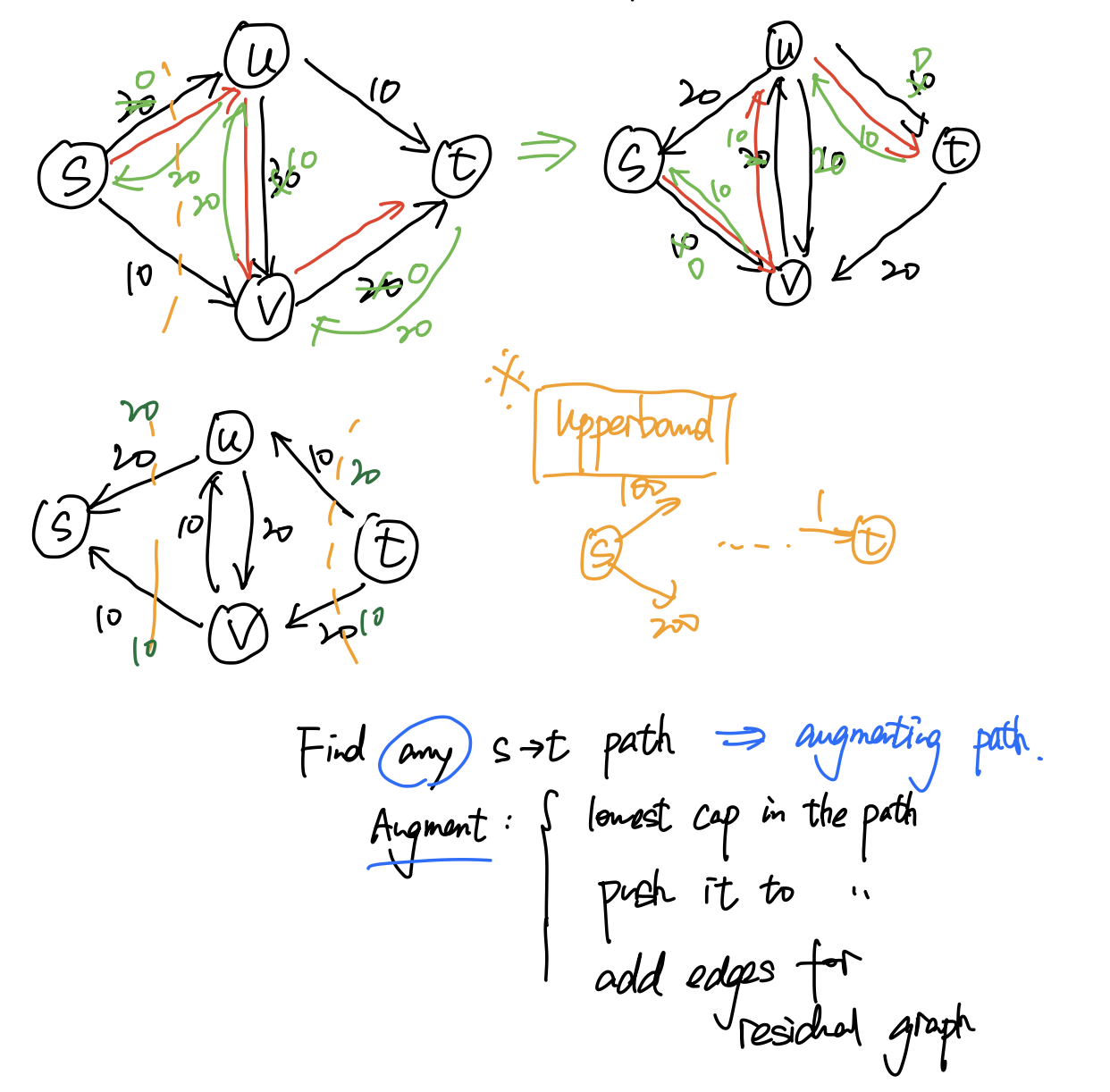
Residual graph: A graph with edges that represent the difference between an edges capacity and its flow.

**Preflow-Push maximum flow algorithm:O(V E^2)**

In>out convert to in=out; Hight = lable(relable)

source-h = n (# of node) All-other-h = 0 (initial)

**Ford-Fulkerson Algorithm: O(E f) f is flow**

****

**Bipartite matching**