Eulerian
Hamiltonian } Graphs
Traveling Salesman Problem
minimum spanning Tree
matching

SYSM 6302

CLASS 14

How can we "cover" the nodes in a network?

- → Trails & circuits (no repeat edges)
- → Paths & cycles (no repeat nodes)
- → Trees (single path between all node pairs)
- → Matching (vertex-disjoint edges)
  - ↳ (paths & cycles in a directed graph)

Eulerian Trail – a trail that visits every edge once

→ implies every vertex is visited (possibly multiple times!)

→ Eulerian circuit is a Eulerian Trail beginning & ending at the same node

A connected graph G:

→ has a Eulerian Circuit ⟺ every vertex has an even degree.
(G is "Eulerian")

→ has a Eulerian Trail ⟺ if there are exactly zero or two vertices with odd degree
(G is "Traversable")

Hierholzer's Algorithm for Eulearian Circuits

1) Select any starting vertex, V. Follow a trail that returns to V.
   (such a circuit must exist because each node has even degree)

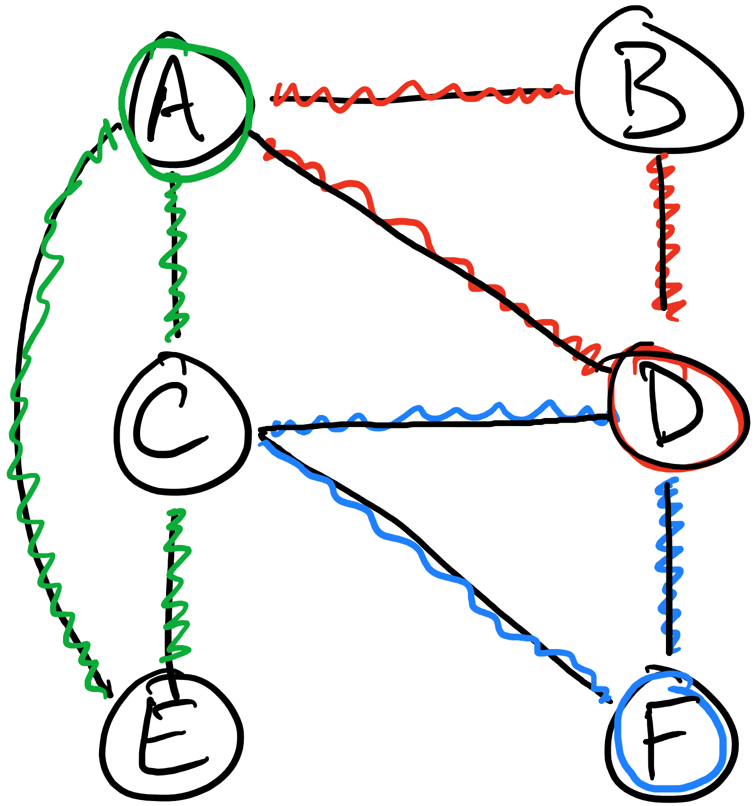
2) If this circuit is Eulerian, Stop. Otherwise select another vertex W in the circuit with some adjacent edges not in the circuit.

3) Find a new trail from W, back to W.

4) Insert this new circuit into the existing circuit.

5) Continue until all edges are used.

Complexity: $O(|E|)$



Select F.

Circuit: FCDF

Select D

new circuit: DABD

Combined circuit: FCDABDF

Select A

new circuit: AECA

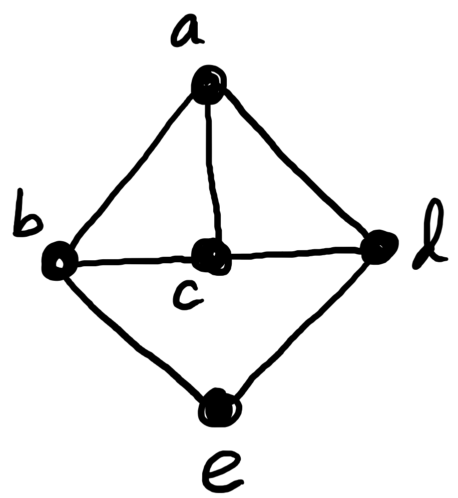
Combined circuit: FCDAECABDF

Applications in: → garbage/postal truck routes

→ checking websites for broken links

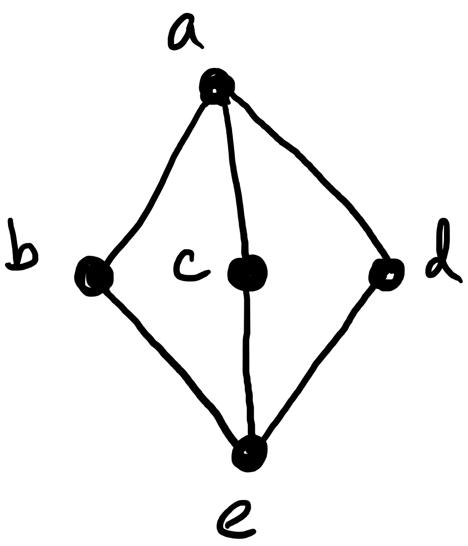
Hamiltonian Cycle a cycle that contains every vertex of a graph

This is a hard problem!



Hamiltonian!

a b e d c a



If G is Hamiltonian $\Rightarrow$ cycle contains all nodes, including:

b, c, d

Since these have exactly 2 edges each, both must be included in the cycle

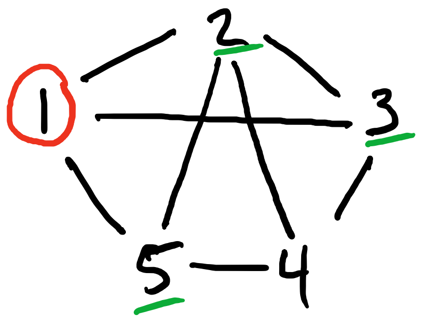
$\rightarrow$ cycle includes: (b,a)
(c,a)
(d,a)

$\rightarrow$ But this contradicts that a is visited once!

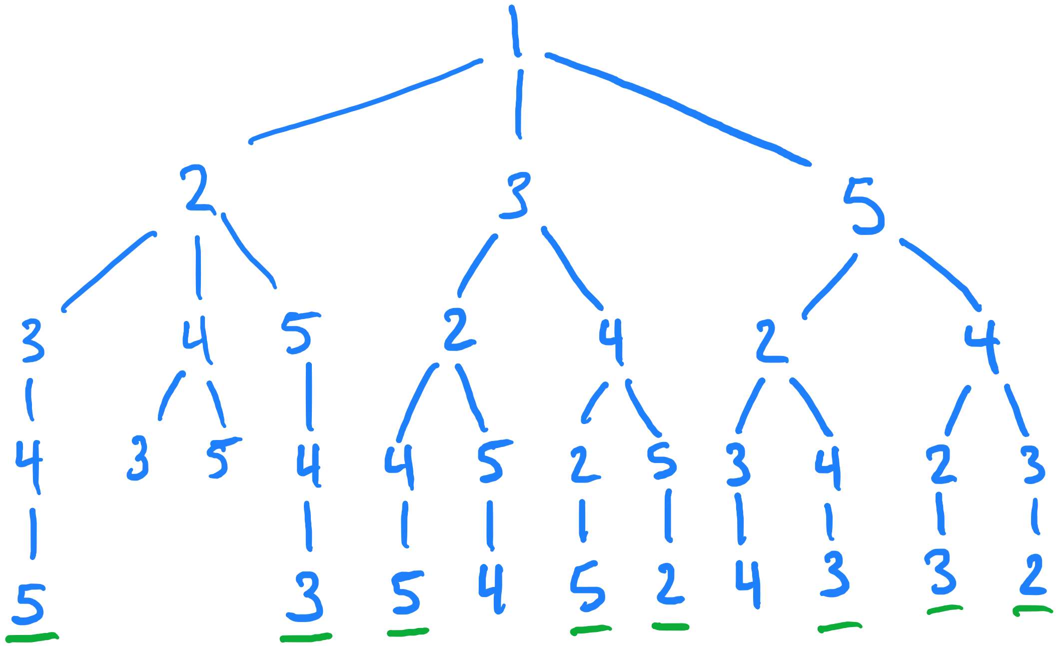
Necessary Conditions:

If a Hamitonian Cycle exists:

1. if undirected – it is connected
   if directed – it is strongly connected
2. every node has degree ≥ 2
3. if undirected – if node has degree $= 2$, then both edges are part of every Hamiltonian cycle



Construct a tree of reachable nodes:



A Hamiltonian cycle will have length n

This combinatorial search is indicative of NP-hard problems

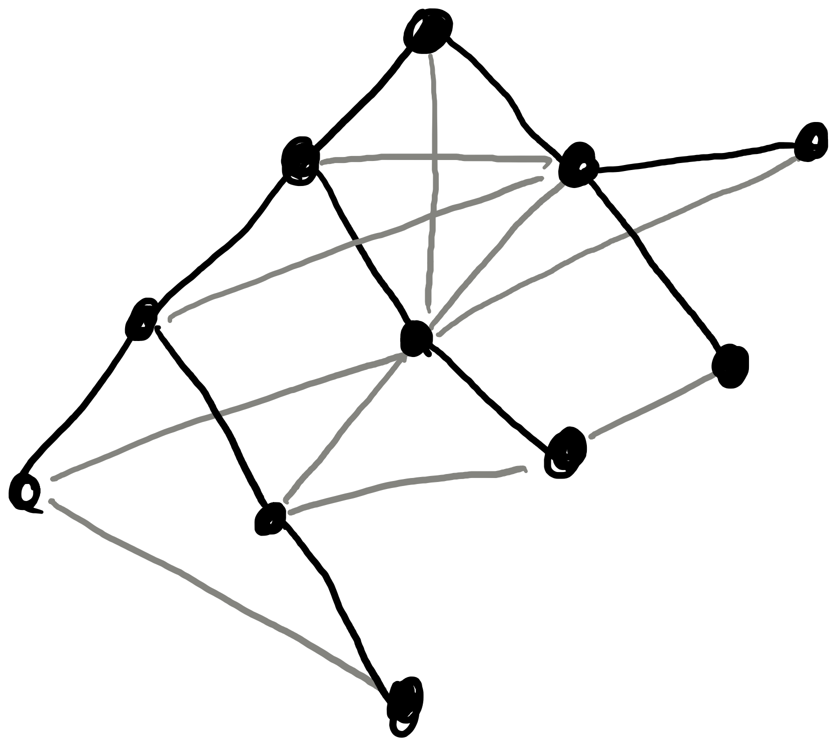
Traveling Salesman Problem

→ Find an optimal (minimal) Hamiltonian cycle on a weighted graph.

→ Core problem in optimization and benchmark for new heuristics

→ Many other applications, including minimizing the time needed to drill holes in a circuit board.

Spanning Tree an acyclic connected subgraph containing all of the vertices of a graph



→ We saw an example of a spanning tree as the result of breadth-first search

Minimum Spanning Tree

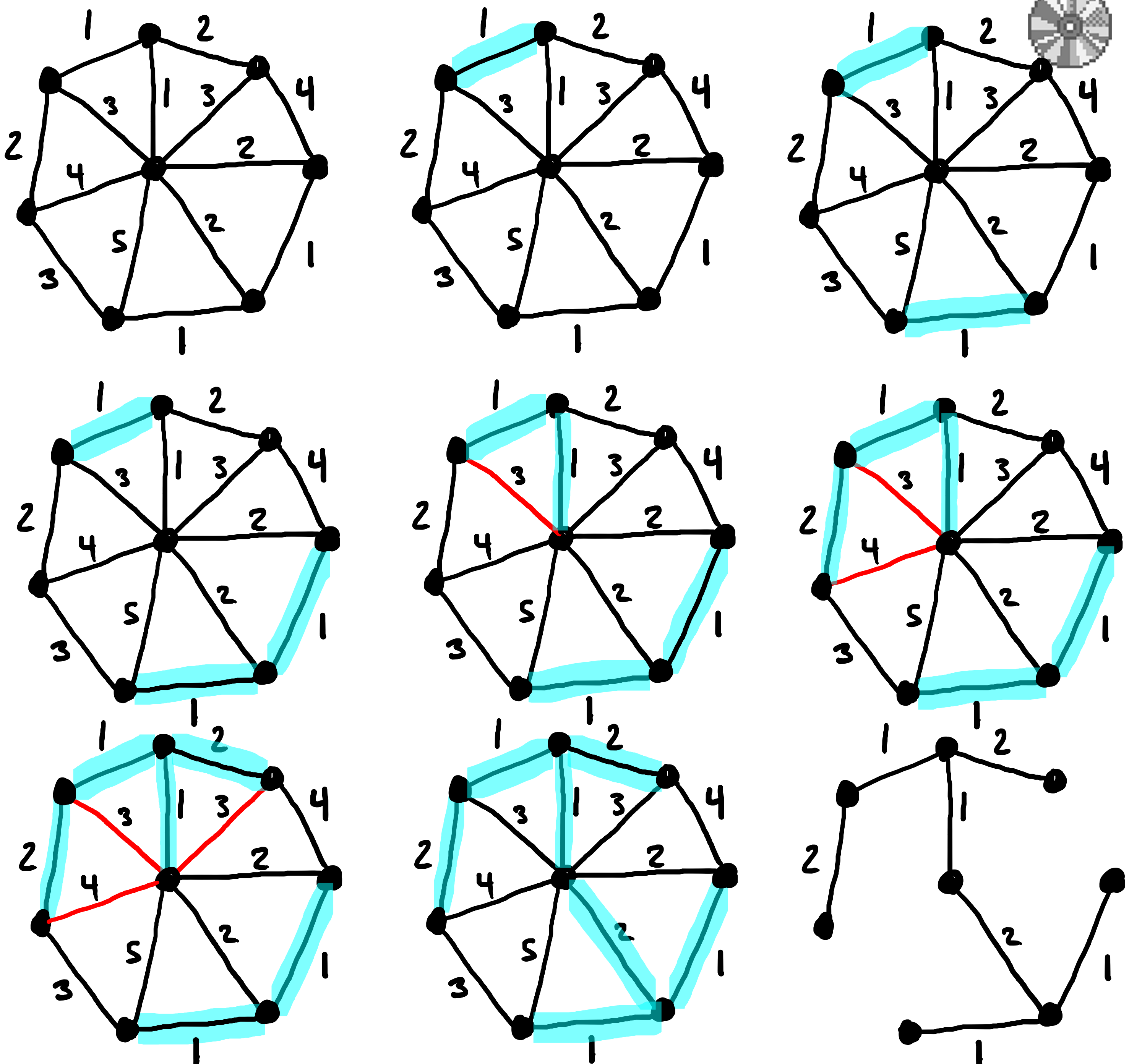
A spanning tree whose weight (the sum of the weights of its edges) is minimum

Kruskal's Algorithm

Graph G, # nodes: n

1. Set $k=1$
2. Pick edge e_k that has the smallest weight in $G-\{e_1,\dots,e_{k-1}\}$ such that e_k does not form a cycle with $e_1,\dots,e_{k-1}$.
3. $k=k+1$.
4. If $k=n-1$, stop. Otherwise repeat starting at 2.

$\Rightarrow$ Minimum Spanning Tree



Why does this work? Let T be the spanning tree found by Kruskal's algorithm and $\tilde{T}$ be a minimum spanning tree.

→ Need to show that the weight of T is $\leq$ weight of $\tilde{T}$:

$$W(T) \leq W(\tilde{T}) \quad , \qquad W(T) = \sum_{k=1}^{n-1} w(e_k)$$

edges in T

weight of an edge

→ Suppose: T has edge sequence: $e_1, e_2, \ldots, e_{n-1}$

with $e_i \in T$ the first edge in the sequence $e_i \notin \tilde{T}$

→ Thus $\tilde{T} + e_i$ is no longer a tree (has a cycle C) (recall tree $\Leftrightarrow$ $|E| = |V| - 1$)

→ Pick $\tilde{e} \in C \subset \tilde{T}$, $\tilde{e} \notin T$ $\Rightarrow$ $\tilde{T} + e_i - \tilde{e}$ is again a tree (and spanning)

→ Note that e_i and $\tilde{e}$ are, thus, "alternative" edges since either can be used to create a spanning tree

→ At the i^{th} step, the algorithm picked e_i, not $\tilde{e}$, thus $w(e_i) \leq w(\tilde{e})$

Thus $w(\tilde{T} + e_i - \tilde{e}) - w(\tilde{T}) = w(e_i) - w(\tilde{e}) \leq 0$

Since $\tilde{T}$ is already minimal, $w(\tilde{T}) \leq w(\tilde{T} + e_i - \tilde{e})$

$\Rightarrow$ $w(\tilde{T} + e_i - \tilde{e}) = w(\tilde{T})$ and $w(e_i) = w(\tilde{e})$

$\Rightarrow$ $\tilde{T} + e_i - \tilde{e}$ is also a minimum spanning tree

Continuing this process for each $e_i \in \tilde{T}$ allows us to conclude that T is minimal, ie $T = \tilde{T} + e_i^{(1)} - \tilde{e}^{(1)} + e_i^{(2)} - \tilde{e}^{(2)} + \dots$

bad notation, but you get the idea...

Matching – matching is a set of non-adjacent edges

→ No two edges share the same node

→ **maximum matching** – a matching with the maximum number/weight of edges

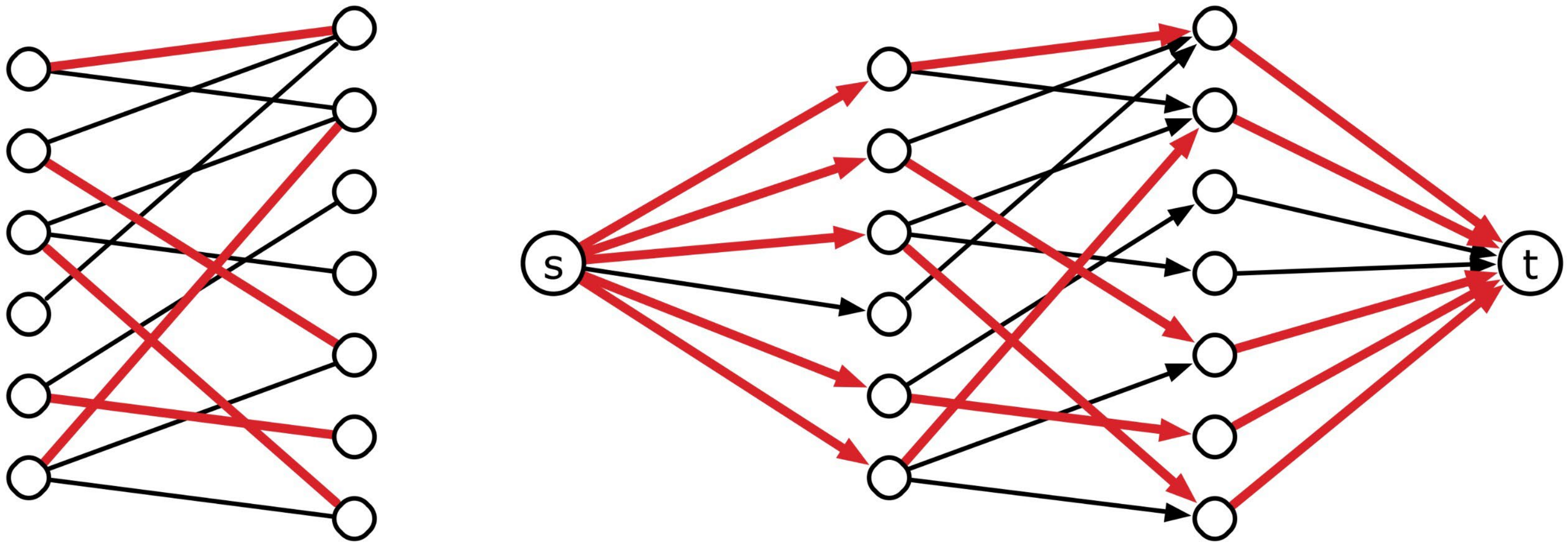
→ perfect matching – all nodes are matched (adjacent to an edge in the matching)

→ alternating path – begins at an unmatched node and the edges alternate between belonging to and not belonging to the matching.

→ augmenting path – begins and ends at unmatched nodes

Finding Maximum Matchings on Bipartite Graphs

$\Rightarrow$ Using maximum flows (eg, Ford-Fulkerson) $O(VE)$



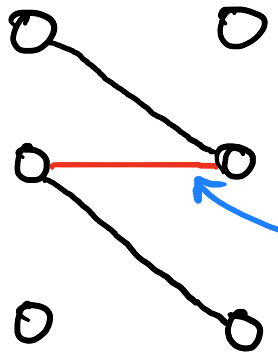
$\rightarrow$ The size of the maximum matching = maximum flow

Finding Maximum Matchings on Bipartite Graphs

Hopcroft-Karp Algorithm $O(E\sqrt{V})$

→ identifies alternating augmenting paths to increase the size of matching

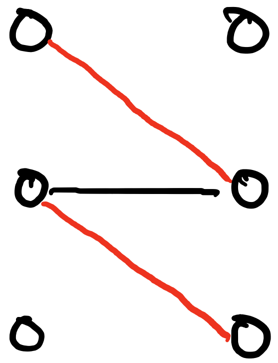
Simplest Alternating augmenting path



alternating augmenting path p

symmetric difference: edges in M and p, but not in both

$M \triangle p =$



$|M \triangle p| = |M| + 1$

optimal matching

⇒ Can be shown that $M \triangle M^* =$ collection of alternating augmenting paths

∴ When no more alternating augmenting paths exist, Matching is optimal