COMP9418: Advanced Topics in Statistical Machine Learning

Graph Decomposition

Instructor: Gustavo Batista

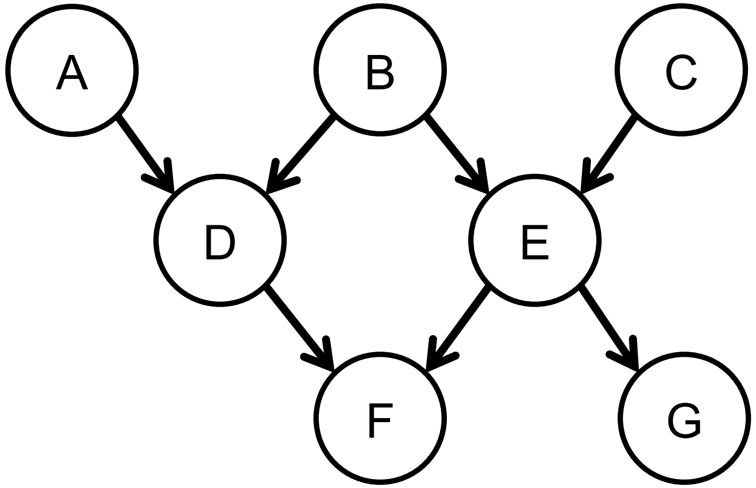
University of New South Wales

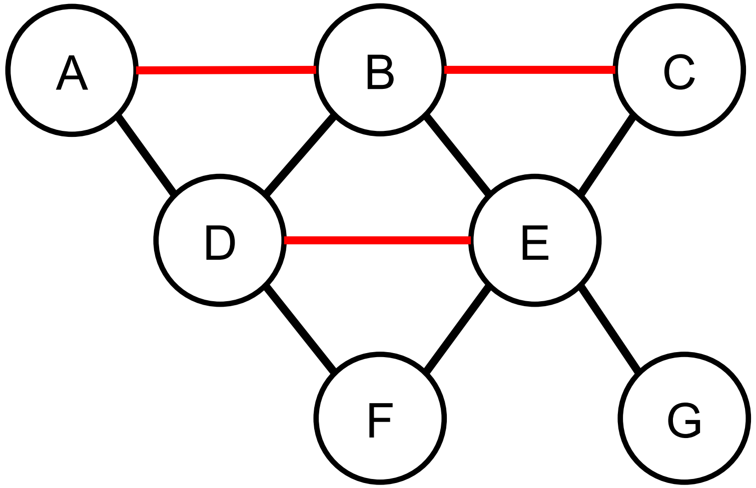
Introduction

- In this lecture, we will study the relationships between Variable Elimination and Jointrees
  - Both can be viewed as decomposing a network in a systematic way
  - VE removes one variable at a time while FE removes one factor at a time
- We will provide more formal definitions to concepts seen in previous lectures about these topics
- And cover polynomial time algorithms to convert elimination orders to jointrees and vice-versa

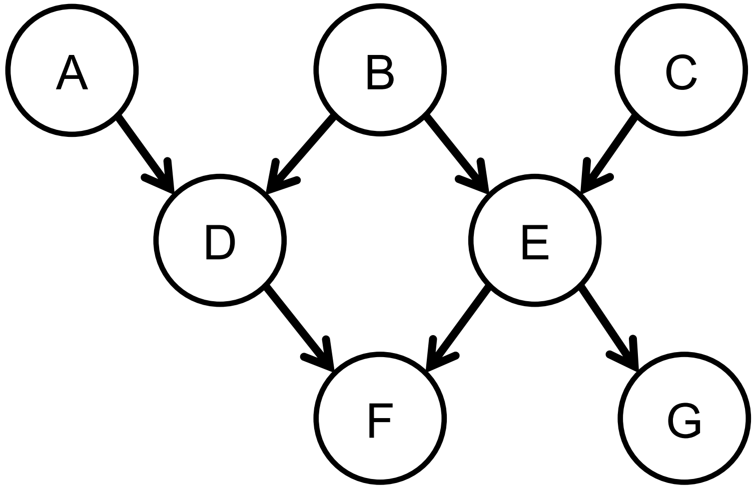
Moral Graph

- An *interaction graph* for factors $\phi_1, \dots, \phi_n$ is an undirected graph
  - Nodes correspond to the variables appearing in $\phi_1, \dots, \phi_n$
  - Edges connect variables in the same factor
- If the factors are CPTs of a Bayesian network with DAG G, the induced graph can be obtained from G by *moralization*
  - Add an undirected edge between every pair of nodes that share a common child
  - Convert every directed edge in G in an undirected edge



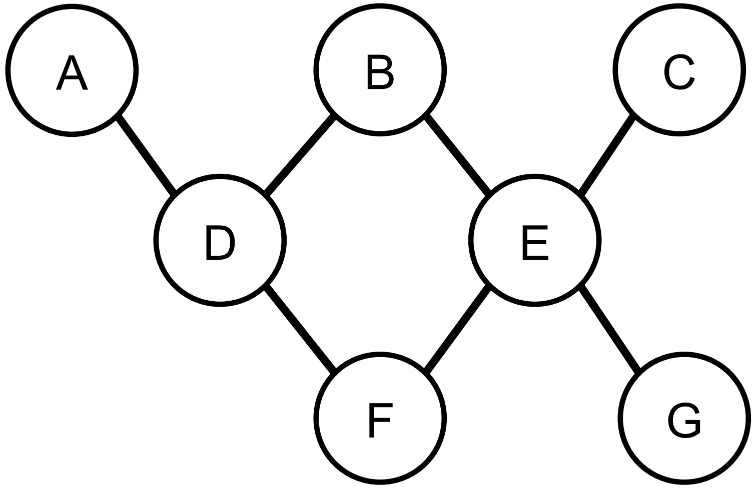


Treewidth



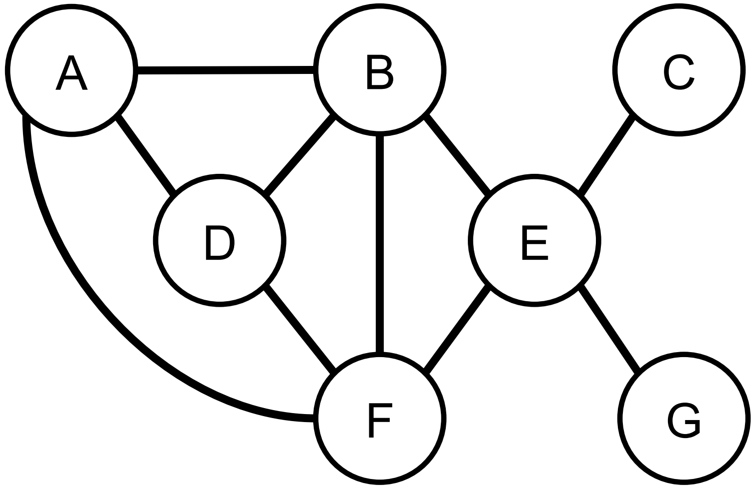
- Treewidth is a central notion to complexity analysis of inference algorithms
  - It is usually defined for undirected graphs
  - It can be extended to DAGs through their moral graphs

- The treewidth of a DAG G is defined as the treewidth of its moral graph
  - We will later define the treewidth for undirected graphs



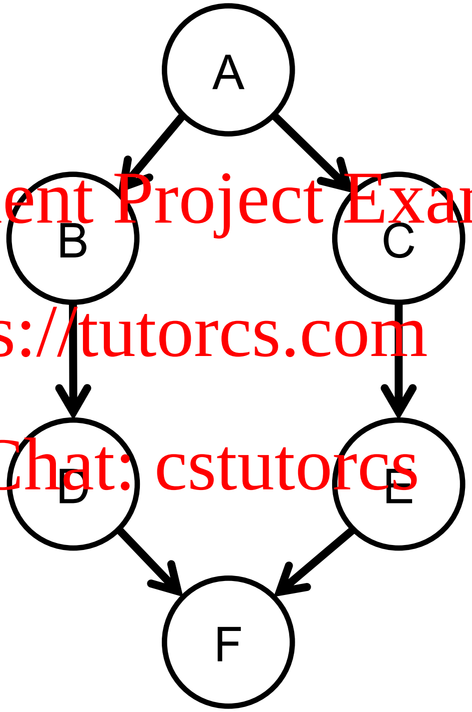
Other Graph-theoretic Definitions

- We adopt the following definitions for a graph G
  - A *neighbor* of a node X is a node Y connected to X by an edge in G. We also say X and Y are *adjacent*
  - The *degree* of a node X is the number of neighbors of X in G
  - A *clique* is a set of nodes in G that are pairwise adjacent, i.e., every pair of nodes in the clique are connected by an edge
  - A *maximal clique* is a clique that is not strictly contained in another clique
  - When we say "graph G" in this lecture, we mean undirected graph. Otherwise, we say "DAG G"

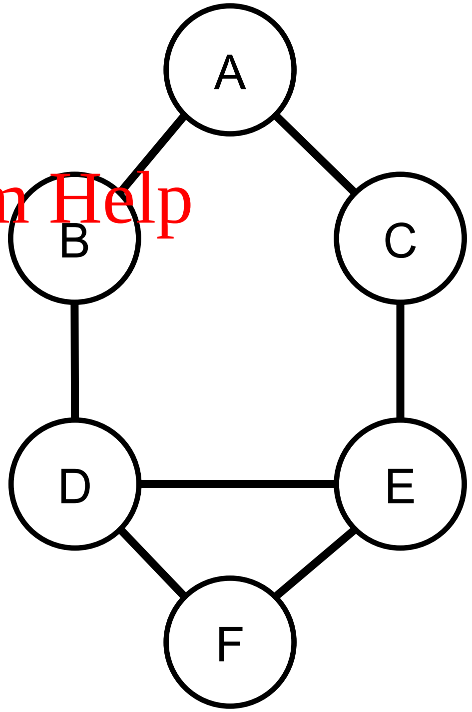


Elimination Order

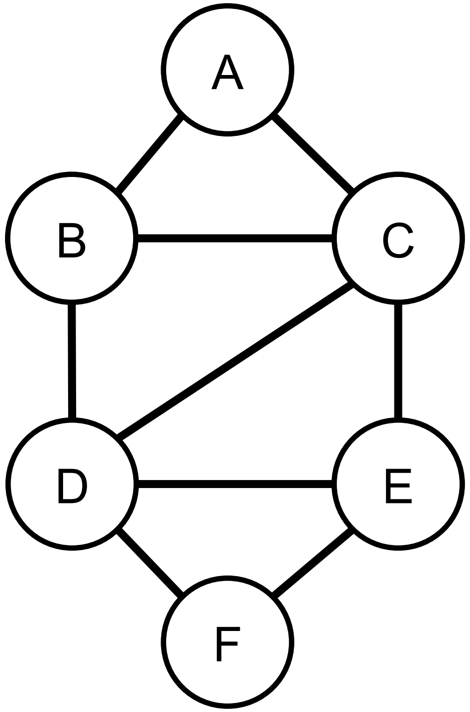
- An elimination order for a graph G is a total ordering π of nodes of G, where $\pi(i)$ is the ith node in the ordering
- The result of eliminating node X from graph G is another graph obtained from G by
  - Adding an edge between every pair of nonadjacent neighbors of X
  - Deleting node X
  - The edges added during the elimination process are called *fill-in edges*



A DAG G



The moral graph G_m

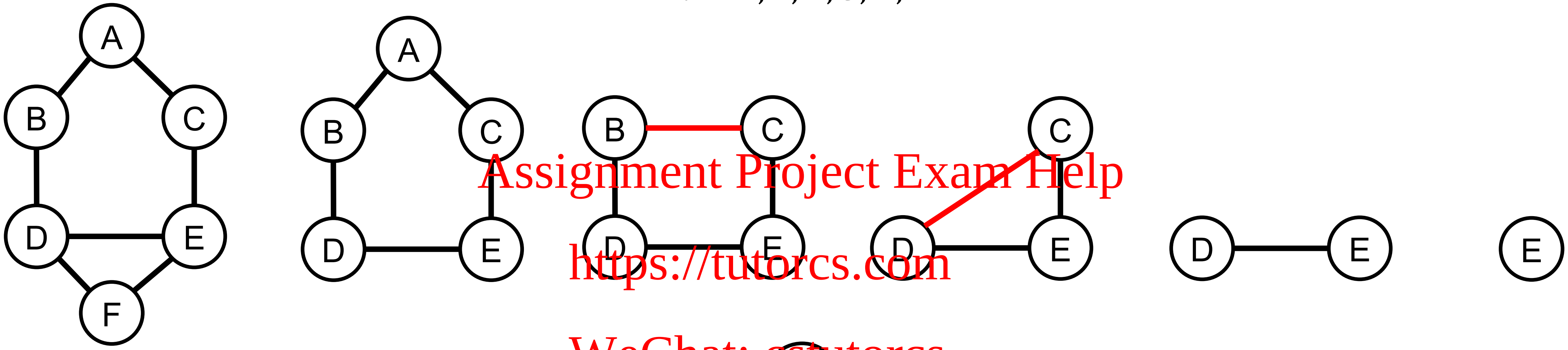


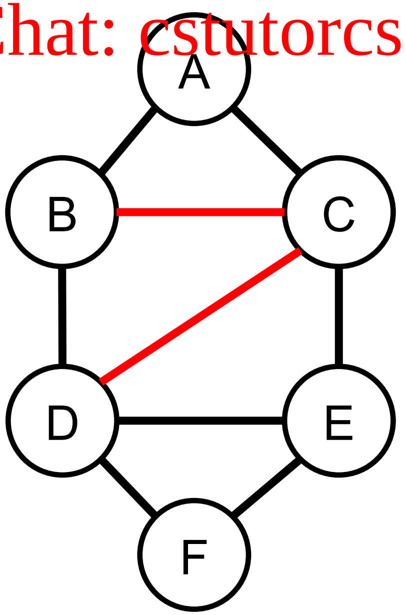
The filled-in graph G_m^π

$\pi = F, A, B, C, D, E$

Node Elimination

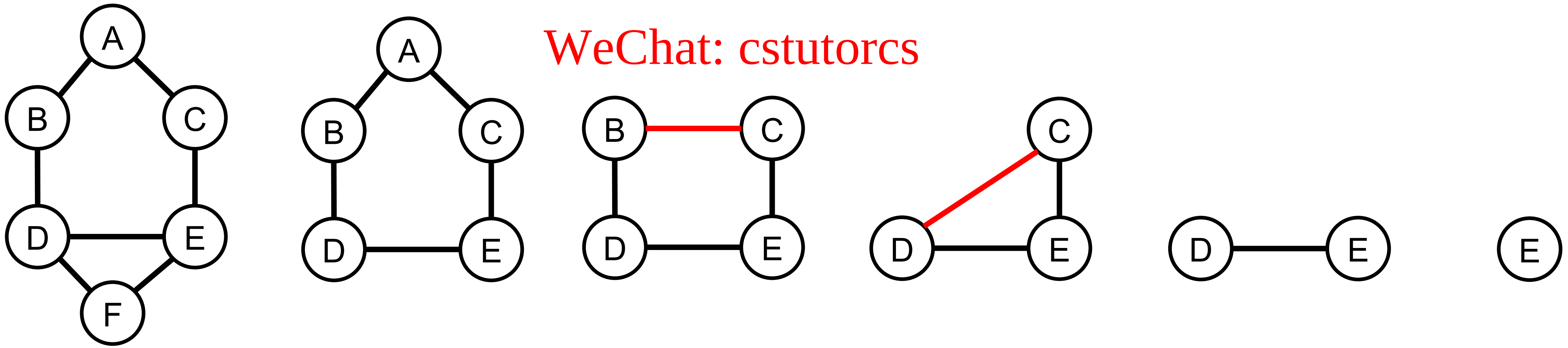
$$\pi = F, A, B, C, D, E$$





Graph and Cluster Sequence

- The elimination of nodes from graph G according to order π induces:
  - A graph sequence $G_1, \ldots, G_n$, where $G_1 = G$ a graph $G_{i+1}$is obtained by eliminating node $\pi(i)$ from graph G_i
  - A cluster sequence $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_n$, where $\boldsymbol{C}_i$ consists of node $\pi(i)$ and its neighbors in graph G_i

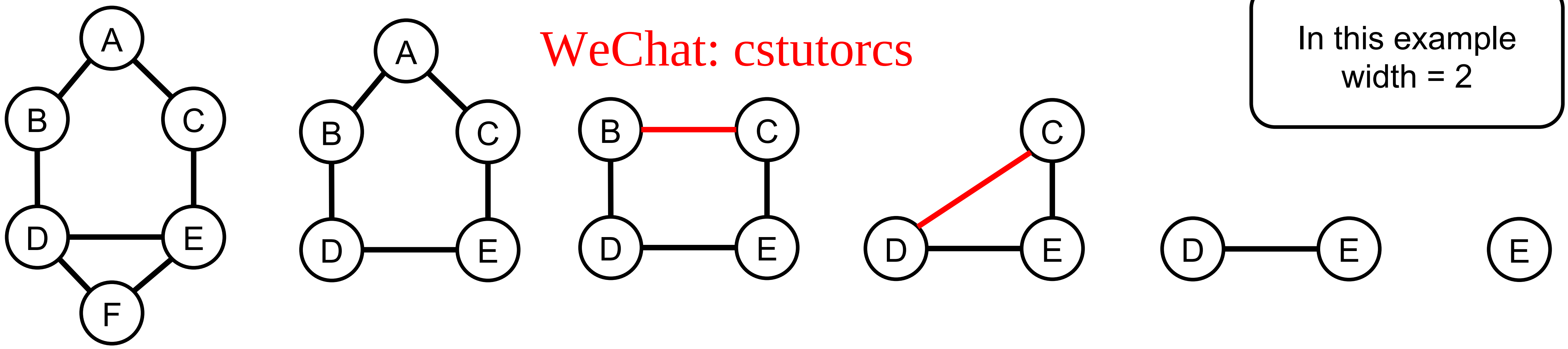


$\boldsymbol{C}_1 = DEF$ $\boldsymbol{C}_2 = ABC$ $\boldsymbol{C}_3 = BCD$ $\boldsymbol{C}_4 = CDE$ $\boldsymbol{C}_5 = DE$ $\boldsymbol{C}_6 = \mathrm{E}$

Elimination Width

- Let π be an elimination order for a graph G, and $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_n$ the induced cluster sequence
  - The *width* of the elimination order π is $width(\pi, G) \stackrel{\text{def}}{=} max_{i=1}^{n} |\boldsymbol{C}_i| - 1$
  - We extend this definition to DAGs: width of π for a DAG is the width of π for the DAG's moral graph

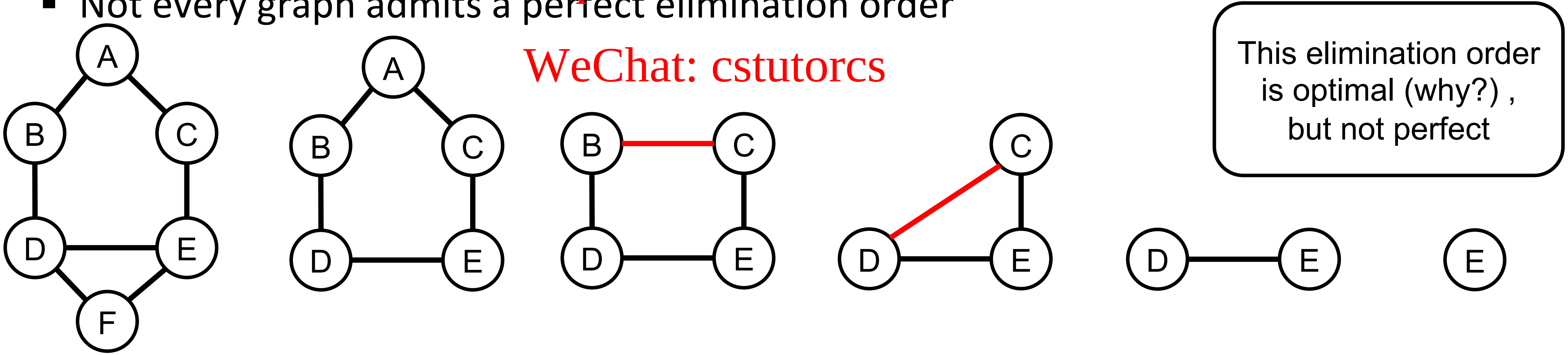
In this example width = 2



$\boldsymbol{C}_1 = DEF$ $\boldsymbol{C}_2 = ABC$ $\boldsymbol{C}_3 = BCD$ $\boldsymbol{C}_4 = CDE$ $\boldsymbol{C}_5 = DE$ $\boldsymbol{C}_6 = \mathrm{E}$

Treewidth

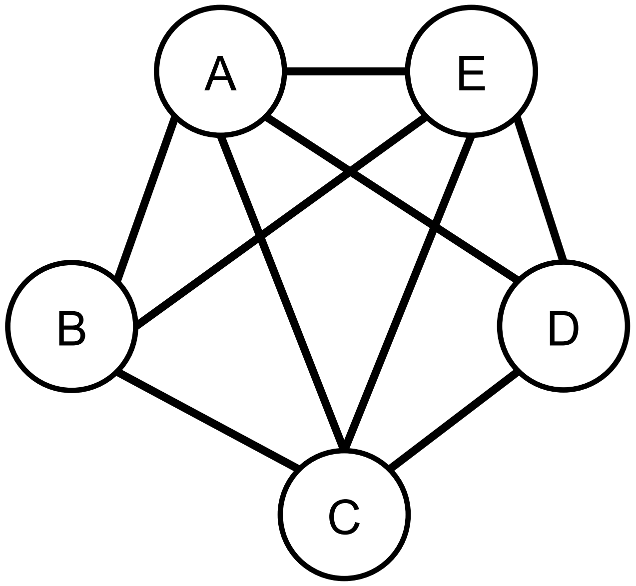
- The width of a graph G is $treewidth(G) \stackrel{\text{def}}{=} \min_{\pi} width(\pi, G)$
  - When the width of an elimination order π equals the treewidth, we say π is *optimal*
  - When an elimination order π does not lead to any fill-in edges, we say π is a *perfect elimination order*
  - Not every graph admits a perfect elimination order



$\boldsymbol{C}_1 = DEF$ $\boldsymbol{C}_2 = ABC$ $\boldsymbol{C}_3 = BCD$ $\boldsymbol{C}_4 = CDE$ $\boldsymbol{C}_5 = DE$ $\boldsymbol{C}_6 = \mathrm{E}$

Elimination Heuristics

- The computation of an optimal elimination order is NP-hard
  - Several greedy elimination heuristics have been proposed
  - These are ways to eliminate nodes based on local considerations
- The two most common heuristics are
  - Min-degree: eliminate the node having the smallest number of neighbors
  - Min-fill: Eliminate the node that leads adding the smallest number of fill-in edges
- Min-fill typically produces better results
- Min-degree is known to be optimal for graphs with treewidth ≤ 2

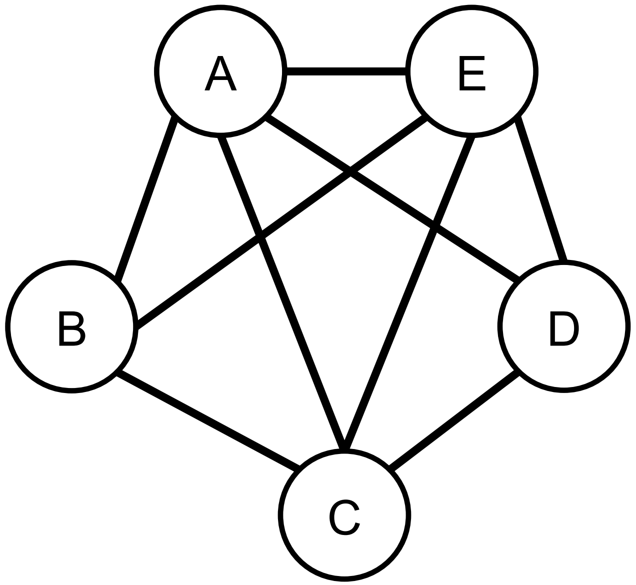


Node C has degree 4, and min-fill score of 1

Elimination Heuristics

- In practice, it is common to combine heuristics
  - Select nodes with min-fill and break ties with min-degree
  - Select nodes with min-degree and break ties using min-fill

- Stochastic techniques can be powerful to combine heuristics
  - Same elimination heuristic to eliminate every node but ties are broken stochastically
  - Different heuristics to eliminate different nodes, where the choice of a heuristic at each node is made stochastically



Node C has degree 4, and min-fill score of 1

Optimal Elimination Prefixes

- A *prefix* of elimination order π is a sequence of variables τ that occurs in the beginning of π
  - For example if $\pi = A, B, C, D, E$ then two prefixes are $\tau = A, B, C$ and $\tau = A, B$
- If τ is a prefix of some optimal elimination order, then τ is an *optimal elimination prefix*
  - It can be completed to yield an optimal order
  - The notion of width can be extended to prefixes by considering the cluster sequence of applying the prefix to a graph
- We can generate optimal prefixes
  - There are preprocessing rules that eliminate a subset of variables
  - While guaranteeing they represent the prefix of some optimal elimination order
  - These rules are not complete, i.e., we cannot always use them to produce a complete elimination order
  - Yet, we can use these rules to eliminate as many nodes as possible before using a heuristic

Optimal Elimination Prefixes

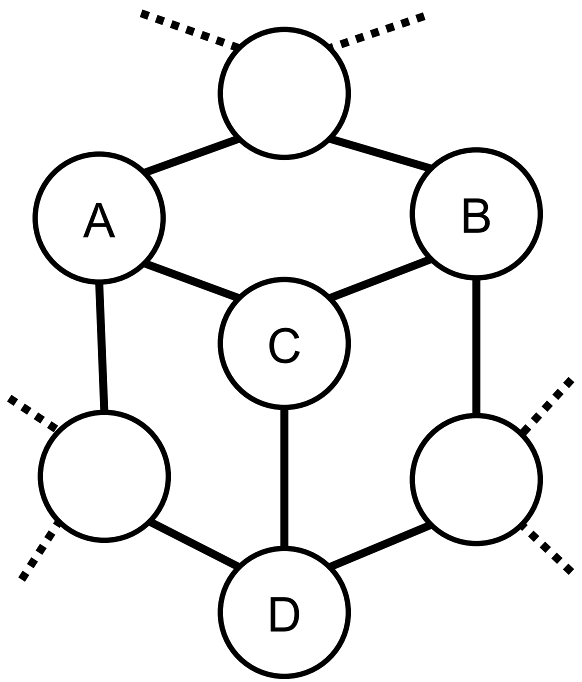
- We apply the rules in any order
  - A lower bound (low) is maintained on the treewidth
  - Some rules update this bound
  - While others use it as a condition for applying the rule

- If a graph G' is the result of applying any of these rules to G and if low is updated accordingly
  - $Treewidth(G) = \max(treewidth(G'), low)$
  - Therefore, these rules reduce the computation of treewidth of G into the computation of treewidth of a smaller graph G'
  - Moreover, they are guaranteed to generate only optimal elimination prefixes

- A *simplicial node* is a node that all its neighbors are pairwise adjacent, forming a clique
  - An *almost simplicial* node is one that all but one neighbor form a clique

Optimal Elimination Rules

- There are four rules
  - **Simplicial rule:** Eliminate any simplicial node with degree d, updating low to $\max(low, d)$
  - **Almost simplicial rule:** Eliminate any almost simplicial node with degree d as long as $low \geq d$
  - **Buddy rule:** If $low \geq 3$, eliminate any pair of nodes X and Y that have degree 3 each and share the same set of neighbors
  - **Cube rule:** If $low \geq 3$, eliminate any set of four nodes A, B, C, D forming the structure in the figure

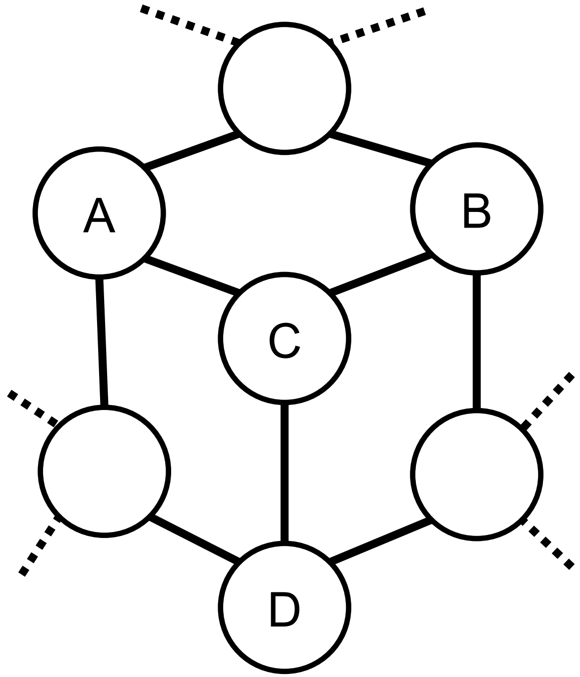


- These rules are complete for graphs of treewidth ≤ 3

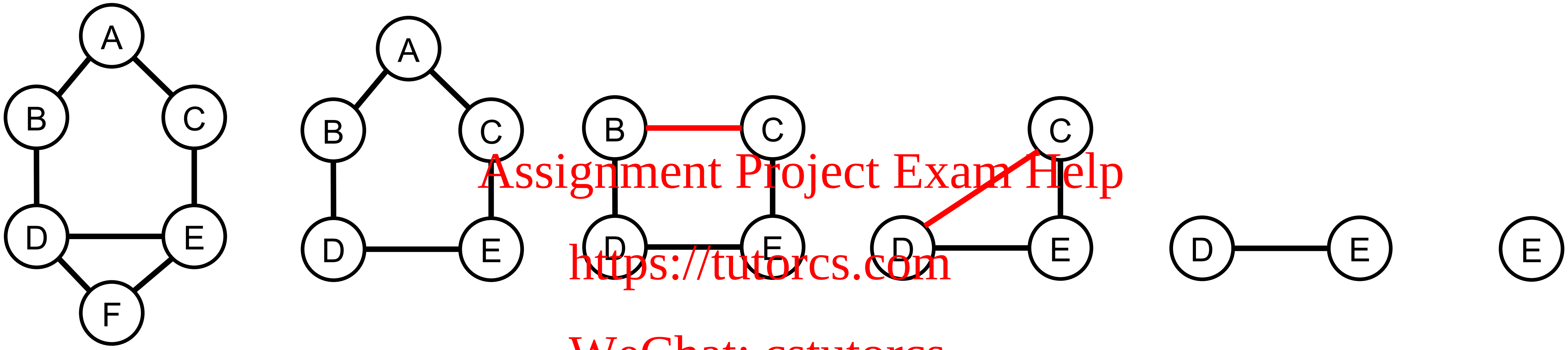
Optimal Elimination Rules

- Special cases for simplicial rule
  - **Isler rule:** Eliminate nodes with degree 0
  - **Twig rule:** Eliminate nodes with degree 1

- Special cases for almost simplicial rule
  - **Series rule:** eliminate nodes with degree 2 if $low \geq 2$
  - **Triangle rule:** eliminate nodes with degree 3 if $low \geq 3$ and if at least two of the neighbors are connected by an edge



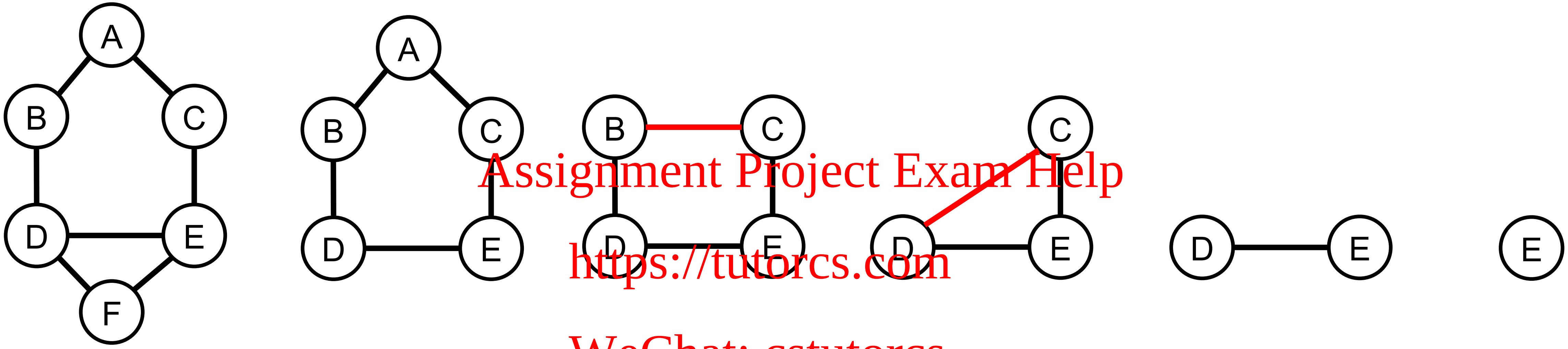
Optimal Elimination Prefixes: Example



$\tau =$

$low = 1$

Optimal Elimination Prefixes: Example

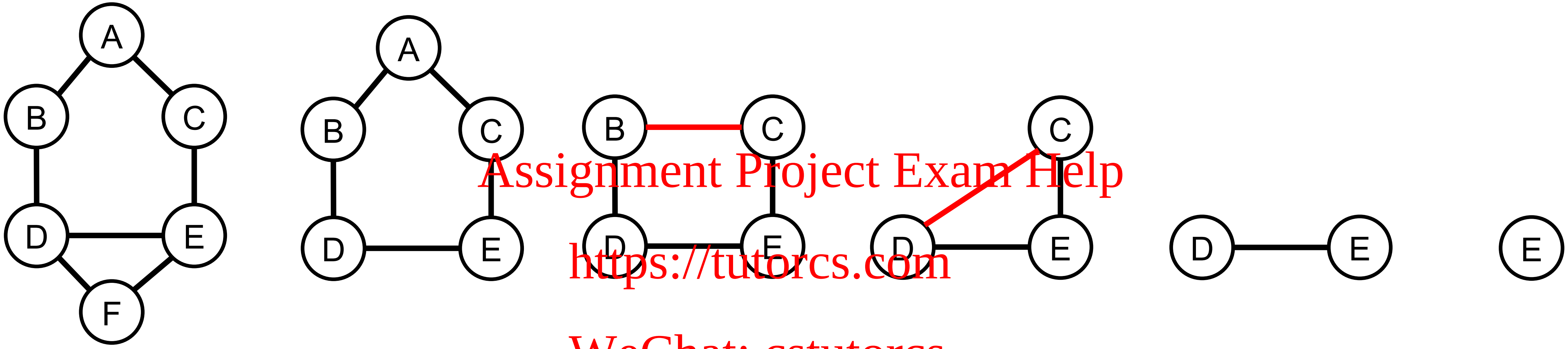


Simplicial rule

$\tau = F$

$low = 2$

Optimal Elimination Prefixes: Example

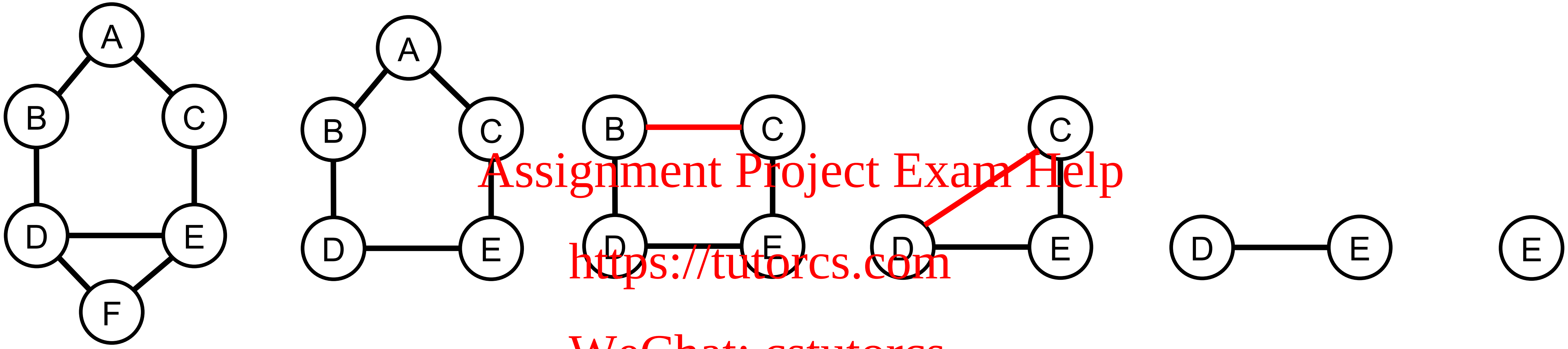


Simplicial rule **Series rule**

$\tau = F, A$

$low = 2$

Optimal Elimination Prefixes: Example

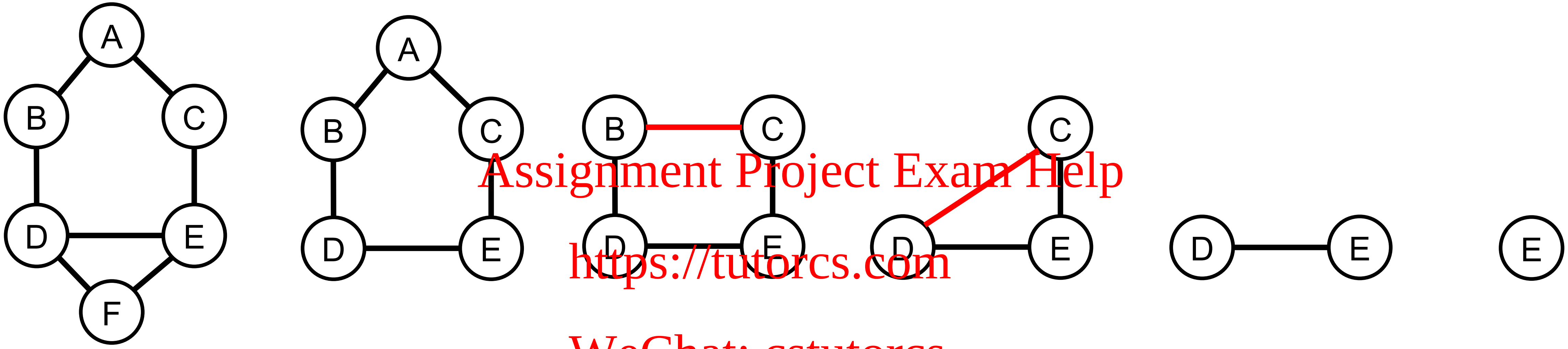


Simplicial rule **Series rule** **Series rule**

$\tau = F, A, B$

$low = 2$

Optimal Elimination Prefixes: Example

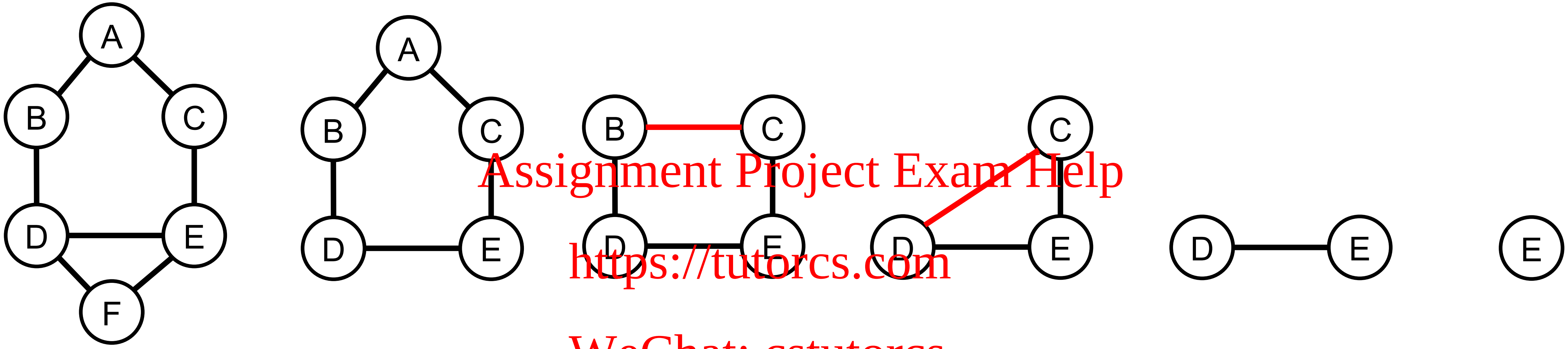


Simplicial rule **Series rule** **Series rule** **Series rule**

$\tau \ = F, A, B, C$

$low \ = 2$

Optimal Elimination Prefixes: Example

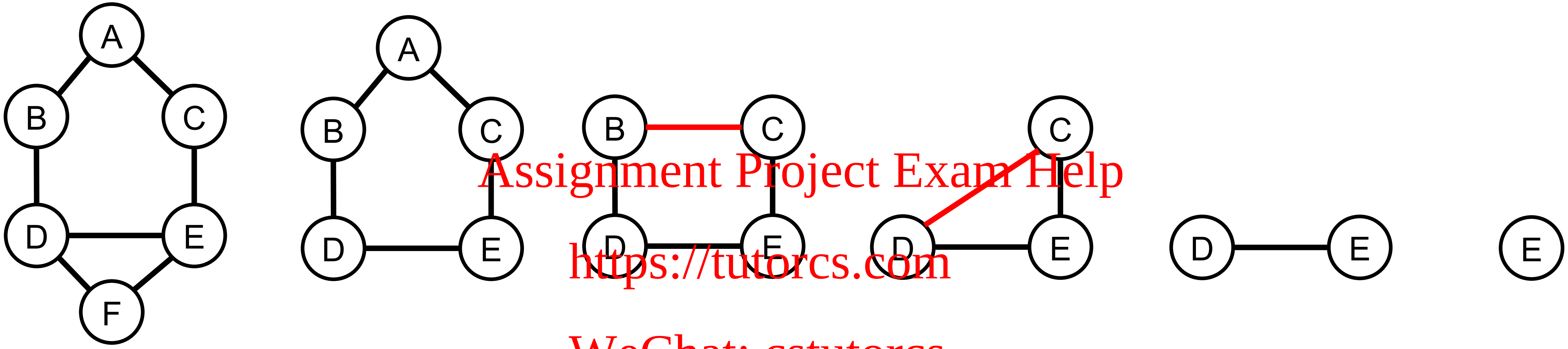


Simplicial rule   Series rule   Series rule   Series rule   Twig rule

$\tau = F, A, B, C, D$

$low = 2$

Optimal Elimination Prefixes: Example



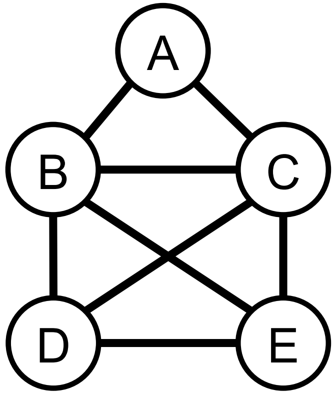
Simplicial rule **Series rule** **Series rule** **Series rule** **Twig rule** **Isler rule**

$\tau = F, A, B, C, D, E$

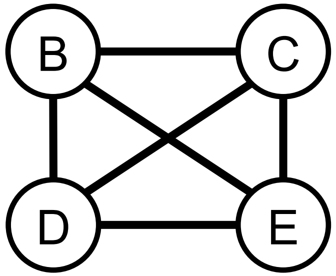
$low = 2$

Lower Bounds on Treewidth

- Lower bounds are used to empower the pre-processing rules
  - We can use then to set low to a larger initial value
  - This may allow us to apply more pre-processing elimination rules
- Two well-know, but typically weak lower bounds are
  - If a graph G has a clique of size n, then $treewidth(G) \geq n - 1$
  - The *degree of a graph* is a lower bound for treewidth. The degree of a graph is the minimum number of neighbours attained by any of its nodes
- The *degeneracy* of a graph is the maximum degree attained by any of its subgraphs
  - It is a better bound. It is based on two observations
  - First, the treewidth of any subgraph cannot be larger than the treewidth of the graph containing it
  - Second, the degree of a subgraph may be higher than the degree of the graph containing it



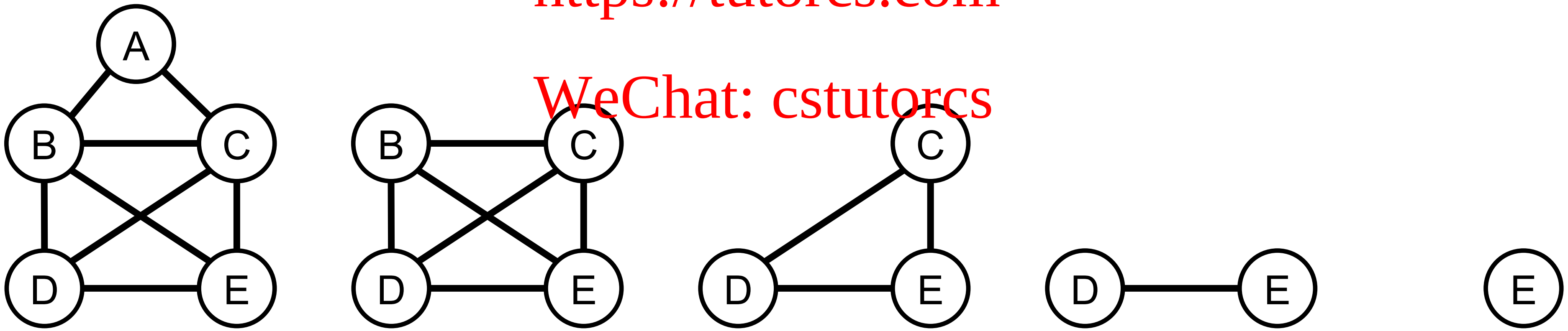
Graph degree is 2



Graph degree is 3

Lower Bounds on Treewidth

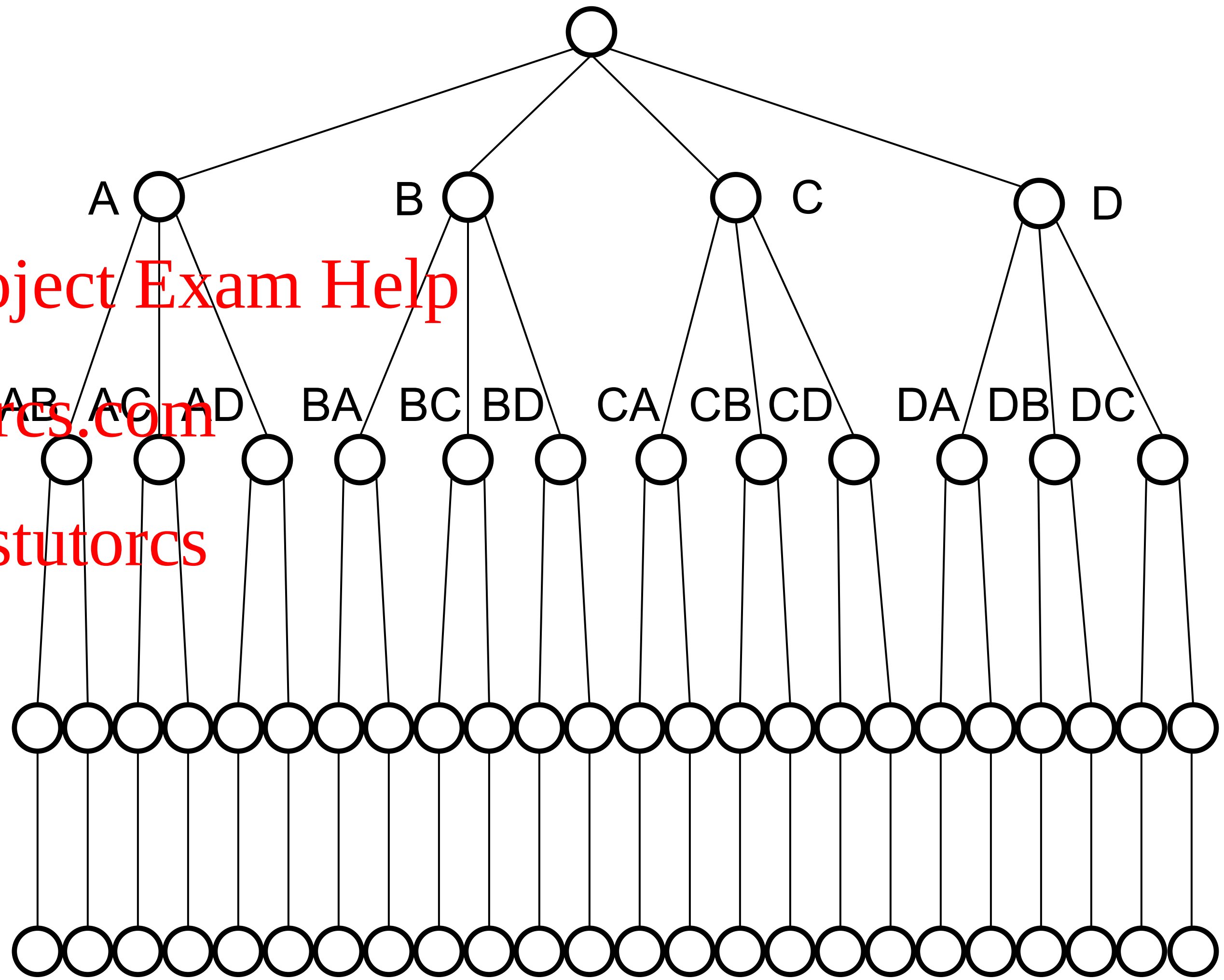
- The degeneracy is also known as maximum minimum degree (MMD)
  - MMD is easily computed by generating a sequence of subgraphs
  - Start with the original graph and remove a minimum-degree node
  - The MMD is the maximum degree attained by any of the generated subgraphs



Degree = 2 Degree = 3 Degree = 2 Degree = 1 Degree = 0

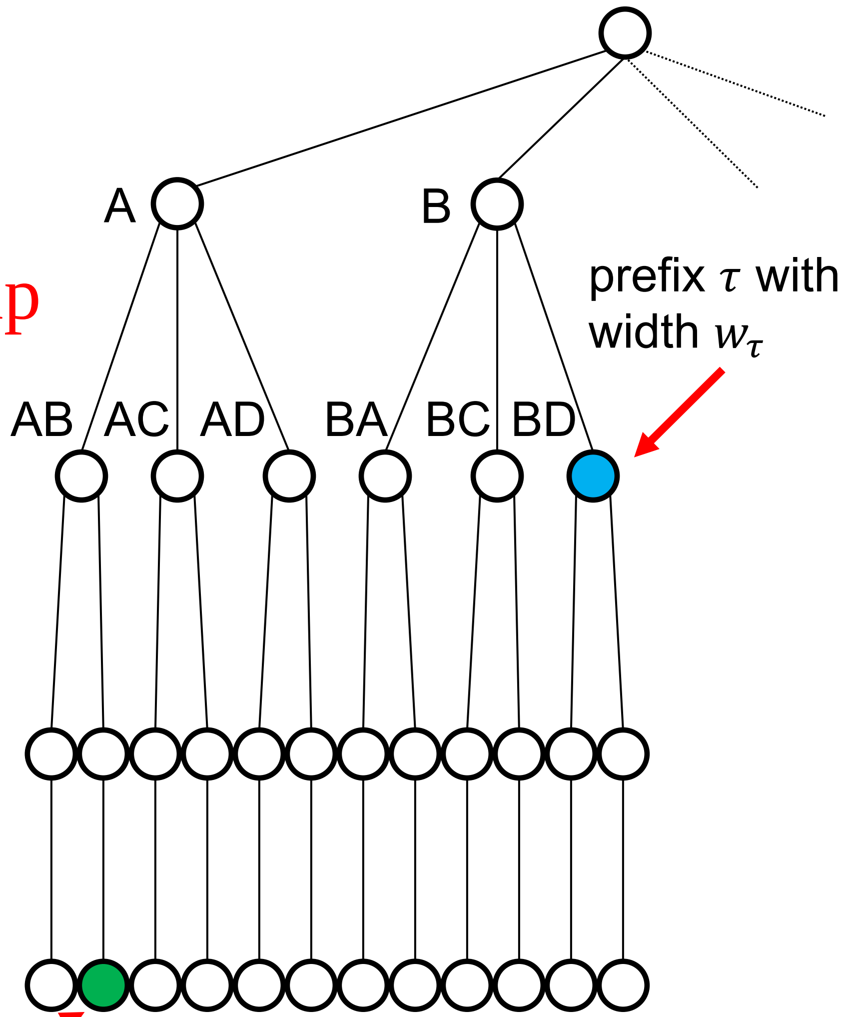
Optimal Elimination Order

- We can use search methods to find an optimal elimination order
  - For instance, depth-first search to generate all possible elimination orders
  - The search space has size $O(n!)$, where n is the number of variables
  - Therefore, such methods are limited to small number of variables
  - However, we can explore optimal elimination prefixes and lower bound to prune the search space



Optimal Elimination Order: Pruning

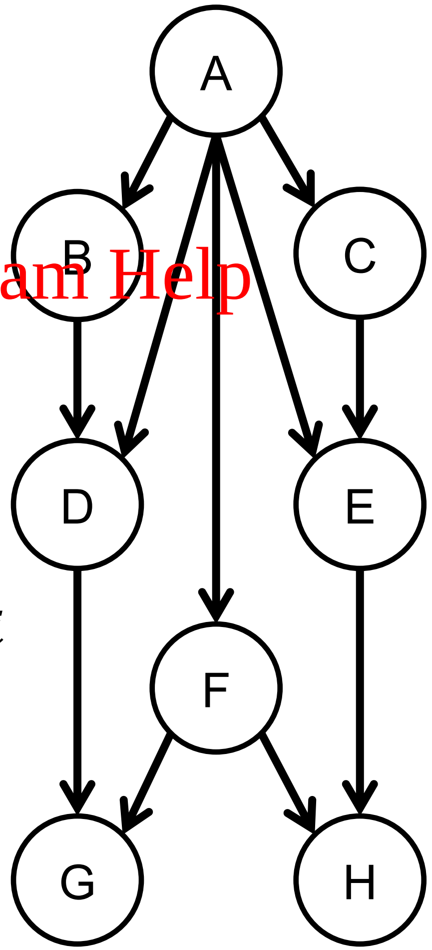
- Pruning is essential to improve the efficiency of these search methods
  - If you have a query, use the query pruning techniques of the lecture 7
  - Use optimal prefix rules to further reduce the graph size and provide an elimination prefix and lower bound
- In a depth-first search, each time we reach a leaf node in the search tree, we obtain an elimination order
- Suppose we have an elimination order π with width w_π
  - We are currently exploring a prefix τ with width w_τ
  - We also have a lower bound b on graph G_τ
  - If $\max(w_\tau, b) \geq w_\pi$, then we cannot improve on order π

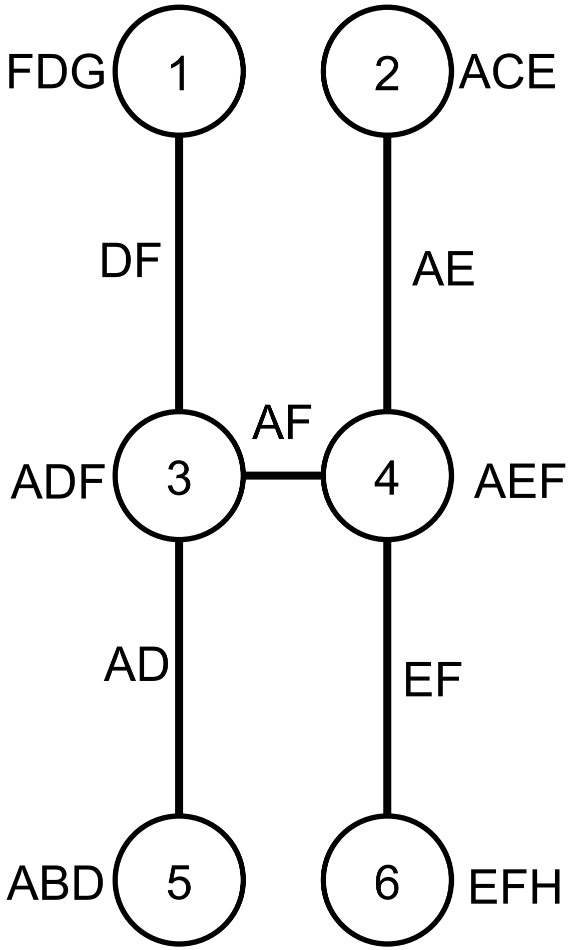


Elimination order π with width w_π
This is our best-so-far width

Jointree: Recap

- A *jointree* for a network G is a pair $(T, \boldsymbol{C})$ where T is a tree and $\boldsymbol{C}$ is a function that maps each node i in the tree T into a label $\boldsymbol{C}_i$, called *cluster*. The jointree must satisfy the following properties
  - The cluster $\boldsymbol{C}_i$ is a set of nodes from G
  - Each factor in G must appear in some cluster $\boldsymbol{C}_i$
  - If a variable appears in two clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$, it must appear in every cluster $\boldsymbol{C}_k$ on the path connecting nodes i and j in the jointree. This is known as *jointree* or *running intersection* property



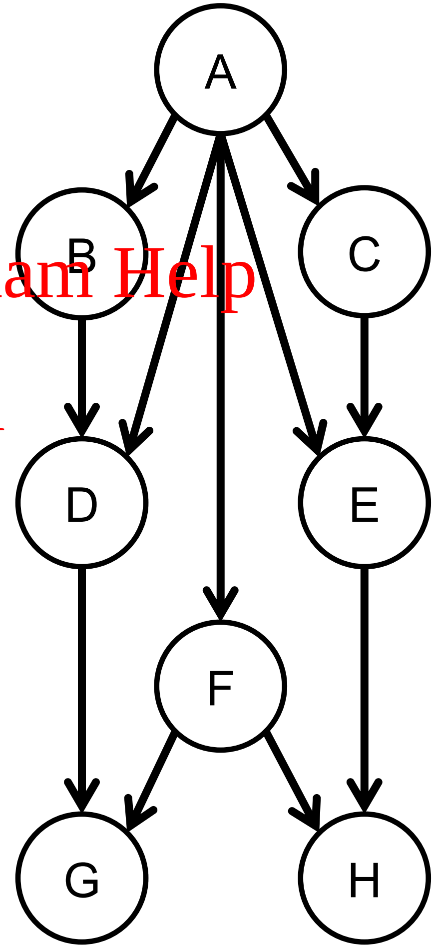


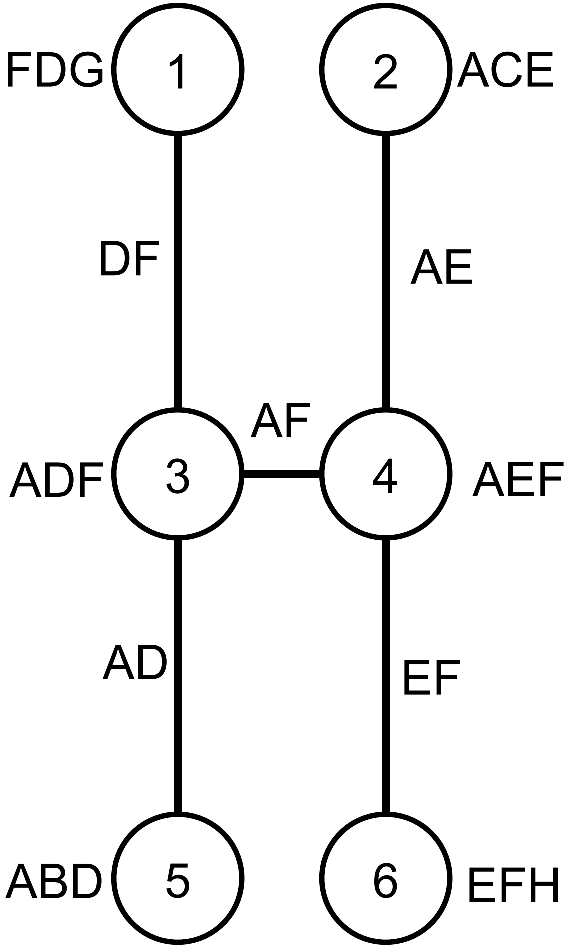
Jointree: Recap

- The *separator* of edge $i - j$ in a jointree is a set of variables defined as follows
$$\boldsymbol{S}_{ij} = vars(i,j) \cap vars(j,i)$$

- The *width* of a jointree is defined as the size of its largest cluster minus one

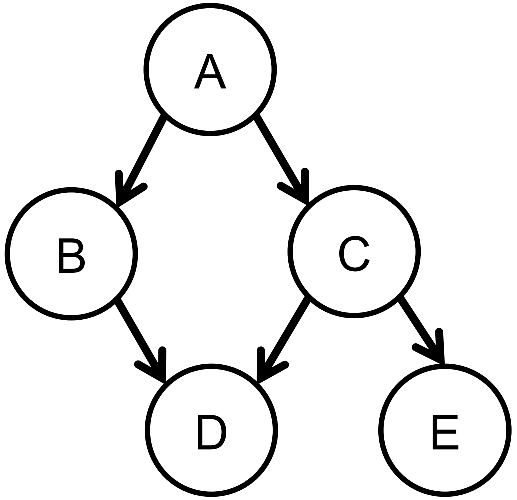
- A jointree is also called a *junction tree* or a *cluster tree*

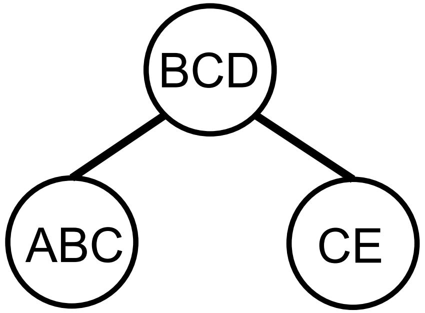


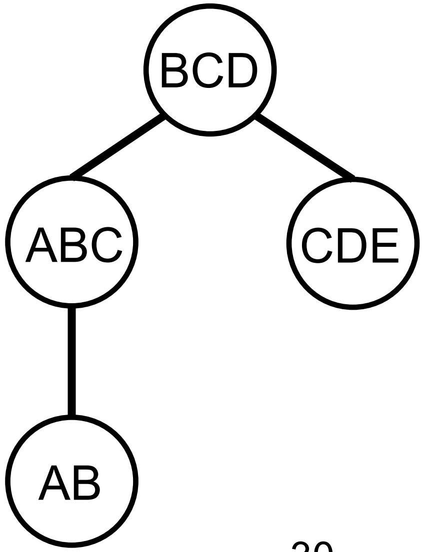


Jointree Operations

- The following transformations preserve all three properties of a jointree
  - **Add variable:** We can add a variable X to a cluster $\boldsymbol{C}_i$ if $\boldsymbol{C}_i$ has a neighbour $\boldsymbol{C}_j$ that contains X
  - **Merge clusters**: We can merge two neighbouring clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ into a single cluster $\boldsymbol{C}_k = \boldsymbol{C}_i \cup \boldsymbol{C}_j$, where $\boldsymbol{C}_k$ inherits the neighbours of $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$
  - **Add cluster:** We can add a new cluster $\boldsymbol{C}_j$ and make it a neighbour of an existing cluster $\boldsymbol{C}_i$ if $\boldsymbol{C}_j \subseteq \boldsymbol{C}_i$
  - **Remove cluster:** We can remove cluster $\boldsymbol{C}_j$ if it has a single neighbour $\boldsymbol{C}_i$ and $\boldsymbol{C}_j \subseteq \boldsymbol{C}_i$

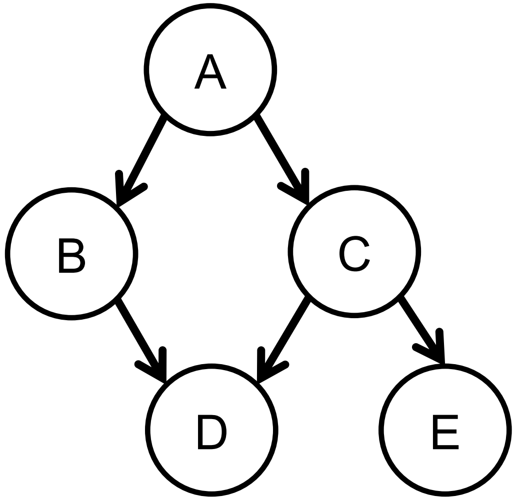


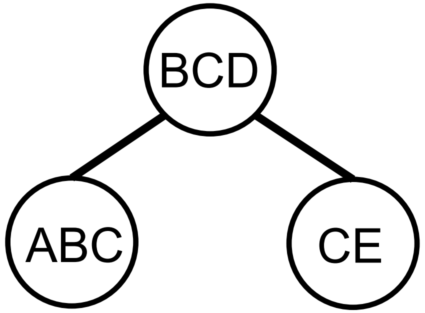


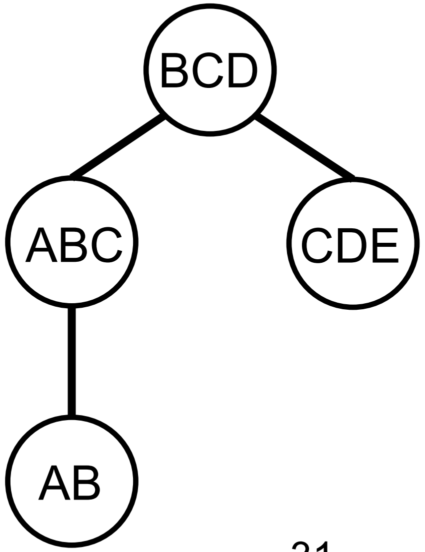


Jointree Operations

- These transformations have practical applications
  - The addition of variable D allows the jointree to compute marginal over variables CDE
  - This marginal will not be computed if the algorithm is applied to the left jointree
  - Merging two clusters eliminates the separator connecting them
  - As the Shenoy-Shafer algorithm creates factors over each separator, this transformation can be used to reduce space requirements
  - This will also typically increase the running time, as merging clusters can lead to larger clusters and the algorithm is exponential to cluster size
  - This transformation can be the basis for time-space trade-offs







Jointree to Elimination Orders

$\pi \leftarrow []$

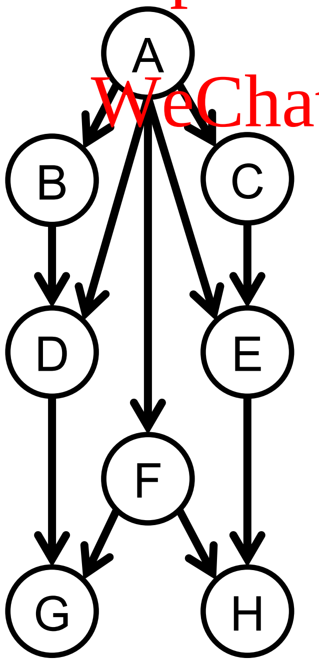
while not empty(T) **do**

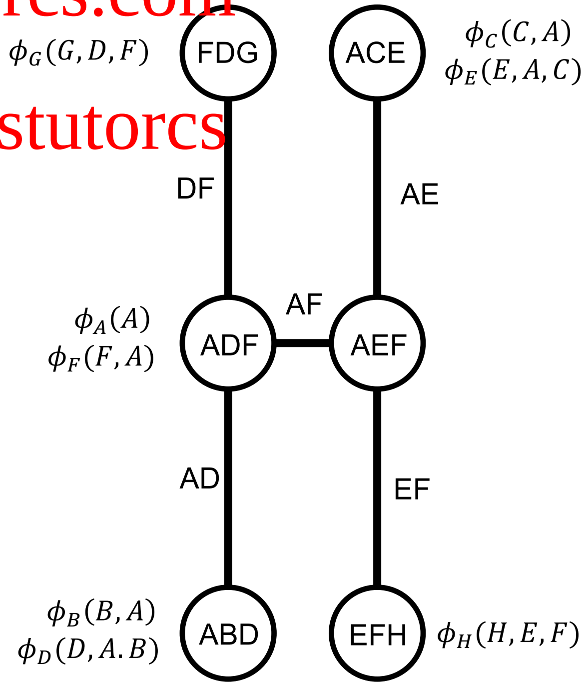
remove node i from T that has a single neighbour j

append variables $\boldsymbol{C}_i \backslash \boldsymbol{C}_i \cap \boldsymbol{C}_j$ to π

append variables $\boldsymbol{C}_r$ to π, where r is the remaining node in T

return π





Jointree to Elimination Orders

$\pi \leftarrow []$

while not empty(T) **do**

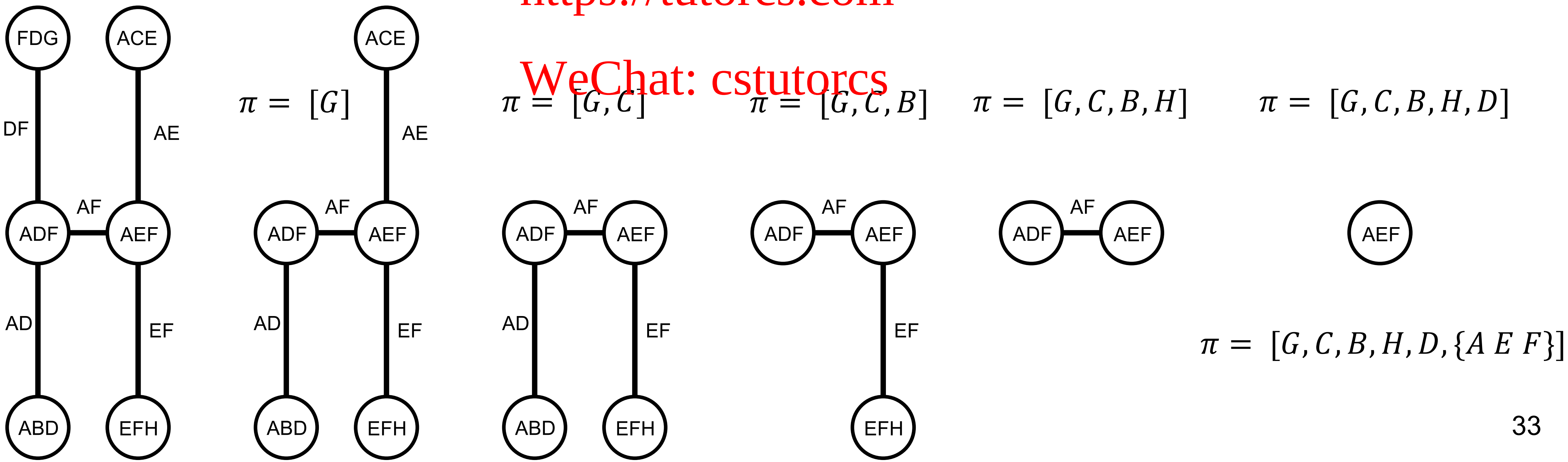
  remove node i from T that has a single neighbour j

  append variables $\boldsymbol{C}_i \backslash \boldsymbol{C}_i \cap \boldsymbol{C}_j$ to π

append variables $\boldsymbol{C}_r$ to π, where r is the remaining node in T

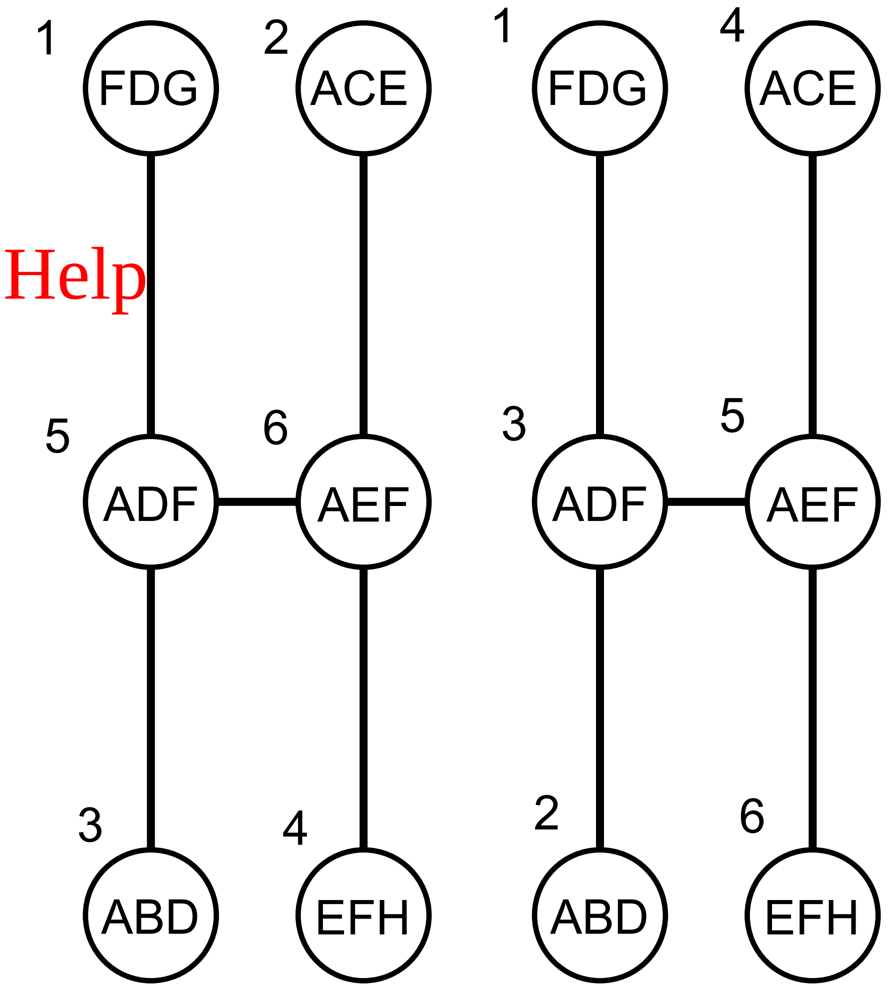
return π

$\pi = []$



Jointree to Elimination Orders

- This algorithm simulates the process of factor elimination
  - It runs in polynomial time
  - It provides an elimination order of no greater width than the jointree width
- The algorithm leaves a few choices undetermined
  - Choosing the node i to remove next from the jointree
  - Choosing the order in which variables $\mathbf{C}_i \backslash \mathbf{C}_i \cap \mathbf{C}_j$ are appended to π
- None of these choices matter to the guarantees provided by the algorithm



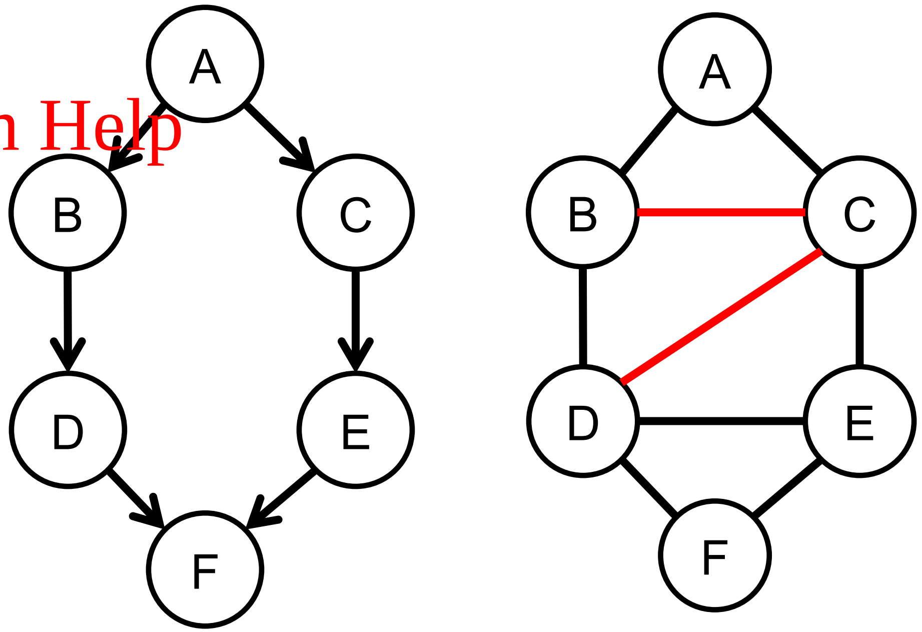
Elimination Orders to Jointrees

- We present a polynomial time, width-preserving algorithm for generating a jointree from an elimination order

- The algorithm consists of two parts
  - Constructing the clusters
  - Assembling the jointree

Elimination Orders to Jointrees: Clusters

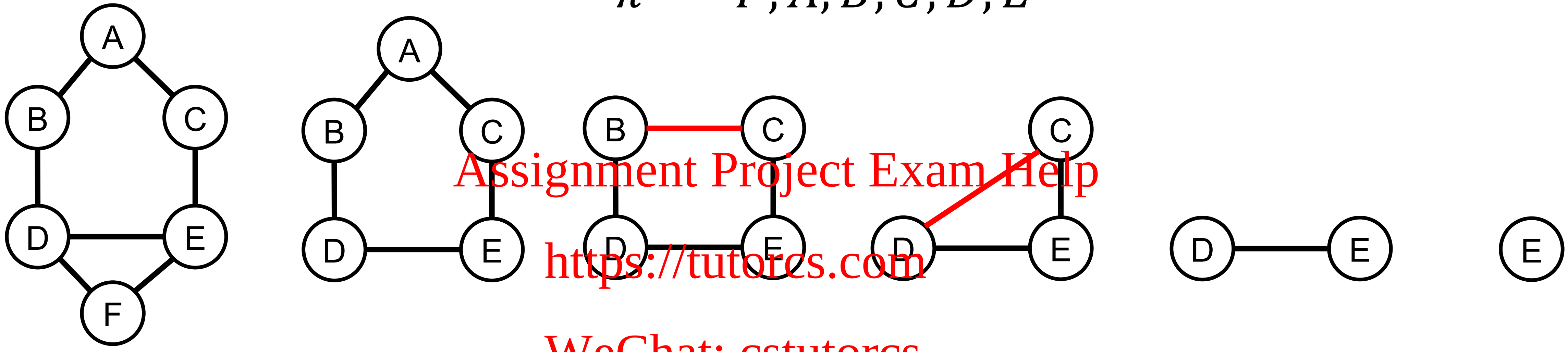
- We show how to generate clusters with the following properties
  - The size of every cluster is $\leq width(\pi, G) + 1$
  - The clusters satisfy conditions 1 and 2 of the jointree definition



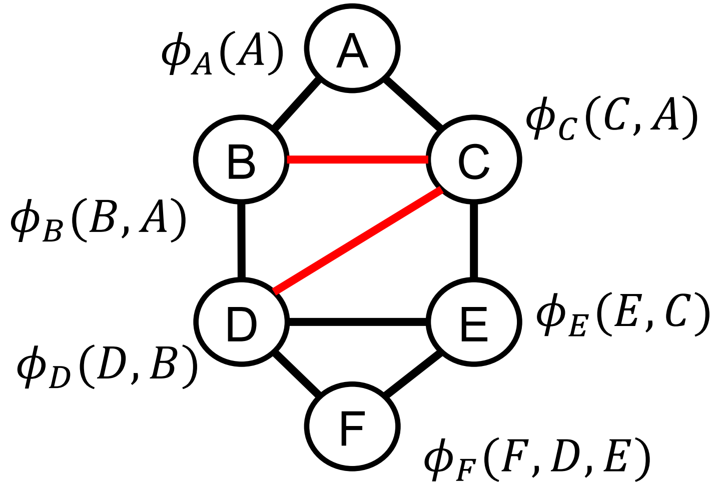
- Let $\boldsymbol{C}_i, \ldots, \boldsymbol{C}_n$ be the cluster sequence that results from applying the elimination order π to the undirected graph (or moral graph of a DAG) G. Every family of G is contained in some cluster in the sequence
  - $\boldsymbol{C}_i, \ldots, \boldsymbol{C}_n$ satisfy the first two conditions of the jointree
  - The size of each cluster is $\leq width(\pi, G) + 1$

Elimination Orders to Jointrees: Clusters

$$\pi = F, A, B, C, D, E$$

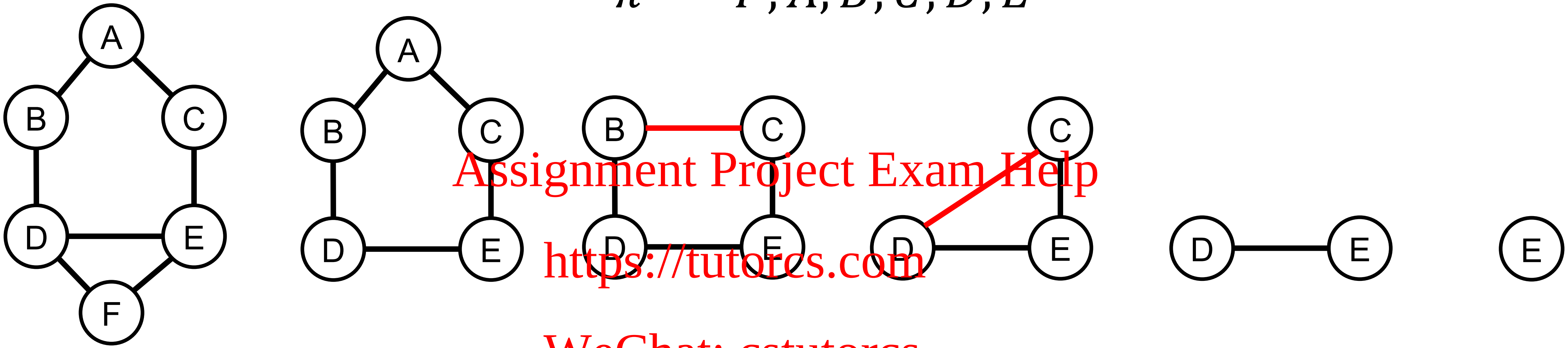


$\boldsymbol{C}_1 = DEF$ $\boldsymbol{C}_2 = ABC$ $\boldsymbol{C}_3 = BCD$ $\boldsymbol{C}_4 = CDE$ $\boldsymbol{C}_5 = DE$ $\boldsymbol{C}_6 = \mathrm{E}$



Elimination Orders to Jointrees: Clusters

$\pi = F, A, B, C, D, E$



$\boldsymbol{C}_1 = DEF$ ← $\phi_F(F, D, E)$

$\boldsymbol{C}_2 = ABC$ ← $\phi_A(A)$, $\phi_B(B, A)$, $\phi_C(C, A)$

$\boldsymbol{C}_3 = BCD$ ← $\phi_D(D, B)$

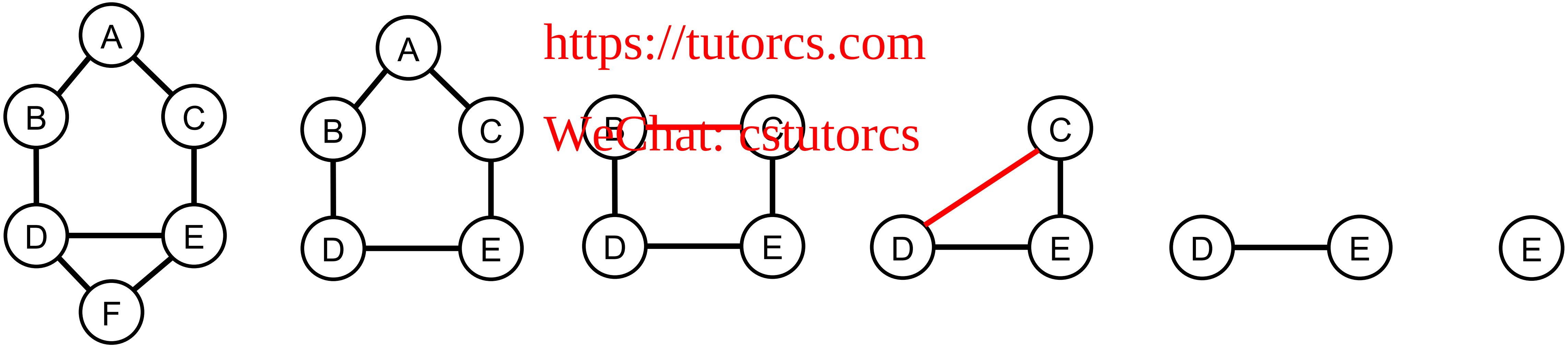
$\boldsymbol{C}_4 = CDE$ ← $\phi_E(E, C)$

$\boldsymbol{C}_5 = DE$

$\boldsymbol{C}_6 = \mathrm{E}$

Running Intersection Property

- Let $\boldsymbol{C}_i,\ldots,\boldsymbol{C}_n$ be the cluster sequence induced by applying elimination order π to graph G.
  - For every $i < n$, the variables $\boldsymbol{C}_i \cap (\boldsymbol{C}_{i+1} \cup \cdots \cup \boldsymbol{C}_n)$ are contained in some cluster $\boldsymbol{C}_j$ where $j > i$
  - This is known as the *running intersection property of the cluster sequence*

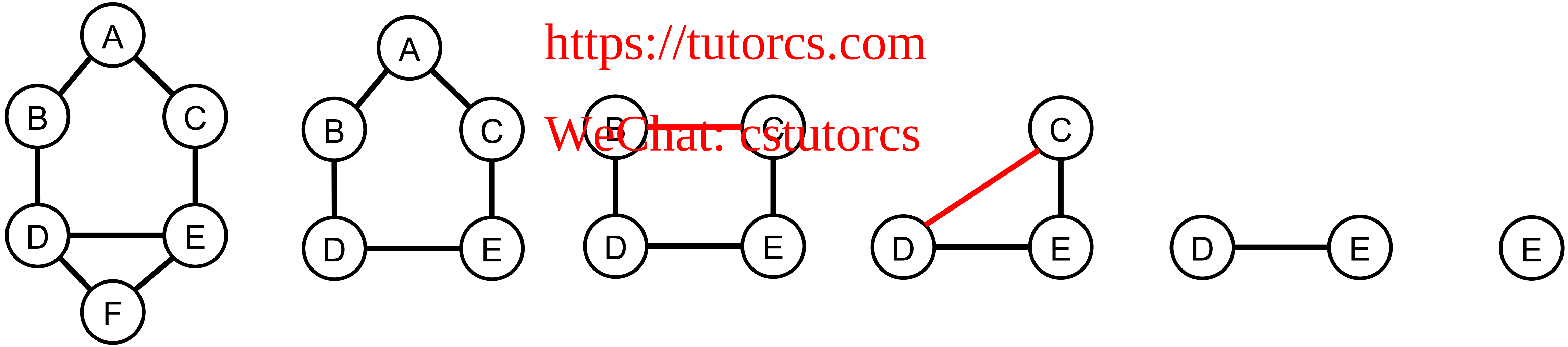


$\boldsymbol{C}_1 = DEF$ $\boldsymbol{C}_2 = ABC$ $\boldsymbol{C}_3 = BCD$ $\boldsymbol{C}_4 = CDE$ $\boldsymbol{C}_5 = DE$ $\boldsymbol{C}_6 = \mathrm{E}$

$\boldsymbol{C}_1 \cap (\boldsymbol{C}_2 \cup \cdots \cup \boldsymbol{C}_6) = DE$

Running Intersection Property

- Let $\boldsymbol{C}_i$,…, $\boldsymbol{C}_n$ be the cluster sequence induced by applying elimination order π to graph G.
  - For every $i < n$, the variables $\boldsymbol{C}_i \cap (\boldsymbol{C}_{i+1} \cup \cdots \cup \boldsymbol{C}_n)$ are contained in some cluster $\boldsymbol{C}_j$ where $j > i$
  - This is known as the *running intersection property of the cluster sequence*

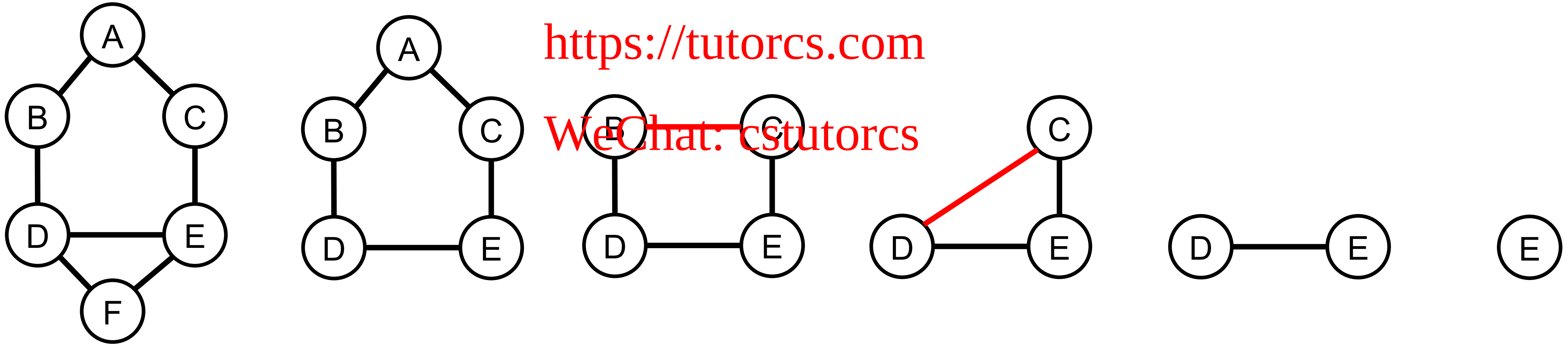


$\boldsymbol{C}_1 = DEF$ $\boldsymbol{C}_2 = ABC$ $\boldsymbol{C}_3 = BCD$ $\boldsymbol{C}_4 = CDE$ $\boldsymbol{C}_5 = DE$ $\boldsymbol{C}_6 = \mathrm{E}$

$\boldsymbol{C}_2 \cap (\boldsymbol{C}_3 \cup \cdots \cup \boldsymbol{C}_6) = BC$

Running Intersection Property

- Let $\boldsymbol{C}_i, \ldots, \boldsymbol{C}_n$ be the cluster sequence induced by applying elimination order π to graph G.
  - For every $i < n$, the variables $\boldsymbol{C}_i \cap (\boldsymbol{C}_{i+1} \cup \cdots \cup \boldsymbol{C}_n)$ are contained in some cluster $\boldsymbol{C}_j$ where $j > i$
  - This is known as the *running intersection property of the cluster sequence*

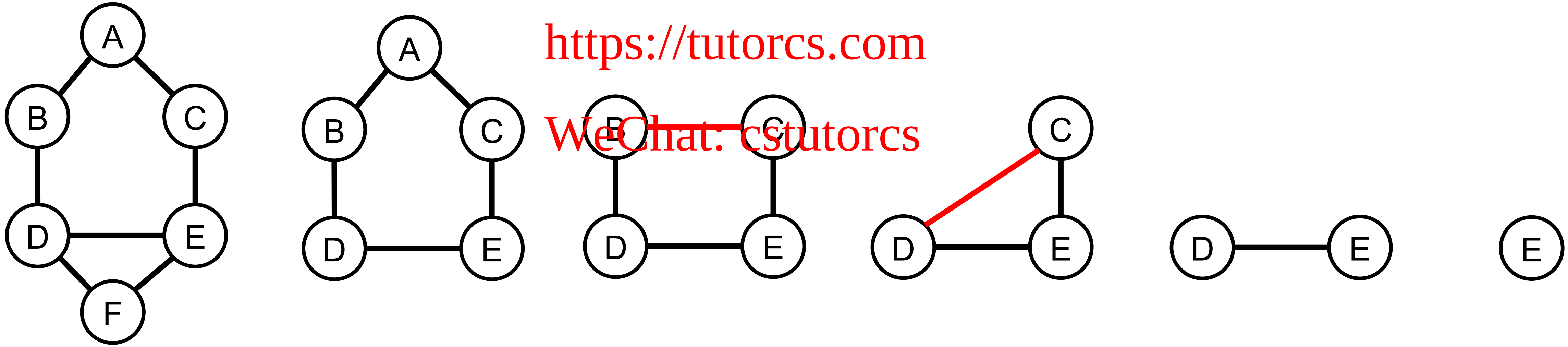


$\boldsymbol{C}_1 = DEF$ $\boldsymbol{C}_2 = ABC$ $\boldsymbol{C}_3 = BCD$ $\boldsymbol{C}_4 = CDE$ $\boldsymbol{C}_5 = DE$ $\boldsymbol{C}_6 = \mathrm{E}$

$\boldsymbol{C}_3 \cap (\boldsymbol{C}_4 \cup \cdots \cup \boldsymbol{C}_6) = CD$

Nonmaximal Clusters

- Nonmaximal clusters are the ones contained in previous clusters of the sequence
  - We can remove a nonmaximal cluster from the sequence
  - But we must reorder the sequence to maintain the running intersection property (RIP)
  - Keeping the RIP is important. We will use it to connect the cluster into a jointree later



$\boldsymbol{C}_1 = DEF$ $\boldsymbol{C}_2 = ABC$ $\boldsymbol{C}_3 = BCD$ $\boldsymbol{C}_4 = CDE$ $\boldsymbol{C}_5 = DE$ $\boldsymbol{C}_6 = \mathrm{E}$

Nonmaximal clusters

Nonmaximal Clusters

- Nonmaximal clusters are the ones contained in previous clusters of the sequence
  - We can remove a nonmaximal cluster from the sequence
  - But we must reorder the sequence to maintain the running intersection property (RIP)
  - Keeping the RIP is important. We will use it to connect the cluster into a jointree later

$\boldsymbol{C}_1 = DEF$ $\quad$ $\boldsymbol{C}_2 = ABC$ $\quad$ $\boldsymbol{C}_3 = BCD$ $\quad$ $\boldsymbol{C}_4 = CDE$ $\quad$ $\boldsymbol{C}_5 = DE$ $\quad$ $\boldsymbol{C}_6 = \mathrm{E}$

Nonmaximal clusters

Nonmaximal Clusters

- Nonmaximal clusters are the ones contained in previous clusters of the sequence
  - We can remove a nonmaximal cluster from the sequence
  - But we must reorder the sequence to maintain the running intersection property (RIP)
  - Keeping the RIP is important. We will use it to connect the cluster into a jointree later

$\boldsymbol{C}_1 = DEF$ $\quad \boldsymbol{C}_2 = ABC$ $\quad \boldsymbol{C}_3 = BCD$ $\quad \boldsymbol{C}_4 = CDE$ $\quad \boldsymbol{C}_5 = DE$ $\quad \boldsymbol{C}_6 = \mathrm{E}$

Remove $\boldsymbol{C}_5$
Move $\boldsymbol{C}_1$ to $\boldsymbol{C}_5$ position

$\boldsymbol{C}_2 = ABC$ $\quad \boldsymbol{C}_3 = BCD$ $\quad \boldsymbol{C}_4 = CDE$ $\quad \boldsymbol{C}_1 = DEF$ $\quad \boldsymbol{C}_6 = \mathrm{E}$

Nonmaximal Clusters

- Nonmaximal clusters are the ones contained in previous clusters of the sequence
  - We can remove a nonmaximal cluster from the sequence
  - But we must reorder the sequence to maintain the running intersection property (RIP)
  - Keeping the RIP is important. We will use it to connect the cluster into a jointree later

$\boldsymbol{C}_1 = DEF$ $\quad \boldsymbol{C}_2 = ABC$ $\quad \boldsymbol{C}_3 = BCD$ $\quad \boldsymbol{C}_4 = CDE$ $\quad \boldsymbol{C}_5 = DE$ $\quad \boldsymbol{C}_6 = \mathrm{E}$

Remove $\boldsymbol{C}_5$
Move $\boldsymbol{C}_1$ to $\boldsymbol{C}_5$ position

$\boldsymbol{C}_2 = ABC$ $\quad \boldsymbol{C}_3 = BCD$ $\quad \boldsymbol{C}_4 = CDE$ $\quad \boldsymbol{C}_1 = DEF$ $\quad \boldsymbol{C}_6 = \mathrm{E}$

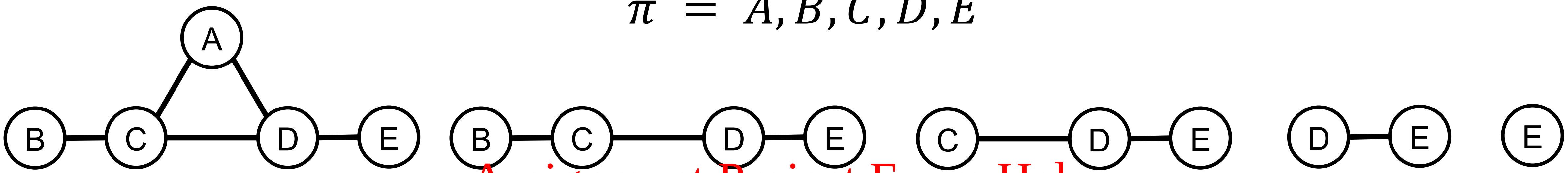
Nonmaximal Clusters

- Nonmaximal clusters are the ones contained in previous clusters of the sequence
  - We can remove a nonmaximal cluster from the sequence
  - But we must reorder the sequence to maintain the running intersection property (RIP)
  - Keeping the RIP is important. We will use it to connect the cluster into a jointree later

$\boldsymbol{C}_1 = DEF$	$\boldsymbol{C}_2 = ABC$	$\boldsymbol{C}_3 = BCD$	$\boldsymbol{C}_4 = CDE$	$\boldsymbol{C}_5 = DE$	$\boldsymbol{C}_6 = \mathrm{E}$
Remove $\boldsymbol{C}_5$ Move $\boldsymbol{C}_1$ to $\boldsymbol{C}_5$ position	$\boldsymbol{C}_2 = ABC$	$\boldsymbol{C}_3 = BCD$	$\boldsymbol{C}_4 = CDE$	$\boldsymbol{C}_1 = DEF$	$\boldsymbol{C}_6 = \mathrm{E}$
Remove $\boldsymbol{C}_6$	$\boldsymbol{C}_2 = ABC$	$\boldsymbol{C}_3 = BCD$	$\boldsymbol{C}_4 = CDE$	$\boldsymbol{C}_1 = DEF$	
	$\boldsymbol{C}_1 = ABC$	$\boldsymbol{C}_2 = BCD$	$\boldsymbol{C}_3 = CDE$	$\boldsymbol{C}_4 = DEF$	

Nonmaximal Clusters: Another Example

$$\pi = A, B, C, D, E$$



$\boldsymbol{C}_1 = ACD$ $\boldsymbol{C}_2 = BC$ $\boldsymbol{C}_3 = CD$ $\boldsymbol{C}_4 = DE$ $\boldsymbol{C}_5 = \mathrm{E}$

$\boldsymbol{C}_1 = ACD$ $\boldsymbol{C}_2 = BC$ $\boldsymbol{C}_3 = CD$ $\boldsymbol{C}_4 = DE$ $\boldsymbol{C}_5 = E$

Remove $\boldsymbol{C}_3$
Move $\boldsymbol{C}_1$ to $\boldsymbol{C}_3$ position

$\boldsymbol{C}_2 = BC$ $\boldsymbol{C}_1 = ACD$ $\boldsymbol{C}_4 = DE$ $\boldsymbol{C}_5 = E$

Remove $\boldsymbol{C}_5$

$\boldsymbol{C}_2 = BC$ $\boldsymbol{C}_1 = ACD$ $\boldsymbol{C}_4 = DE$

Elimination Orders to Jointrees: Assembly (1)

$\boldsymbol{C}_1, \ldots, \boldsymbol{C}_n \leftarrow$ maximal cluster sequence induced by elimination order π
$T \leftarrow \{\boldsymbol{C}_n\}$
for $i \leftarrow n-1, \ldots, 1$ **do**
    $T \leftarrow T \cup \{\boldsymbol{C}_i\}$
    add edge between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ that contains $\boldsymbol{C}_i \cap (\boldsymbol{C}_{i+1} \cup \cdots \cup \boldsymbol{C}_n)$
return T

$\boldsymbol{C}_1 = ABC$ $\boldsymbol{C}_2 = BCD$ $\boldsymbol{C}_3 = CDE$ $\boldsymbol{C}_4 = DEF$

Elimination Orders to Jointrees: Assembly (1)

$\boldsymbol{C}_1, \dots, \boldsymbol{C}_n \leftarrow$ maximal cluster sequence induced by elimination order π

$T \leftarrow \{\boldsymbol{C}_n\}$

for $i \leftarrow n-1, \dots, 1$ **do**

$\quad T \leftarrow T \cup \{\boldsymbol{C}_i\}$

$\quad$ add edge between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ that contains $\boldsymbol{C}_i \cap (\boldsymbol{C}_{i+1} \cup \cdots \cup \boldsymbol{C}_n)$

return T

$\boldsymbol{C}_1 = ABC$ $\quad$ $\boldsymbol{C}_2 = BCD$ $\quad$ $\boldsymbol{C}_3 = CDE$ $\quad$ $\boldsymbol{C}_4 = DEF$

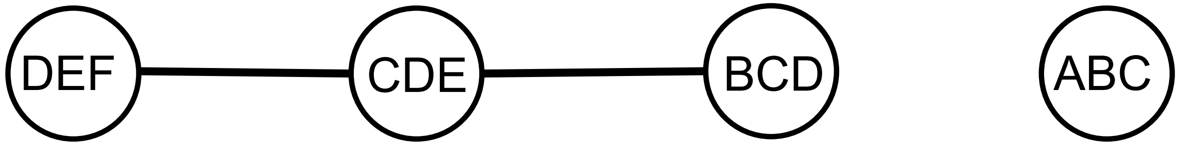
$\boldsymbol{C}_3 \cap \boldsymbol{C}_4 = DE$

Elimination Orders to Jointrees: Assembly (1)

$\boldsymbol{C}_1, \ldots, \boldsymbol{C}_n \leftarrow$ maximal cluster sequence induced by elimination order π
$T \leftarrow \{\boldsymbol{C}_n\}$
for $i \leftarrow n-1, \ldots, 1$ **do**
    $T \leftarrow T \cup \{\boldsymbol{C}_i\}$
    add edge between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ that contains $\boldsymbol{C}_i \cap (\boldsymbol{C}_{i+1} \cup \cdots \cup \boldsymbol{C}_n)$
return T

$\boldsymbol{C}_1 = ABC$ $\boldsymbol{C}_2 = BCD$ $\boldsymbol{C}_3 = CDE$ $\boldsymbol{C}_4 = DEF$

$\boldsymbol{C}_2 \cap (\boldsymbol{C}_3 \cup \boldsymbol{C}_4) = CD$



Elimination Orders to Jointrees: Assembly (1)

$\boldsymbol{C}_1, \ldots, \boldsymbol{C}_n \leftarrow$ maximal cluster sequence induced by elimination order π

$T \leftarrow \{\boldsymbol{C}_n\}$

for $i \leftarrow n-1, \ldots, 1$ **do**

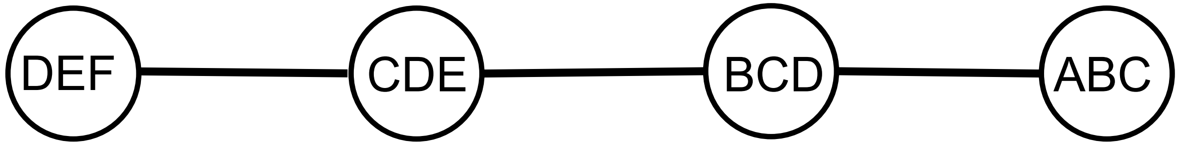
$\quad T \leftarrow T \cup \{\boldsymbol{C}_i\}$

$\quad$ add edge between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ that contains $\boldsymbol{C}_i \cap (\boldsymbol{C}_{i+1} \cup \cdots \cup \boldsymbol{C}_n)$

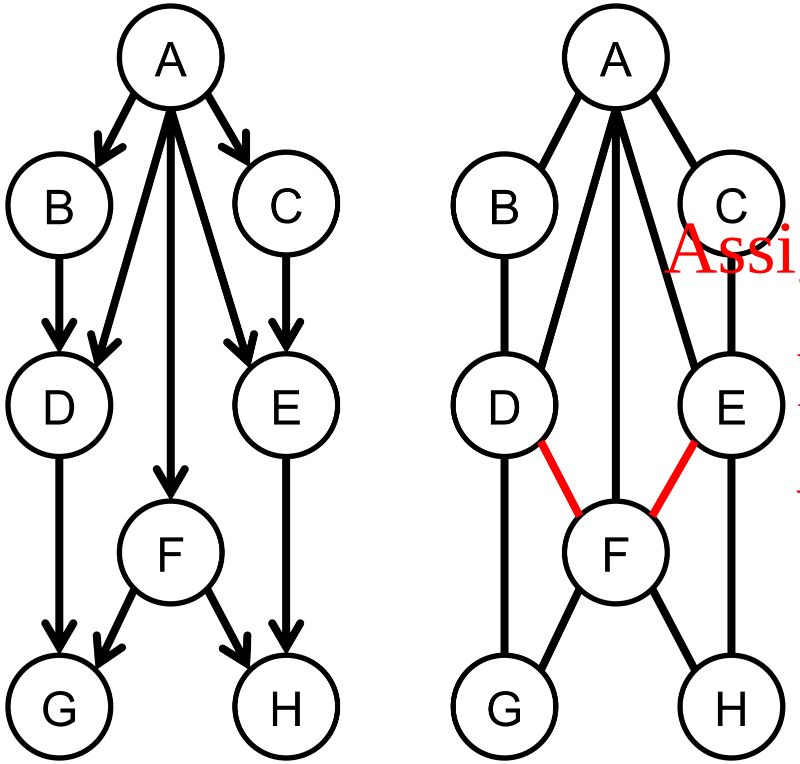
return T

$\boldsymbol{C}_1 = ABC$ $\quad \boldsymbol{C}_2 = BCD$ $\quad \boldsymbol{C}_3 = CDE$ $\quad \boldsymbol{C}_4 = DEF$

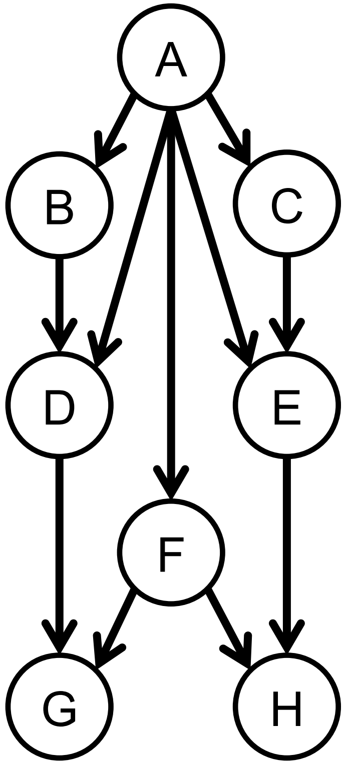
$\boldsymbol{C}_1 \cap (\boldsymbol{C}_2 \cup \ldots \cup \boldsymbol{C}_4) = BC$

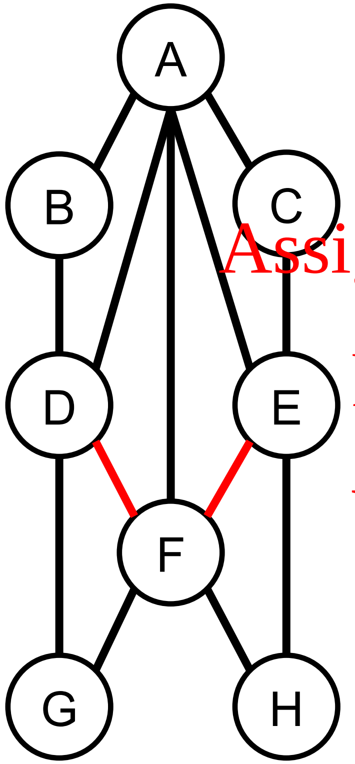


Elimination Orders to Jointrees: Example



Elimination Orders to Jointrees: Example

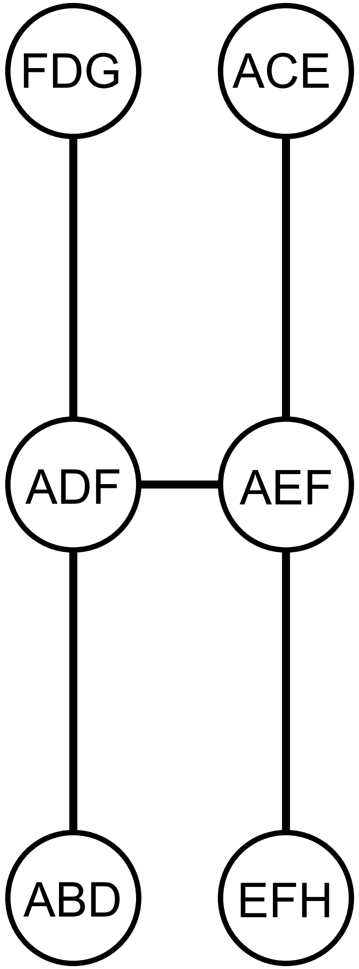




$\pi = G, H, B, C, D, E, F, A$

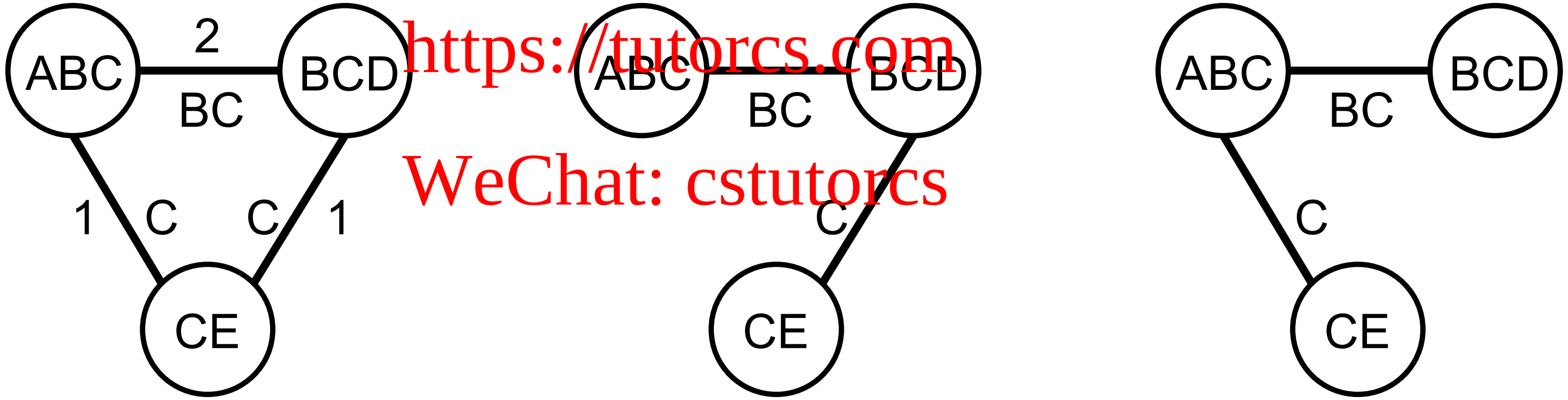
$\boldsymbol{C} = \{DFG, EFH, ABD, ACE, ADF, AEF, AF, A\}$

$\boldsymbol{C}' = \{DFG, EFH, ABD, ACE, ADF, AEF\}$



Elimination Orders to Jointrees: Assembly (2)

$\boldsymbol{C}_1, \ldots, \boldsymbol{C}_n \leftarrow$ maximal cluster sequence induced by elimination order π
$G_c \leftarrow$ cluster graph of $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_n$
$G_c^{MST} \leftarrow$ maximum spanning tree of G_c
return G_c^{MST}

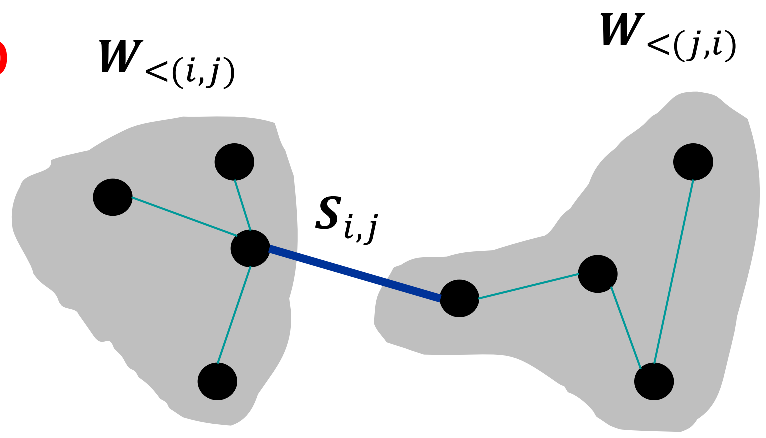


A *cluster graph* is a complete weighted undirected graph with cost $|\boldsymbol{C}_i \cap \boldsymbol{C}_j|$ assigned to each edge $\boldsymbol{C}_i - \boldsymbol{C}_j$

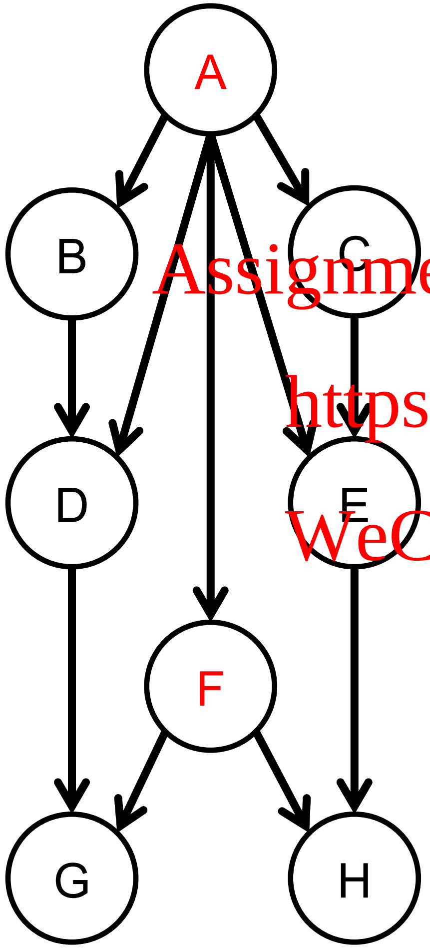
Jointrees and Independence

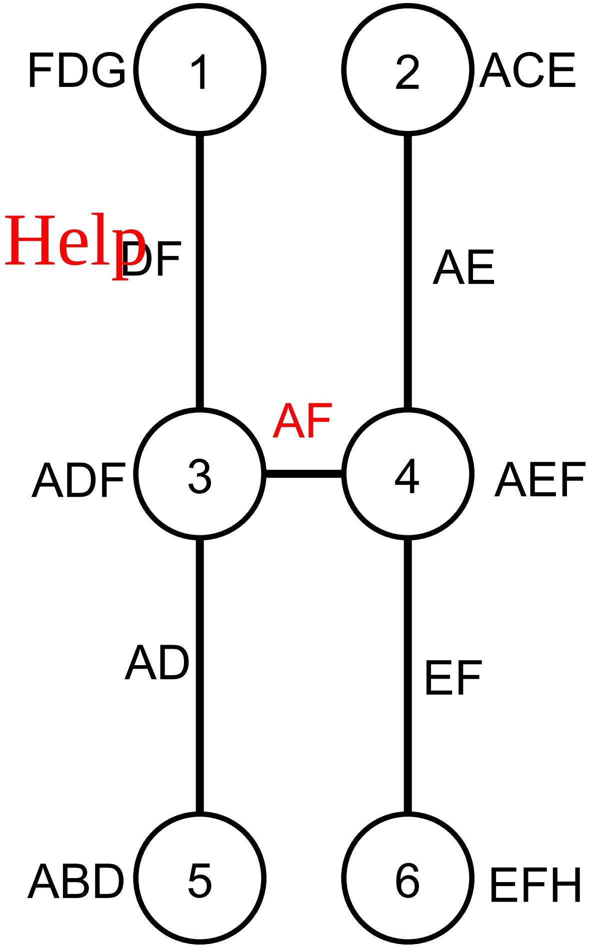
- For an edge (i, j) be the set of all variables that appear only on the $\boldsymbol{C}_i$ side of the jointree T, let
  - $\boldsymbol{W}_{<(i,j)}$ be the set of all variables that appear only on the $\boldsymbol{C}_i$ side of T
  - $\boldsymbol{W}_{<(j,i)}$ be the set of all variables that appear only on the $\boldsymbol{C}_j$ side of T
  - Variables on both sides are in the separation $\boldsymbol{S}_{i,j}$
- Then

$$\boldsymbol{W}_{<(i,j)} \perp \boldsymbol{W}_{<(j,i)} | \boldsymbol{S}_{i,j}$$



Jointrees and Independence



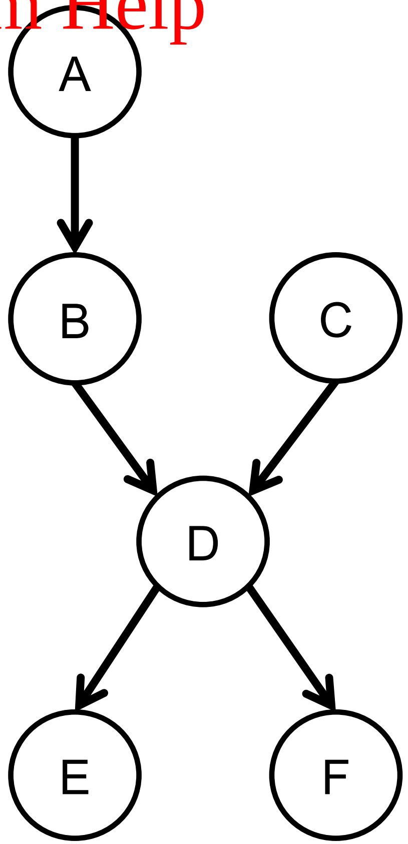


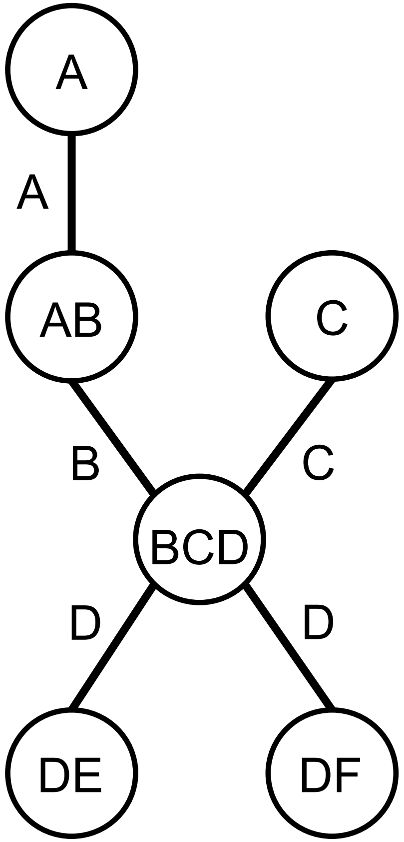
Jointrees as a Factorization Tool

- Given a probability distribution $P(X)$ induced by a graph and given a corresponding jointree
  - The distribution can be expressed in the following form

$$P(X) = \frac{\prod_{jointree\ node\ i} P(\boldsymbol{C}_i)}{\prod_{jointree\ edge\ i-j} P(\boldsymbol{C}_i \cap \boldsymbol{C}_j)}$$

$$P(A,B,C,D,E,F) = \frac{P(A)P(C)P(B,A)P(D,B,C)P(E,D)P(F,D)}{P(A)P(B)P(C)P(D)P(D)}$$





Conclusion

- In this lecture we formalized several concept related to VE and FE
- We defined polynomial time width-preserving algorithms to convert
  - Elimination orders to Jointrees
  - Jointrees to elimination orders
- We also discussed approaches to find optimal prefixes and treewidth lower bound
  - These algorithms have practical implications since they can be used with heuristics and search methods
- Tasks
  - Read Chapter 9 from the textbook (Darwiche)