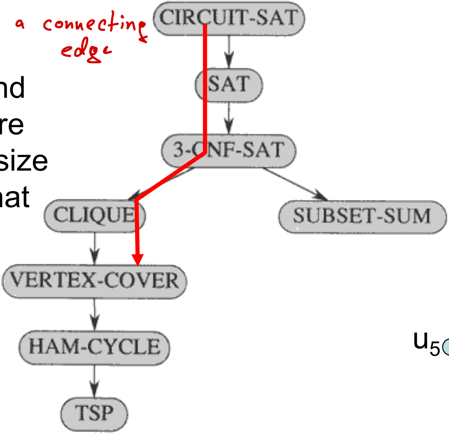
combinatorial optimization problems

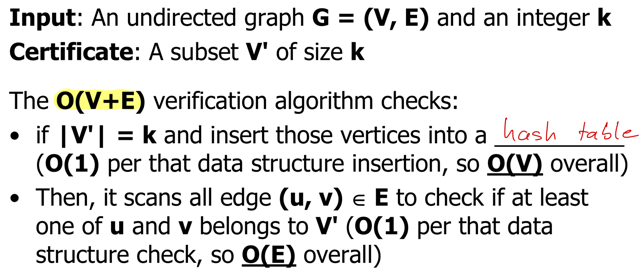
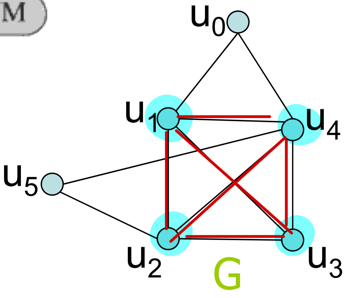
# Vertex cover

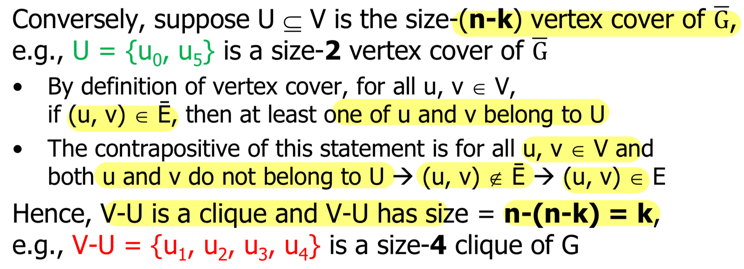
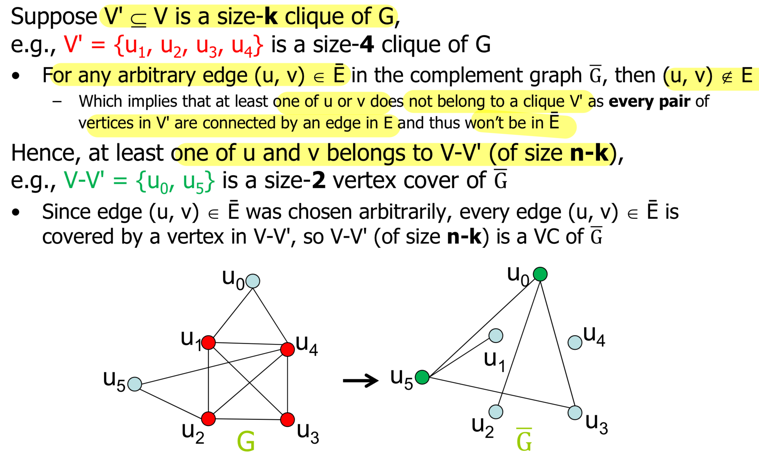
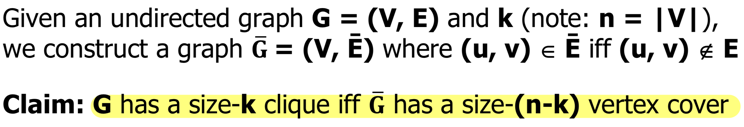
## A screenshot of a computer Description automatically generateddefinition

Vertex Cover of Graph G = (V, E): subset S ⊆ V such that **for every e = (u, v) ∈ E**, **u ∈ S or v ∈ S**

## proof np-complete

**NP** = can be solved in polynomial time by a non-deterministic machine and verified in polynomial time by a deterministic machine  
**NP-Hard** = every problem in NP is reducible to L in polynomial time  
**NP-Complete** = NP && NP-Hard

**1. Vertex-Cover in NP:** (verify in polynomial time)

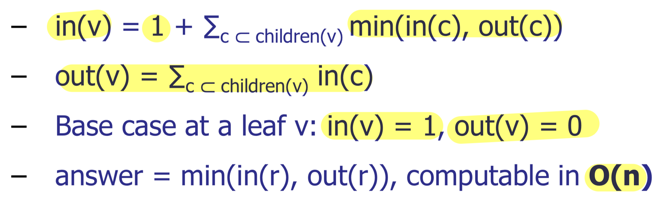
**2. Vertex-Cover in NP-Complete:** (polynomial time reduction)  
Clique of Graph G = (V, E): subset C ⊆ V such that every pair of vertices in C has a connecting edge  
🡪 reduce Clique-Problem (NP-Hard) into Vertex Cover

MVC is NP-hard: O(1) polynomial reduction VC <=p MVC (check if min <= k)  
MVC is not NP-complete: no polynomial verifier (have to try every possible cover)

🡪 MVC solves MIS (Max-Independent-Set: no connecting edges between vertices) 🡪 complement V \ MVC

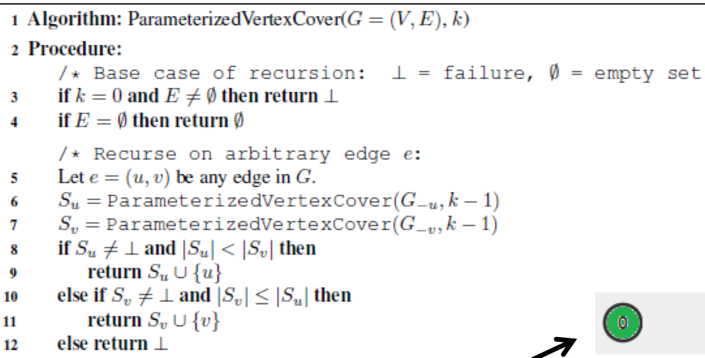
**Fast, optimal, universal 🡪 special case, parameterized solution (assume), approximate solution**

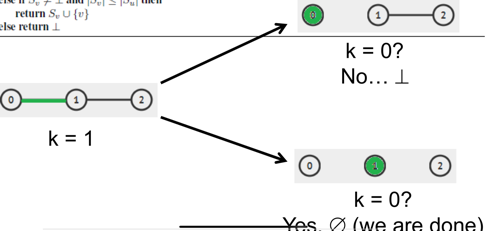
## MVC on tree (DP)

**No Cycles, small graph size**Only 2xV States, at most two incoming edges -> O(V)

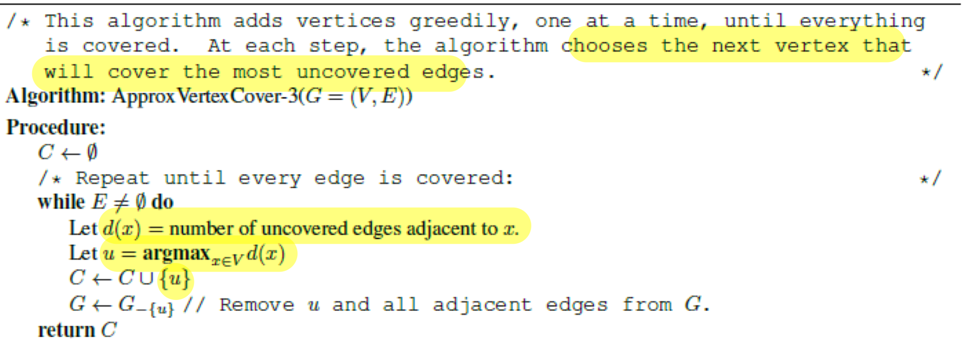
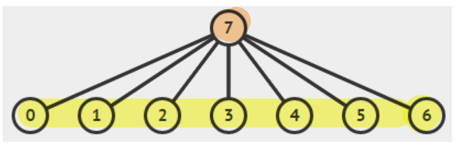
MVC solvable in polynomial (also if no odd cycles)

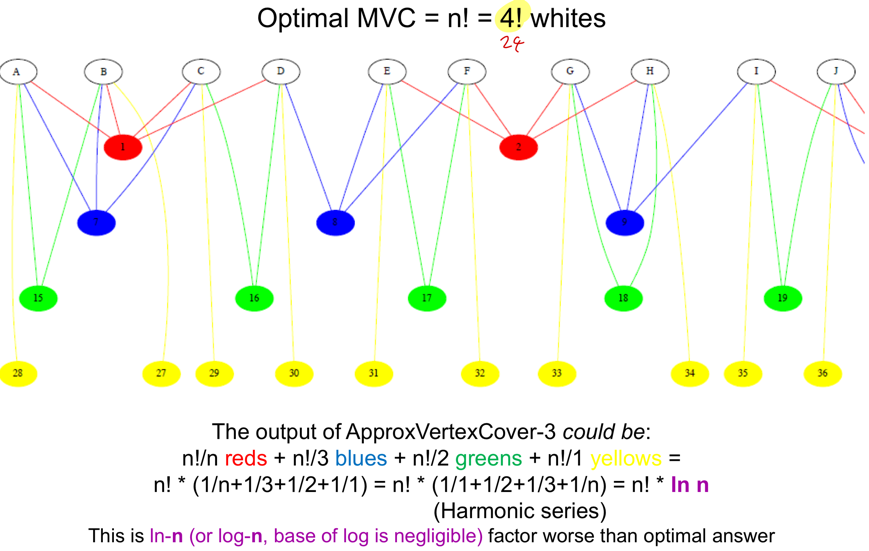
## Parameterized MVC

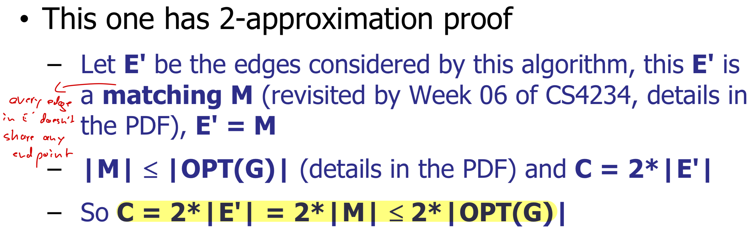
Pure Brute Force: O(2nm) 🡪 **NO Cycles**  
Parameterized: k << n 🡪 Naive O(nkm) algorithm or O(2km)

T(k, m) <= 2 T(k-1, m) + O(m)  
O(2km)

## Approx mvc

adding arbitrary end point:

Randomized: P(right choice) = 0.5 🡪 C at most 2 \* OPT (expeced) vs. Star graph

**Deterministic 2: adding both ends (only this one is 2-Approximation)**  
Deterministic 3: O(log n) approx.

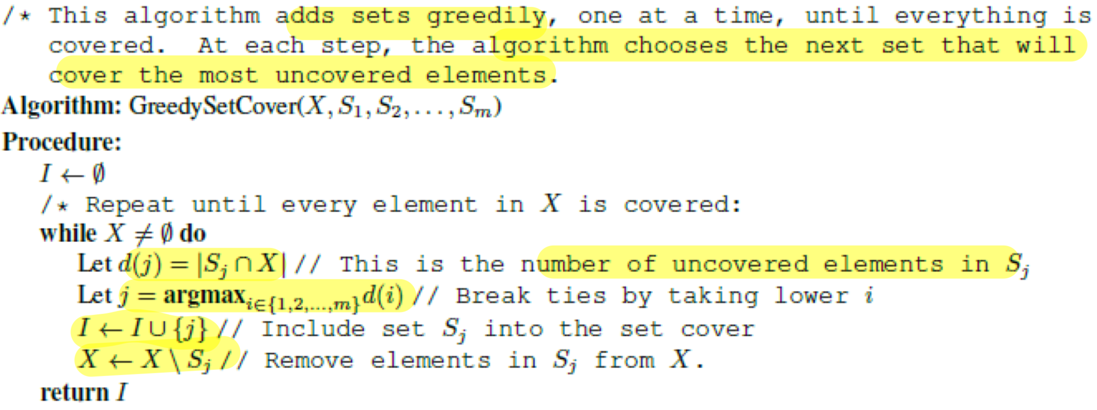
A couple of black circles with white text

Description automatically generatedAll run in polynomial time (fast) 🡪 run all of them and report best

# Min Set Cover

Set Cover of X = {x1, x2, … xn} with subsets S1, S2, … Sm: set I ⊆ {1, 2, … m} such that ∪j∈I Sj = X  
**VC <=p SC 🡪 SC is** **NP-Hard**

## Greedy set cover

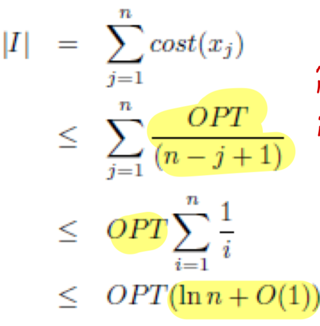


A math equations and formulas

Description automatically generated with medium confidence**O(log n) Approximation:**

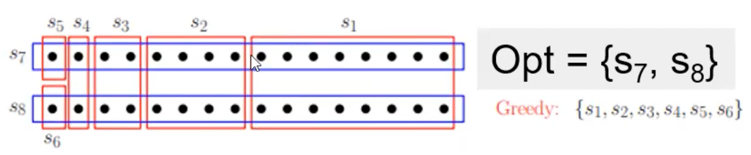
A math equation with black text

Description automatically generated



A math equations on a white background

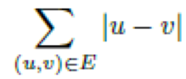
Description automatically generated



3 times more subsets than optimal answer

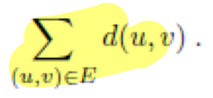
# Steiner tree

## Euclidean-steiner-tree

Definition: given set R of n 2D-points in **Euclidean plane**, find set of additional Steiner points S and spanning Tree T = (R ∪ S, E) such that weight of tree is minimized (NP-Hard)

* Each Steiner Point has **degree 3**
* Lines form **120 degree** angles
* At most **n-2 Steiner points**

## Metric-steiner-tree

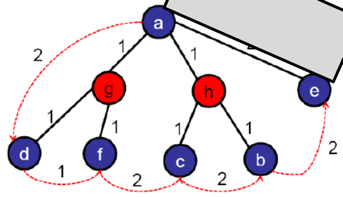
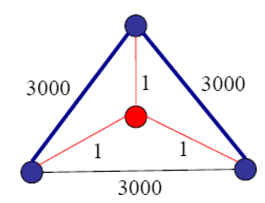
Given required vertices R, set of Steiner vertices S, distance function d find subset S’ ⊂ S and spanning tree T = (R ∪ S’, E) of min weight

Metric easier than Euclidean (criterion on how many Steiner points needed and where to place them)  
Generalized: given an arbitrary graph with edge weights

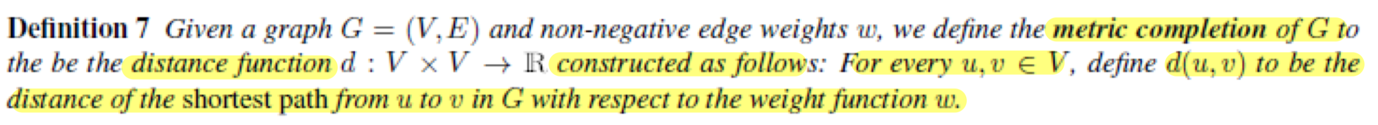
## Complete search solution

Try all possible subsets of Steiner vertices (2|n-s|) 🡪 run MST algo O(E log V) 🡪 O(2|n-s| \* n2 \* log n)

## metric MST 2-approximation (no Steiner point)

Create a cycle bypassing all Steiner vertices and then removing duplicate vertices (generate short-cutting paths, triangle inequality) 🡪 break one edge to obtain acyclic spanning tree  
**cost(M) <= cost(C) <= 2\*cost(T)** with cost(T) = cost of DFS

## General MST 2-approximation (gap-preserved)

**Metric completion:** non-metric edge weights into metric ones with All-Pairs Shortest Path algorithm (e.g. Floyd-Warshall O(V3)) 🡪 preserves metric properties *(e.g., proof triangle inequality by contradiction)*

**Convert back to General-ST:** replace virtual edges in Metric-ST with the actual shortest paths, remove overlapping edges and cycles 🡪 cost equal or lower in General ST-version

Use **Theorem:** given an α-approx. algo. for finding a metric ST, we can find an α-approx. algo for a general ST  
**Gap-preserving:** reduction that preserves approximation ratios

# Traveling salesman (TSP)

A grid of black text

Description automatically generated**Definition:** given a set V of n points and a distance function d, find cycle C of minimum cost that contains all points; Complete Graph O(V2)  
Botanic Variant: solvable in O(N2)

4 Variants (3 equivalent), all NP-hard (G-NR-TSP even NP-hard to approx)

## Brute Force & dP

(N-1)! Permutations if we fix one node 🡪 O(N! \* N) time, O(N) dist. sum calc.  
Improvement: memorize repeated sub-tours O(N2 \* 2N-1), Held-Karp DP for TSP

🡪 small graph, TSP is bitonic, each vertex visited exactly once

## 2-Approx for G-R-TSP

A diagram of a number

Description automatically generated with medium confidenceRun MST of input graph 🡪 Run DFS on resulting MST 🡪 Output vertices in cycle induced by DFS (no repeat)

**Proof:**

* C\* = OPT(V, d); E\* = edges in optimal cycle; T\* is MST of G = (V, E\*) 🡪 d(T\*) <= d(C\*) = OPT
* T = MST of G = (V, E) 🡪 E\* ⊆ E 🡪 d(T) <= d(T\*)

🡪 d(C) = 2 \* d(T) //see Metric ST Analysis <= 2 \* d(T\*) <= 2 \* d(C\*) <= 2\* OPT

## 1.5-Approx for m-NR-tsp

**Eulerian Cycle:** a cycle that crosses each edge exactly once (connected + every vertex even degree)  
**Perfect Matching:** subset M of edges in a graph so that no two edges share an endpoint; perfect: |M| = |V|/2

**Christofides’s Algorithm:**

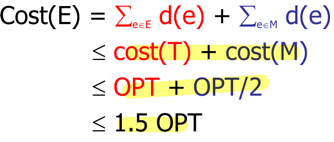
1. T = MST(G) and E = edges of T
2. O = set of vertices with odd degree (even number, Handshaking lemma)
3. M = Min-Weight-Perfect-Matching on subgraph G\* induced by O
4. Combine T+M (all vertices have even degree)
5. Output vertices in Eulerian Cycle (no repeat)

A close up of text

Description automatically generated**A close-up of a math equation

Description automatically generatedAnalysis:**

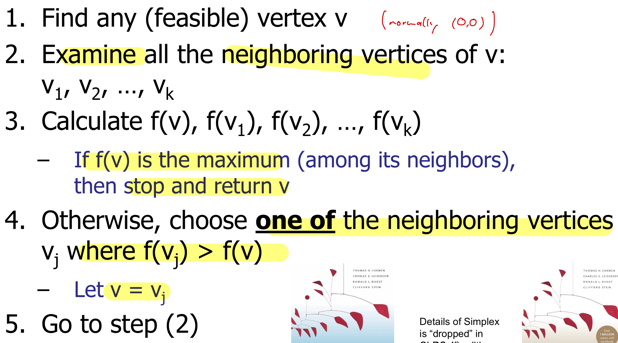
* A white paper with black text and black text

  Description automatically generatedCycle **C’ on odd vertices** (even, with no repeats): **cost (C’) <= cost(C)**, as we only skipped vertices (triangle inequality)
* M = min cost perfect matching
* Cost(M) <= cost(M1)
* Cost(M) <= cost(M2) (each has |O|/2 edges)
* 2 cost(M) <= cost(m1) + cost(M2) <= cost(C’) <= cost(C) = OPT
* Cost(M) <= OPT/2

Linear programming

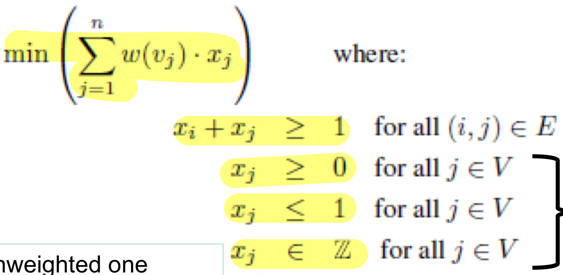
# LPs

## Simplex method (LP: polynomial time)

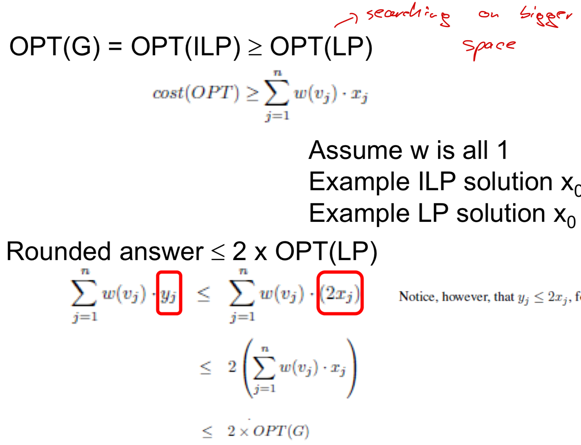
Vertex = intersection of some constraints

If equations are not linearly separable you will get many solutions  
m constraints, n variables 🡪 mCn = O(mn)

## MWVC as ILP

🡪 MVC <=(p) ILP 🡪 ILP is also np-hard

## Relaxations (ILP: nP-hard)

No integer constraint 🡪 round up x if it is <= 0.5  
2-Approximation algorithm:

Flows & Matching

# Max flow – not np hard!

**st-cut:** partitions vertices of a graph into 2 disjoint sets S and T (source s ∈ S, sink t ∈ T)  
**cap of st-cut:** sum of capacities of edges that cross cut **from S to T  
net-flow:** flow on edges from S->T minus flow from T->S

**flow f** = net-flow of any st-cut <= cap of st-cut (**Weak duality**)

**Induction-Proof:** start with S = {s}, T= V\S 🡪 take node x, add/subtract outgoing/incomding edges, subtract/add edges from X to S 🡪 flow into X = flow out of X 🡪 F unchanged 🡪 same for all cuts

## MaxFlow-MinCut Theorem

f is max flow ⬄ cut whose capacity equals value of f (min cap) ⬄ no augmenting paths in residual graph

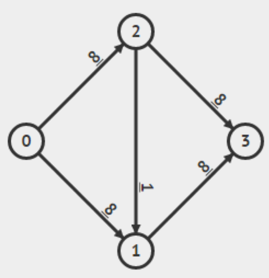
* 2 -> 1: st-cut with min cap 🡪 for all flows g: value(g) <= cap(S, T) = value(f) 🡪 f is max flow
* 1 -> 3: assume 1 more augmenting path 🡪 send one flow and improve flow 🡪 f not max flow 🡪 contradiction
* 3 -> 1: source cannot reach sink anymore

## A screenshot of a computer Description automatically generatedFord-fulkerson

**Idea:** find augmenting path (from s to t through edges with residual capacity left) along which flow++

**If Ford-Fulkerson terminates there is no augmenting path left 🡪 flow is max**

**FF always terminates (if cap integers):**

* Every iteration finds new augmenting path 🡪 bottleneck cap of at least 1
* Each iteration increases flow of at least one edge by at least 1
* Finite number of edges, finite max cap per edge 🡪 termination

Complexity: **O(m2U)**

* O(m) for finding path p in R and updating caps (m >> n)
* U = max cap of outgoing edge connected to s 🡪 MF <= m\*U

## Edmonds karp

Run O(E) BFS to find the shortest (in terms of edges used) augmenting path  
Complexity: O(m2n) 🡪 strongly polynomial algorithm 🡪 NOT NP-hard

## Dinic

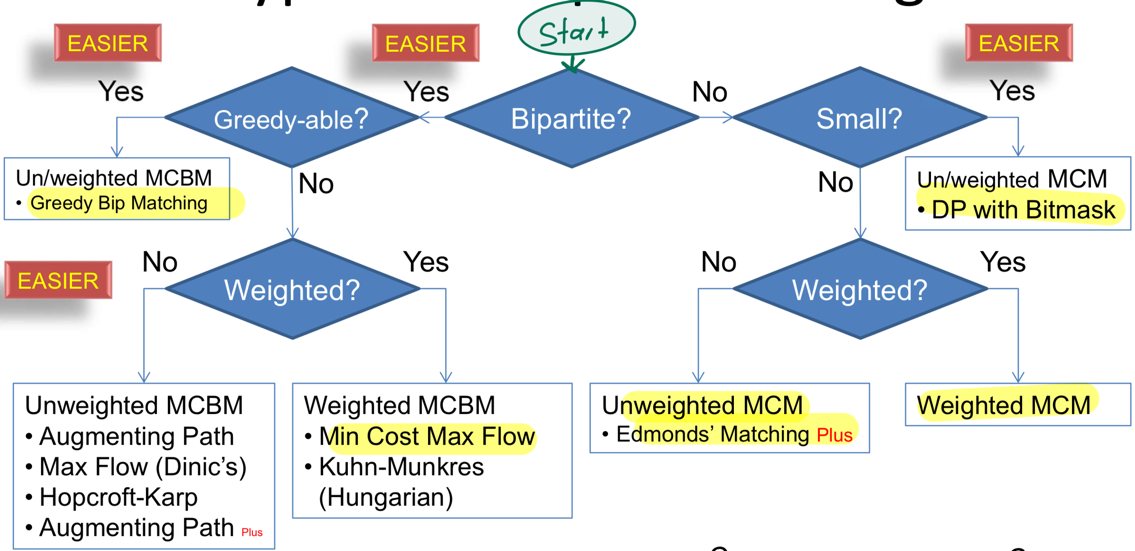
Uses BFS information in a better way than Edmonds-Karp (90% identical) 🡪 O(V2\*E)

## Finding edges in min-cut

Run Maxflow algo until termination 🡪 find vertices S that are still reachable from source (DFS) 🡪 T = V\S 🡪 for every edge in S: if endpoint in T add to min-cut

# (Weighted) max-cardinanlity (bipartite) matching

*MCBM Keywords: Left/Right, Row/Col, alternate Row/Col, Prime/Coprime, Odd/even, male/female, job/employee, bi-coloring, out/in-degree only, no-odd-length cycle, Tree\**

**Matching:** subset M of edges in a graph G = (V, E) so that no two edges share an endpoint  
**Bipartite:** vertices partitioned into 2 disjoint sets U and V, such that every edge can only connect from U to V

## MCBM by reducing into maxflow

A diagram of a network

Description automatically generatedDirected, bipartite!, O(sqrt(V)\*E) for Dinic

## Augmenting path algorithm

**Berges theorem:**

* Matching M is maximum if and only if there is no augmenting path with M
* Augmenting path: starts and ends on unmatched vertices and alternates between edges in and not in the matching

**Proof:**

* Max 🡪 no augmenting path:

Contradiction: 1 more augmenting path -> flip it -> get one more matching -> not max

* No Augmenting -> Max: suppose not max 🡪 M’ > M 🡪 Symmetric difference (edges that is not covered by both) 🡪 consists of paths or cycles (degree <= 2)
* Even length path/cycle 🡪 |M| = |M’| -> M’ not > M
* Odd length cycle not possible (triangle graph: cannot assign last edge)
* A close-up of a computer code

  Description automatically generatedOdd length path: starts with edges from larger M’ and edges in M are inside 🡪 aug path 🡪 contradiction

For each vertex in the left:  
- if there is an augmenting path of 1+ edges -> flip edge status along path

**O(VE),**

**Weakness:** for very connected graphs augmenting paths will be very long in the last stages 🡪 randomly O(V+E) select neighbour 🡪 O(V^2+kE)

Maxflow: for variation (use left or right multiple times), multiple layers

## Hopcroft karp (HK)

Identical to Dinic Max Flow 🡪 prioritize shortest augmenting paths (number of edges) 🡪 O(sqrt(V)\*E)

## Hall’s marriage theorem

Bipartite Graph with sets U and V 🡪 a matching covering U exists if and only if for each subset W of U: |W| <= |N(W)| 🡪 2W checks required

Random

DAG = Directed acyclic graph  
n <= 25 is upper limit of what O(2n) algorithm can do in 1s

# MST

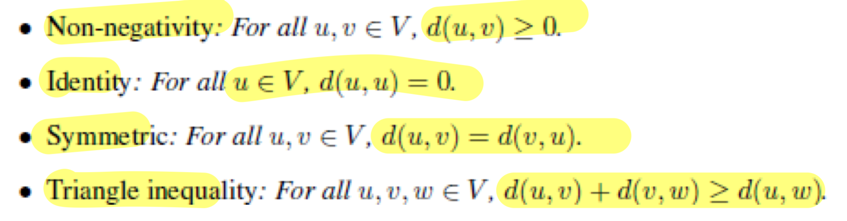
Prim, Kruskal: O(V^2 log V)

# DFS, BFS

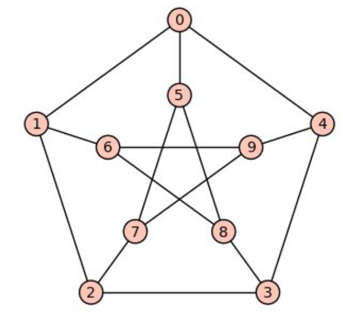
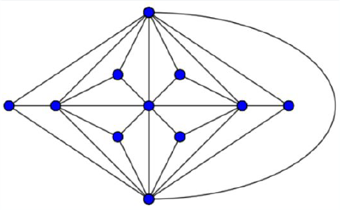
# Iterative brute force COPS

O(2V) and use Bitmask (normally V > 10 is too large

# Metric



# Planar graph criteria

Kuratowski & Wagner: A graph is planar if and only if it does not contain K5 and K(3, 3) minor 🡪 Biggest Clique a planar graph can have is of size 4

Number of edges <= 3n-6

4-Colour-Theorem: in any planar graph you need at most 4 color to color the graph

# List of NP-hard COPs

