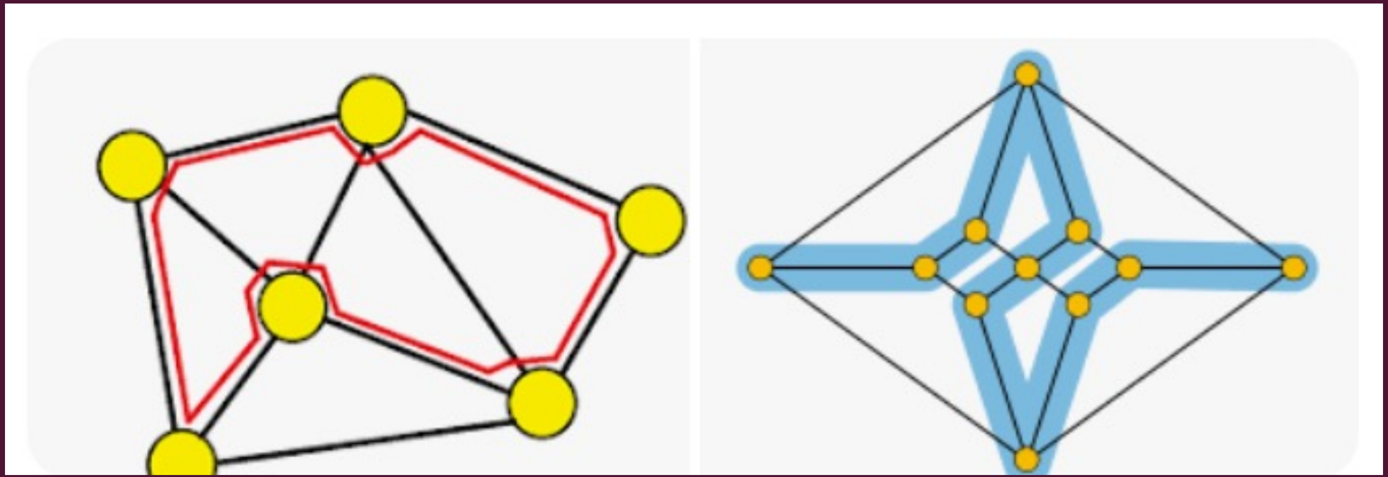
NP COMPLETE PROBLEMS

REDUCTIONS

REDUCTION OF 3-SAT TO HAMILTONIAN CYCLE (HC) PROBLEM

– **a simple cycle including all the vertices of the graph (start and end vertex are same)**



3-SAT AND HC

- For a 3-SAT expression containing n variables, there are 2^n possible assignments

- We model these 2^n possible truth assignments using a graph with 2^n different Hamiltonian cycles

CONSTRUCTION OF PATHS

Construct n paths P_1, P_2, ..., P_n corresponding to the n variables.

Each path P_i should consist of $2k$ nodes ($v_{i,1}$, $v_{i,2}$, ..., $v_{i,2k}$) where k is the number of clauses in the expression.

For example, consider the following boolean expression with 4 variables:

x_1, x_2, x_3, x_4

Expression: $(x_1 + x_2 + \overline{x_3}).(\overline{x_2} + x_3 + x_4).(x_1 + \overline{x_2} + x_4)$

CONSTRUCTION OF PATHS

CONSTRUCTION OF PATHS

We construct 4 paths with 6 nodes each

P_1 with nodes $v_{1,1}, v_{1,2}, v_{1,3}, v_{1,4}, v_{1,5}, v_{1,6}$

P_2 with nodes $v_{2,1}, v_{2,2}, v_{2,3}, v_{2,4}, v_{2,5}, v_{2,6}$

P_3 with nodes $v_{3,1}, v_{3,2}, v_{3,3}, v_{3,4}, v_{3,5}, v_{3,6}$

P_4 with nodes $v_{4,1}, v_{4,2}, v_{4,3}, v_{4,4}, v_{4,5}, v_{4,6}$

For example, consider the following boolean expression with 4 variables:

x_1, x_2, x_3, x_4

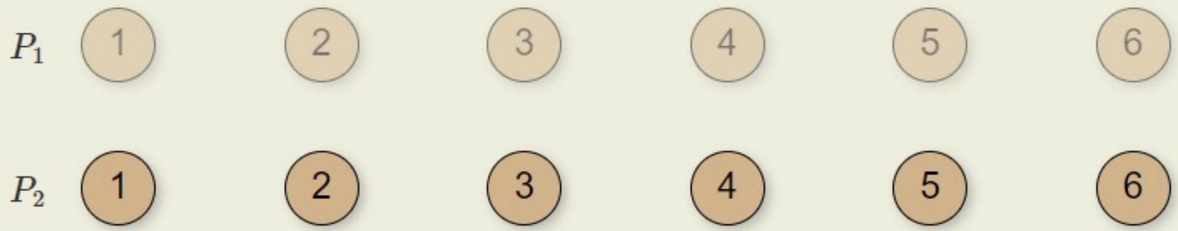
Expression: $(x_1 + x_2 + \overline{x_3}).(\overline{x_2} + x_3 + x_4).(x_1 + \overline{x_2} + x_4)$

Step 1a: Adding nodes for the paths

Variables: x_1 , x_2 , x_3 , x_4

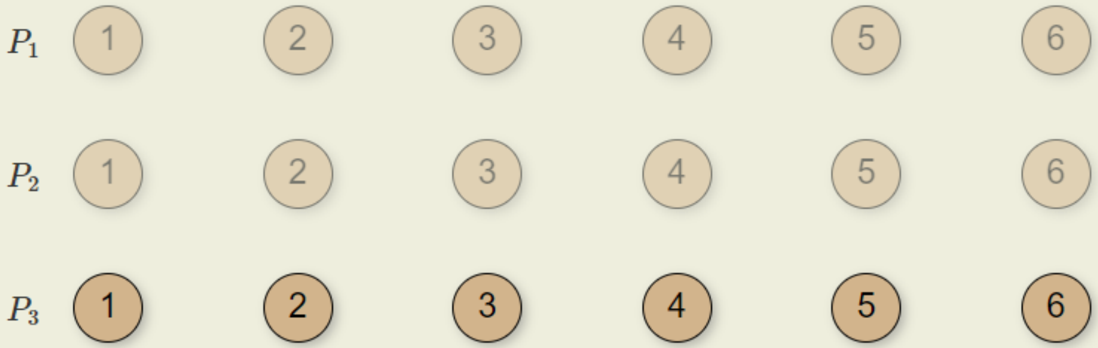
Step 1a: Adding nodes for the paths

Variables: x_1 , x_2 , x_3 , x_4



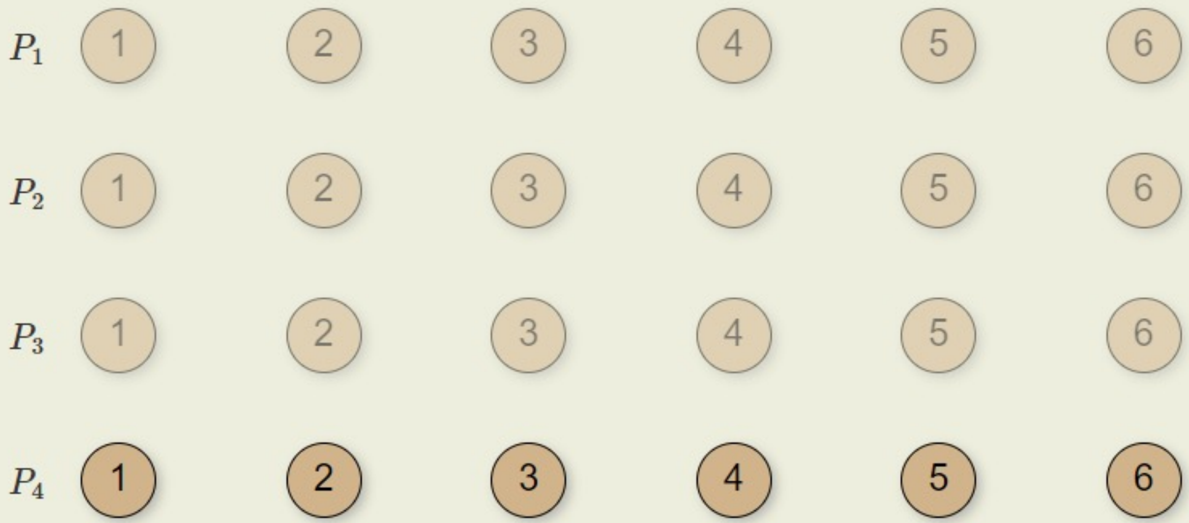
Step 1a: Adding nodes for the paths

Variables: x_1 , x_2 , x_3 , x_4



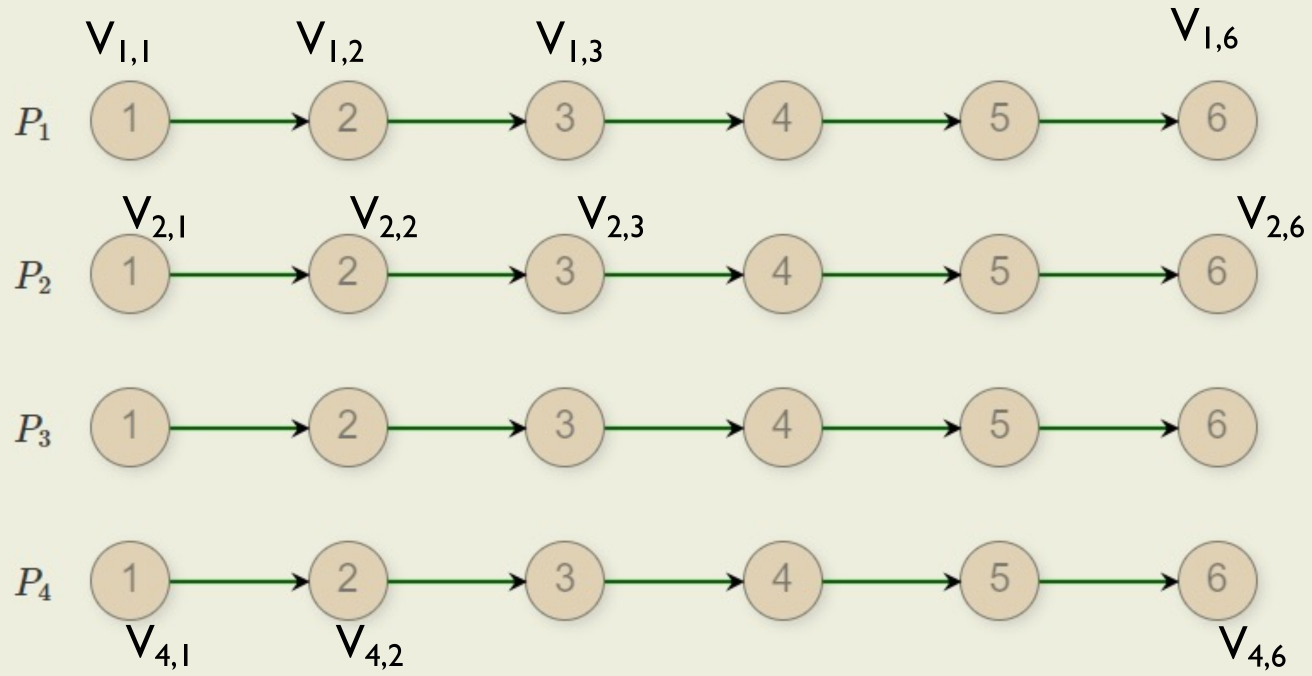
Step 1a: Adding nodes for the paths

Variables: x_1 , x_2 , x_3 , x_4



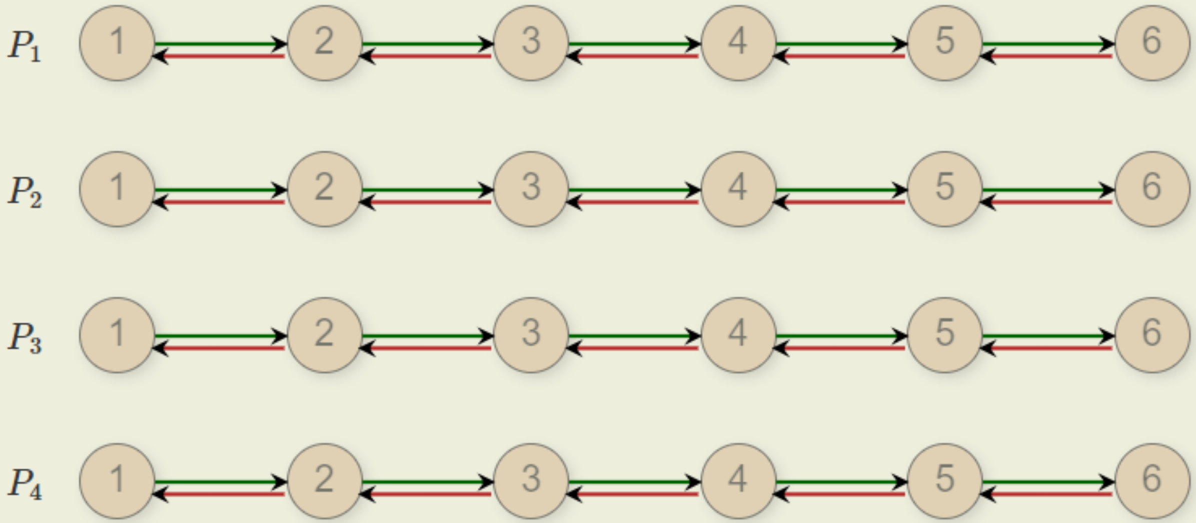
Step 1b: Adding edges to the paths

Add edges from $v_{i,j-1}$ to $v_{i,j}$ (i.e. left to right) on P_i to correspond to the assignment $x_i = True$



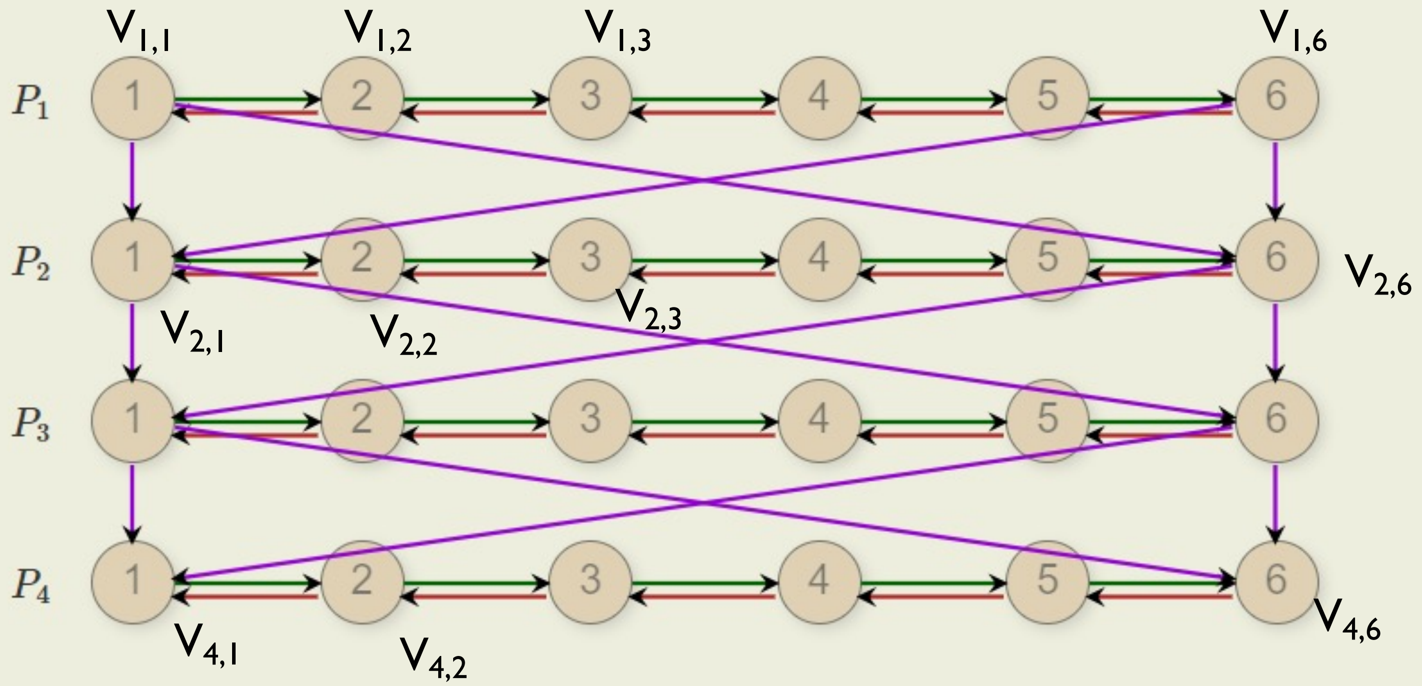
Step 1b: Adding edges to the paths

Add edges from $v_{i,j}$ to $v_{i,j-1}$ (i.e. right to left) on P_i to correspond to the assignment $x_i = False$

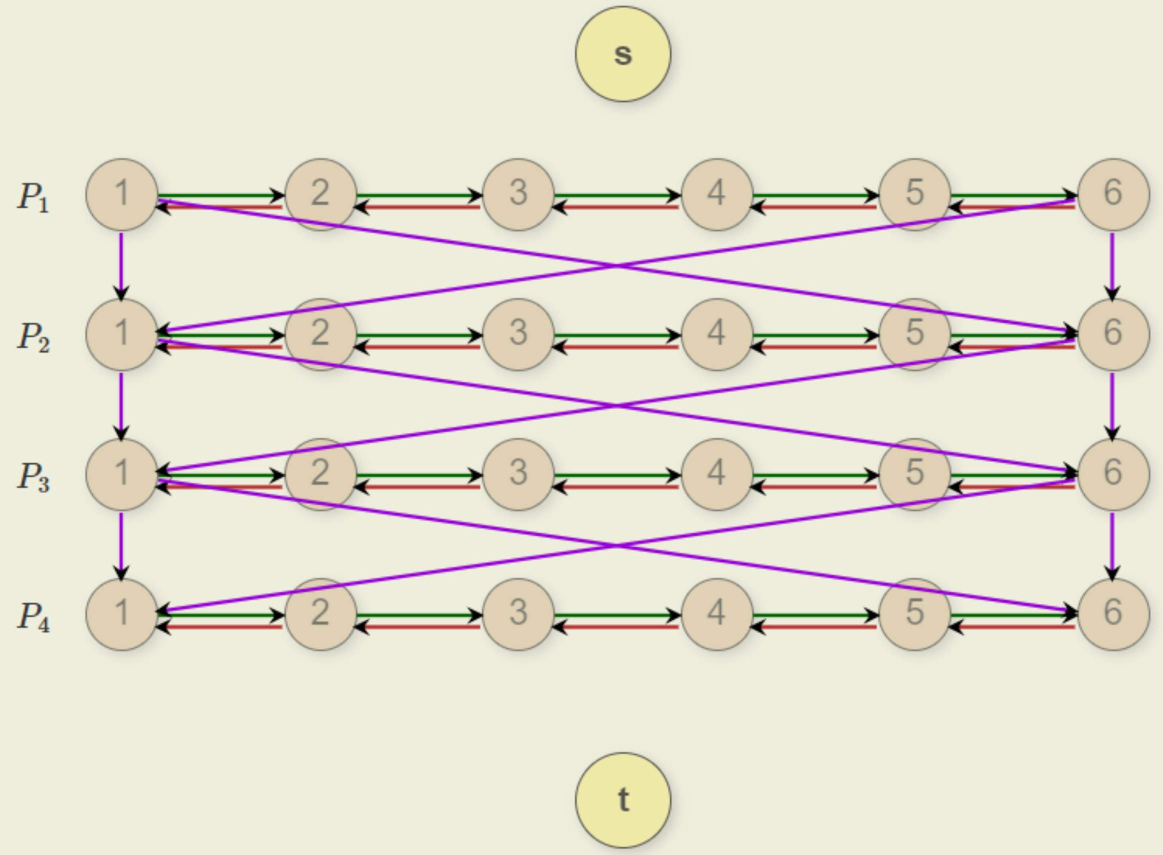


Step 2: Inter-connecting the paths

Add edges from $v_{i,1}$ and $v_{i,6}$ to $v_{i+1,1}$ and $v_{i+1,6}$

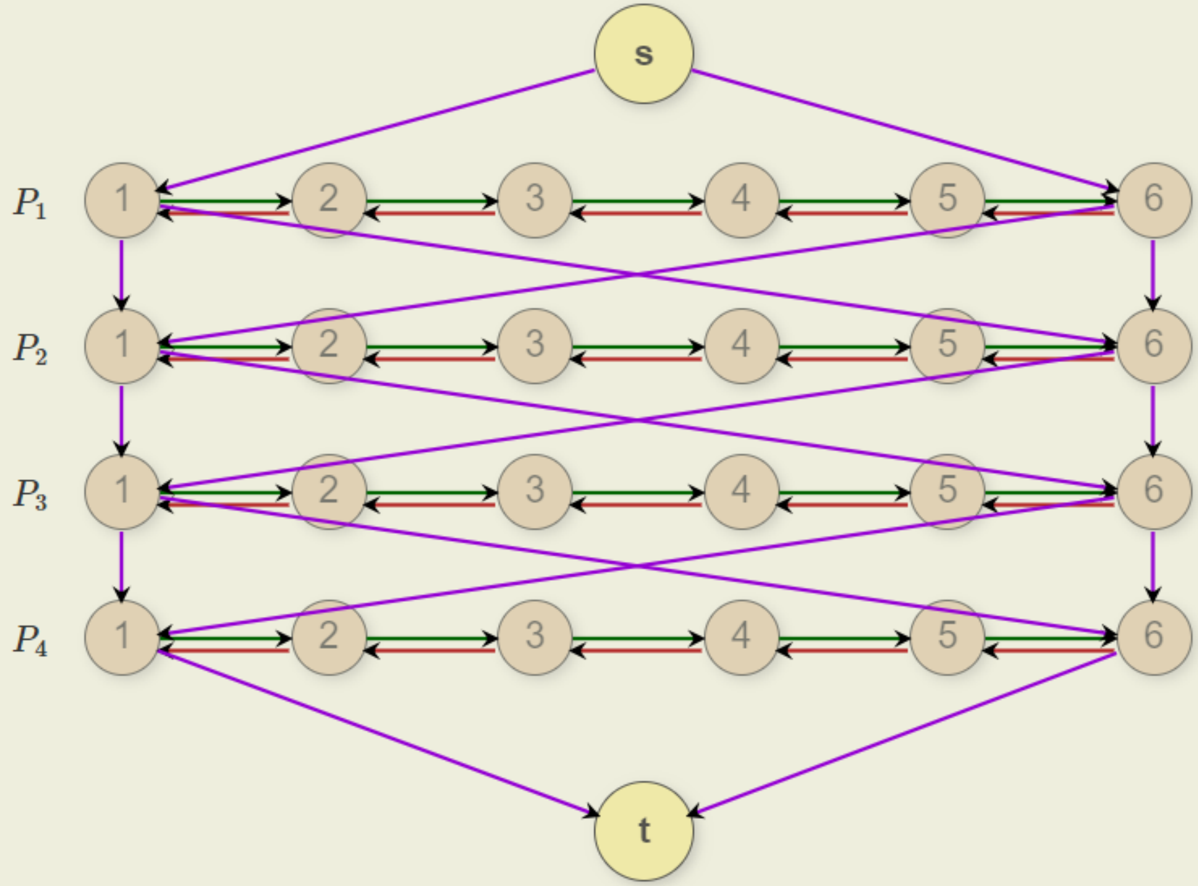


Step 3: Adding source and target nodes



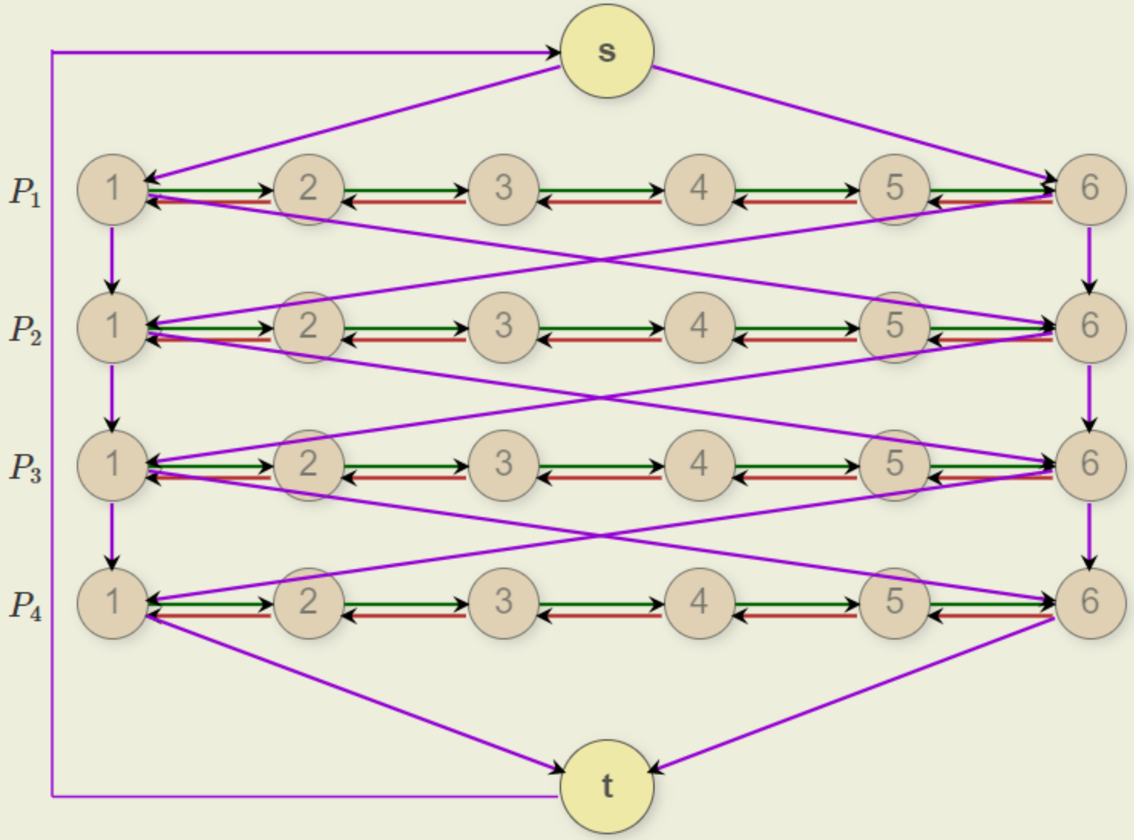
Step 4: Connecting source and target nodes to the paths

Add edges from 's' to $v_{1,1}$ and $v_{1,6}$ and from $v_{4,1}$ and $v_{4,6}$ to 't'



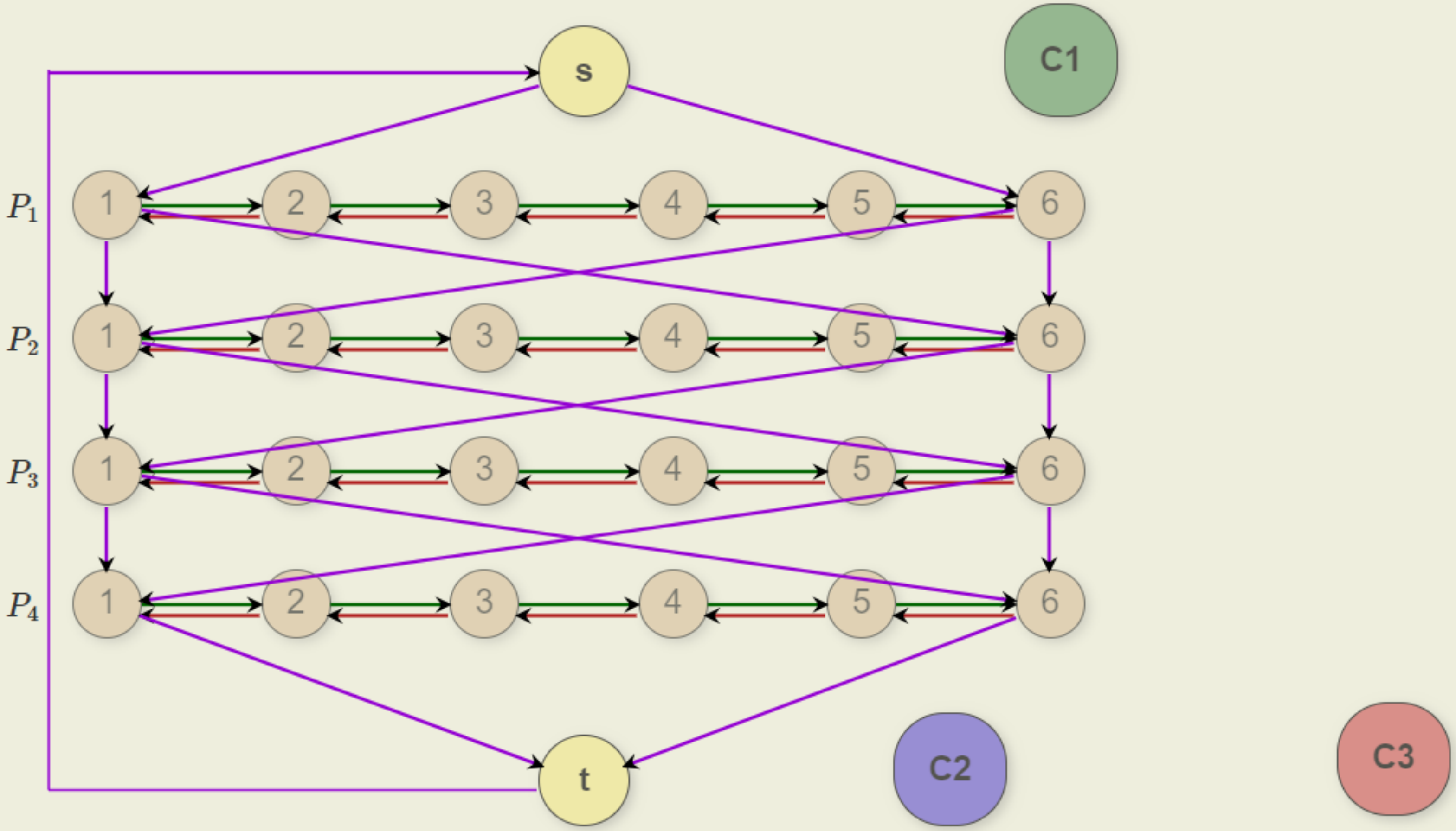
Step 5: Adding a backpath from target to source

Being the only path from target to source, this path will always be present in any Hamiltonian Cycle of the graph.



Step 6: Adding nodes corresponding to clauses

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_2} + x_3 + x_4) \cdot (x_1 + \overline{x_2} + x_4) \cdot$



Step 7: Connecting clauses to the paths

If a clause C_j contains the variable x_i,

1. Connect C_j to $v_{i,2j-1}$ and $v_{i,2j}$
2. The direction of the path connecting C_j, $v_{i,2j-1}$ and $v_{i,2j}$ should be:

Step 7: Connecting clauses to the paths

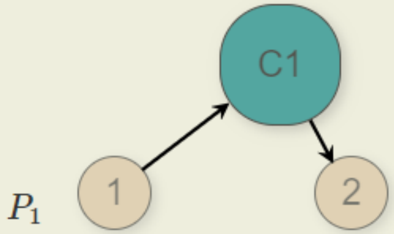
If a clause C_j contains the variable x_i,

1.Connect C_j to $v_{i,2j-1}$ and $v_{i,2j}$

2.The direction of the path connecting C_j,$v_{i,2j-1}$ and $v_{i,2j}$ should be:

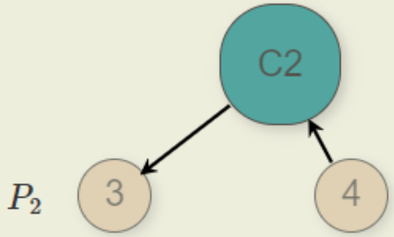
a. left to right if C_j contains x_i

For example : $C_1 = (x_1 + x_2 + \overline{x_3})$ contains x_1. So C_1 should be connected as:



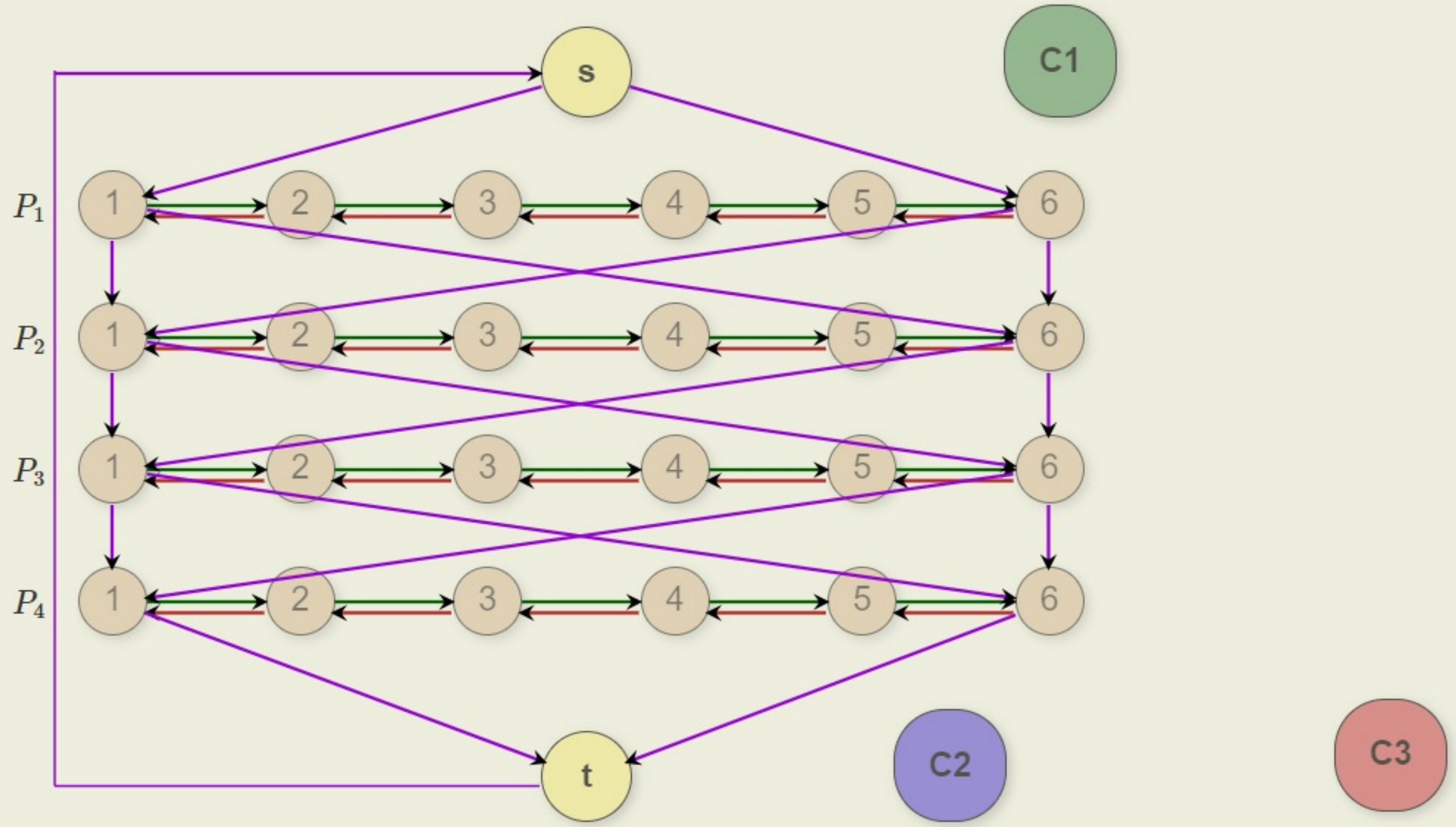
b. right to left if C_j contains $\overline{x_i}$

For example : $C_2 = (\overline{x_2} + x_3 + x_4)$ contains $\overline{x_2}$. So C_2 should be connected as:



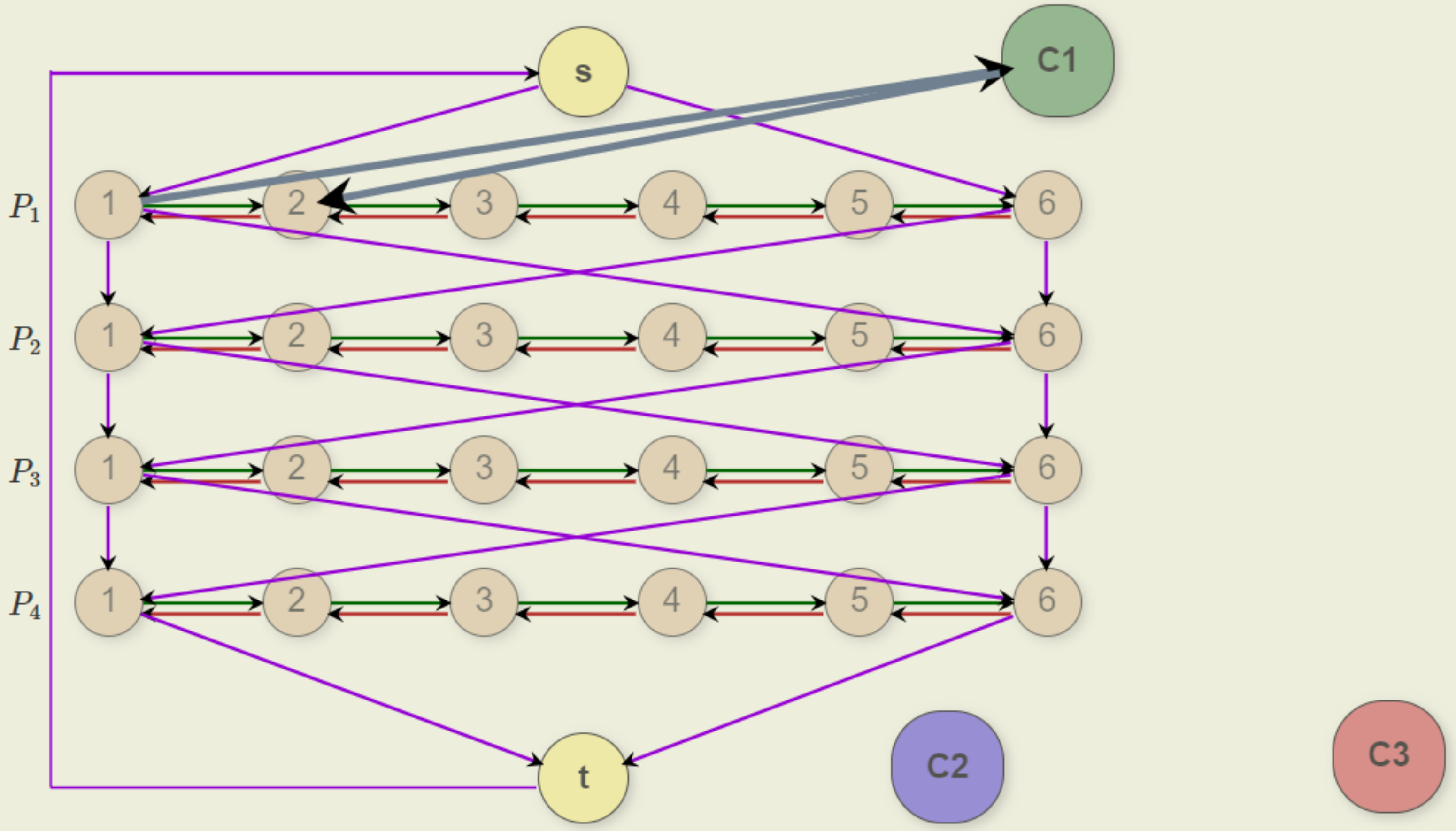
Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_2} + x_3 + x_4) \cdot (x_1 + \overline{x_2} + x_4)$.



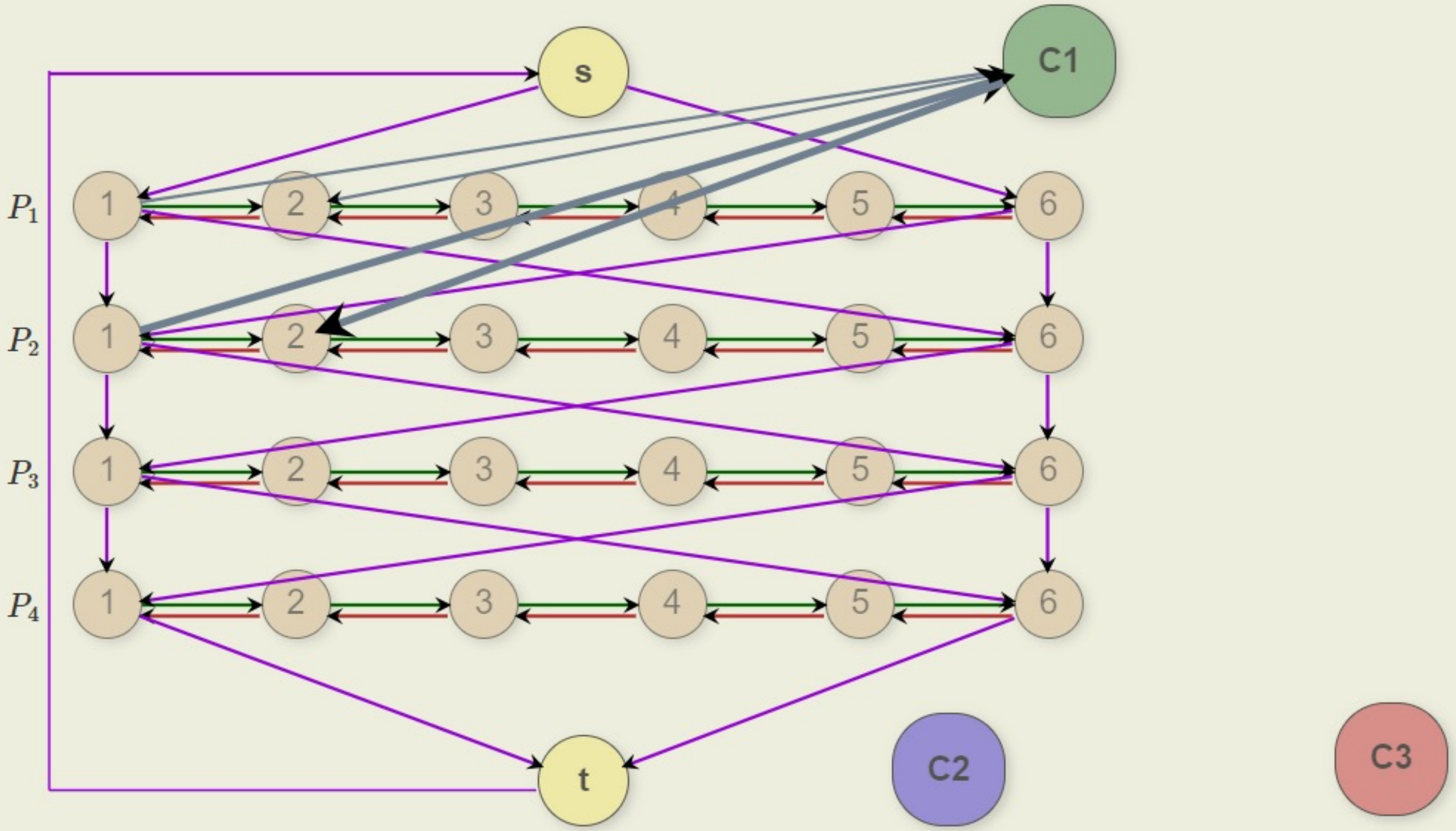
Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) . (\overline{x_2} + x_3 + x_4) . (x_1 + \overline{x_2} + x_4) .$



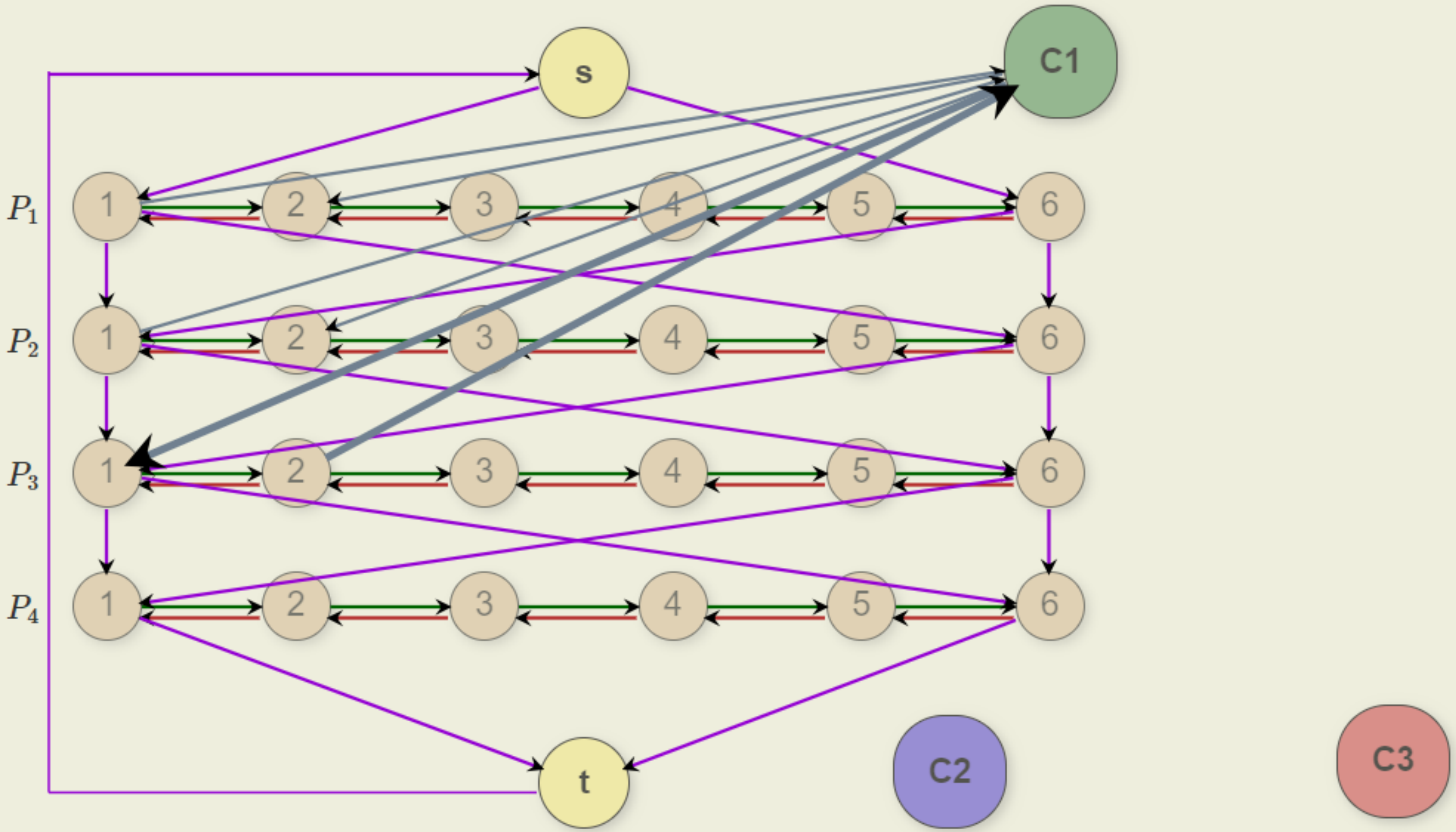
Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_2} + x_3 + x_4) \cdot (x_1 + \overline{x_2} + x_4)$.



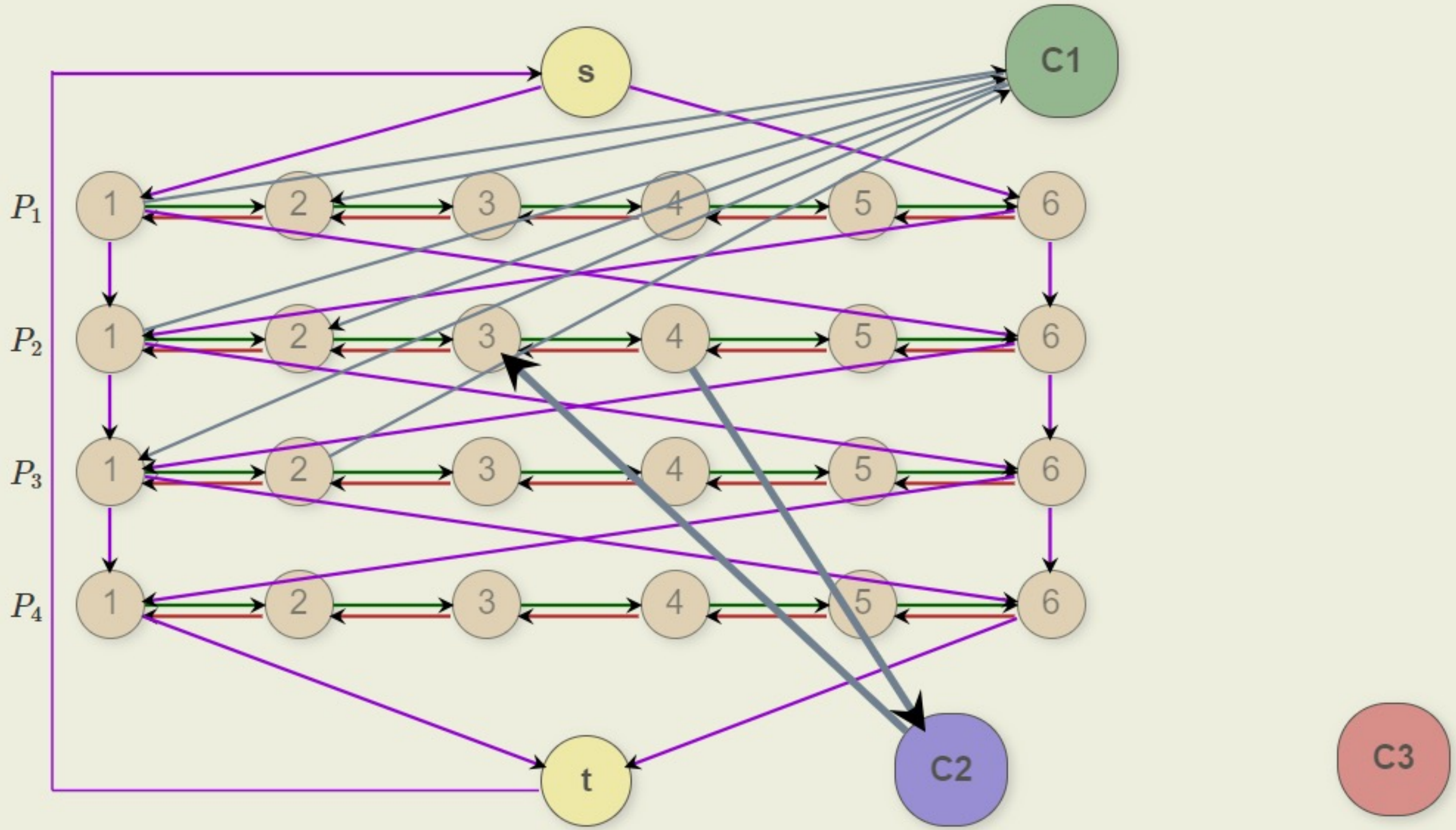
Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_2} + x_3 + x_4) \cdot (x_1 + \overline{x_2} + x_4) \cdot$



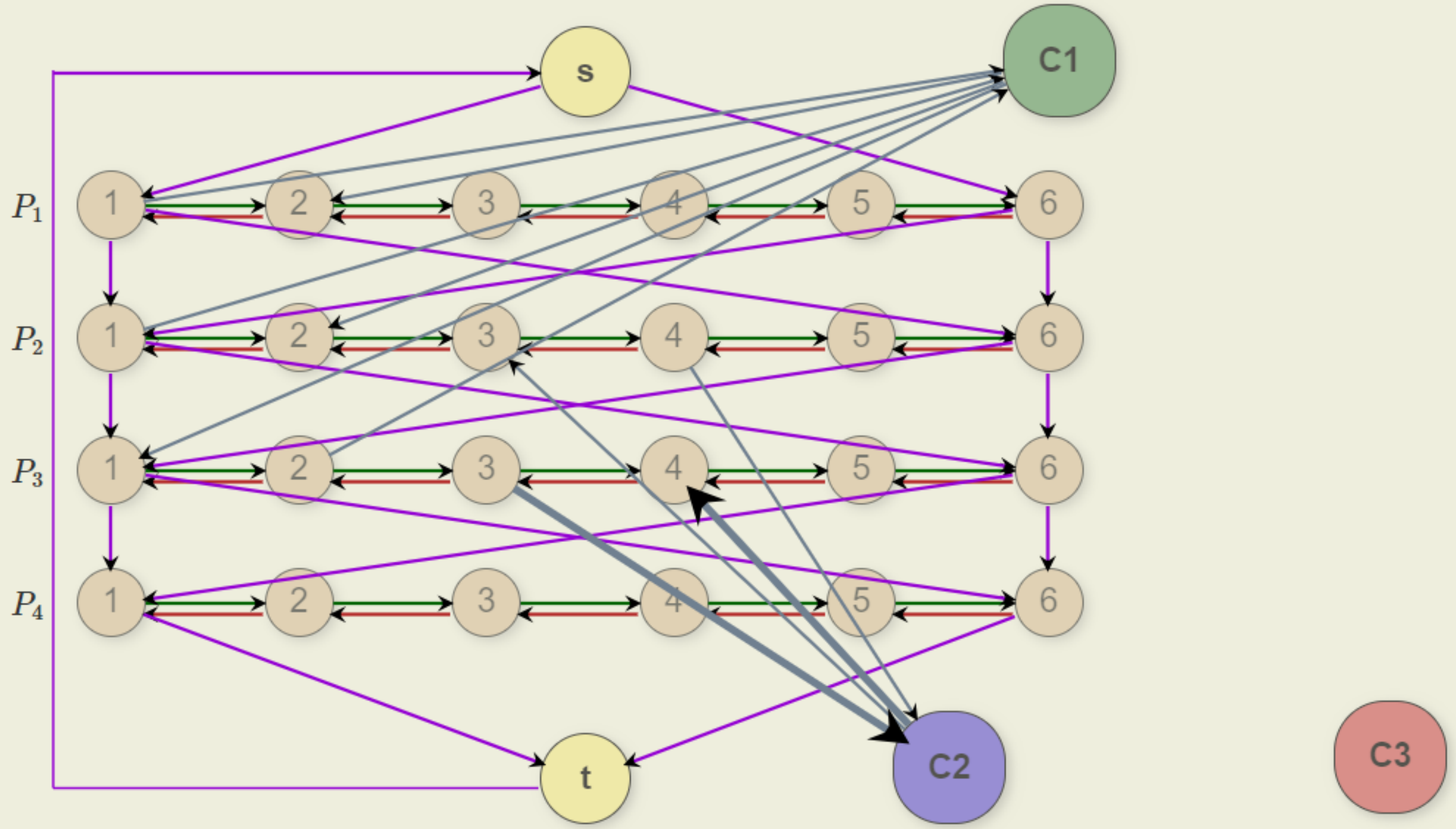
Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_2} + x_3 + x_4) \cdot (x_1 + \overline{x_2} + x_4)$.



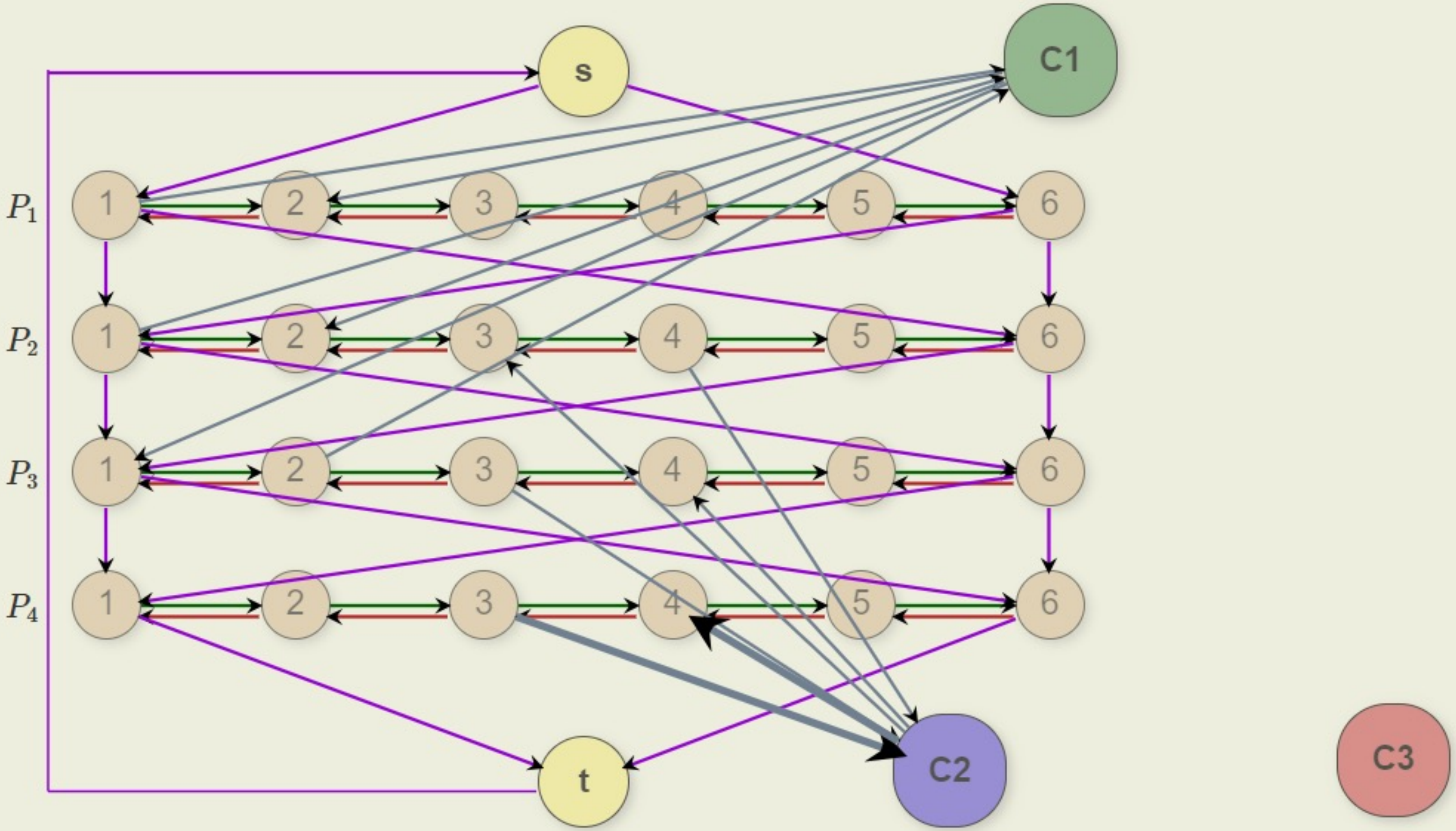
Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_2} + x_3 + x_4) \cdot (x_1 + \overline{x_2} + x_4)$.



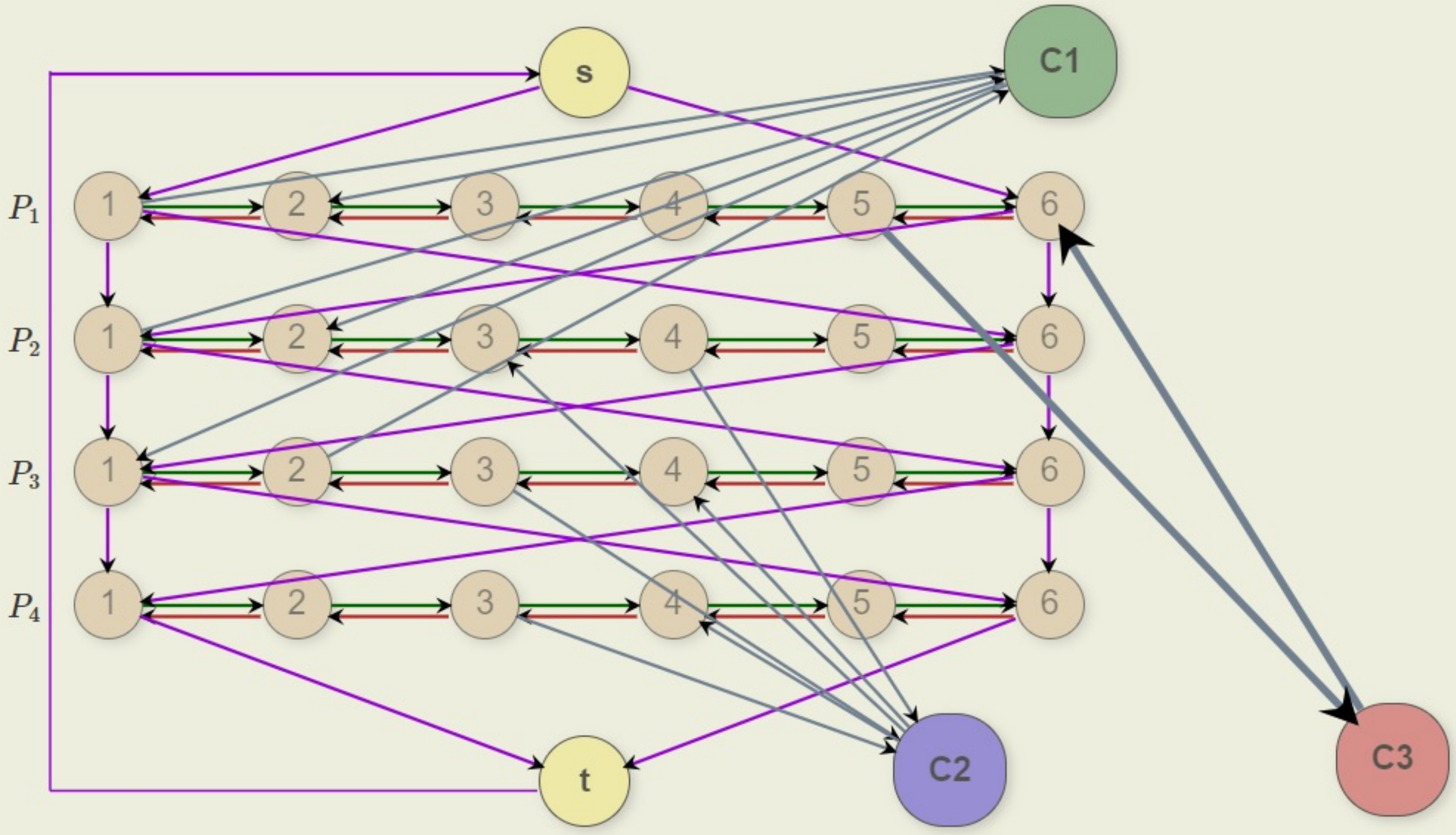
Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_2} + x_3 + x_4) \cdot (x_1 + \overline{x_2} + x_4) \cdot$



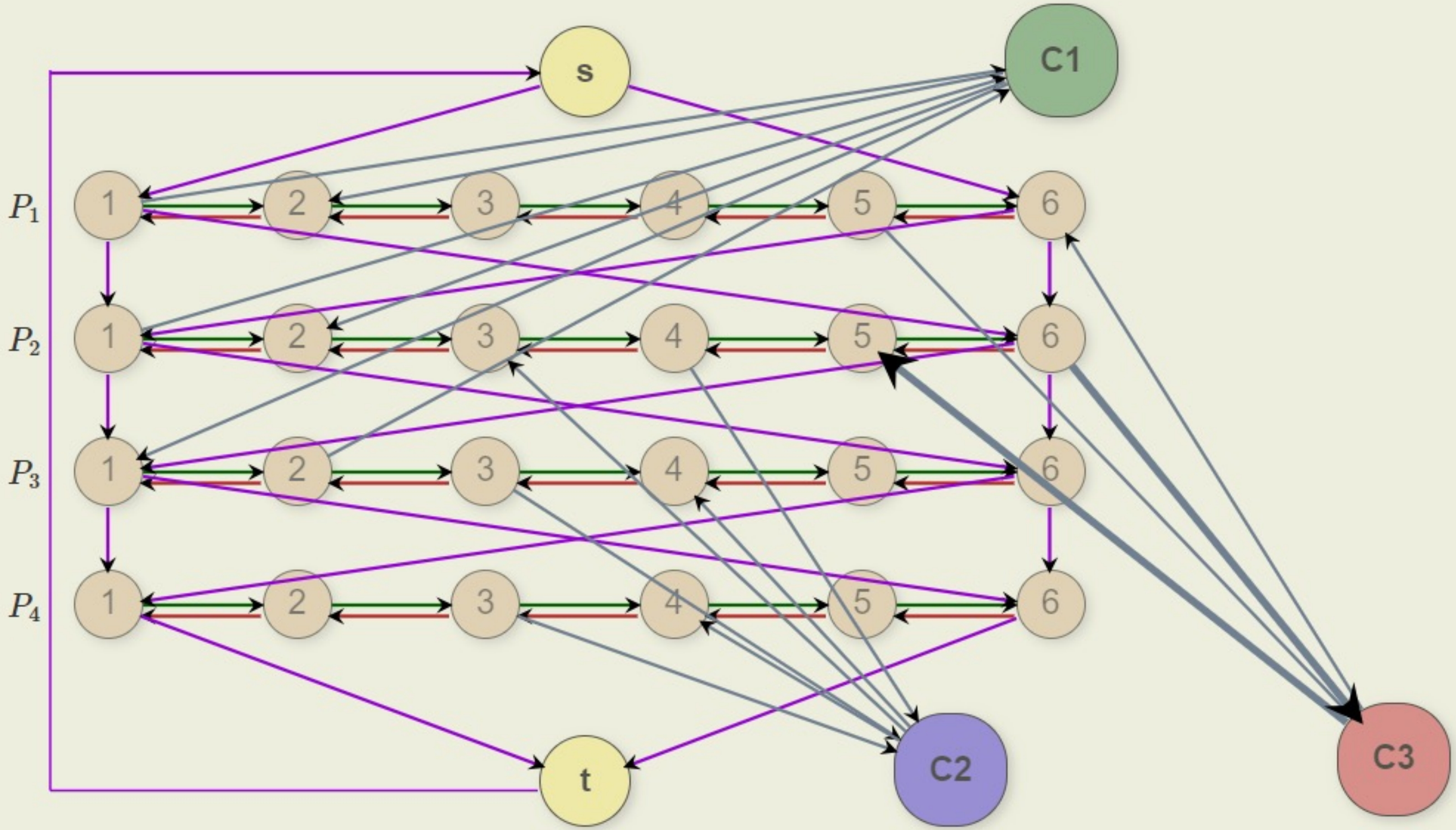
Step7: Connecting clauses to the paths

3-CNF expression: $(\ x_1\ +\ x_2\ +\ \overline{x_3}\ )\ .\ (\ \overline{x_2}\ +\ x_3\ +\ x_4\ )\ .\ (\ x_1\ +\ \overline{x_2}\ +\ x_4\ )\ .$



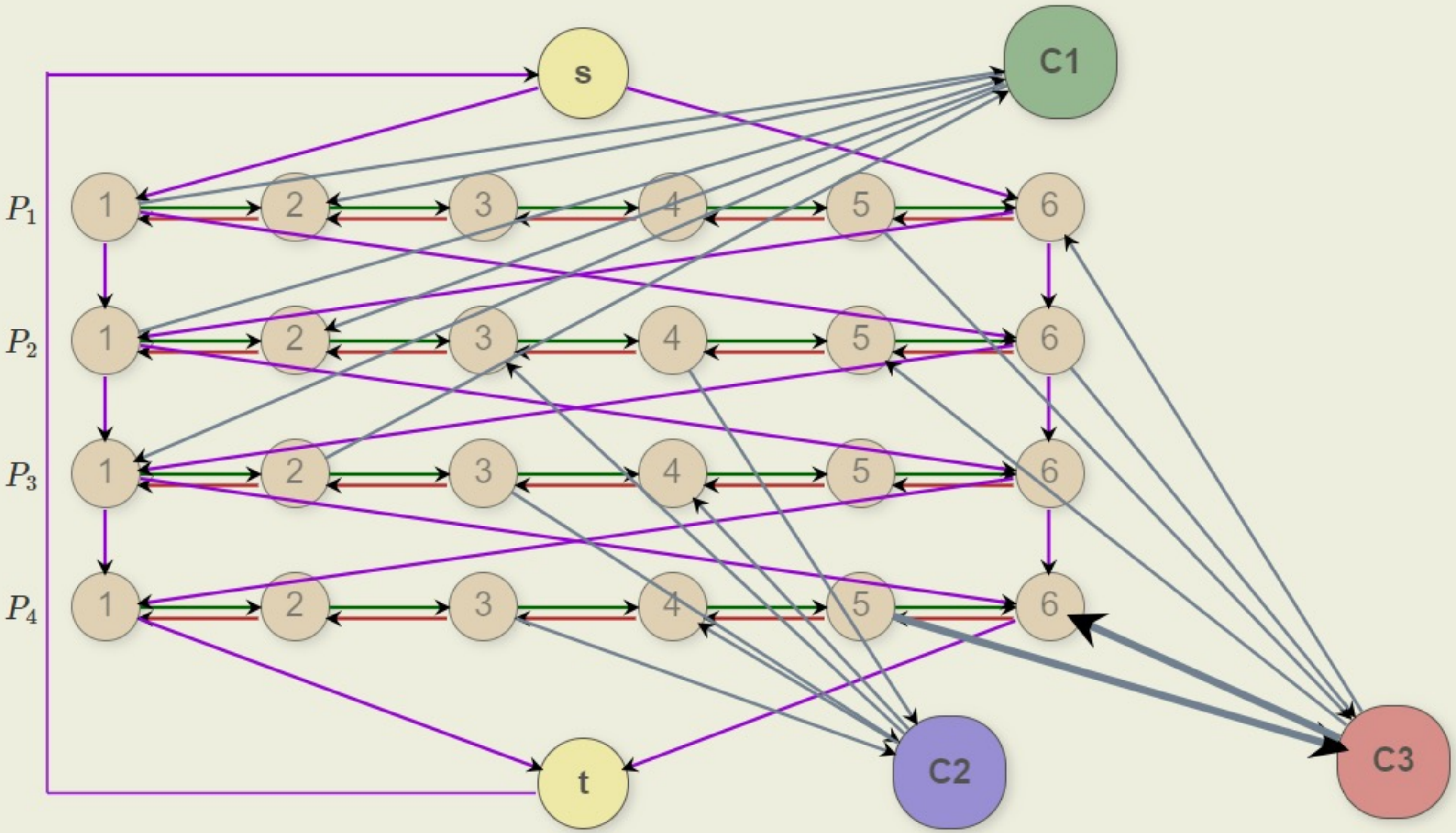
Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_2} + x_3 + x_4) \cdot (x_1 + \overline{x_2} + x_4)$.



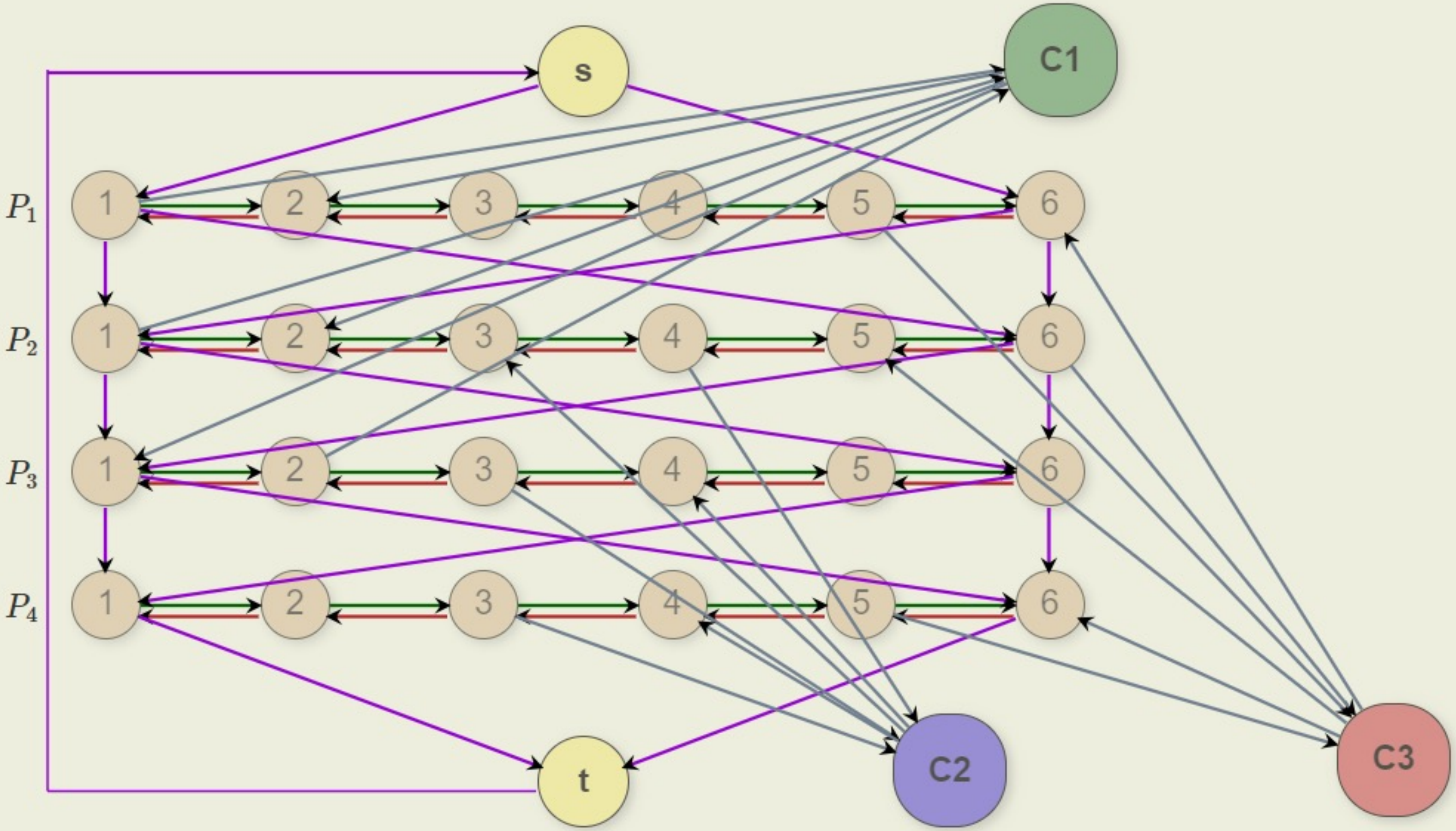
Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_2} + x_3 + x_4) \cdot (x_1 + \overline{x_2} + x_4)$.



Step7: Connecting clauses to the paths

3-CNF expression: $(x_1 + x_2 + \overline{x_3}) \,.\, (\overline{x_2} + x_3 + x_4) \,.\, (x_1 + \overline{x_2} + x_4) \,.$



INSIGHTS ABOUT THE CONSTRUCTED GRAPH

1. Any Hamiltonian Cycle in the constructed graph (G) traverses P_i either from right-to-left or left-to-right. This is because any path entering a node $v_{i,j}$ has to exit from $v_{i,j+1}$ either immediately or via one clause-node in between, in order to maintain Hamiltonian property
Similarly all paths entering at $v_{i,j-1}$ has to exit from $v_{i,j}$.

2. Since each path P_i can be traversed in 2 possible ways and we have n paths mapping to n variables, there can be 2^n Hamiltonian cycles in the graph G - $\{C_1, C_2 \cdots C_k\}$.
Each one of this 2^n Hamiltonian cycles corresponds to a particular assignment for variables $x_1, x_2 \cdots x_n$.

3. **This graph can be constructed in polynomial time.**

1. **If there exists a Hamiltonian cycle H in the graph G,**

If H traverses P_i from left to right, assign $x_i = True$

If H traverses P_i from right to left, assign $x_i = False$

Since H visits each clause node C_j, at least one of P_i was traversed in the right direction relative to the node C_j

The assignment obtained here satisfies the given 3 CNF.

3-SAT AND HAMILTONIAN CYCLE

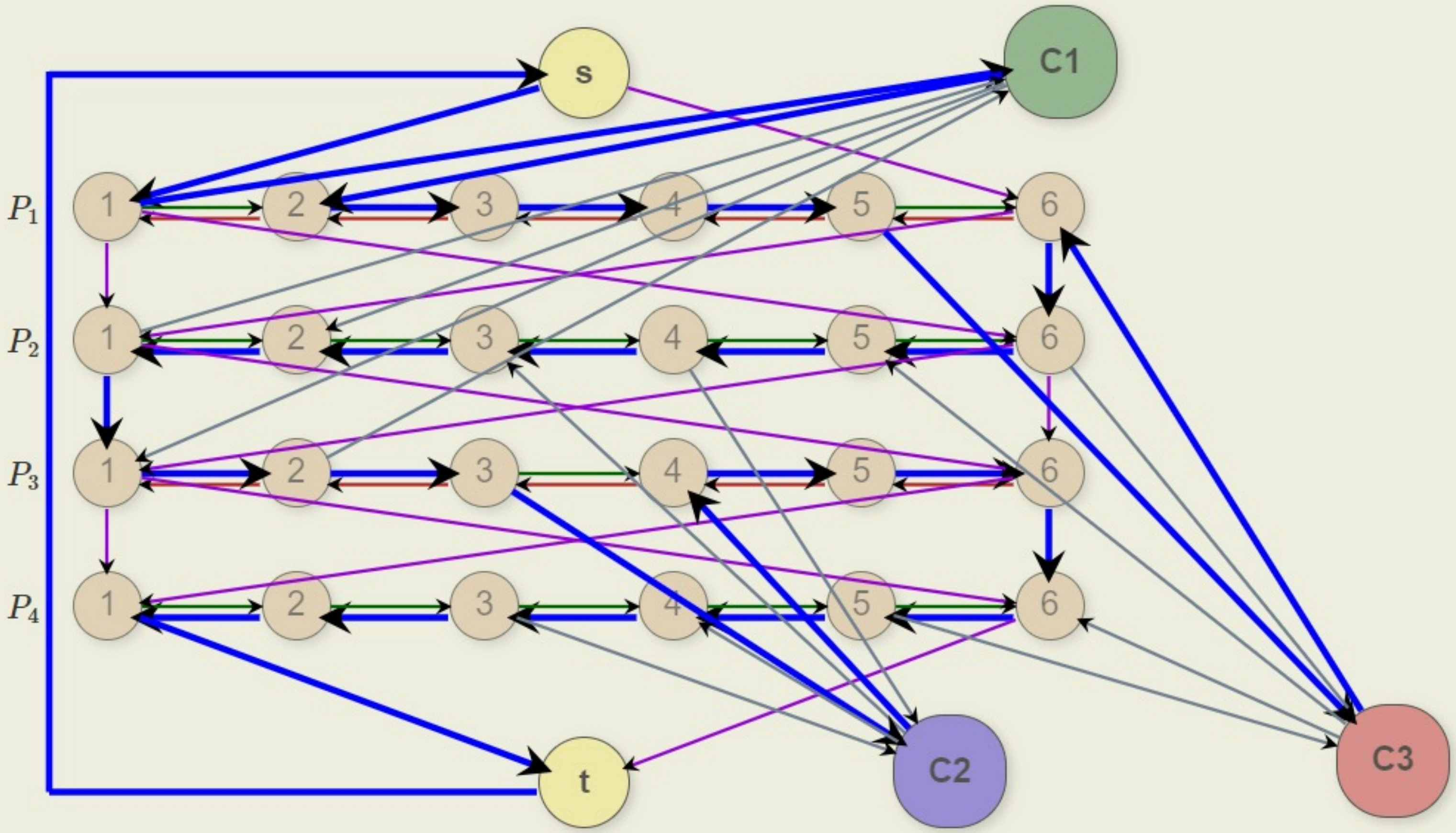
2. **If there exists a satisfying assignment for the 3 CNF**,
Select the path that traverses P_i from left-to-right if $x_i = True$ or right-to-left if $x_i = False$
Include the clauses in the path wherever possible.
Connect the source to P_1, P_n to target and P_i to P_{i+1} appropriately so as to maintain the continuity of the path
Connect the target to source to complete the cycle

Since the assignment is such that every clause is satisfied, all the clause-nodes are included in the path.
The P_i nodes and source and target are all included and since the path traverses unidirectional, no node is repeated twice
The path obtained is a Hamiltonian Cycle

3-SAT AND HAMILTONIAN CYCLE

Hamiltonian Cycle in the constructed graph

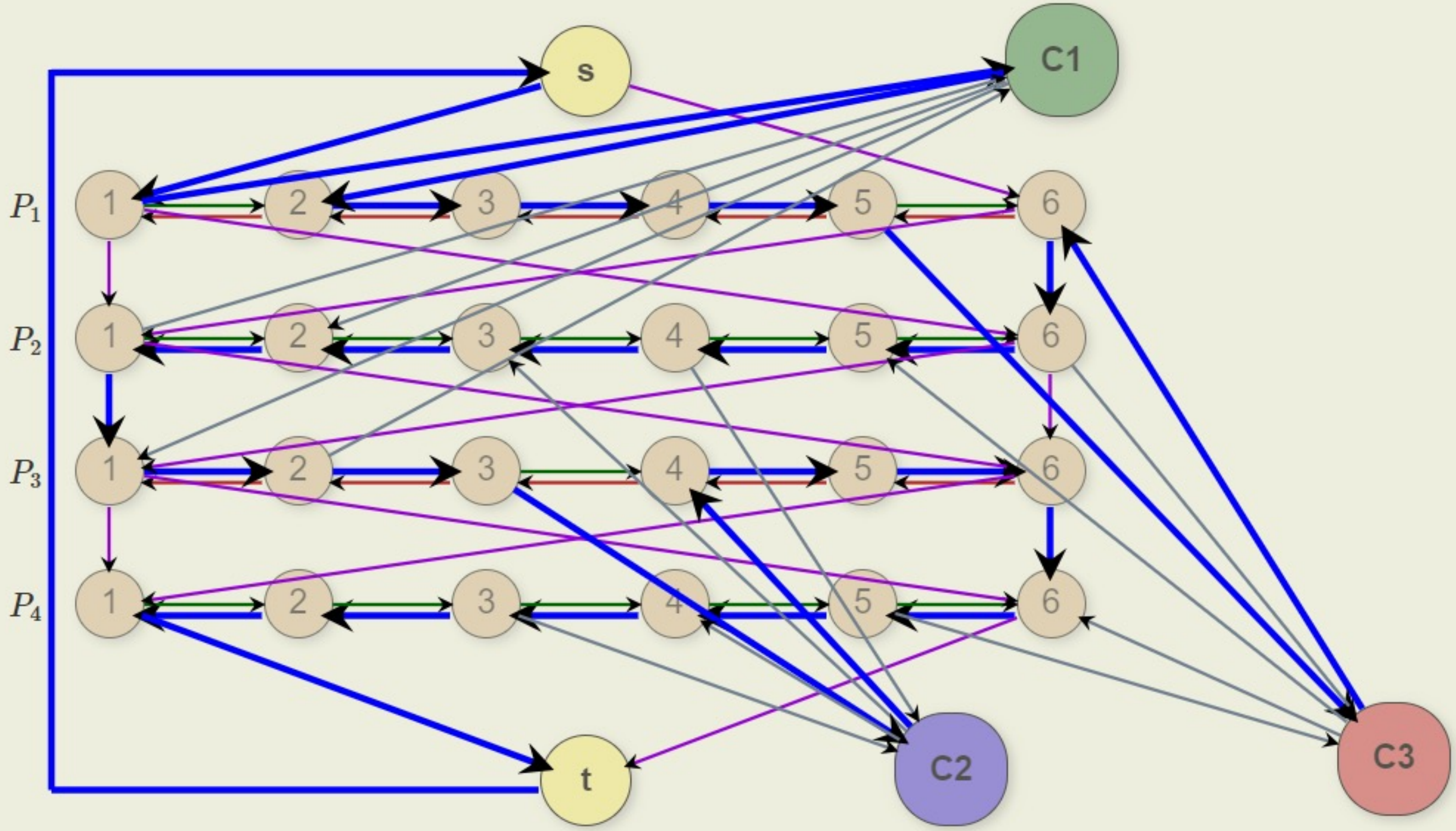
The graph G has a Hamiltonian cycle



Assignment for 3-SAT

From the Hamiltonian cycle below the assignment is :

$x_1 = True$, $x_2 = False$, $x_3 = True$, $x_4 = False$



REDUCTION TO HC TO TRAVELLING SALESMAN PROBLEM (TSP)

Given a complete graph with edge weights, determine the shortest tour that includes all of the vertices (visit each vertex exactly once, and get back to the starting point)

REDUCTION TO HC TO TSP

To reduce the Hamiltonian Cycle Problem to the Traveling Salesman problem for a given graph $G = (V, E)$, complete the graph G, by adding edges between all pairs of vertices that were not connected in G

Let the new graph be $G' = (V', E')$ where $V' = V$ and E'={(u,v)} for any $u, v \in V'$.

For edges in G' that were also present in G , we assign a weight 0.
For other edges we assign weight 1

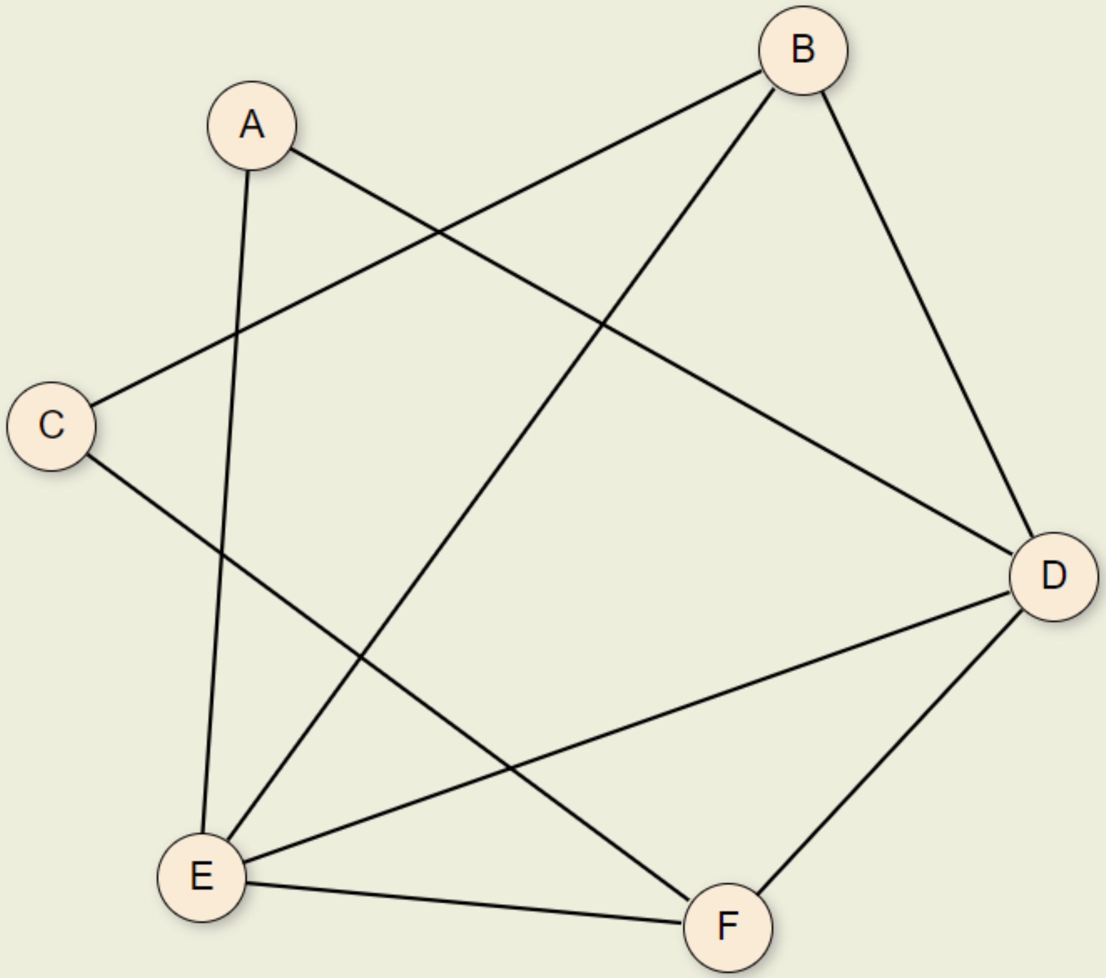
that is , $\forall e = (u, v) \in E'$,

$$W(e) = 0, \text{ if } (u, v) \in E$$
$$W(e) = 1, \text{ if } (u, v) \notin E$$

.

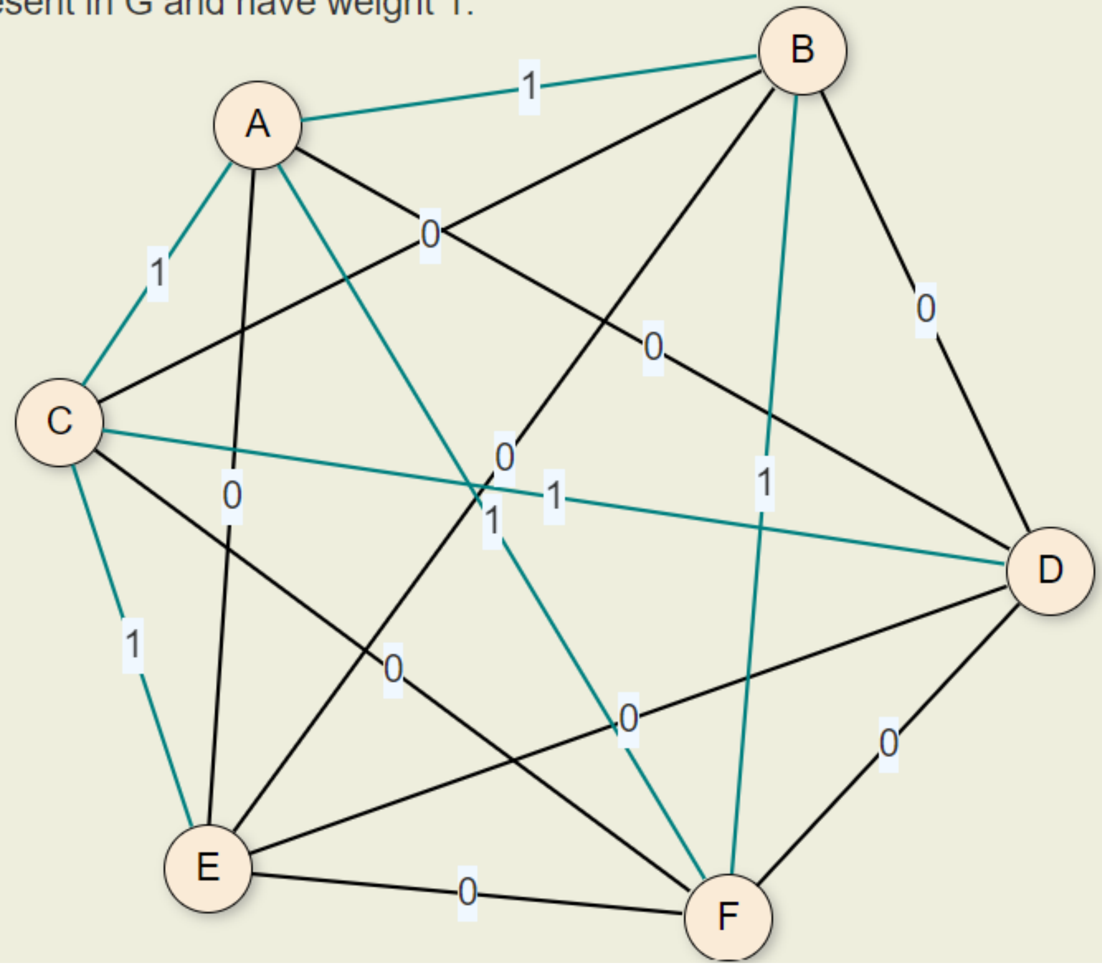
Note: Construction of the complimentary graph can be done in polynomial time

Let this graph G be an input to the Hamiltonian Cycle problem



The constructed graph G' is as below.

The blue edges were not present in G and have weight 1.



Hamiltonian Cycle problem reduced to an instance of Traveling Salesman Problem

The graph G has a Hamiltonian Cycle if and only if there exists a cycle in G' passing through all vertices exactly once, and that has a length $<= 0$ (i.e. has a solution for the instance of Traveling Salesman Problem where $k = 0$

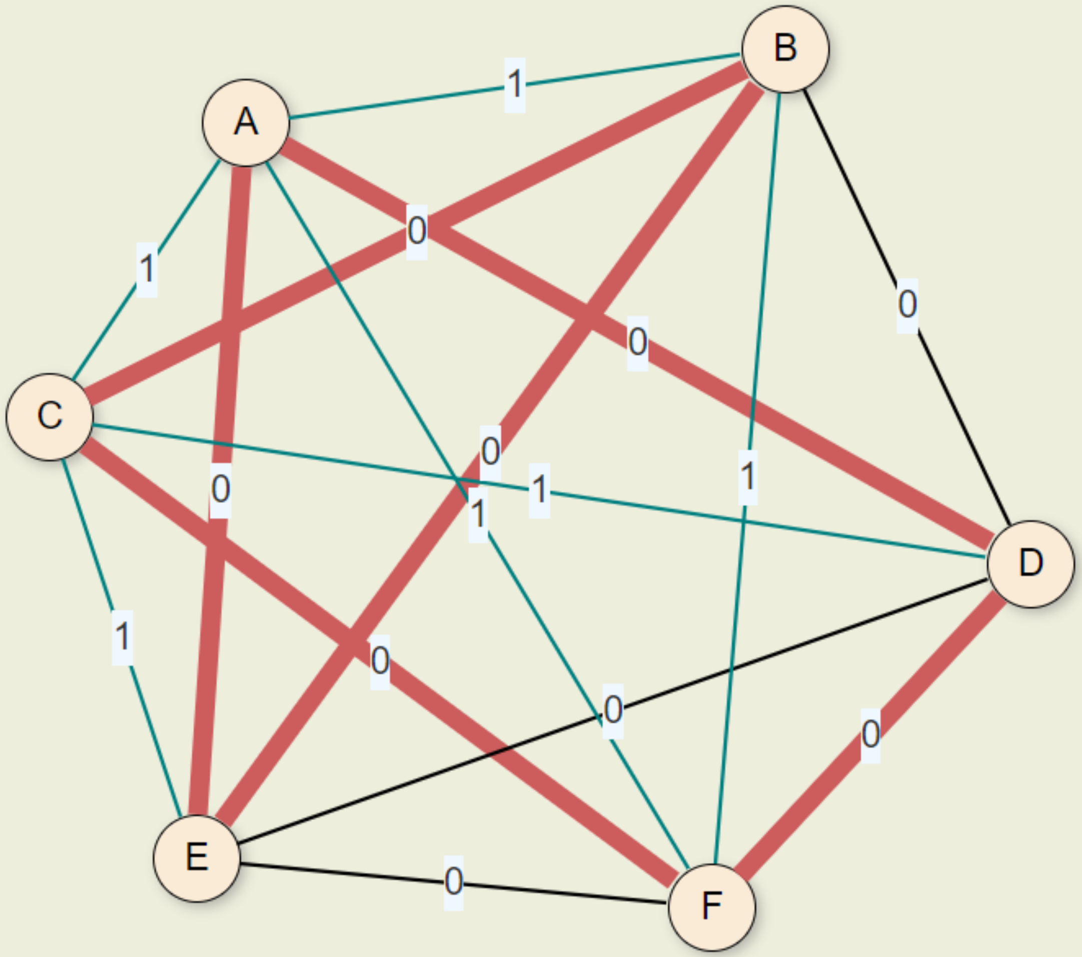
1. **If there is a cycle that passes through all vertice exactly once, and has length $<= 0$ in graph G'**, the cycle contains only edges that were originally present in graph G. (The new edges in G' have weight 1 and hence can not be part of a cycle of length $<= 0$.

Hence **there exist a Hamiltonian cycle in G**

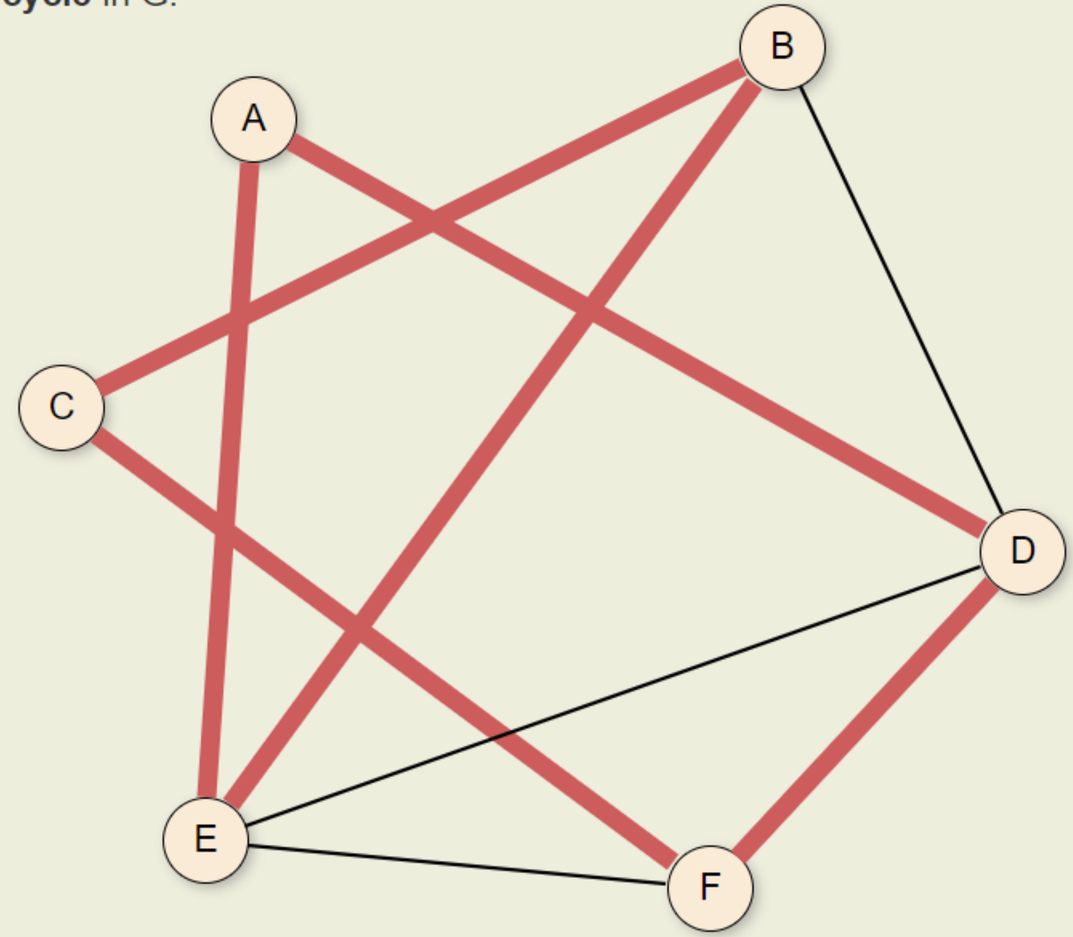
2. **If there exists a Hamiltonian Cycle in the graph G**, it forms a cycle in G' with length $= 0$, since a weights of all the edges is 0.

Hence **there exists a solution for Traveling Salesman Problem in G' with length $<= 0$**

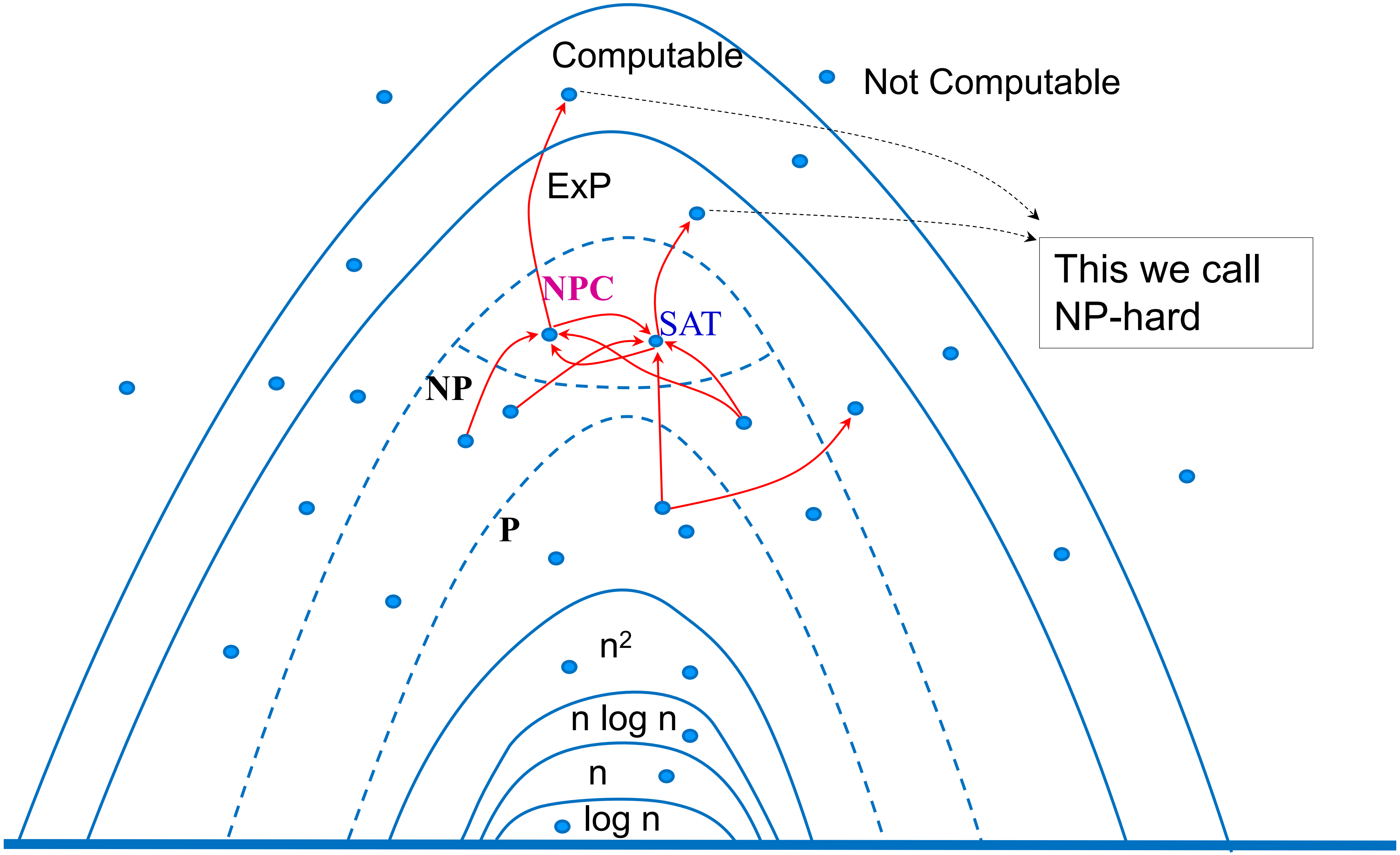
G' has a cycle passing through all vertices exactly once with length $<= 0$



G' has a cycle passing through all vertices exactly once with length $<= 0$
This cycle is a **Hamiltonian cycle** in G.



Complexity Structure



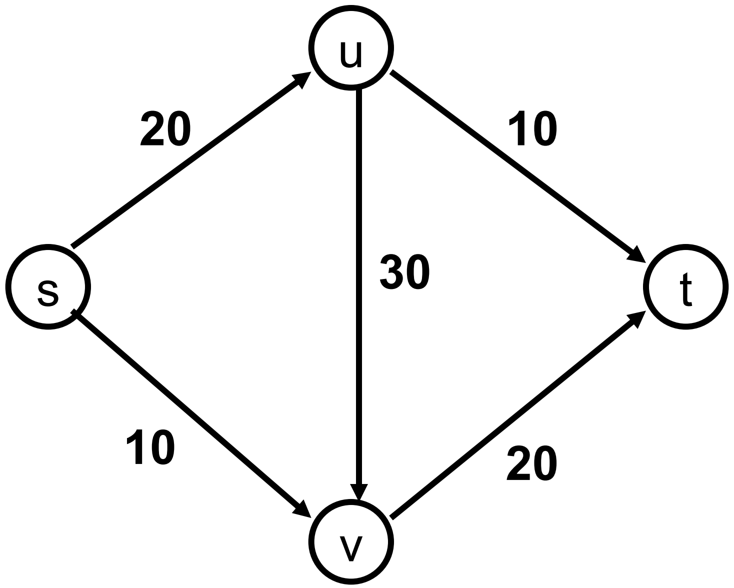
Universe of Problems to be solved

FLOW NETWORK

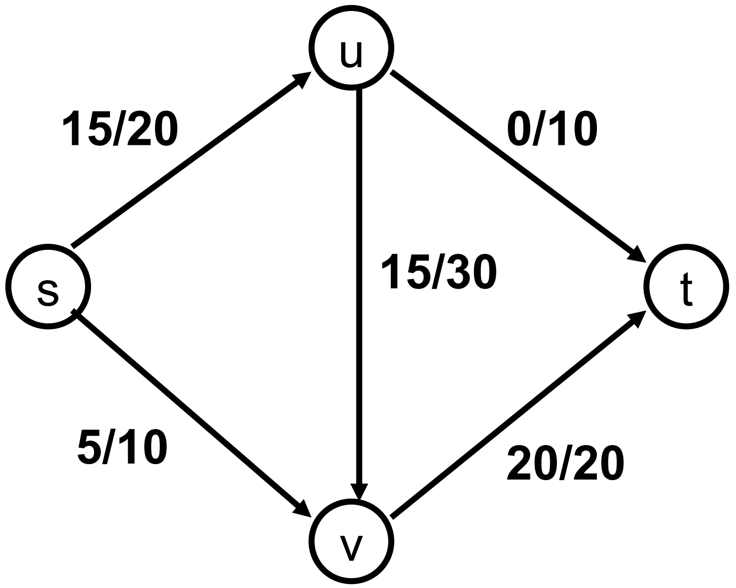
Network Flow Problems

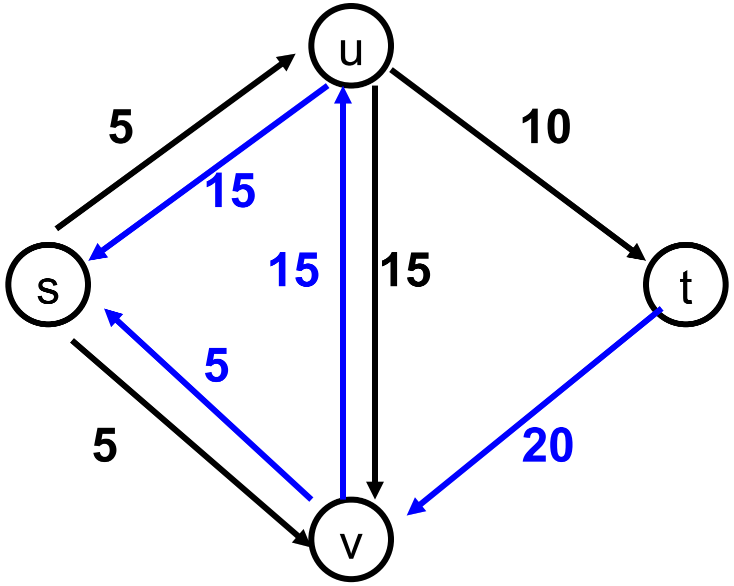
- Weighted, digraph G, or network
- May have cost per unit flow for edges
- May have maximum flow per edge
- May have max production rates
- May have required consumption rate per sink

FLOW EXAMPLE



FLOW ASSIGNMENT AND THE RESIDUAL GRAPH





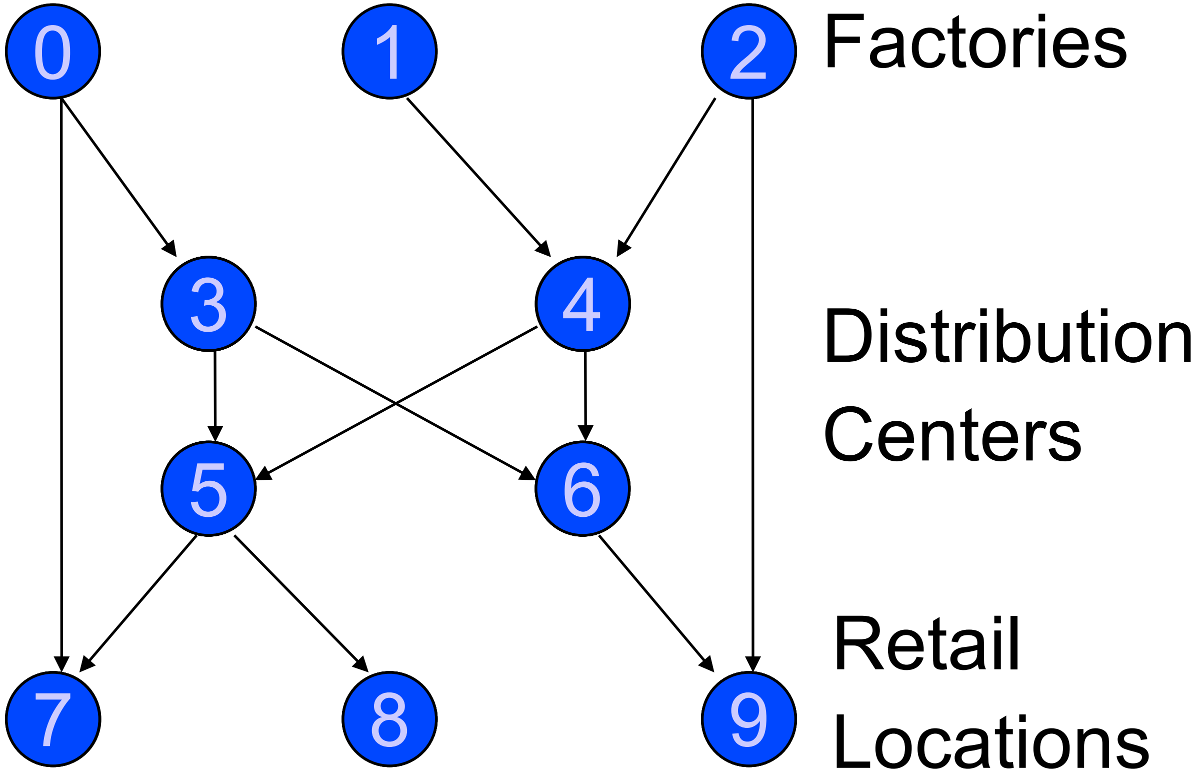
NETWORK FLOW DEFINITIONS

- Flowgraph: Directed graph with distinguished vertices s (source) and t (sink)
- Capacities on the edges, $c(e) >= 0$
- Problem, assign flows $f(e)$ to the edges such that:
  - $0 <= f(e) <= c(e)$
  - Flow is conserved at vertices other than s and t
    - Flow conservation: flow going into a vertex equals the flow going out
  - The flow leaving the source is a large as possible

Distribution Problems

- Merchandise distribution
  - Sources with production rates
  - Sinks with consumption rates
  - Distribution centers
    - Input rate = output rate
  - Channels with maximum rate and unit cost for distribution

Merchandise distribution

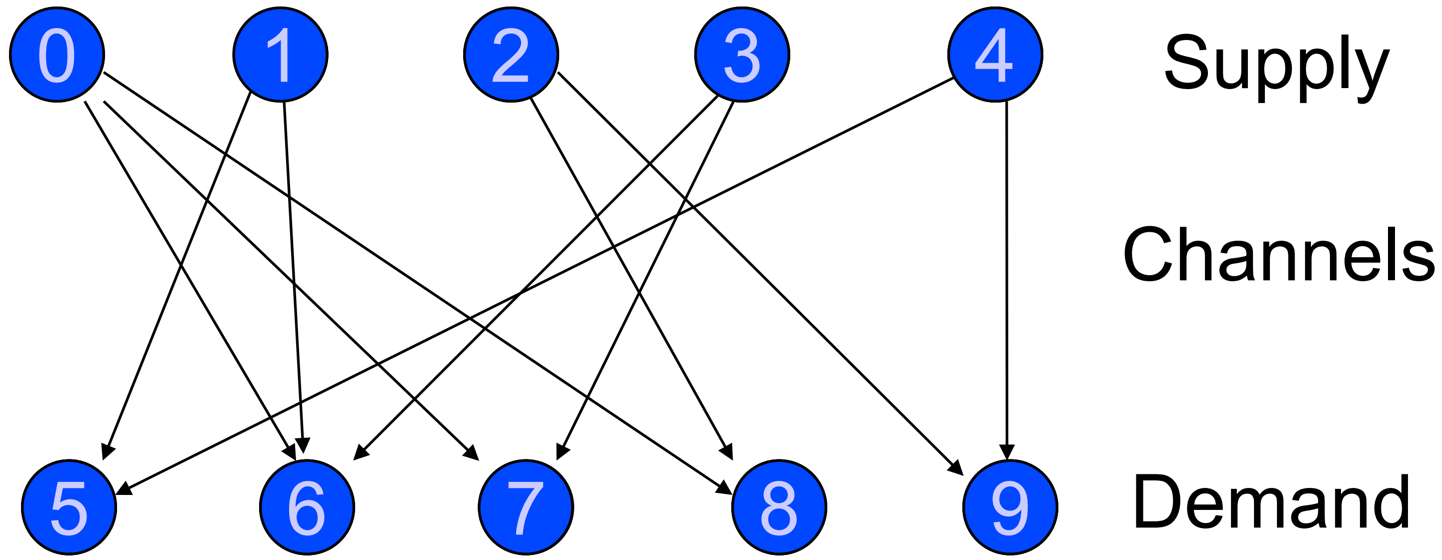


Get product to retail locations cheaply

Transportation Problems

- Communications
  - Max total data rate between a source and sink
  - Cheapest way to move a given amount of data from s to t
- Traffic flow
  - Minimize evacuation time
  - Minimize total cost

Transportation Problem



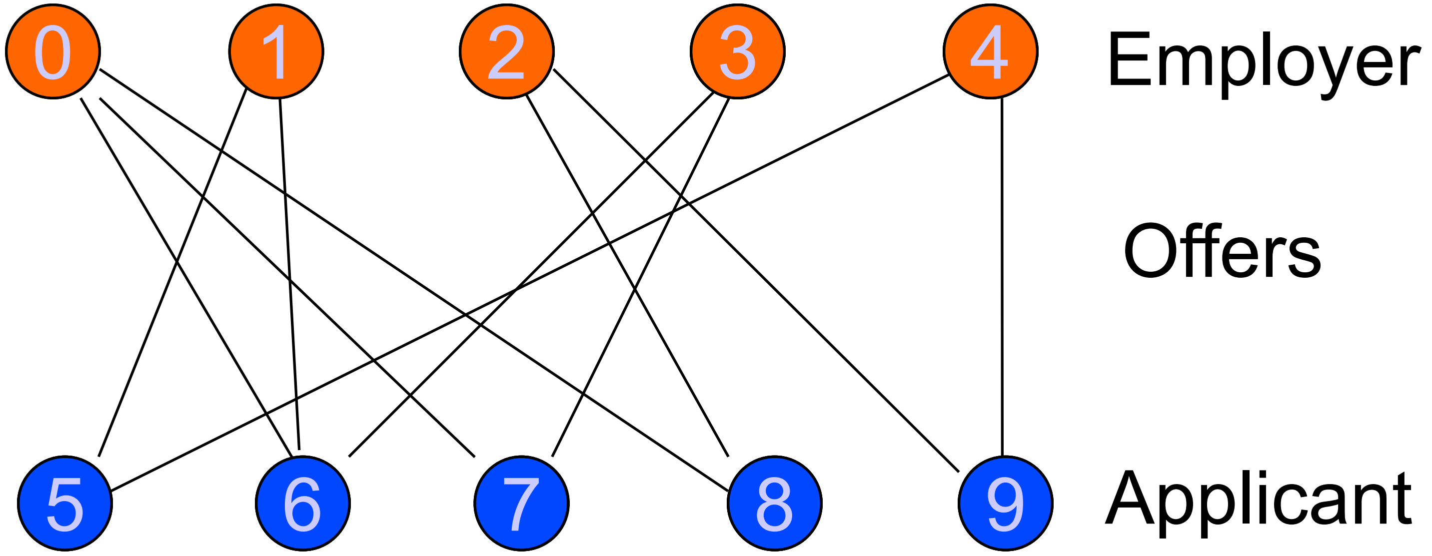
No channel capacity restrictions

Get product to retail locations cheaply

Matching Problems

- Job placement
  - Interviews + job offers
  - Maximize number of placements
- Min-distance point matching
  - Two sets A and B of N points each
  - Find set of N segments matching an element from A with one from B that has lowest cost

Matching Problem

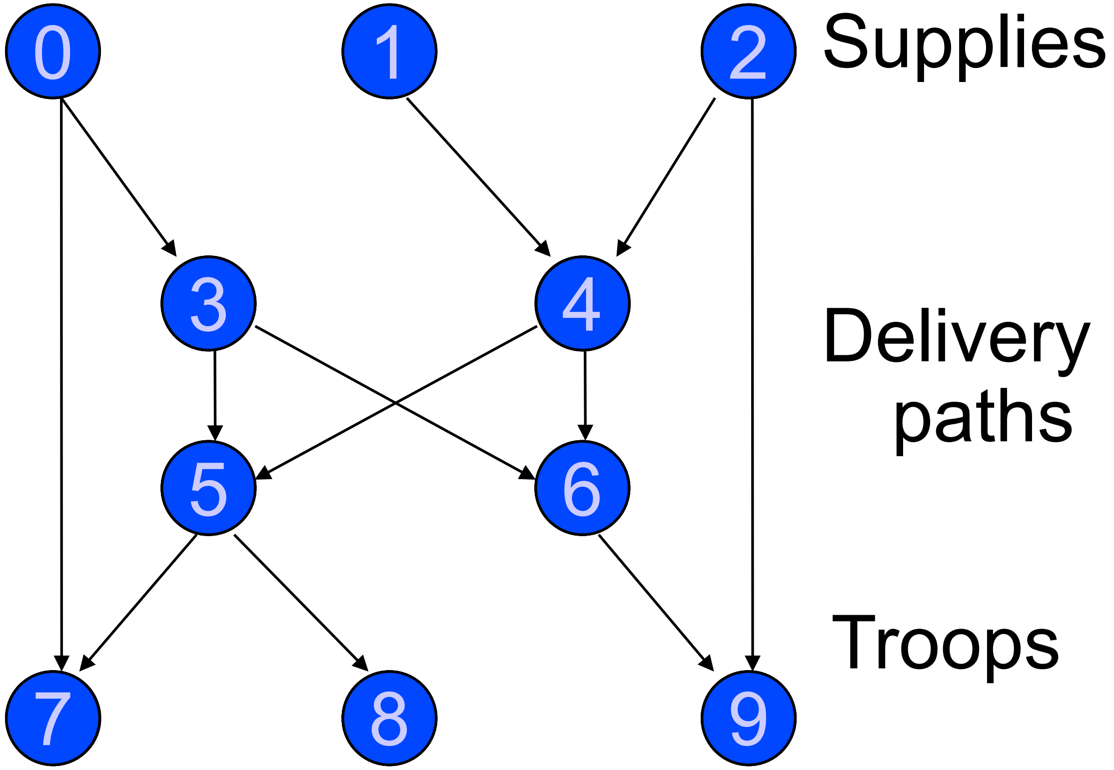


Maximize placements (matching)

Cut Problems

- Network reliability
  - What is minimum number of lines that must be cut to disconnect two switches?
- Supply line cutting
  - What is the minimum supply line destruction required to ensure no troops get supplies?

Cut Problem



How few edges must be cut to disrupt delivery

May have edge weights also

Network Flow Problems

- Generic problems
- Maxflow
  - What is maximum flow between s and t?
- Mincost-flow
  - What is cheapest cost way to achieve a particular flow?

Network Flow

- Flow Networks
- Maxflow Algorithms
- Maxflow Reductions
- Mincost Flows
- Network Simplex Algorithm

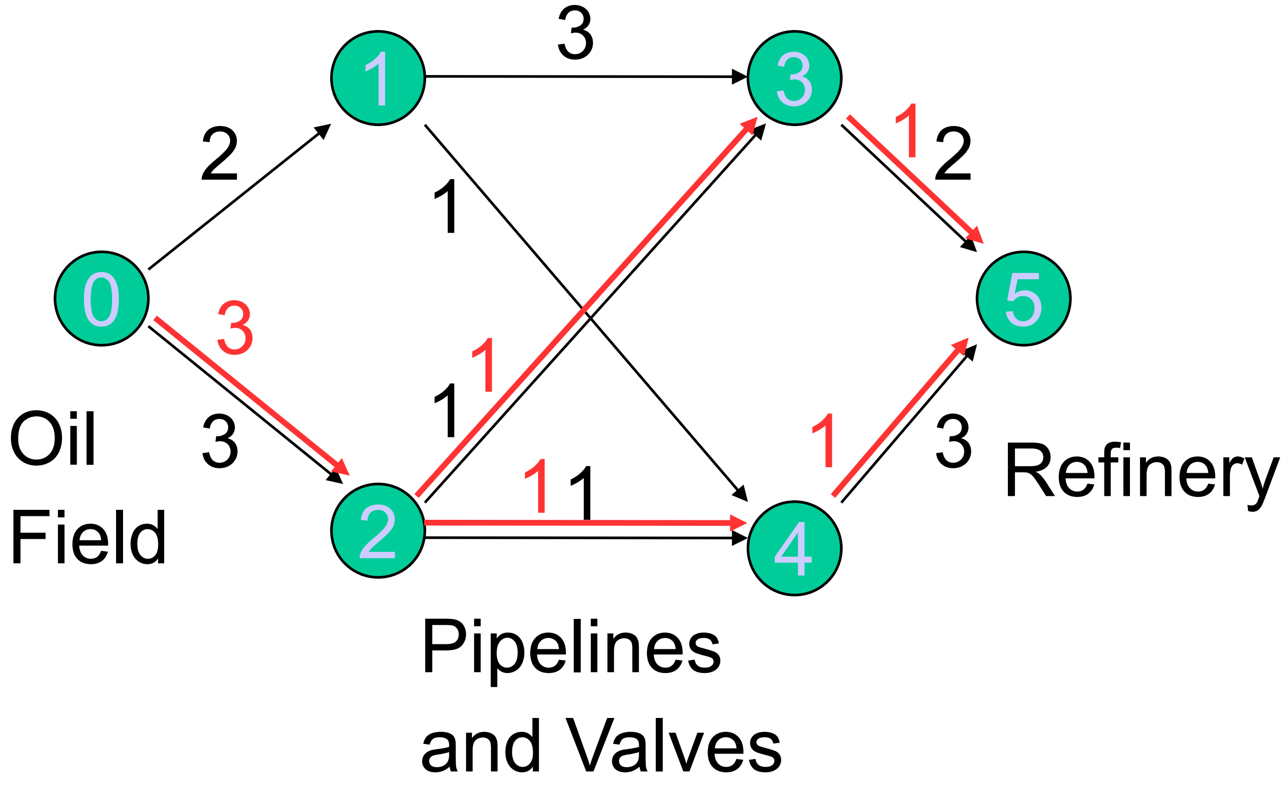
Flow Networks

- A network with a single source and a single sink is an s-t network

Flow Networks

- A flow network is an s-t network with positive edge weights, called capacities.
- A flow in a flow network is a set of non-negative edge weights called edge flows satisfying:
    - No edge flow exceeds capacity
    - In flow = out flow for interior nodes

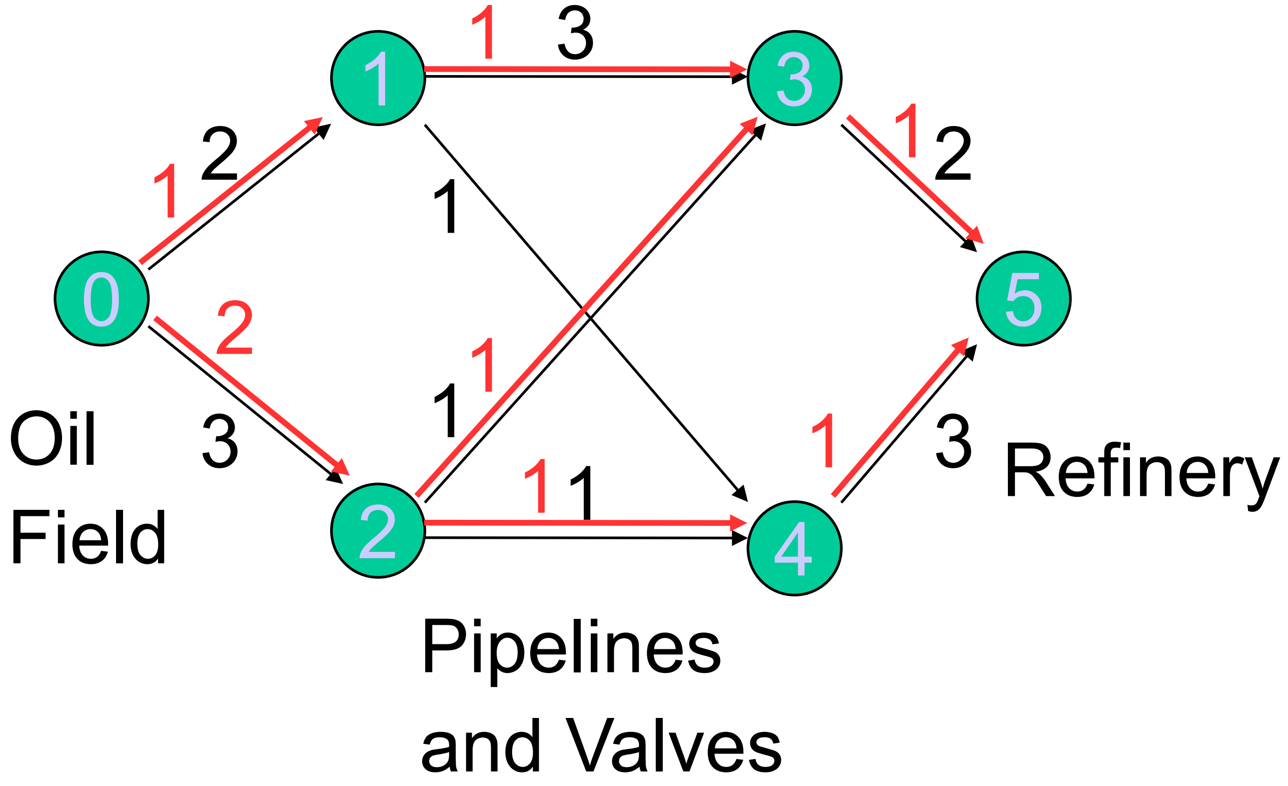
Flow Network



Maximize flow subject to capacity
and conservation of flow

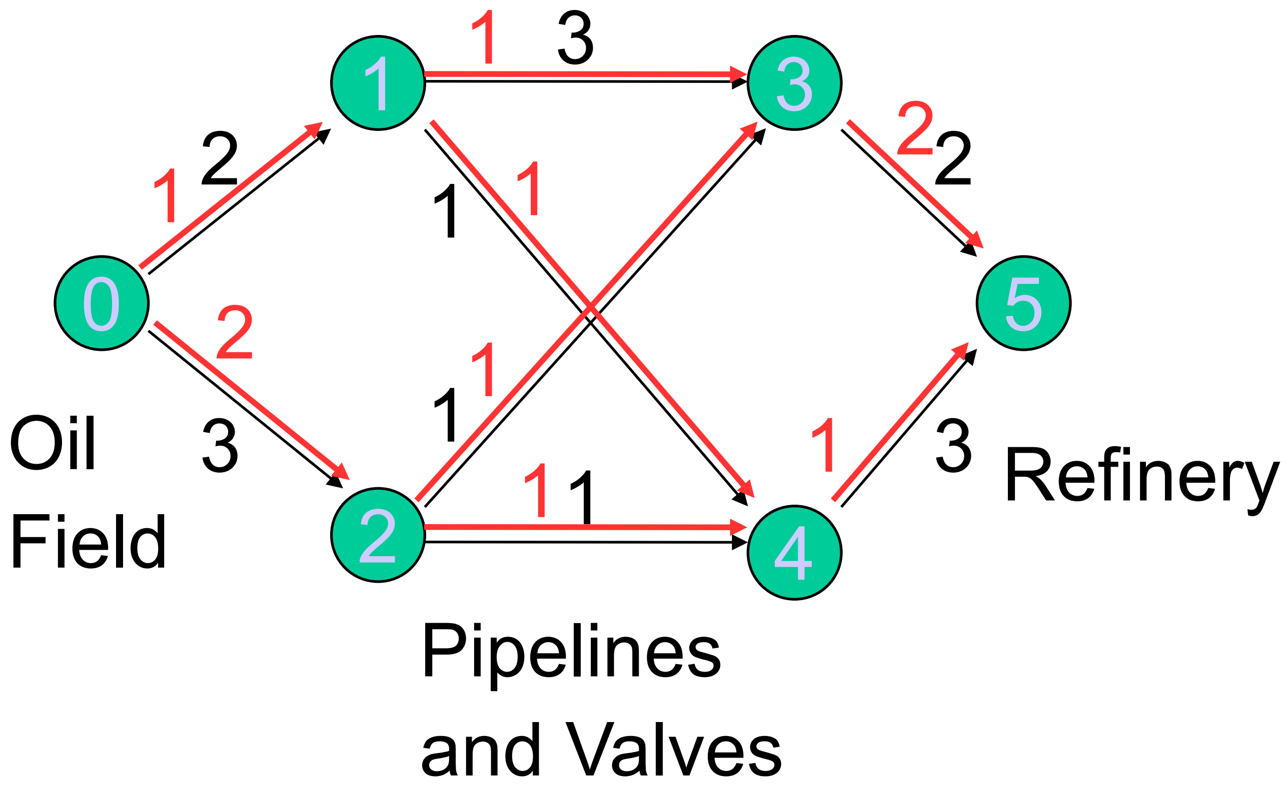
Is this OK?

Flow Network



Can we do better?

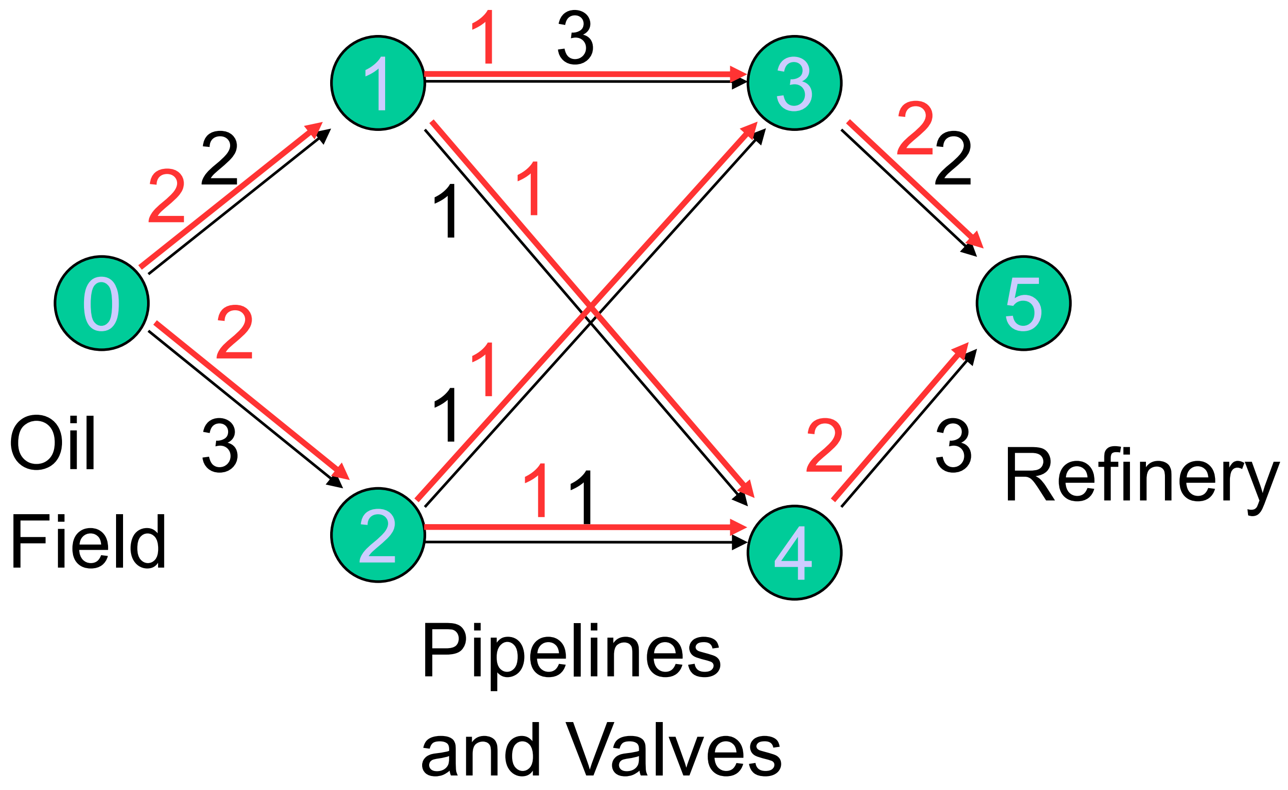
Flow Network



Are we done?

Is this OK?

Flow Network



Now are we done?

Yep

Flow Network

- Sum of flows into a node is called *inflow*
- Sum of flows out of a node is called *outflow*
- *Conservation of flow*: except for source and sink, inflow = outflow
- *Feasible* flow = obeys constraints (max flow and conservation of flow)

Flow Network

- Set outflow from sink to zero
- Set inflow to source to zero
- Outflow of source = inflow of sink
- This is called network's *value*

Maximum Flow

- Given an s-t network, find a flow such that no other flow from s to t has a larger value.
- A flow like this is called a *maxflow*.
- Problem of finding one is called the *maxflow problem*.