Tool



Pass



Minimum Spanning Tree

MST

## Outline



**Goal:** Understand interaction o n/Communication".



- Shortest Paths
  - Dijkstra[a] (or BFS with weights)
- Centralized MST[b] (Minimum Spanning Tree)



  - Prim (Outlin
  - Kruskal (Out



- Distributed MST



  - Gallager-Humblet-Spira (SynGHS)
- Appendix

Will focus: Communication and Computation

[a] This can be used as a review since the non-distributed Dijkstra algorithm may have already been covered in other courses,

[b] This can be used as a review since the non-distributed MST material may have also been covered in other courses.

## Motivation: Shortest Paths



- Consider a strongly connecte raph, with unidirectional communicat gubors.



- BFS finds shortest path with the hop distance. How do we generalize BFS?



  - Assume that each (un)directed edge has an associated nonnegative





distance between u, v is the number of hops from u to v

- The weight of a path is the sum of the weights on its edges.

## Shortest Path Problem



- Problem: find a shortest path fro ed source node to every other node, where



  – a shortest path is a path with minimum weight.

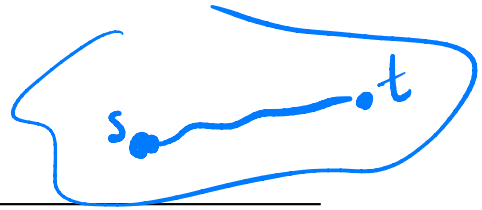
- A collection of shortest paths from the source to all the other nodes in the digraph constitutes a subtree of the digraph, all of whose edges are ori





  – Does the collection of shortest paths f e node in an undirected graph form a tre r a bidirectional ring of $n$ nodes.) How about if the edge weights are pairwise different?[a]







[a] Why?

## Applications: Shortest Paths

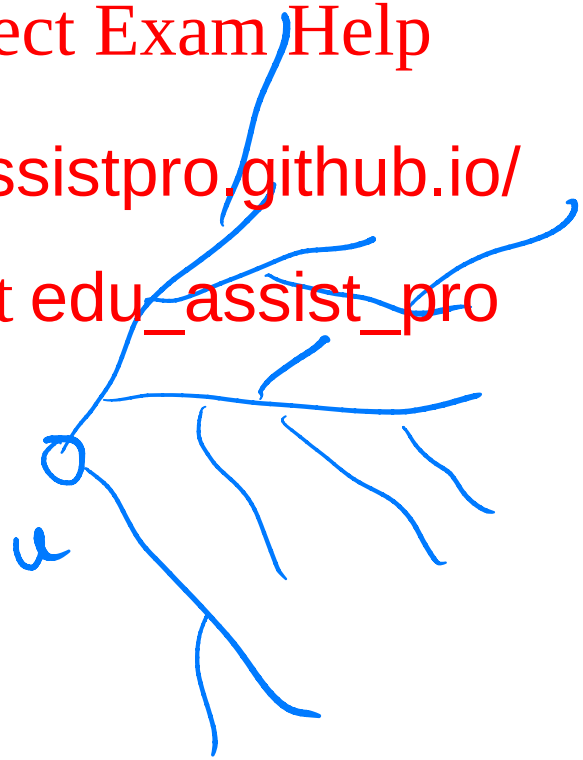


- Motivation for constructing s om the desire to have a convenient structure f munication.



- Weights represent costs associated with the traversal of edges, for instance,
  - communication delay,
  - bandwidth,
  - monetary charge.



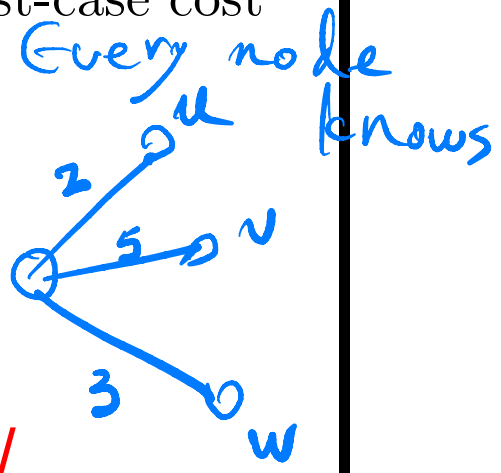


## Shortest Paths' Trees (SPTs)



- A shortest paths' tree minimiz worst-case cost of communicating with any pr rk.






- We assume that every process initially knows
  1. the weight of all its incident edges.
  2. the number





- We require that ea
  1. its parent in a particular shortest paths' t
  2. its distance (i.e., the total weight of its shortest path) from the source.

## Construction of BFS



- If all edges are of equal weight, the shortest paths tree.



  - a trivial modification of the simple SynchBFS tree construction can be made to produce the distance information as well as the parent pointers.



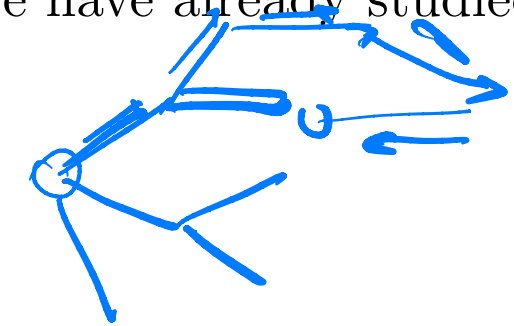
- In synchronous sy ple yet efficient method t



  - in asynchronous systems the spann cted by the flooding algorithm may be far from BFS.



- In standard BFS constructions (that we have already studied) all edges have weight 1.



## Construction of BFS (with weights)



- A classic BFS construction is



  – Dijkstra's Algorithm

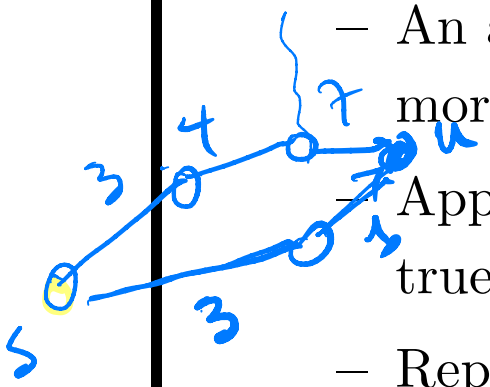
and can be a sychronous/asynchronous algorithm

## Dijkstra's Algorithm and Relaxation



- Based on the **principle of** .
  - An approximation to corr ually replaced by more accurate values until reaching the optimum solution.
  - Approximate distance to each vertex is an overestimate of true distance.
  - Replaced by t newly found p








$true\text{-}dist_s(u) \le approx\text{-}dist_s(u)$





- Greedily selects a node "correspondin dge" that has not yet been processed, and performs this relaxation process on all of its outgoing edges.
- Processing (usually done with a heap) which also counted as part of the cost

## Basic Idea of Constructing Shortest Paths

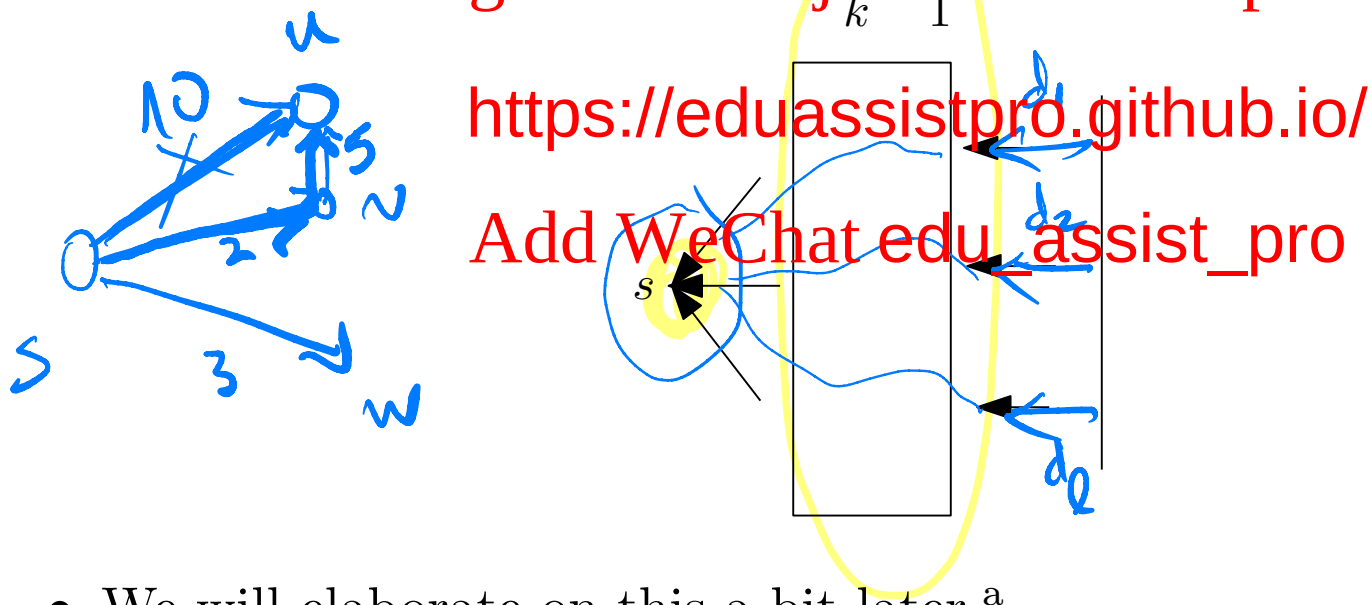


- Let $S_k$ be the set of $k-$ he destination $s$.



- During the $k$th step the o the destination $s$ is found by considering
  - the distance of nodes in $N \setminus S_k$ to any node in $S_k$.










- We will elaborate on this a bit later.[a]

[a] See discussion on *blue* edges later.

## Dijkstra's Algorithm: Adding Edges (1/3)



- Finds the shortest path from the $s$ to all other nodes in a weighted graph.



  - Similar to BFS, except that it keeps track of a distance $d(j)$ (length of the shortest path known so far to node from a "root node" $j$ for each node $j$.

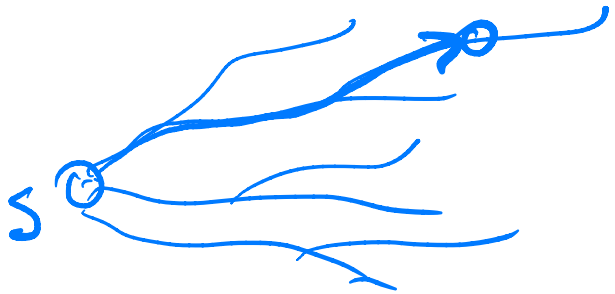


- Instead of examin izes them by the distance $d$ and picks j d node $i$ with the smallest $d(i)$, whose distance , and updates tentative distance $d$ for all its neighbors.





- The algorithm uses a heap to keep track of its unvisited nodes $j$, each with a metric $d(j)$.



## Dijkstra's Algorithm: Forming a Heap (2/3)



- Removing the item with small $O(\log n)$ time if the heap[a] contains $n$ ite



  - If an item's metric changes but it remains in the heap, it takes $O(\log n)$ time to adjust its position in the heap.
  - Initializing a heap of $n$ items takes $O(n)$ time.



- Nodes no longer in t $d(j)$ is the shortest path from $s$ to $j$.





  - The shortest path can be traced back $j$ to $s$ by walking the tree from $j$ to its parent $p(j)$, then to $p(p(j))$, and so on until reaching $s$.

---

[a] A tree-based data structure satisfying the heap property. In a max heap, for any given node $C$, if $P$ is a parent node of $C$, then the key (the value) of $P$ is greater than or equal to the key of $C$. In a min heap, the key of $P$ is less than or equal to the key of $C$.

## Dijkstra's Algorithm: Intuition (3/3)



- Nodes in heap have not been visit $d(j)$ is tentative.
  - They split into two kinds of n nite $d$ (discovered in the frontier), and those with infinite $d$ (undiscovered).
  - Each node in the frontier is incident on at least one edge from the visite
  - The node in the f $d(j)$ has a very useful property on which the algorit $d(j)$ is the true shortest distance of the path fro to $j$.



maintain information







- The algorithm selects this node and then updates its neighbors as $j$ moves from the frontier into the set of visited nodes.
- Algorithm finds the shortest path from s to all other nodes in $O((|V| + |E|) \log |V|)$ time; asymptotic time can be reduced with a Fibonacci heap; in practice a conventional heap is faster.

## Dijkstra's Algorithm: Formally



- $N$ set of all the nodes in (undirect

- $l(i,j)$ (non-negative) cost ass e $\{i,j\}$.
  - $l(i,j) = +\infty$ if there is no edge between $i,j$.
- Let $s$ be the node executing the algorithm in order to find shortest paths fro ork.

  - $s$ is the start or s

- Algorithm constructs incremental $M$ of nodes:

  - $M$ is the set of nodes incorporated so far, and
  - algorithm stops when $M = N$.
- $C(n)$ is the cost of the path from $s$ to node $n$.

## Dijkstra's Algorithm: Formally



- **Dijkstra's Algorithm**



1. $M = \{s\}$
2. **for each** $n \in N \setminus \{s\}$
3. $\quad C(n) = l(s, n)$
4. **while** $N \neq M$ **do**



5. $\quad M =$ that $C(w)$ is m

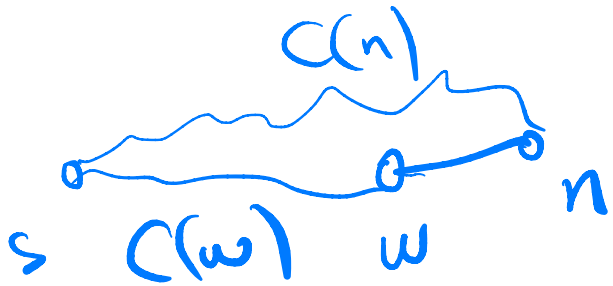


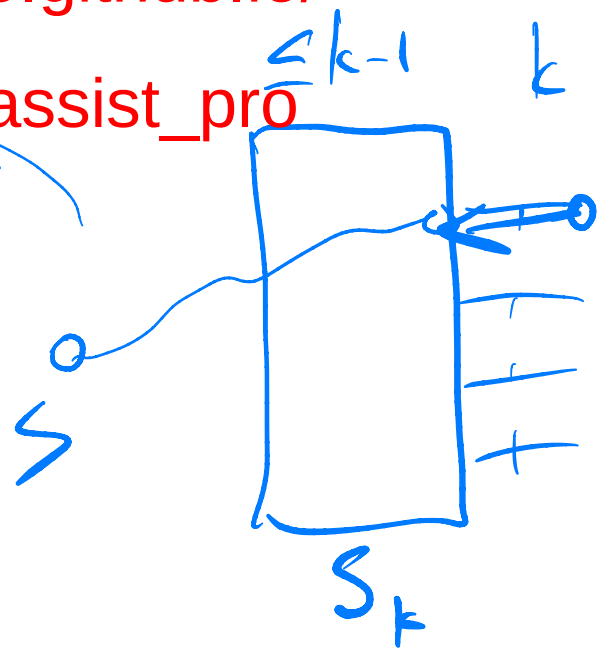
6. $\quad$ **for each** $n \in N \setminus M$



7. $C(n) = \min\{C(n), C(w) + l($

See example in appendix.



## Discussion of Analysis



- Algorithm can be implemente $O(|V|^2)$, where $V$ is the number of node



- As presented, the algorithm computes weights of paths, not the paths themselves.



- Can be easily modi[a]
  - The last edge to
    
    shortest path to destination,
    
  - can be used to compute a shortest path t
  - compute routing tables.

[a] We discuss this later.

## Application: Route Calculation in LSP



- Dijkstra's Algorithm used for r n in Link State Protocol (LSP)



- Finds shortest paths from all nodes to some fixed destination (or source)



- Requires that all ed restriction for mos



- Shortest paths found in order of increasi



Dijkstra Tree

## Dijkstra BFS Tree: Constructing the Routes (1/2)



- The algorithm proceeds in pha g trees.
  - In phase $p$ the nodes at di from the root are detected.
  - Let $T_p$ denote the tree constructed in phase $p$.



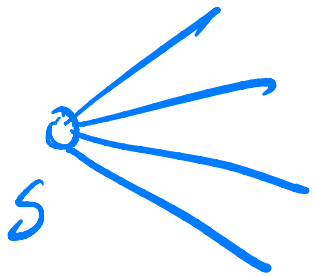


- The starting phas
  - Tree $T_1$ is th us all direct neighbors of the root which ha





- We now determine how to update from phase $p$ to phase $p+1$;




$p \leftarrow p+1$

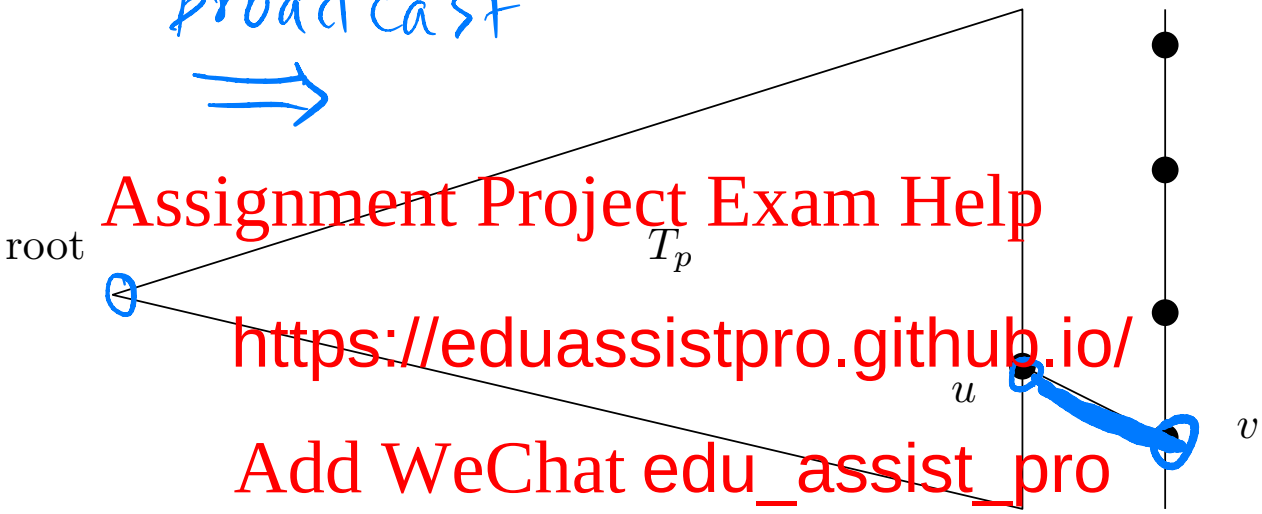
## Dijkstra's Algorithm Tree Construction



- Construction of tree $T_p$ i rithm



broadcast






- Broadcast from the root in phase $p$ and echo.
- The root decides which vertex $v$ is selected with a echo/broadcast subroutine.

## Dijkstra BFS: Constructing the Routes (2/2)



**repeat**



1. The root starts phase $p$ g "start $p$" within $T_p$.
2. When receiving "start p" a leaf node $u$ of $T_p$ (that is, a node that was newly discovered in the last phase) sends a "join $p+1$" message to all quiet neighbors. (A neighbor $v$ is quiet if $u$ has not yet "talk



3. Node $v$ receivi lies with ACK and becomes a leaf of the tree $T$ es decline */





4. The leaves of $T_p$ collect all the answers of their neighbors; then the leaves start an echo algorithm back to the root.
5. When the echo process terminates at the root, the root increments the phase

**until** there was no new node detected

## Analysis of Dijkstra BFS



- **Theorem 1** *In Dijkstra's a*
  - *the time complexity is* $O(n \log n + m)$ *a*
  - *the message complexity is* $O(m + nD)$,

  *where* $D$ *is the diameter of the graph,* $n$ *the number of nodes,*[a] *and* $m$ *the number of edges. Adjusting the position of an element (vertex)* $i$ *...* $O(\log n)$.









$1 + \cdots + k + (k+1) + (k+2) + \cdots + D$

[a] Note that earlier we used $V$ for the number of nodes; here we use $n$.

## Analysis of Dijkstra BFS



- Time Complexity
  - A broadcast/echo algorit[boilerplate] needs at most time $2D$.
  - Finding new neighbors at the leaves costs 2 time units.
  - Since the BFS tree height is bounded by the diameter, we have $D$ phases, giving a total time complexity of $O(D^2)$.
- Message Comple[boilerplate]
  - Each node par[boilerplate]ives at most 1 message and sends (echoes) at most $\cdots$
  - There are $D$ phases, so message cost is bounded by $O(nD)$.
  - On each edge there are at most 2 "join" messages.
  - Replies to "join" request are answered by 1 "ACK/NACK" (so we have at most 4 additional messages per edge).
  - Message complexity is $O(m + nD)$. Processing using the heap costs a factor $O(\log n)$.

## Applications: RIP and BGP



- A distributed variant of the Bell orithm is used in distance-vector routing prot le the Routing Information Protocol (RIP).



- The algorithm is distributed and involves a number of nodes (routers) within an Autonomous System (AS), a collection of IP networks typic following steps:







  1. Each node calculates distances betwe er nodes within the AS and stores this information as a table.
  2. Each node sends its table to all neighboring nodes.
  3. When a node receives distance tables from its neighbors, it calculates the shortest routes to all other nodes and updates its own table to reflect any changes.

## Questions in Graph Search



- When you ask your smartphon est route from Ottawa to Vancouver, ho route?



- If you ask it to help you drive from Ottawa to Paris, how does it know you cannot do that?



- If you post somethi nds and friends of frien



- These questions can all be posed in terms of s h (also called graph traversal).

## Graph Representation



- For an unweighted graph $n$ nodes, storing a binary adjacency matrix nal array using $O(n^2)$ memory works well in a graph traversal algorithm if the graph is small or if there are edges between many of the nodes.



  - In this matrix $A = (a_{ij})$, $a_{ij} = 1$ if $\{i$





  - With edge wei al matrix can represent the weight of the edge gh this assumes that in the problem at hand an edge with zero weight is the same as no edge at all.

## Example Graph Representation



- A graph and its adjacency matri

## Graph Representation



- A road network can be represen irected graph, with each node an interse ge a road between them.



- The graph is directed because some roads are one-way, and it is unconnected because you cannot drive from Ottawa to Paris.



  - The edge weig e along the road from $i$ to $j$;



  - all edge weights must therefore be gre [a]



- An adjacency matrix works well for a single small town, but it takes too much memory for the millions of intersections in the North American road network.

[a] There are situations where negative weights are natural to use.

## Graph Representation



- Fortunately, only a handful of r t any one node, and thus the adjacency matrix i arse.



- This kind of matrix or graph is best represented with adjacency lists, where each node $i$ has a list of nodes $j$ that it is adjacent to, along with the weights of the corresponding edges.

## Searching a Graph



- Searching a graph from a single s $s$, discovers all nodes reachable from $s$ v e graph.



- Nodes are marked when visited, so to search the entire graph, a single-source algorithm can be repeated by starting a new search from each node in the graph, ignoring nodes that have already been visit





- The graph traversal then performs acti isited, and also records the order in which the grap .



- Often, only a small part of the graph needs to be searched.
- This can greatly speed up the solution of problems such as route planning.

## Spanning Trees



- A spanning tree of a graph is a subgr
  - that is a tree (i.e., contains no c
  - includes all of the nodes of the graph



- If the edges of the network are weighted (e.g., representing average delay exp[a] inimum weight spanning tree is of





- Two non-distributed algorithms for co :
  - Prim's algorithm
  - Kruskal's algorithm



- A distributed algorithm for computing MST:
  - GHS (Gallagher, Humblet, Spira).

[a] Local Area Network.

## Spanning Tree Terminology



- $G = (V, E)$ is an undirected gr
  - $V$ is the set of nodes (vertice
  - $E$ is the set of edges



- $w_{i,j}$ is the weight of the edge $\{i, j\}$

$w_{ij} \geq 0$

- A *spanning tree* is an acyclic subgraph containing all nodes



- Weight of a tree



$$w(T) =$$



$$\{i,j$$

where $E(T)$ is the set of edges in $T$.

- MST (Minimum weight Spanning Tree) is a ST (Spanning Tree) of minimum weight

$$\min_{T} w(T)$$

## Two Basic Sequential Spanning Tree Algorithms



1. Prim's Algorithm (Jarnik, 1930



   - Start from any root node.
   - Always maintain a connected subtree (check for cycles).
   - Among possible choices add a "min weight" edge at a time



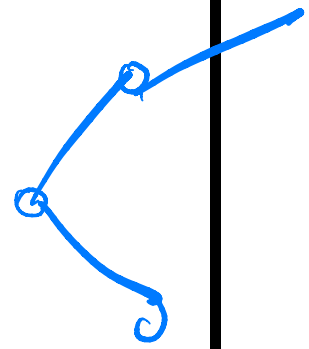
2. Kruskal's Algorith



   - Sort the edges i
   - Always maintain a forest (check for c



   - Add edges in order as long as no cycles created



Both are centralized

## Prim's Algorithm (Jarnik, 1930)[a]



- **Prim's Algorithm**



- $P$ is current set of nodes in tree
- $D_i$ is min weight edge from node $i$ to a node in $P$
- Initially $P = \{1\}$ and $D_i = w_{i,1}$ if $\{i, 1\}$ exists, $\infty$ otherwise



  1. Find $i \notin P$
  2. $P = P \cup \{i$



  3. For $j \notin P$, $D_j = \min\{D_j, w_j,$



  4. Go back to 1

example in the appendix

Can be implemented in $O(|E| + |V| \log |V|)$ time. See example in appendix.

---

[a]1) Jarnk, V. (1930), "O jistm problmu minimlnm" [About a certain minimal problem], Prce Moravsk Prodovdeck Spolenosti (in Czech), 6 (4): 57-63. 2) Prim, R. C. (November 1957), "Shortest connection networks And some generalizations", Bell System Technical Journal, 36 (6): 1389-1401,

## Kruskal's Algorithm (Boruvka, 1926)[a]



- **Kruskal's Algorithm**



1. Sort the edges of $G$ i i r
2. Consider edges in order and add edge to tree if the result does not form a cycle



- Time complexit i e in appendix.



Example in the appendix

[a] 1) Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". Proceedings of the American Mathematical Society. 7: 48-50. 2) Borvka, Otakar. O Jistm Problmu Minimlnm. Prce Moravsk Prodovdeck Spolenosti III, no. 3 (1926): 37-58.

Assignment Project Exam Help

# Dis



# T



1. Communication & Computation
2. Concept of the round.

## Distributed MST Algorithm



- Given a graph $G$, the goal is to ted algorithm that always terminates, and co of $G$.



- At the end of an execution, each processor knows which of its incident edges belong to the MST and which do not (i.e. the processor writes in a local output register the corresponding incident edges).





- In the distributed version of the MST, a co network is solving a problem where the in itself.[a]



- This is one of the fundamental starting points of (distributed) network algorithms.

[a] A "local" version of the MST is not possible: We won't discuss this here.

**Main Idea: Select**



- Recall





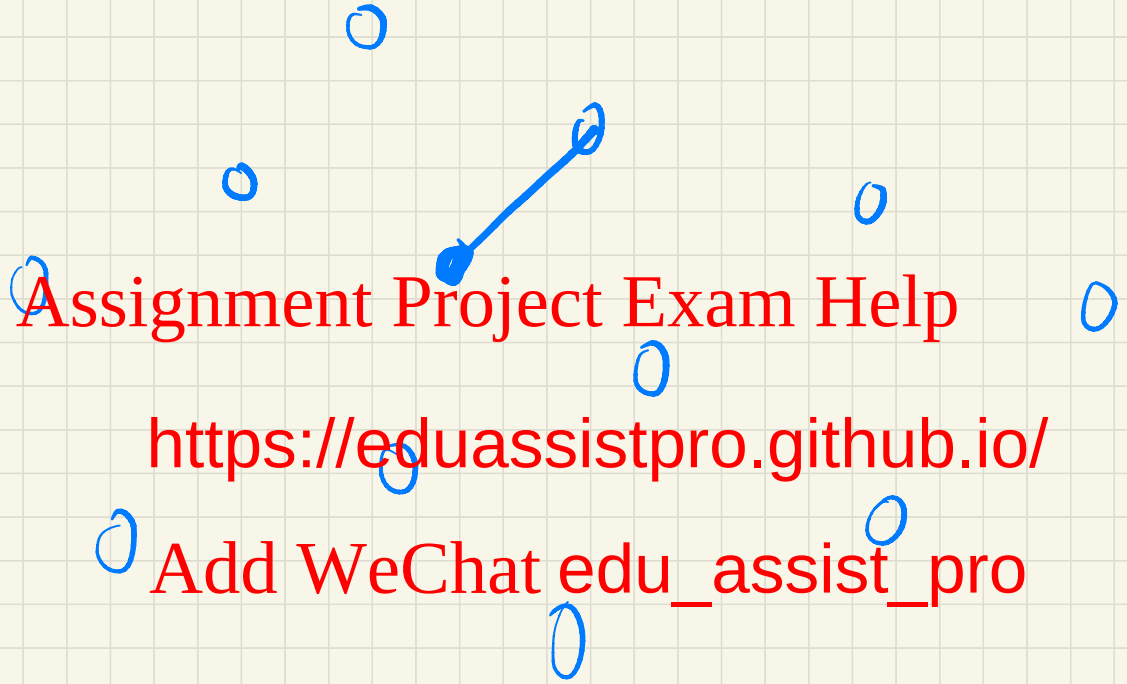
(a) is a tree,

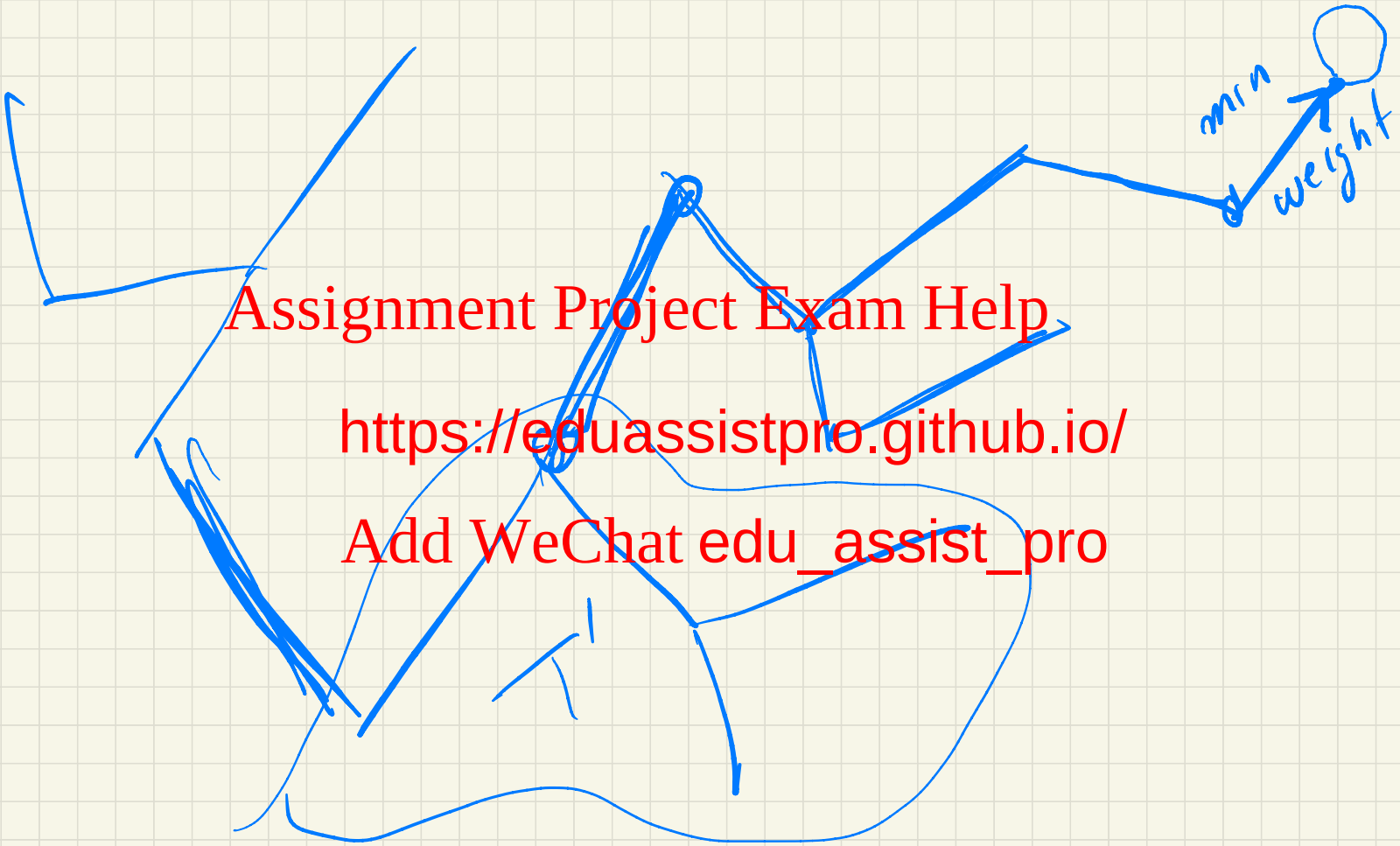


1. Start with trivial spa $n$ individual nodes and no edges: at the start each vertex i



2. Repeatedly do the following: Select
   (a) an arbitrary component $C$ (i.e., tree) in the forest, and
   (b) an arbitrary outgoing edge $e \in C$ having minimum weight among the outgoing edges of $C$.



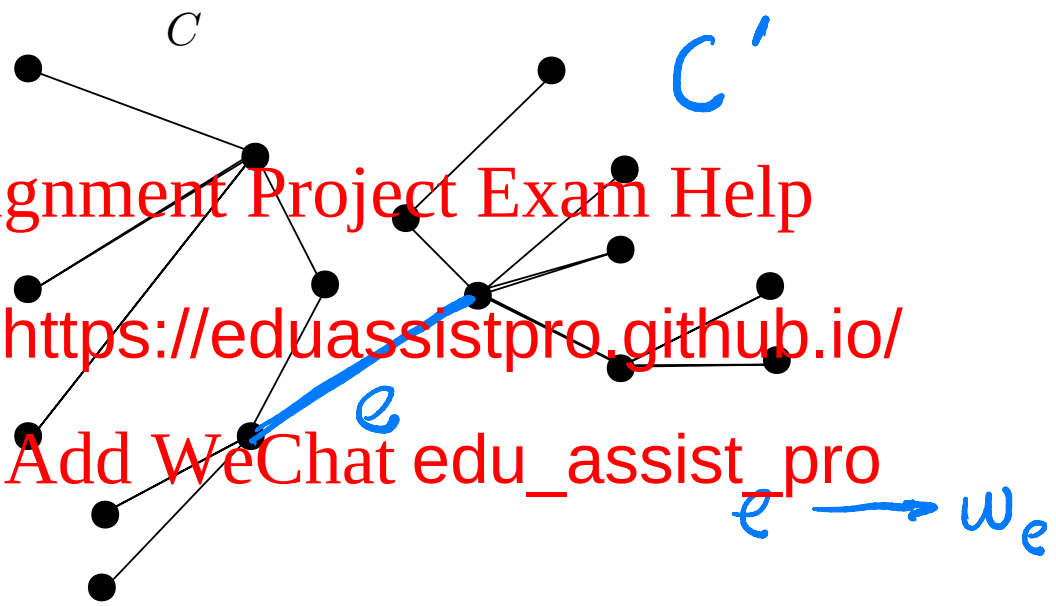


## Main Idea: Merge



1. For such an edge $e$, combi component at the other end of $e$, including edge ined component.








2. Stop when the forest has a single component.

Merge C & C'

## Assumptions



- Now we investigate how to desi lgorithm.



- We assume that no two edges of th ame weight.
  - This simplifies the problem
  - simplificatio



  - one can always nt vertices to the weight.





Assume weights are pairwise different!

Otherwise: tag $\langle ID_u, w_{\{u,v\}} \rangle$

## Concept of Blue Edges



- Let $T$ be a spanning tree of the w $G$
  - A subgraph $T' \subseteq T$ o gment.



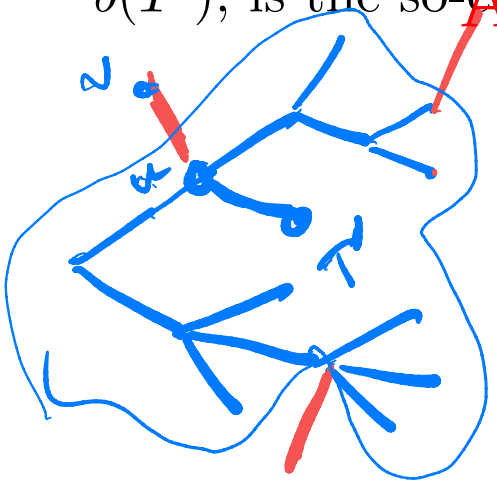
- Edge $e = \{u, v\}$ is an outgoing edge of $T'$ if either
  - $u \in T'$ and $v \notin T'$ or
  - $u \notin T'$ and





- The minimum we $T'$, denoted by $b(T')$, is the so-called blue edge of






$T' \subseteq T$

$\{u, v\}$

$b(T')$ blue edge

red

## Examples: Concept of Blue Edges



- Example 1:

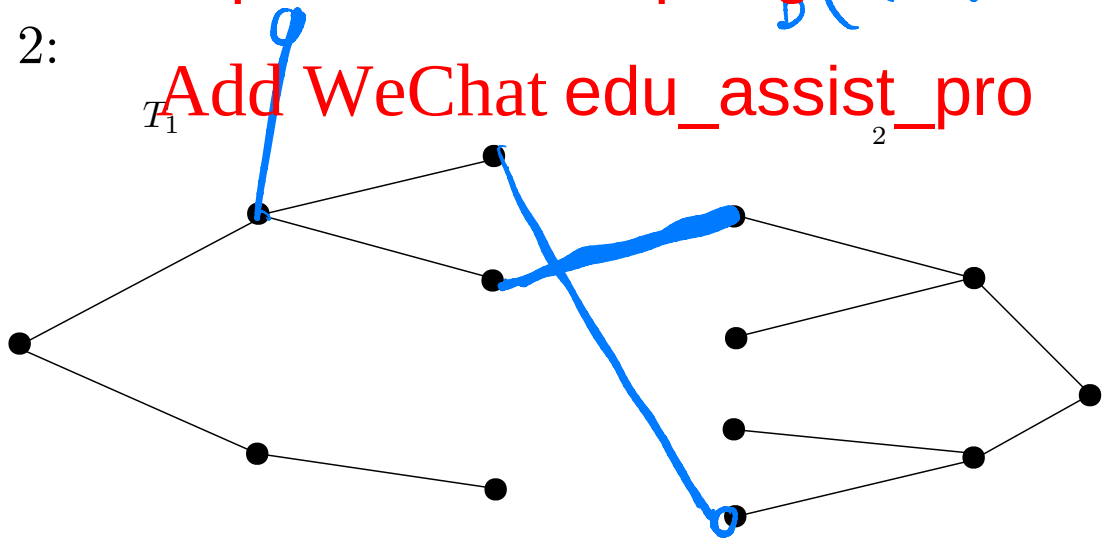






- Example 2:





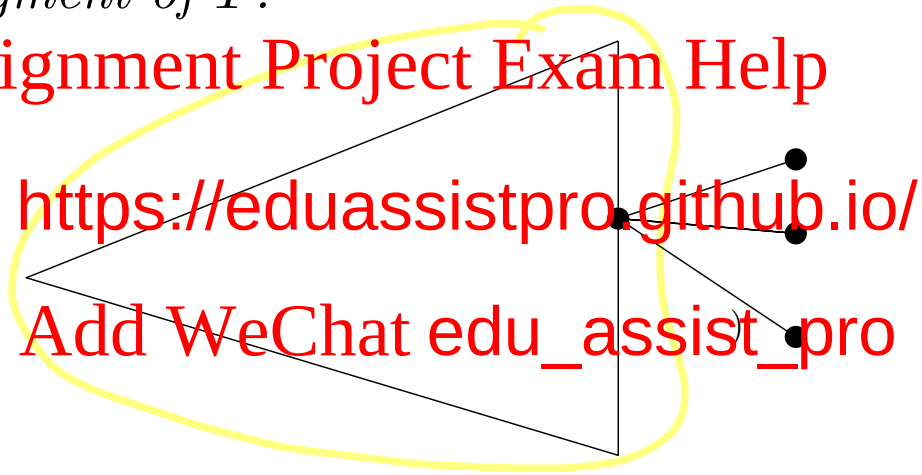
## A Lemma on Adding Blue Edges



- **Lemma 1** *For a given weigh* $G$ *(such that no two weights are the same),*
  - *let* $T$ *denote the MST, and*
  - $T'$ *be a fragment of* $T$*.*











*Then the blue edge of* $T'$ *(denoted by* $b(T')$*) is also part of* $T$*,*
  - *i.e.,* $T' \cup \{b(T')\} \subseteq T$*.*

- So, the Lemma says that blue edges can be added to an already constructed MST fragment and maintain the MST property.

## Proof of Lemma 1



- For the sake of contradiction, s ST $T$ there is edge $e \neq b(T')$ connecti e remainder of $T$.



- Adding the blue edge $b(T')$ to the MST $T$ we get a cycle including both $e$ and $b(T')$.



- If we remove $e$
  - we still have a sp
  - since by the definition of the blue edge $w_e > w_{b(T')}$, the weight of that new spanning tree is less t f $T$.[a]





- We have a contradiction.

[a] Here we used the fact that the edge weights are different!

## Issues



- Blue edges
  - allow a fragment to grow in a g
  - seems to be the key to a "distributed algorithm" for the "distributed MST" problem.





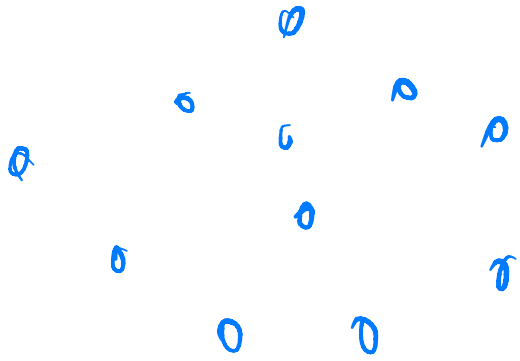
- Since every node it e directly has a blue e



- All we need to do now is to grow these fragme



  - Essentially this is a distributed version of Kruskal's sequential algorithm.



## A Distributed Algorithm

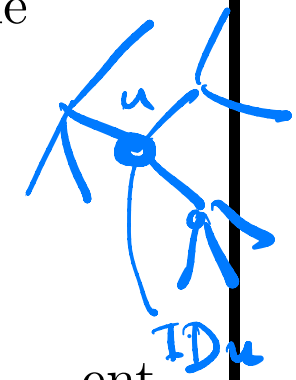


- At any given time the nodes of the g oned into fragments (rooted subtrees of t



- Each fragment has a designated vertex called root (of the fragment):
  - ID of fragment is defined to be the ID of its root.








- In the course of the al
  - each node knows its parent and its chil ent.



- The algorithm operates in phases.
- At the beginning of a phase, nodes know the IDs of the fragments of their neighbor nodes.

**SynGHS (Gallager, Humblet, Spira)**



- The algorithm builds the comp s" (or phases).



- For each $k$, the lev $k$ c stitute a spanning forest, where each level $k$ component consists of a tree that is a subgraph of the MST.



- Each level $k$ co $k$ des.



- Each component der node.



- The processes a ow a fixed number of rou $O(n)$, to complete each level.

## SynGHS: Base and Inductive Steps

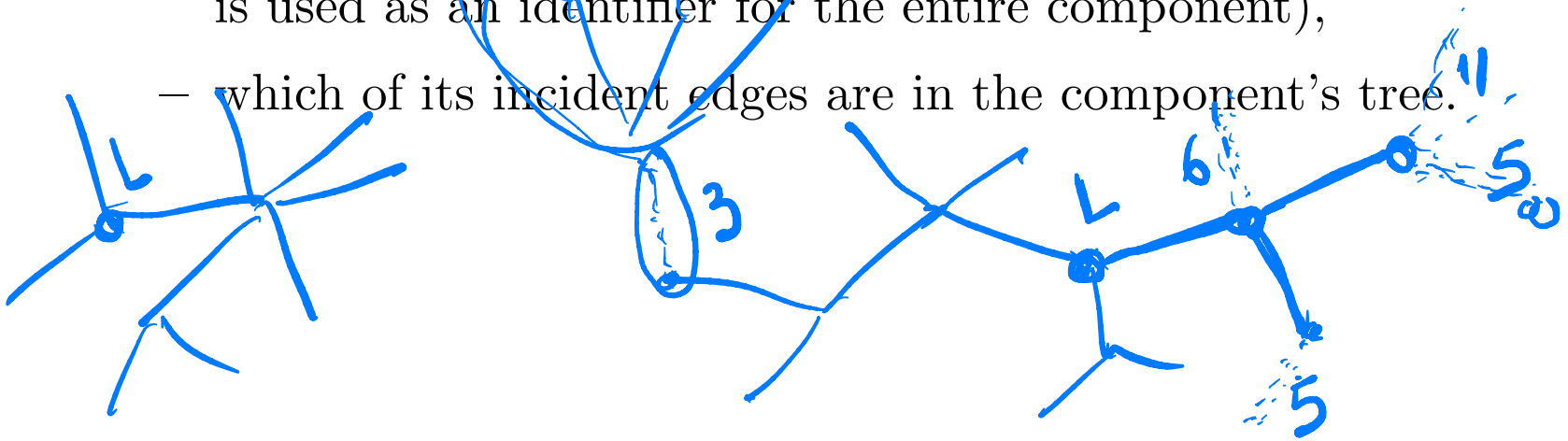


- **Base Step:**
  The algorithm starts with level $0$ components consisting of individual nodes and no edges.



- **Inductive Step:**
  Suppose inductively that the level $\kappa$ components have been determined (alo ally, suppose that each p
  - the UID (User ID) of the leader of its com D is used as an identifier for the entire component),
  - which of its incident edges are in the component's tree.





## SynGHS: Inductive Step $k \to k+1$



- To get the level $k+1$ comp el $k$ component conducts a search along its span r the MWOE (Minimum-Weight Outgoing Edge)[a] of the component.



- The leader broadcasts search requests along tree edges, using a message broadcast strategy (BFS).



- Each process finds minimum weight that is outgoing from t f there is any such edge);





  - it does this by sending test messages along all non-tree edges, asking whether or not the other end is in the same component.

[a] Recall that we called such edges: "blue" edges of the fragment.

## SynGHS: Inductive Step $k \to k+1$



- Then the processes convergec -weight edge information toward the leade a along the way).



- The minimum obtained by the leader is the MWOE[a] of the entire component.



- When all level WOEs, the components are orm the level $k+1$ components.





  - This involves the leader of each level mponent communicating with the component process adjacent to the MWOE, to tell it to mark the edge as being in the new tree;
  - the process at the other end of the edge is also told to do the same thing.

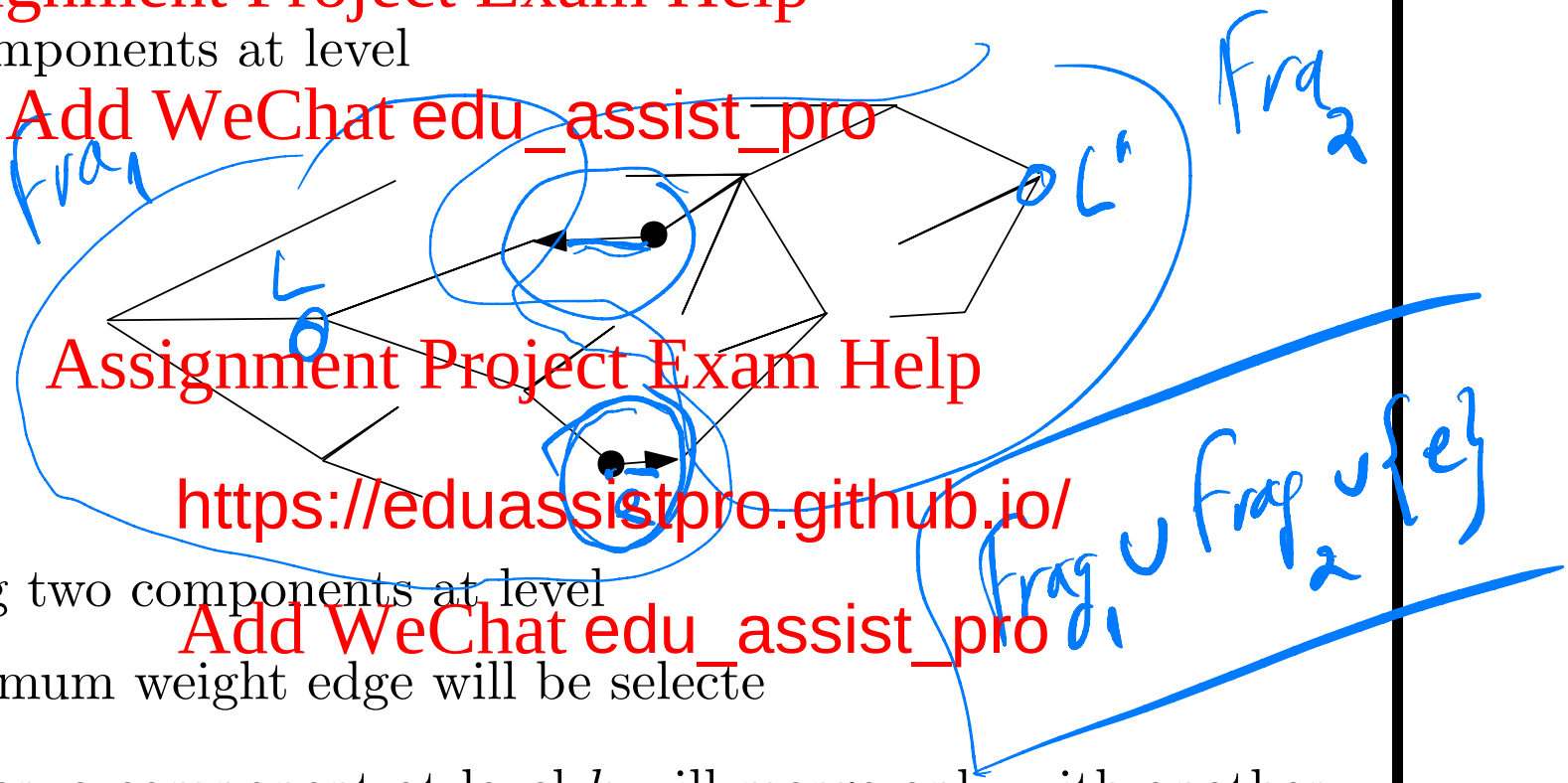
[a] I.e., "blue" edge

## SynGHS: New Leader



- Two components at level










- Merging two components at level



  - minimum weight edge will be selecte

- Moreover, a component at level $k$ will merge only with another component at level $k$ which corresponds to a minimum weight outgoing edge!

## SynGHS: New Leader



- Then a new leader is chosen for eac $k+1$ component, as follows. 
  - It can be shown that for each group of level $k$ components that get combined into a single level $k+1$ component, there is a unique edge $e$ that is the common MWOE of two of the level $k$ com 2).
  - New leader: is th larger UID.
  - **NB:** this new leader can identify its information available locally.







- Finally, the UID of the new leader is propagated throughout the new component, using a broadcast.

[a] Recall that edge weights are pairwise distinct.

## SynGHS: Conclusion



- Eventually, after some numb nning forest consists of only a single compon ll the nodes in the network.



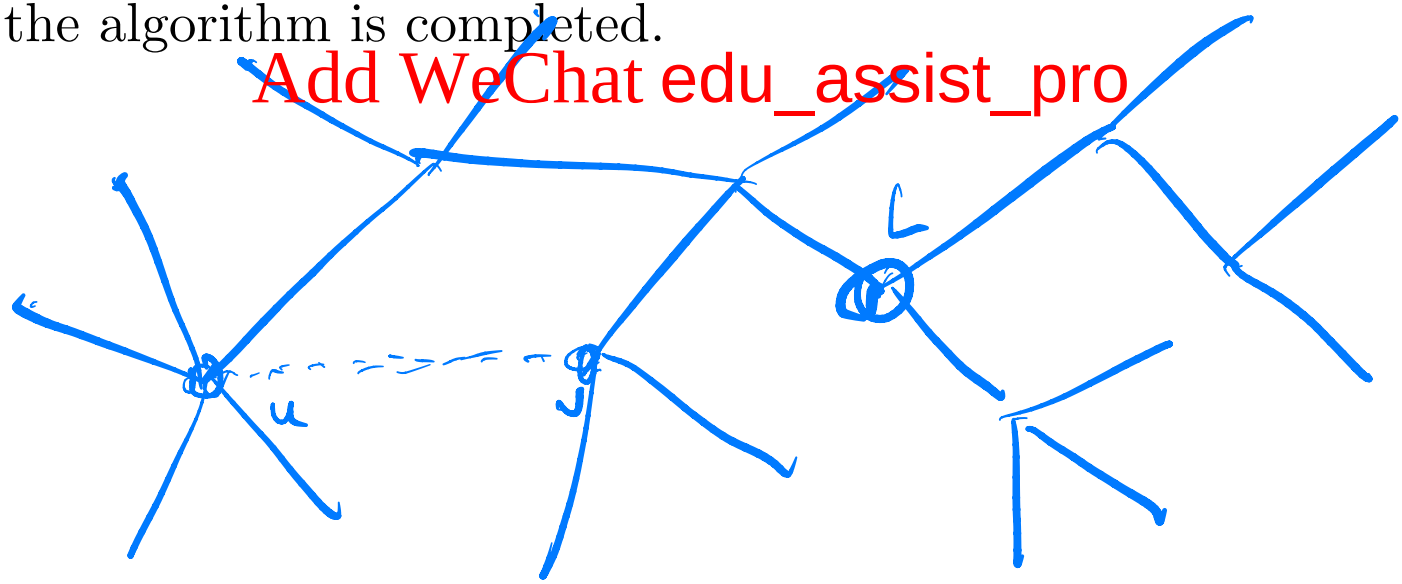
- Then a new attempt to find a MWOE will fail, because no process will find an outgoing edge.



- When the leader he ing that the algorithm is completed.







## A Key Idea of the Algorithm (1/2)



- Among each group of level nts that get combined, there is a unique (unrooted) e mon MWOE of both endpoint components.
  - To see this: consider the **component digraph** $G'$, whose nodes are the level $k$ components that combine to form one level $k+1$ co OEs.
  - $G'$ is a weakly c de has exactly one outgoing edge.[a]









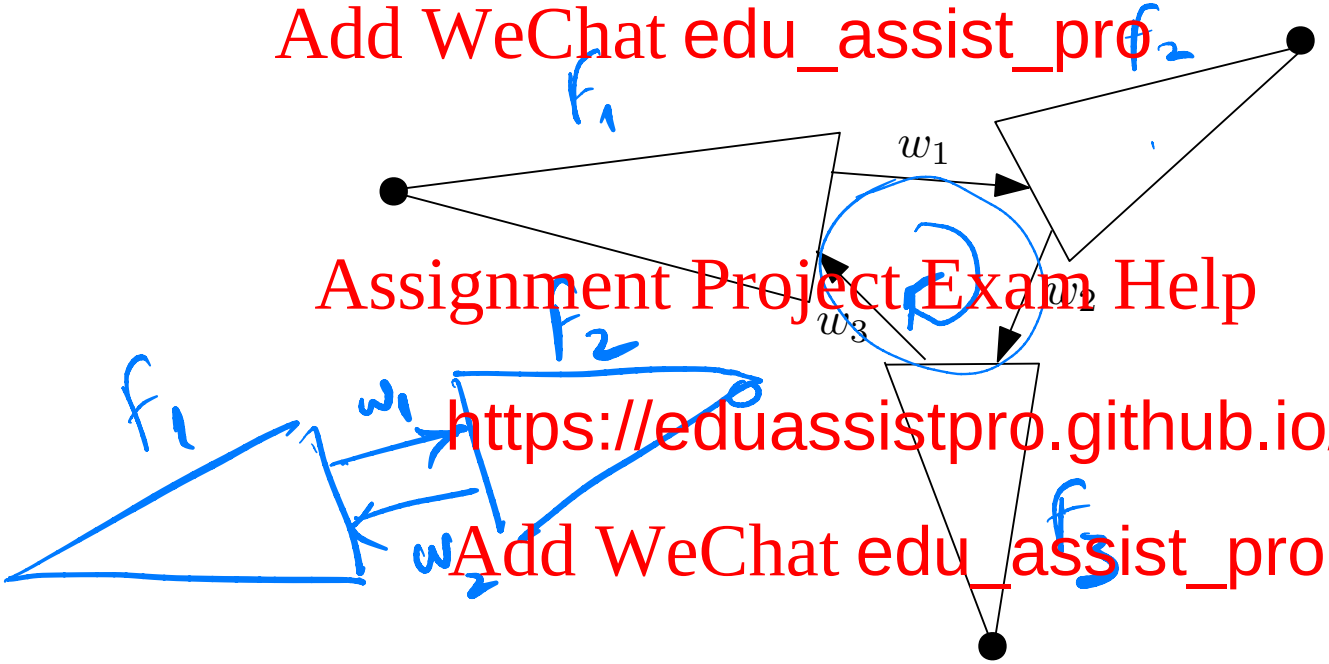
[a] A digraph is weakly connected if its undirected version, obtained by ignoring the directions of all the edges, is connected.

## A Key Idea of the Algorithm: Example



- Can we have a cycle of length omponent graph















- If yes then, $w_1 \not< w_3$ and $w_2 \not< w_1$ and $w_3 \not< w_2$!

$w_1 < w_3$ $w_2 < w_1$ $w_3 < w_2$

## A Key Idea of the Algorithm (2/2)



- So we have the following proper



**Lemma 2** *If for a weakly con- $G$ each node has exactly one outgoing edge then $G$ contains exactly one cycle.*

- We apply Lemma 2 to the component digraph $G'$ to obtain the unique cycle of components.



- Because of the way ssive edges in the cycle must have non-increasing weigh length of this cycle cannot be greater than 2.





- So the length of the unique cycle is exactly 2.
- But this corresponds to an edge that is the common MWOE of both adjacent components.

## Importance of Synchrony in SynGHS



- Synchrony ensures when a pro ries to determine whether or not the other endpoint of edge is in the same component, both $i$ and $j$ have up-to-date component UIDs.



- If the UID at $j$ is observed to be different from that at $i$, we would like to be certain that $i$ and $j$ really are in different components, not j eir component UIDs.





- In order to execute the levels synchronou low a predetermined number of rounds for ea



- To be certain that all the computation for the round has completed, this number will be $O(n)$; note that $O(diameter)$ rounds are not always sufficient.
- Need to count this number of rounds is only reason that nodes need to know $n$.

## Complexity Analysis of SynGHS



- Note first that the number of nod $k$ component (with the possible exception of t st $2^k$.



- This can be shown by induction, using the fact that at each level, each component is combined with at least one other component at the same level.



- Therefore, the m $n$.



- Since each level takes time $O(n)$ e time complexity of SynchGHS is $O(n$



- The communication complexity is $O((n + |E|) \cdot \log n)$, since at each level, $O(n)$ messages are sent in total along all the tree edges, and $O(|E|)$ additional messages are required for finding the local minimum-weight edges.

## Issues: Asynchronous Case



- More details are needed for the as mmunication model.



  - It may be that some fragments (subtrees) are much larger than others, and because of that some nodes may need to wait for others, e.g., if node $u$ needs to find out whether neighbor $v$ e $b = (u, v)$.





- These details can b



  - We can bound the asynchronicity b hat nodes only start the new phase after the last phase is done, similarly to the phase-technique of Dijkstra's Algorithm.

- This gives rise to the idea of levels which will not be discussed any further.

## Issues



- The GHS algorithm is also kno re as BigMerge.



- The GHS algorithm can be appli ys.
- GHS for instance directly solves leader election in general graphs: The leader is simply the last surviving root!



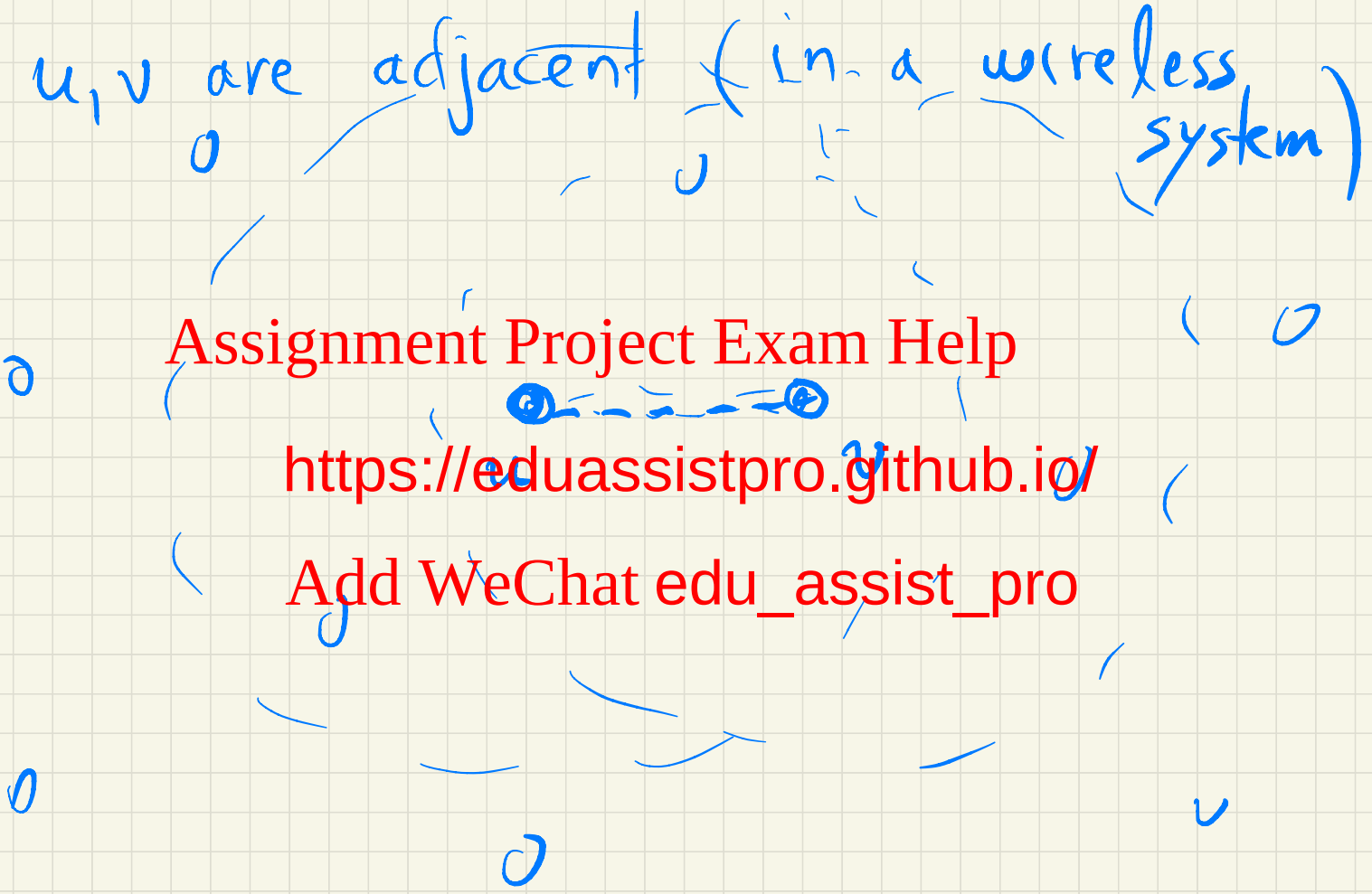
- GHS is distribute em in that the number o



  - In general, if you restrict the number o ot construct a spanning tree.



  - There exist constant round algorithms on geometric graphs that construct spanners with good spanning properties.



## Application



- Construct "local" planar spa ant stretch factor in Wireless Networks.

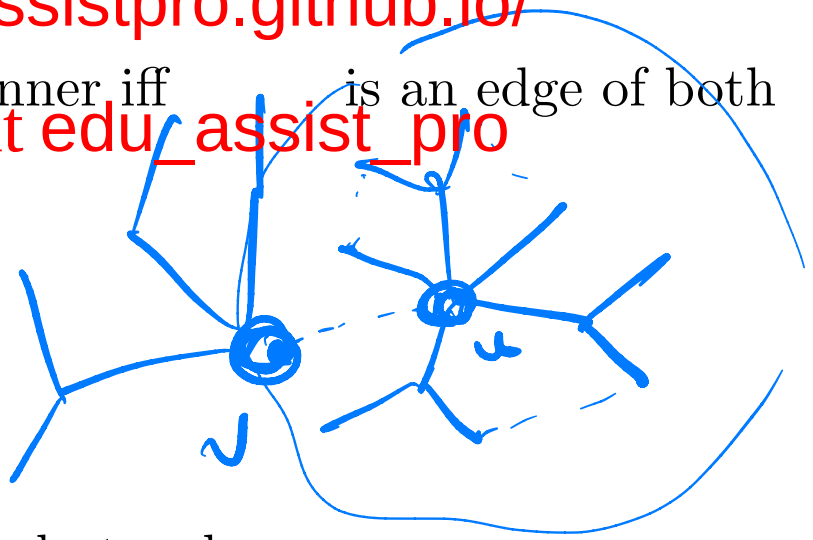


- Here is a simple algorithm.
  1. Each node $u$ finds its distance 2 neighborhood $N_2(u)$.
  2. Each node h distance wightss) of its distance 2 n
  3. $\{u, v\}$ is an edge of the spanner iff is an edge of both $T_u$ and $T_v$.










- The resulting spanner is
  1. planar,
  2. has small stretch factor,
  3. requires information from only two hops away.

## Exercises[a]



1. Consider the following traversa itiator sends out a token to discover the travers e the parent of a node as one from which the token is received for the first time. All other neighboring nodes will be called neighbors. By definition, the initiator does not have a parent. The following two rules define the







   (a) Send the token to

   (b) If Rule (1a) cannot be used to send the toke token to its parent.



   Show that when the token returns to the root, the entire graph has been traversed by proving the following two claims.

   (a) The token has a valid move until it returns to the root.

   (b) Eventually every node is visited by the token.

[a] No to hand in!

2. Let $G = (V, E)$ be a directed graph. A maximal strongly connected component of $G$ is a subgraph $G'$ such that 1) for every pair of vertices $u, v$ rected path from $u$ to $v$ and a directed path from nd 2) no other subgraph of $G$ has $G'$ as its subgraph. Propose a distributed algorithm to compute the maximal strongly connected component of a graph.



3. Propose an algorith e by restoring connectivity when a single no algorithm should complete the repair b st number of edges. Compute the time complexity of your algorithm.



4. In a spanning tree of a graph, there is exactly one path between any pair of nodes. If a spanning tree is used for broadcasting a message, and a process crashes, some nodes will not be able to receive the broadcast. Our goal is to improve the

connectivity of the subgraph used for broadcast, so that it can tolerate the crash of one process.



Given a connected graph, what minimal subgraph would you use for broadcasting, so that messages will reach every process even if one process fails? Suggest a distributed algorithm for constructing such a subgraph. Argue why your algorithm will work, and analyze its complexity.





5. The eccentricity of a vertex $v$ is the maximum distance from $v$ to any other vertex. The minimum eccentricity form the center.





   (a) Show that a tree can have at most two centers.

   (b) Design a distributed algorithm to find the center of a tree.

6. Given an undirected graph $G = (V, E)$, a matching $M$ is a subset of $E$, such that no two edges are incident on the same vertex. A matching $M$ is called maximal if there is no other

matching $M'$ such that $M \subset M'$. Suggest a distributed algorithm for computing a maximal matching. When the algorithm terminates, each n s matching neighbor, if such a match exists.



Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

## Sources




- R. Wattenhofer, Lecture Not f Distributed Computing, ETH Spring 201





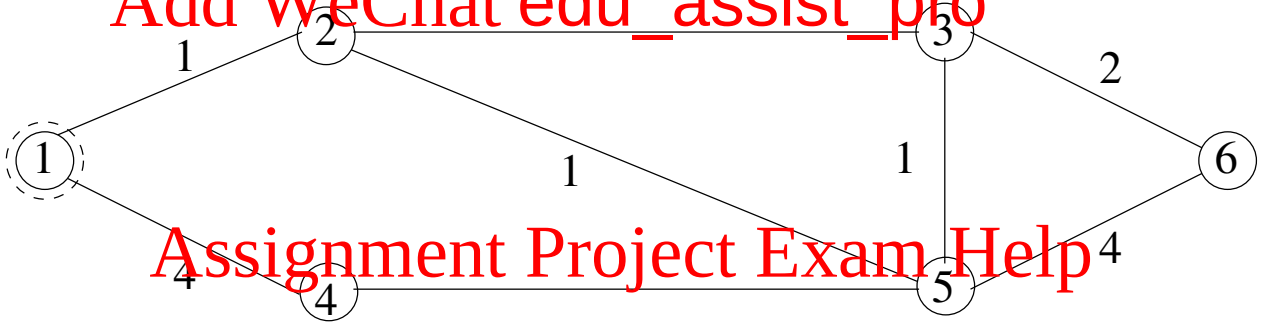
- N. Lynch, Distributed Algorithms, Morgan-Kaufmann, 1996.
- Gallager, R.G., Humblet, P.A., Spira, P.M., A distributed algorithm for mini M Trans. Prog. Lang. Syste

## Example: Dijkstra's Algorithm



Start node $s := 1$










**Iteration 1:**

Compute all costs to 1.



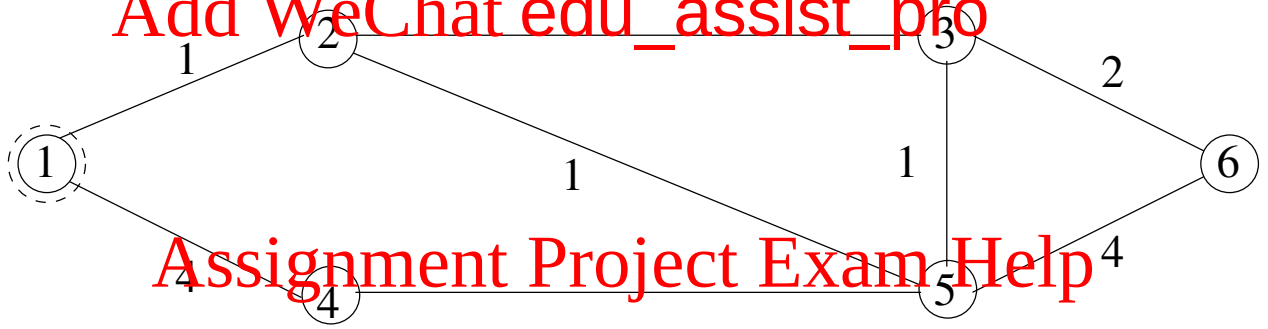

Update min cost routes to node 1.

## Example: Dijkstra's Algorithm



Start node is $s = 1$





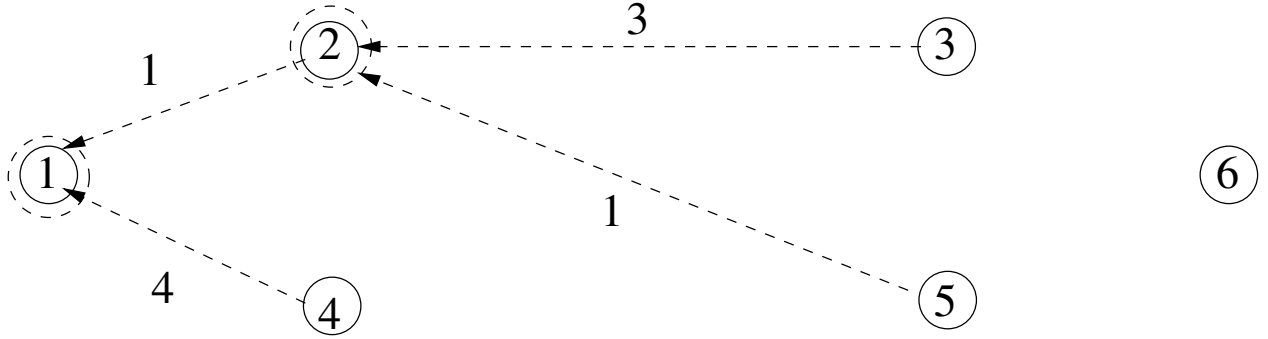



**Iteration 2:**



Add min cost node to $M$ (node 2).





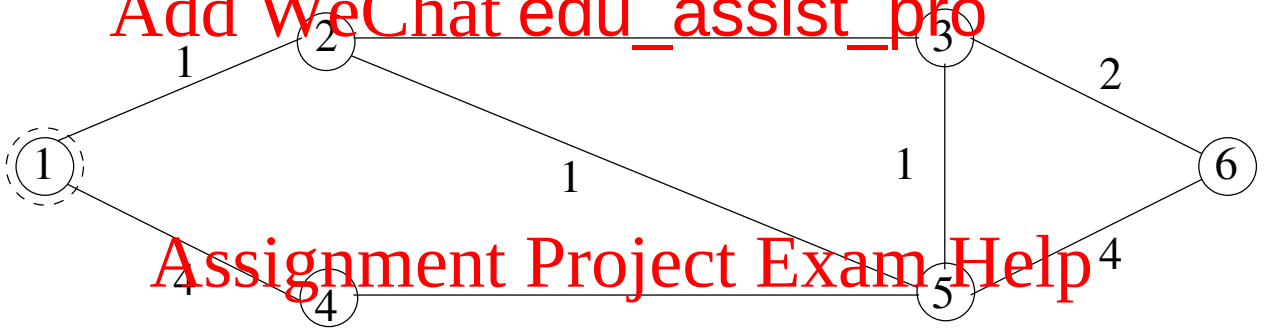

Update min cost routes to node 1.

## Example: Dijkstra's Algorithm



Every node executes Dijkstra's algorithm





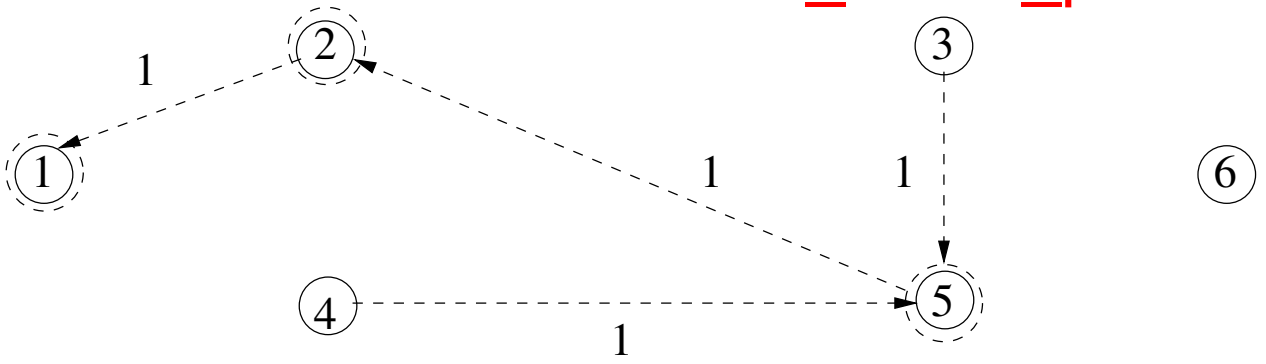



**Iteration 3:**



Add min cost node to $M$ (node 5).





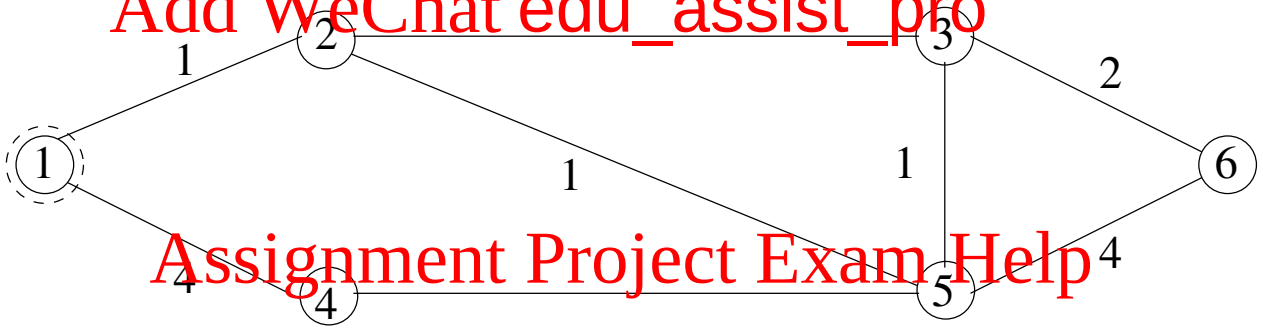

Update min cost routes to node 1.

## Example: Dijkstra's Algorithm



Every node executes Dijkstra's alg





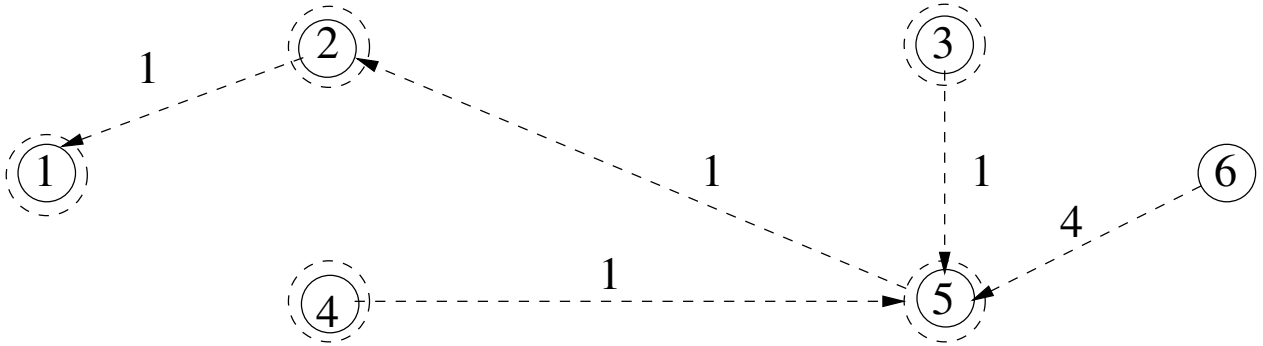



**Iterations 4:**



Add min cost node to $M$ (node 3).





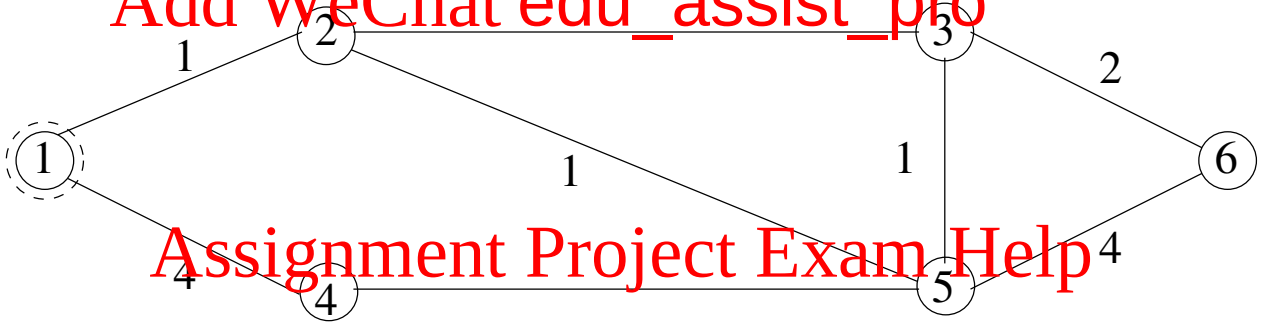

Update min cost routes to node 1.

## Example: Dijkstra's Algorithm



Every node executes Dijkstra's alg





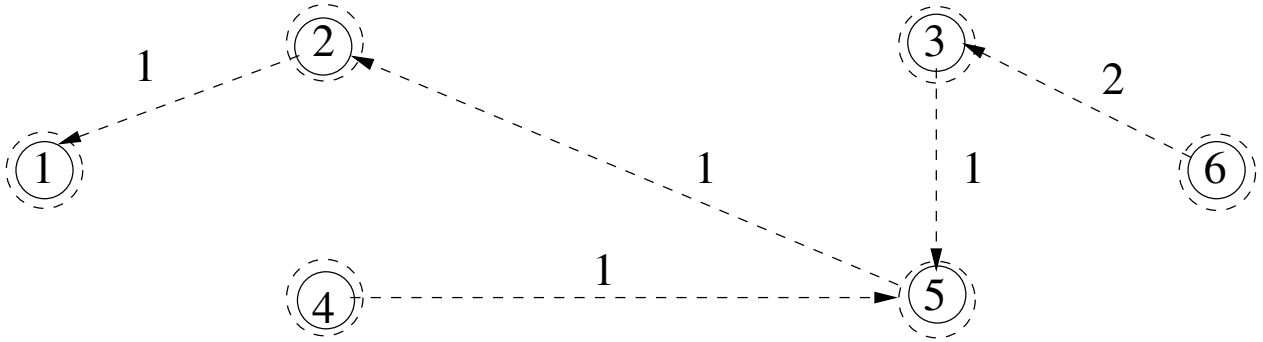



**Iteration 5:**



Add min cost node to $M$ (node 6).

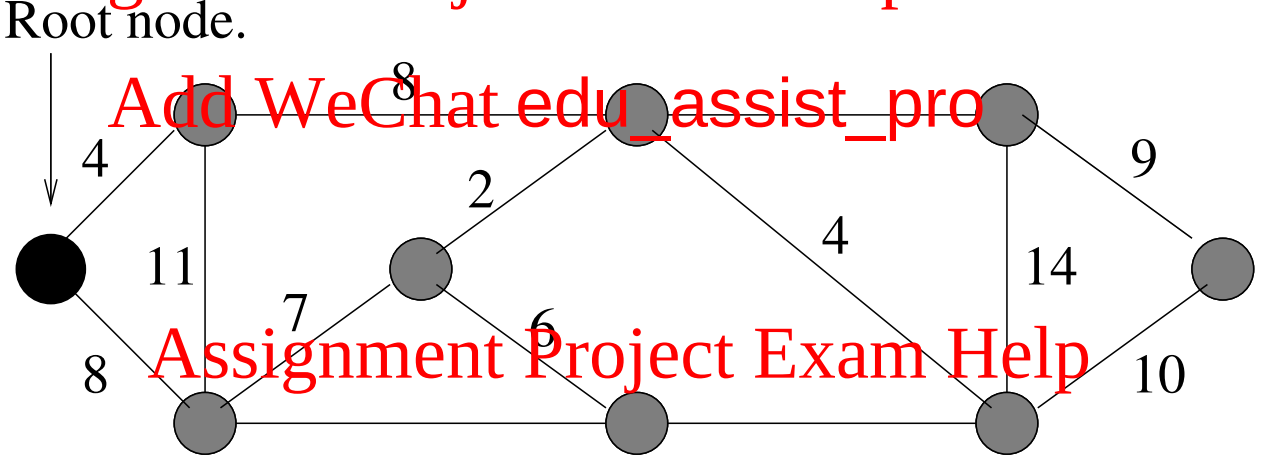




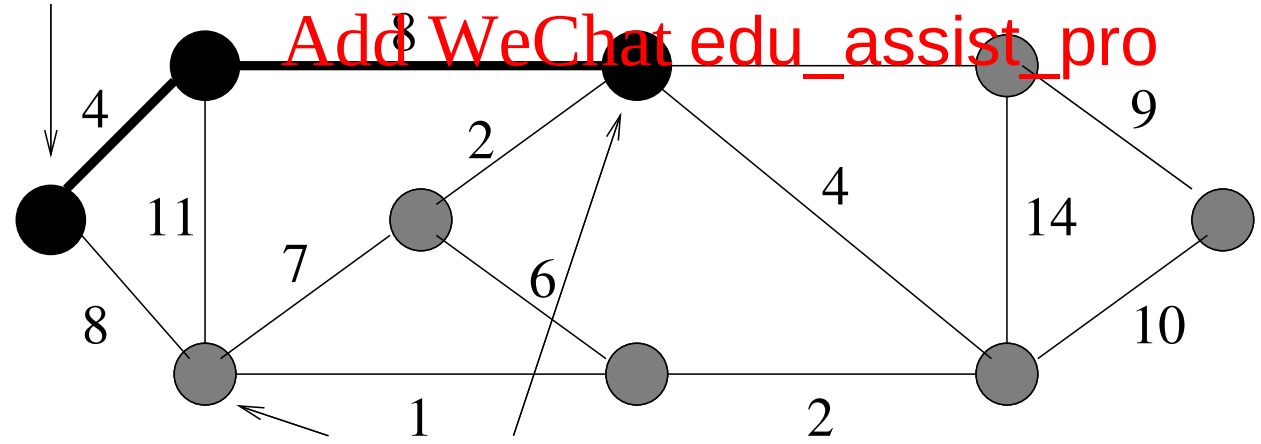

Update min cost routes to node 1.

## Example: Prim's Algorithm





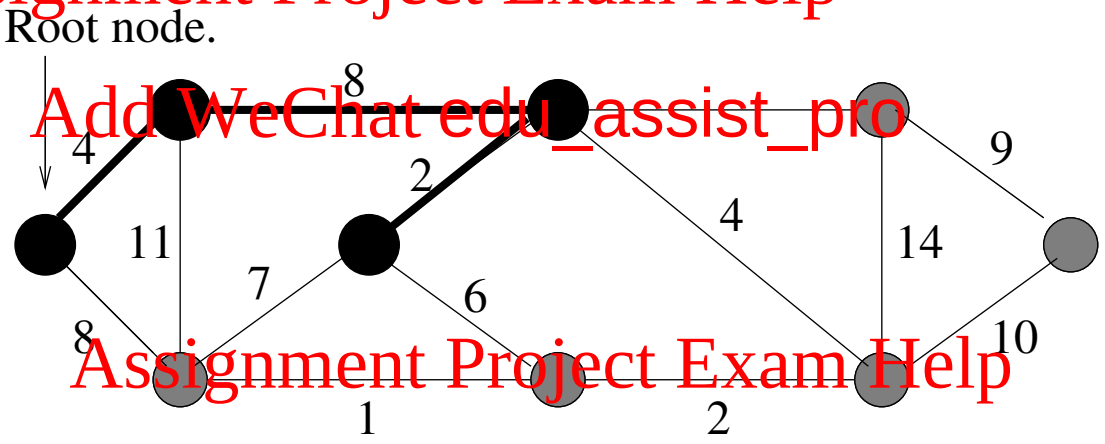


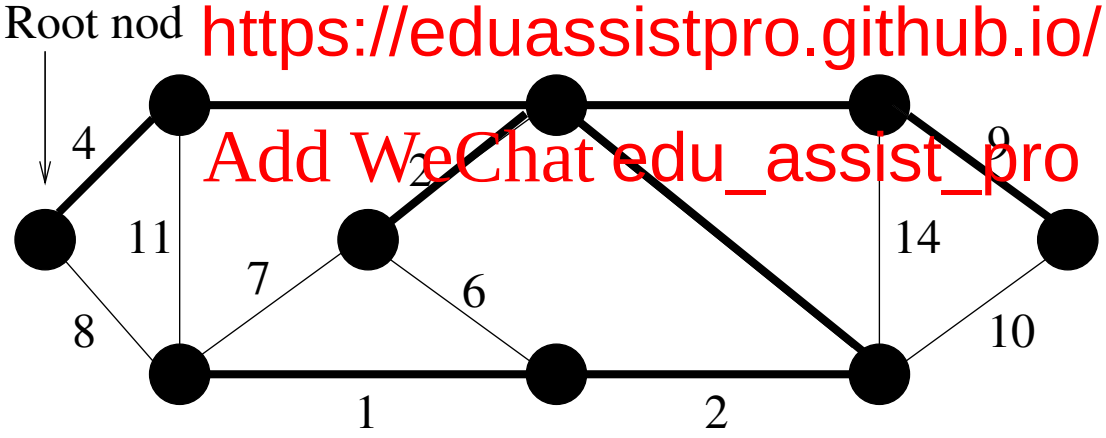

We have a choice: can add either of these two nodes.

## Example: Prim's Algorithm







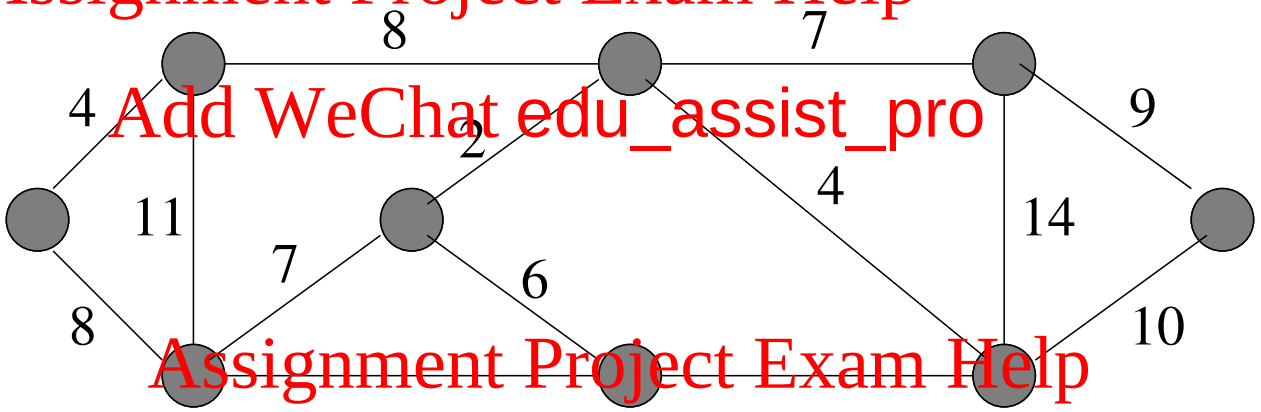





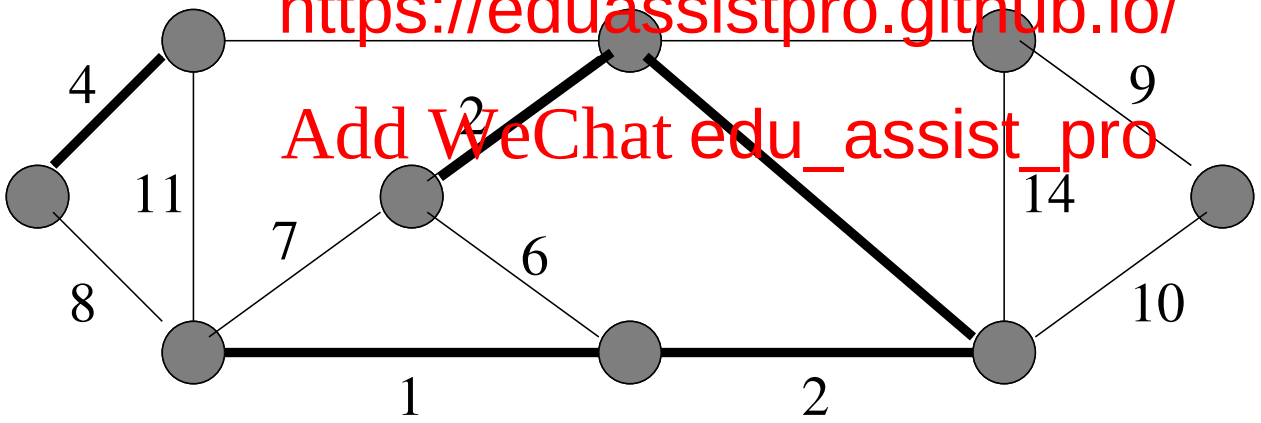

Then we add nodes adjacent to links
4. 2. 1. 7. 9 in this order

## Example: Kruskal's Algorithm





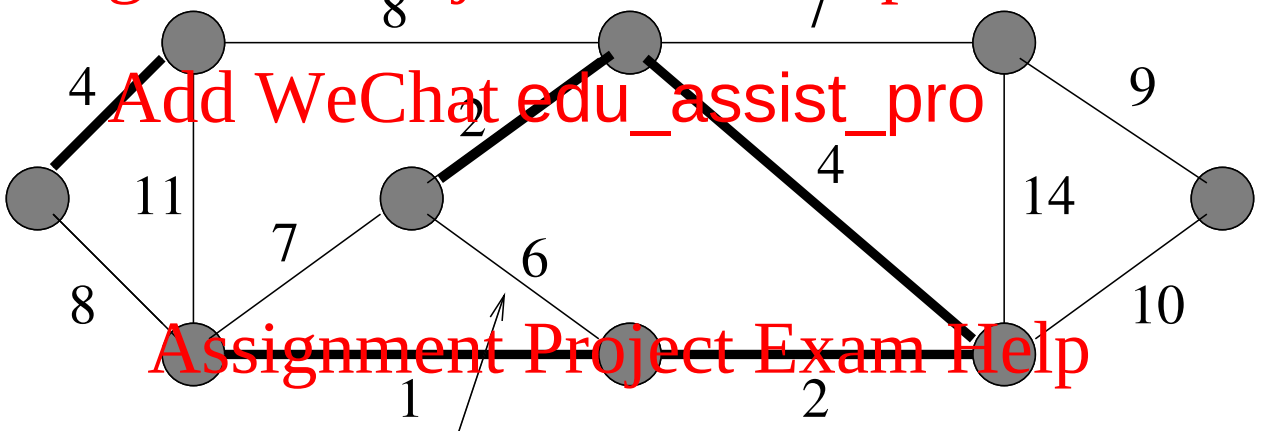



Links Added: 1. 2. 2. 4. 4

## Example: Kruskal's Algorithm

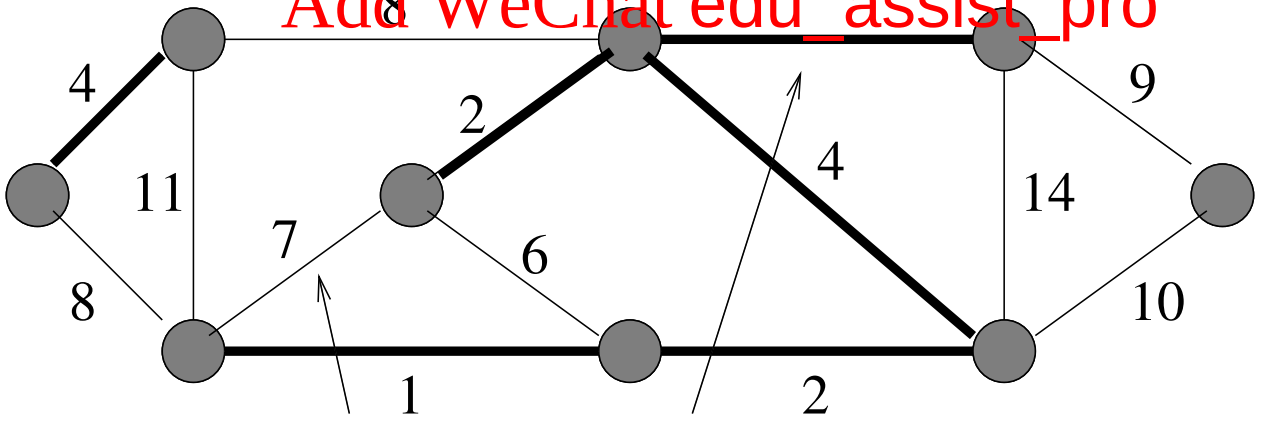







We c ycle

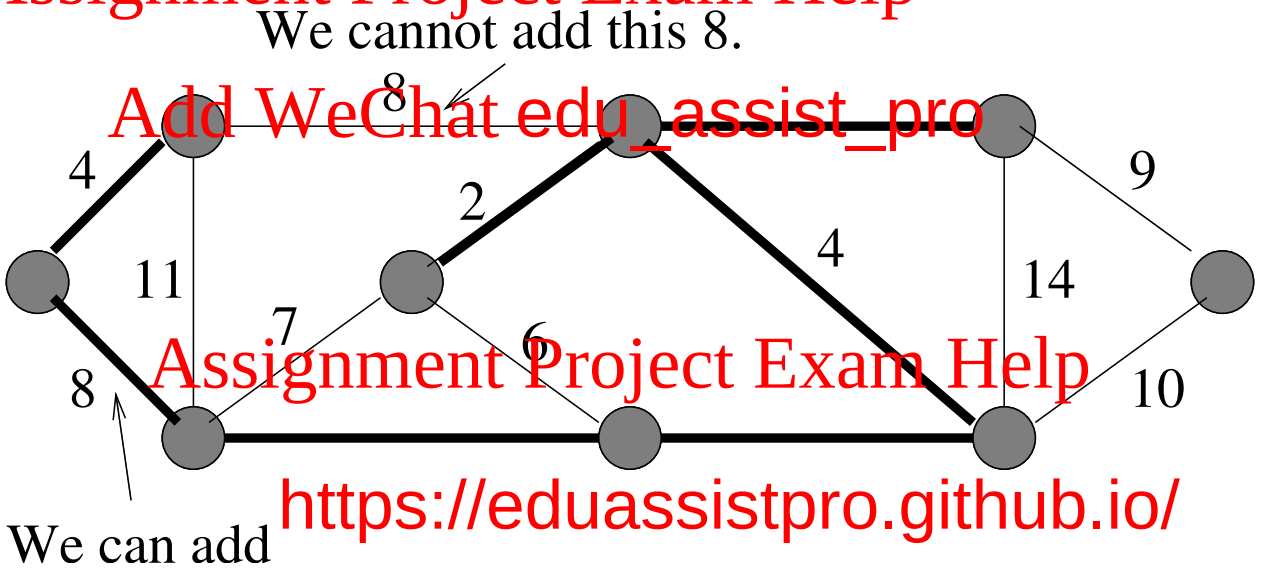





We can add the 7 that does not create a cycle

## Example: Kruskal's Algorithm





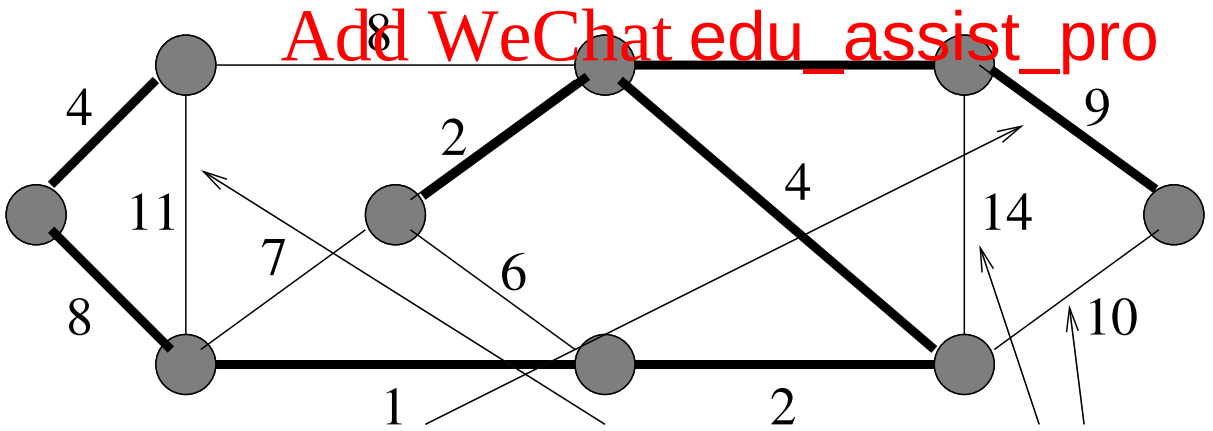





We can add








We add the 9. None of 10. 11. 14 can be added