Chapter 3: Greedy Algorithm Design Technique - II

Devesh C Jinwala, IIT Jammu, India

November 21, 2022

Design and Analysis of Algorithms
IIT Jammu, Jammu

1 Applications of MST approach

Application of MST approach : The K-clustering Problem

Issues with the MST algorithms...

- MST problem is only a particular formation of a broader network design goal

Issues with the MST algorithms...

- MST problem is only a particular formation of a broader network design goal
- it is only a good way to connect a set of sites by installing edges between them such that the TOTAL edge cost is the minimum

Issues with the MST algorithms...

- MST problem is only a particular formation of a broader network design goal
- it is only a good way to connect a set of sites by installing edges between them such that the TOTAL edge cost is the minimum
- But, does it ensure that

Issues with the MST algorithms...

- MST problem is only a particular formation of a broader network design goal
- it is only a good way to connect a set of sites by installing edges between them such that the TOTAL edge cost is the minimum
- But, does it ensure that
  - point-to-point distance between the edges it has, is also the minimum ?

Issues with the MST algorithms...

- MST problem is only a particular formation of a broader network design goal
- it is only a good way to connect a set of sites by installing edges between them such that the TOTAL edge cost is the minimum
- But, does it ensure that
  - point-to-point distance between the edges it has, is also the minimum ?
  - sometimes, one is willing to pay more for the total cost but reduce the cost between specific nodes

Issues with the MST algorithms...

- Alternately, does it ensure that

Issues with the MST algorithms...

- Alternately, does it ensure that
  - the congestion on the edges is textcolorredminimized

Issues with the MST algorithms...

- Alternately, does it ensure that
  - the congestion on the edges is textcolorredminimized
    - i.e. given traffic that needs to be routed between pairs of nodes, one desires a ST in which no single edge carries more than a certain amount of traffic

Issues with the MST algorithms...

- Alternately, does it ensure that
  - the congestion on the edges is textcolorredminimized
    - i.e. given traffic that needs to be routed between pairs of nodes, one desires a ST in which no single edge carries more than a certain amount of traffic
  - the network design problem is mitigated wholly with robustness. . . . . ? i.e. is the tree formed - robust against any failures ?

Issues with the MST algorithms...

- Alternately, does it ensure that
  - the congestion on the edges is textcolorredminimized
    - i.e. given traffic that needs to be routed between pairs of nodes, one desires a ST in which no single edge carries more than a certain amount of traffic
  - the network design problem is mitigated wholly with robustness. . . . . ? i.e. is the tree formed - robust against any failures ?
    - especially since a tree has the property that destroying any edge disconnects it

Issues with the MST algorithms...

- Alternately, does it ensure that
  - the congestion on the edges is textcolorredminimized
    - i.e. given traffic that needs to be routed between pairs of nodes, one desires a ST in which no single edge carries more than a certain amount of traffic
  - the network design problem is mitigated wholly with robustness. . . . . ? i.e. is the tree formed - robust against any failures ?
    - especially since a tree has the property that destroying any edge disconnects it
  - what if network resilience is the explicit goal at the same time i.e.

Issues with the MST algorithms...

- Alternately, does it ensure that
  - the congestion on the edges is textcolorredminimized
    - i.e. given traffic that needs to be routed between pairs of nodes, one desires a ST in which no single edge carries more than a certain amount of traffic
  - the network design problem is mitigated wholly with robustness. . . . . ? i.e. is the tree formed - robust against any failures ?
    - especially since a tree has the property that destroying any edge disconnects it
  - what if network resilience is the explicit goal at the same time i.e.
    - seeking the cheapest connected network on the set of sites that remain connected even after deletion of one site ?

Issues with the MST algorithms...

- Alternately, does it ensure that
  - the congestion on the edges is textcolorredminimized
    - i.e. given traffic that needs to be routed between pairs of nodes, one desires a ST in which no single edge carries more than a certain amount of traffic
  - the network design problem is mitigated wholly with robustness. . . . . ? i.e. is the tree formed - robust against any failures ?
    - especially since a tree has the property that destroying any edge disconnects it
  - what if network resilience is the explicit goal at the same time i.e.
    - seeking the cheapest connected network on the set of sites that remain connected even after deletion of one site ?

Issues with the MST algorithms...

- Alternately, does it ensure that
  - the congestion on the edges is textcolorredminimized
    - i.e. given traffic that needs to be routed between pairs of nodes, one desires a ST in which no single edge carries more than a certain amount of traffic
  - the network design problem is mitigated wholly with robustness. . . . . ? i.e. is the tree formed - robust against any failures ?
    - especially since a tree has the property that destroying any edge disconnects it
  - what if network resilience is the explicit goal at the same time i.e.
    - seeking the cheapest connected network on the set of sites that remain connected even after deletion of one site ?

All these lead to various variations. . . . . .

The K-Clustering Problem

- arises when a collection of objects are to be classified or organized into coherent groups

The K-Clustering Problem

- arises when a collection of objects are to be classified or organized into coherent groups
- e.g. it is natural to look for how similar or dissimilar each pair of objects is

The K-Clustering Problem

- arises when a collection of objects are to be classified or organized into coherent groups
- e.g. it is natural to look for how similar or dissimilar each pair of objects is
- that is to define a distance function, which is such that....

The K-Clustering Problem

- arises when a collection of objects are to be classified or organized into coherent groups
- e.g. it is natural to look for how similar or dissimilar each pair of objects is
- that is to define a distance function, which is such that....
  - objects at a larger distance from each other are less similar to each other; as compared to the objects closer to each other - that are more similar

The K-Clustering Problem

- arises when a collection of objects are to be classified or organized into coherent groups
- e.g. it is natural to look for how similar or dissimilar each pair of objects is
- that is to define a distance function, which is such that....
  - objects at a larger distance from each other are less similar to each other; as compared to the objects closer to each other - that are more similar
- So, here the objective function for clustering is *maximizing spacing*.

The K-Clustering Problem

- arises when a collection of objects are to be classified or organized into coherent groups
- e.g. it is natural to look for how similar or dissimilar each pair of objects is
- that is to define a distance function, which is such that....
  - objects at a larger distance from each other are less similar to each other; as compared to the objects closer to each other - that are more similar
- So, here the objective function for clustering is *maximizing spacing*.
  - the spacing of a set of clusters as the distance between the closest pair of points in different clusters.

The K-Clustering Problem

- arises when a collection of objects are to be classified or organized into coherent groups
- e.g. it is natural to look for how similar or dissimilar each pair of objects is
- that is to define a distance function, which is such that....
  - objects at a larger distance from each other are less similar to each other; as compared to the objects closer to each other - that are more similar
- So, here the objective function for clustering is *maximizing spacing*.
  - the spacing of a set of clusters as the distance between the closest pair of points in different clusters.
- For a good clustering, one expects the spacing to be large;

The K-Clustering Problem

- arises when a collection of objects are to be classified or organized into coherent groups
- e.g. it is natural to look for how similar or dissimilar each pair of objects is
- that is to define a distance function, which is such that....
  - objects at a larger distance from each other are less similar to each other; as compared to the objects closer to each other - that are more similar
- So, here the objective function for clustering is *maximizing spacing*.
  - the spacing of a set of clusters as the distance between the closest pair of points in different clusters.
- For a good clustering, one expects the spacing to be large;
- hence, maximizing the spacing seems like a reasonable objective function for clustering.

The K-Clustering Problem...

- The distance may not be in terms of physical distance
- distance may take more abstract meaning e.g.
  - distance between two species - to be the no of years in the course of the evolution
  - distance between two images – number of corresponding pixels at which their intensity values differ by at least some threshold
- Thus, Given a distance function on the objects divide the objects into groups so that intuitively,
  - objects within the same group are close whereas
  - objects within the different groups are far apart.

The k-clustering problem: Formal definition

- Formally given a set U of n objects, labeled $p_1, p_2, p_3, p_4 \ldots p_n$ for each pair p_i and p_j, we have a numerical distance with the properties that $d(p_i, p_j)$ with $d(p_i, p_i)$=0 and $d(p_i, p_j) > 0$.

The k-clustering problem: Formal definition

- Formally given a set U of n objects, labeled $p_1, p_2, p_3, p_4 \ldots p_n$ for each pair p_i and p_j, we have a numerical distance with the properties that $d(p_i, p_j)$ with $d(p_i, p_i)$=0 and $d(p_i, p_j) > 0$.
- Also, $d(p_i, p_j) = d(p_j, p_i)$

The k-clustering problem: Formal definition

- Formally given a set U of n objects, labeled $p_1, p_2, p_3, p_4 \ldots p_n$ for each pair p_i and p_j, we have a numerical distance with the properties that $d(p_i, p_j)$ with $d(p_i, p_i)$=0 and $d(p_i, p_j) > 0$.
- Also, $d(p_i, p_j) = d(p_j, p_i)$
- Given a parameter k, a k-clustering of U is a partition of U into k nonempty sets $C_1,\ C_2,\ C_3,\ C_4 \ldots\ldots C_k$

The k-clustering problem: Formal definition

- Formally given a set U of n objects, labeled $p_1, p_2, p_3, p_4 \ldots p_n$ for each pair p_i and p_j, we have a numerical distance with the properties that $d(p_i, p_j)$ with $d(p_i, p_i)$=0 and $d(p_i, p_j) > 0$.
- Also, $d(p_i, p_j) = d(p_j, p_i)$
- Given a parameter k, a k-clustering of U is a partition of U into k nonempty sets $C_1,\ C_2,\ C_3,\ C_4 \ldots\ldots C_k$
- Spacing of a k-clustering is defined as the minimum distance between any two points lying in different clusters.

The k-clustering problem: Applications

- Clustering problems are widespread in many different domains.
- Businesses often want to cluster their customers into different groups in order to better tailor their marketing to specific types of people (market segmentation).
- News aggregation websites like Google Maps, cluster news stories from different sources into groups of stories that are all on the same basic topic.
- Clustering can be use to find epicenters of disease outbreak (such as the the COVID 2019 Or the Swine Fu epidemic).

The K-Clustering Problem

- Given that points in different clusters should be far apart from each other, a natural goal is to seek a k-clustering with the maximum possible spacing.

The K-Clustering Problem

- Given that points in different clusters should be far apart from each other, a natural goal is to seek a k-clustering with the maximum possible spacing.
- How many k-clustering of a set u can be there ?

The K-Clustering Problem

- Given that points in different clusters should be far apart from each other, a natural goal is to seek a k-clustering with the maximum possible spacing.
- How many k-clustering of a set u can be there ?
- exponentially many

The K-Clustering Problem

- Given that points in different clusters should be far apart from each other, a natural goal is to seek a k-clustering with the maximum possible spacing.
- How many k-clustering of a set u can be there ?
- exponentially many
- finding out then one with maximum spacing. How ?

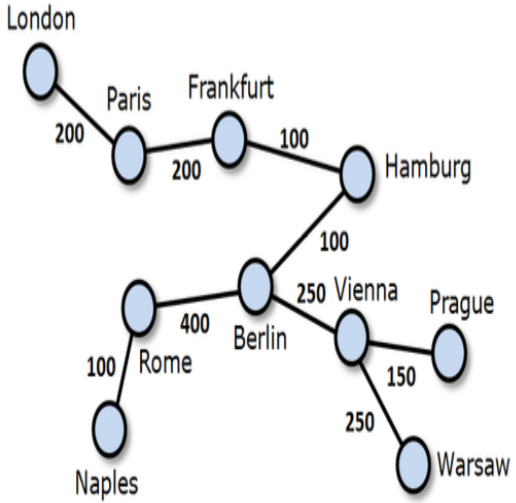
The Algorithm Approach

- consider growing a graph on the vertex set U.
- connected components become the clusters.
  - bring the nearby points into the same cluster.
- draw an edge between the closest pair of points
  - add edges between pairs of points in order of increasing distance $d(p_i, p_j)$.
  - if p_i and p_j belong to the same cluster, do not add that edge.
- How is this related to Kruskal's MST ?

The Algorithm Approach...

- Follow the same method as Kruskal's algorithm, but stop when obtaining k-clustering.
- How to stop at having k connected components ?
- stop just before it adds the last k – 1 edges.
- Run Kruskal's MST on the given graph fully and
  - then delete the k-1 most expensive edges
  - define the resulting structure to be k connected components.
  - an example of 3-clustering

Examples of 2-clustering, 3-clustering



Proving the algorithm

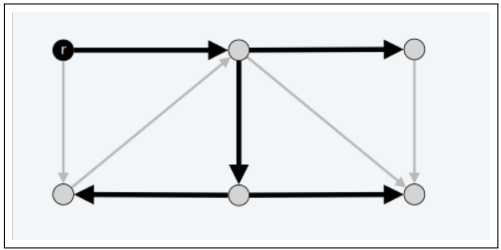
- Prove that the algorithm as outlined before actually works.

Application of MST approach : The Minimum Cost Arborescence

Arborescence

What is an arborescence ?

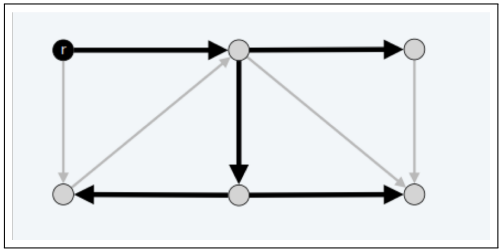
- An arborescence is a directed graph in which, for a vertex v called the root and any other vertex u, there is exactly one directed path from v to u i.e. an arborescence is a directed, rooted tree in which all edges point away from the root.



Arborescence

What is an arborescence ?

- An arborescence is a directed graph in which, for a vertex v called the root and any other vertex u, there is exactly one directed path from v to u i.e. an arborescence is a directed, rooted tree in which all edges point away from the root.

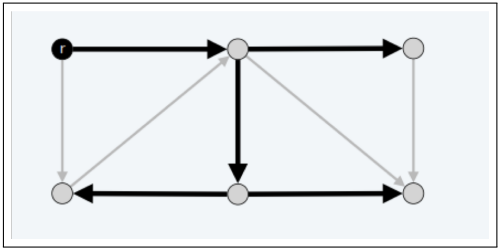


Note that every arborescence is a directed acyclic graph (DAG), but not every DAG is an arborescence.

Arborescence...

Formal definition

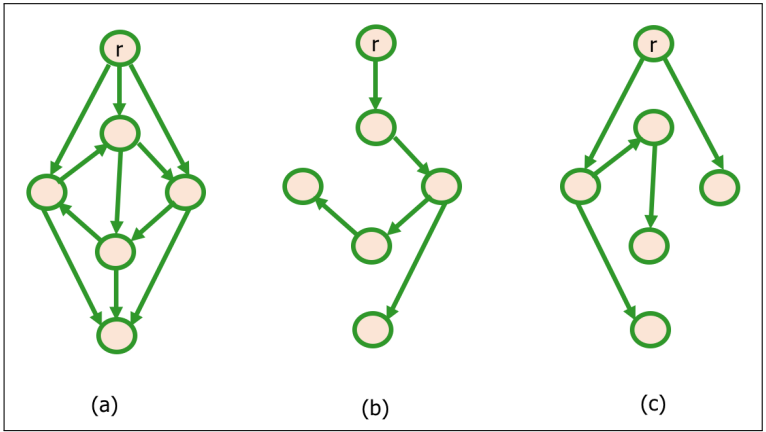
- Given a digraph G = (V, E) and a root r ϵ V, an arborescence (rooted at r) is a subgraph T = (V, F) such that T is a spanning tree of G if we ignore the direction of edges. There is a directed path in T from r to each other node v ϵ V.



Thus, every arborescence is a rooted directed acyclic graph in which the path from the root to any other vertex is unique.

Arborescence...

For the given graph in (a), the two arborescenes are shown in (b) and in (c)



Minimum Cost Arborescence

The Problem Statement

- Given a directed graph G = (V, E) with a distinguished root node r and with a non-negative edge weight $c_e \geq 0$ on each edge, the goal is to compute the optimal (minimum) cost arborescence rooted at r, of the graph.

Theorem: A graph has an arborescence rooted at r if and only if there is a directed path from the root node r to every other node in the graph.

Minimum Cost Arborescence

The Problem Statement

- Given a directed graph G = (V, E) with a distinguished root node r and with a non-negative edge weight $c_e \geq 0$ on each edge, the goal is to compute the optimal (minimum) cost arborescence rooted at r, of the graph.
- Assumption: We assume that there is a directed path from the root node r to every other node in the graph.

Theorem: A graph has an arborescence rooted at r if and only if there is a directed path from the root node r to every other node in the graph.

Minimum Cost Arborescence

The Problem Statement

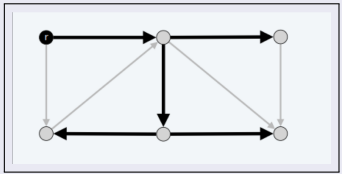
- Given a directed graph G = (V, E) with a distinguished root node r and with a non-negative edge weight $c_e \geq 0$ on each edge, the goal is to compute the optimal (minimum) cost arborescence rooted at r, of the graph.
- Assumption: We assume that there is a directed path from the root node r to every other node in the graph.
- Algorithm approach. BFS or DFS from r is an arborescence (iff all nodes reachable).

Theorem: A graph has an arborescence rooted at r if and only if there is a directed path from the root node r to every other node in the graph.

Minimum Cost Arborescence...

The Problem Statement

- Given a digraph G with a root node r and with a nonnegative cost $c_e \geq 0$ on each edge e, compute an arborescence rooted at r of minimum cost.

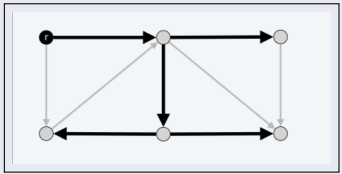


Theorem: A graph has an arborescence rooted at r if and only if there is a directed path from the root node r to every other node in the graph.

Minimum Cost Arborescence...

The Problem Statement

- Given a digraph G with a root node r and with a nonnegative cost $c_e \geq 0$ on each edge e, compute an arborescence rooted at r of minimum cost.



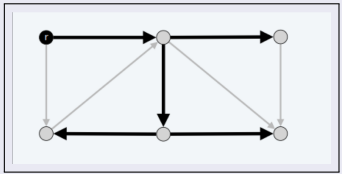
- Assumption 1. G has an arborescence rooted at r.

Theorem: A graph has an arborescence rooted at r if and only if there is a directed path from the root node r to every other node in the graph.

Minimum Cost Arborescence...

The Problem Statement

- Given a digraph G with a root node r and with a nonnegative cost $c_e \geq 0$ on each edge e, compute an arborescence rooted at r of minimum cost.



- Assumption 1. G has an arborescence rooted at r.
- Assumption 2. No edge enters r (safe to delete since they won't help).

Theorem: A graph has an arborescence rooted at r if and only if there is a directed path from the root node r to every other node in the graph.

A Proposition

Claim of Proposition: A subgraph T = (V, F) of G is an arborescence rooted at r iff T has no directed cycles and each node v$\neq$ r has exactly one entering edge.

- Claim#1: If a subgraph T = (V, F) of G is an arborescence rooted at r if T has no directed cycles and each node v$\neq$ r has exactly one entering edge.

A Proposition

Claim of Proposition: A subgraph T = (V, F) of G is an arborescence rooted at r iff T has no directed cycles and each node v$\neq$ r has exactly one entering edge.

- Claim#1: If a subgraph T = (V, F) of G is an arborescence rooted at r if T has no directed cycles and each node v$\neq$ r has exactly one entering edge.
- Claim#2: If a subgraph T = (V, F) of G has no directed cycles and each node v$\neq$ r has exactly one entering edge then it is an arborescence rooted at r.

A Proposition

Claim of Proposition: A subgraph T = (V, F) of G is an arborescence rooted at r iff T has no directed cycles and each node v$\neq$ r has exactly one entering edge.

- Claim#1: If a subgraph T = (V, F) of G is an arborescence rooted at r if T has no directed cycles and each node v$\neq$ r has exactly one entering edge.
- Claim#2: If a subgraph T = (V, F) of G has no directed cycles and each node v$\neq$ r has exactly one entering edge then it is an arborescence rooted at r.
- Proofs of Claim#1 and Claim#2:

Motivating Applications

- In a variety of contexts, a group of users may be jointly responsible for sharing the total cost of a "joint project".

Motivating Applications

- In a variety of contexts, a group of users may be jointly responsible for sharing the total cost of a "joint project".
  - But the issue is how to allocate the total cost to the individual agents ? There is no appropriate "market mechanism" which can allocate the total cost to the individual agents.

Motivating Applications

- In a variety of contexts, a group of users may be jointly responsible for sharing the total cost of a "joint project".
  - But the issue is how to allocate the total cost to the individual agents ? There is no appropriate "market mechanism" which can allocate the total cost to the individual agents.
  - A large number of practical problems exists wherein it is essential to identify the agents with nodes in a graph.

Motivating Applications

- In a variety of contexts, a group of users may be jointly responsible for sharing the total cost of a "joint project".
  - But the issue is how to allocate the total cost to the individual agents ? There is no appropriate "market mechanism" which can allocate the total cost to the individual agents.
  - A large number of practical problems exists wherein it is essential to identify the agents with nodes in a graph.
- Some applications where such a scenario exists are as follows:

Motivating Applications

- In a variety of contexts, a group of users may be jointly responsible for sharing the total cost of a "joint project".
  - But the issue is how to allocate the total cost to the individual agents ? There is no appropriate "market mechanism" which can allocate the total cost to the individual agents.
  - A large number of practical problems exists wherein it is essential to identify the agents with nodes in a graph.
- Some applications where such a scenario exists are as follows:
  - Multicast routing with a directed network connecting the source to all receivers

Motivating Applications

- In a variety of contexts, a group of users may be jointly responsible for sharing the total cost of a "joint project".
  - But the issue is how to allocate the total cost to the individual agents ? There is no appropriate "market mechanism" which can allocate the total cost to the individual agents.
  - A large number of practical problems exists wherein it is essential to identify the agents with nodes in a graph.
- Some applications where such a scenario exists are as follows:
  - Multicast routing with a directed network connecting the source to all receivers
  - Villages and an irrigation system which draws water from a dam, and have to share the cost of the distribution network.

Motivating Applications

- In a variety of contexts, a group of users may be jointly responsible for sharing the total cost of a "joint project".
  - But the issue is how to allocate the total cost to the individual agents ? There is no appropriate "market mechanism" which can allocate the total cost to the individual agents.
  - A large number of practical problems exists wherein it is essential to identify the agents with nodes in a graph.
- Some applications where such a scenario exists are as follows:
  - Multicast routing with a directed network connecting the source to all receivers
  - Villages and an irrigation system which draws water from a dam, and have to share the cost of the distribution network.
  - A capacity synthesis problem..... in which the agents may share a network for bilateral exchange of information, or for transportation of goods between nodes.

Finding Arborescences

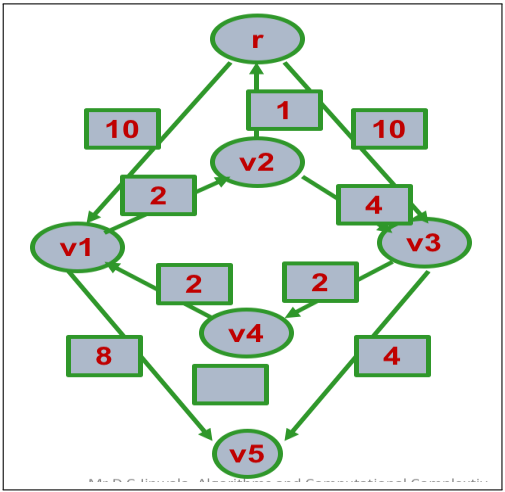
- A subgraph T= (V, F) for a graph G=(V, E) is an arborescence with respect to root r, if and only if T has NO cycles and for each node v $\neq$ r, there is exactly one edge in F that enters v.
- Just as every connected graph has a ST, every directed graph has a arborescence rooted at r, provided that r can reach every node.

Minimum Cost Arborescence

- The problem therefore, is to compute the minimum-cost arborescence of a directed graph.
- This problem is similar to the analogue of the minimum-cost spanning tree problem for directed graphs.
- But raises various issues...
  - must the minimum cost arborescence contain the cheapest edge in the whole graph?
  - can we safely delete the most expensive edge in a cycle, confident that it cannot be in the optimal arborescence?
  - The counter examples are vital. . . . . .

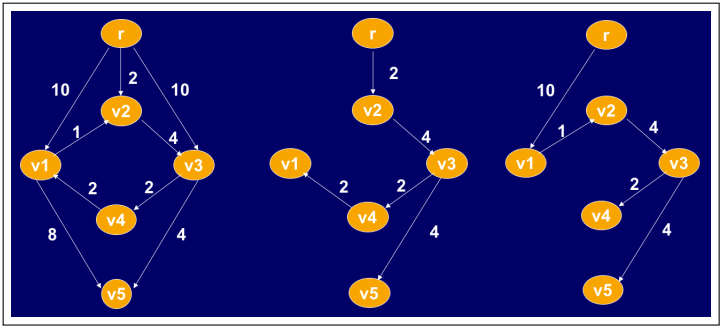
Counterexample1

- Must the minimum cost arborescence contain the cheapest edge in the whole graph?
- What is the cheapest edge ?
- Can it belong to optimal arborescence ?



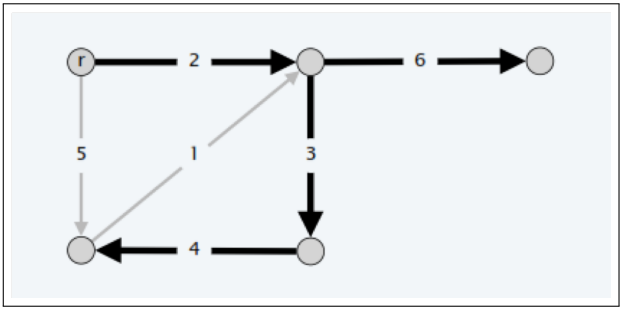
Counterexample2

- Even though some arborescences do contain the cheapest edge e in G rooted at r, it need not belong to the optimal arborescence.



Observation and Inferences

- A min-cost arborescence need not
  - be a shortest-paths tree.
  - include the cheapest edge (in some cut).
  - exclude the most expensive edge (in some cycle).



Tutorial Exercise

Problem definition

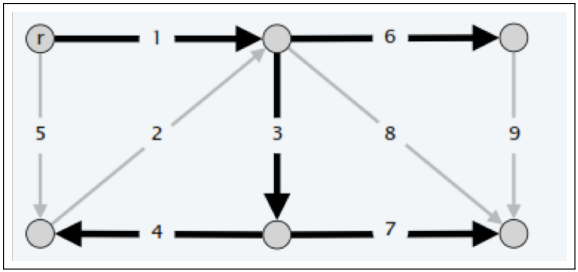
Think of a counterexample each to show that the Kruskal's algorithm and the Prim's algorithm do not work to solve the problem.

Probable Approach

- Property. For each node v $\neq$ r, choose one cheapest edge entering v and let F* denote this set of n - 1 edges. If (V, F*) is an arborescence, then it is a min-cost arborescence.

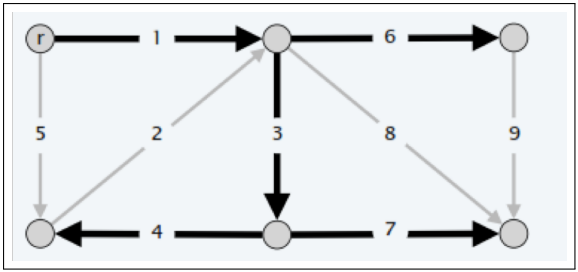
Probable Approach

- Property. For each node v $\neq$ r, choose one cheapest edge entering v and let F* denote this set of n - 1 edges. If (V, F*) is an arborescence, then it is a min-cost arborescence.
- How to argue to prove?



Probable Approach

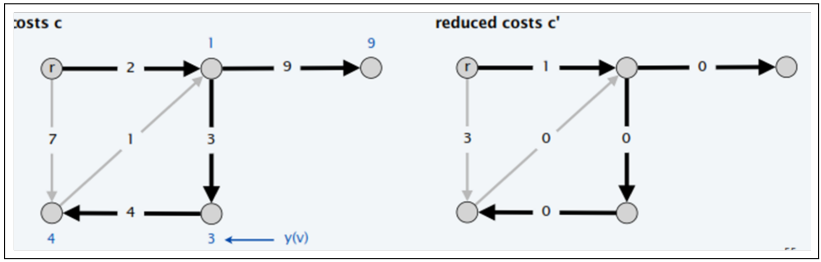
- Property. For each node v $\neq$ r, choose one cheapest edge entering v and let F* denote this set of n - 1 edges. If (V, F*) is an arborescence, then it is a min-cost arborescence.
- How to argue to prove?



- Proof: An arborescence needs exactly one edge entering each node v $\neq$ r and (V, F*) is the cheapest way to make these choices.

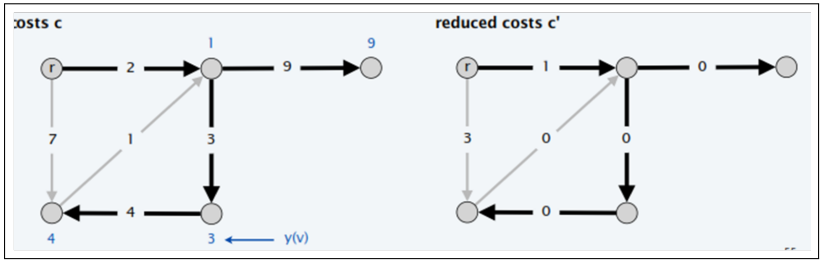
Reduced Costs

- For each v $\neq$ r, let y(v) denote the min cost of any edge entering v. The reduced cost of an edge (u, v) is c'(u, v) = c(u, v) - y(v) $\geq$ 0.



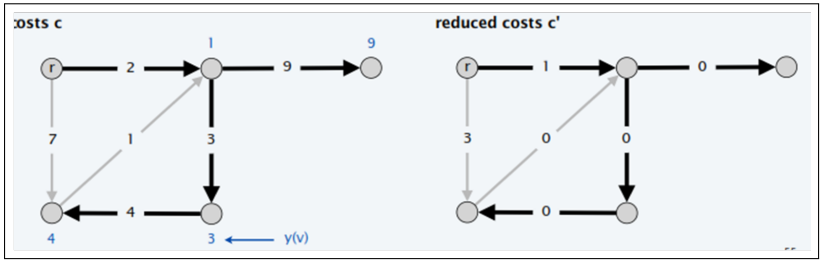
Reduced Costs

- For each v $\neq$ r, let y(v) denote the min cost of any edge entering v. The reduced cost of an edge (u, v) is c'(u, v) = c(u, v) - y(v) $\geq$ 0.
- Observation. T is a min-cost arborescence in G using costs c iff T is a min-cost arborescence in G using reduced costs c'.



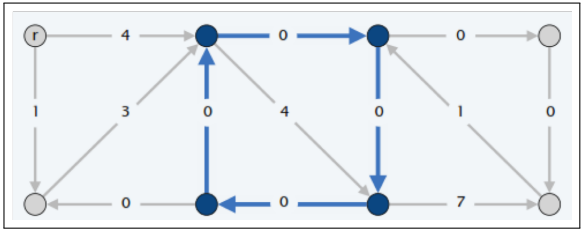
Reduced Costs

- For each v $\neq$ r, let y(v) denote the min cost of any edge entering v. The reduced cost of an edge (u, v) is c'(u, v) = c(u, v) - y(v) $\geq$ 0.
- Observation. T is a min-cost arborescence in G using costs c iff T is a min-cost arborescence in G using reduced costs c'.
- Proof: Each arborescence has exactly one edge entering v.



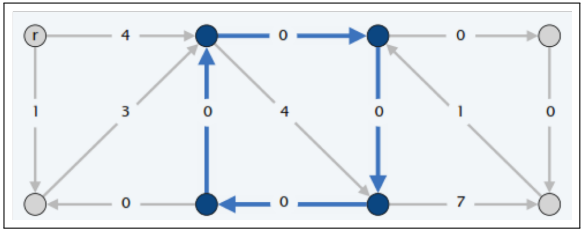
Edmonds Branching Algorithm

- Intuition : Recall F* = set of cheapest edges entering v for each v $\neq$ r.



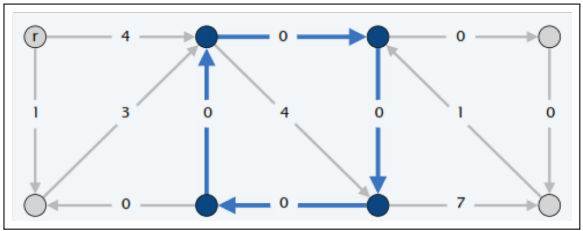
Edmonds Branching Algorithm

- Intuition : Recall F* = set of cheapest edges entering v for each v $\neq$ r.
- Now, all edges in F* have 0 cost with respect to costs c'(u, v).



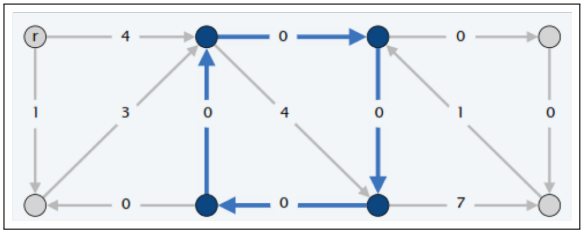
Edmonds Branching Algorithm

- Intuition : Recall F* = set of cheapest edges entering v for each v $\neq$ r.
- Now, all edges in F* have 0 cost with respect to costs c'(u, v).
- If F* does not contain a cycle, then it is a min-cost arborescence.



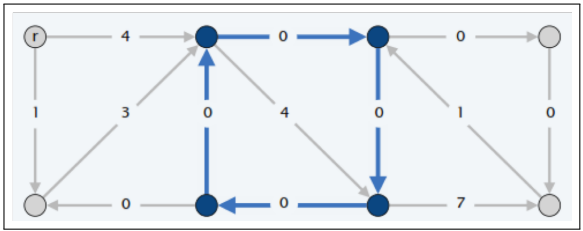
Edmonds Branching Algorithm

- Intuition : Recall F* = set of cheapest edges entering v for each v $\neq$ r.
- Now, all edges in F* have 0 cost with respect to costs c'(u, v).
- If F* does not contain a cycle, then it is a min-cost arborescence.
- If F* contains a cycle C, can afford to use as many edges in C as desired.



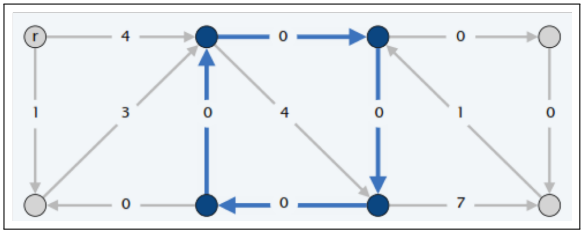
Edmonds Branching Algorithm

- Intuition : Recall F* = set of cheapest edges entering v for each v $\neq$ r.
- Now, all edges in F* have 0 cost with respect to costs c'(u, v).
- If F* does not contain a cycle, then it is a min-cost arborescence.
- If F* contains a cycle C, can afford to use as many edges in C as desired.
- Contract nodes in C to a supernode.



Edmonds Branching Algorithm

- Intuition : Recall F* = set of cheapest edges entering v for each v $\neq$ r.
- Now, all edges in F* have 0 cost with respect to costs c'(u, v).
- If F* does not contain a cycle, then it is a min-cost arborescence.
- If F* contains a cycle C, can afford to use as many edges in C as desired.
- Contract nodes in C to a supernode.
- Recursively solve problem in contracted network G' with costs c'(u, v)



Edmonds Branching Algorithm...

EDMONDSBRANCHING(G, r, c)

FOREACH $v \neq r$

$y(v) \leftarrow$ min cost of an edge entering v.

$c'(u, v) \leftarrow c'(u, v) - y(v)$ for each edge (u, v) entering v.

FOREACH $v \neq r$: choose one 0-cost edge entering v and let F^* be the resulting set of edges.

IF F^* forms an arborescence, RETURN $T = (V, F^*)$.

ELSE

$C \leftarrow$ directed cycle in F^*.

Contract C to a single supernode, yielding $G' = (V', E')$.

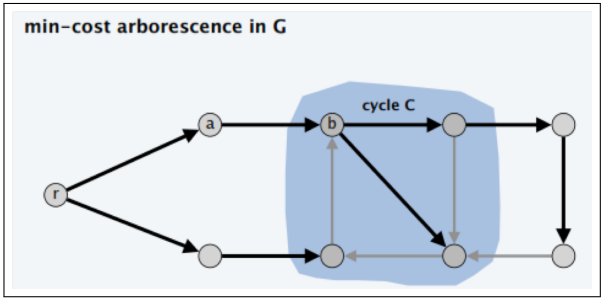
$T' \leftarrow$ EDMONDSBRANCHING(G', r, c')

Extend T' to an arborescence T in G by adding all but one edge

RETURN T.

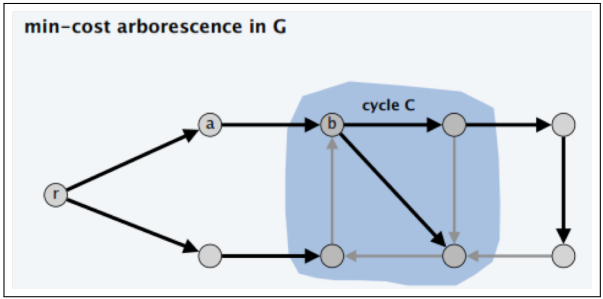
Issues

- What could go wrong?



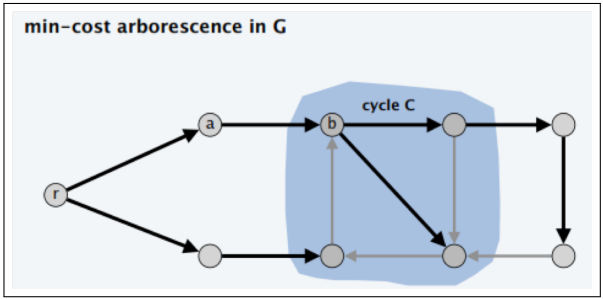
Issues

- What could go wrong?
- Min-cost arborescence in G' has exactly one edge entering a node in C (since C is contracted to a single node)



Issues

- What could go wrong?
- Min-cost arborescence in G' has exactly one edge entering a node in C (since C is contracted to a single node)
- But min-cost arborescence in G might have more edges entering C.



Key Lemma in Edmonds Algorithm

Claim: Let C be a cycle in G consisting of 0-cost edges. There exists a mincost arborescence rooted at r that has exactly one edge entering C.

- Proof: Let T be a min-cost arborescence rooted at r.

Key Lemma in Edmonds Algorithm

Claim: Let C be a cycle in G consisting of 0-cost edges. There exists a mincost arborescence rooted at r that has exactly one edge entering C.

- Proof: Let T be a min-cost arborescence rooted at r.
- Case 0. T has no edges entering C.

Key Lemma in Edmonds Algorithm

Claim: Let C be a cycle in G consisting of 0-cost edges. There exists a mincost arborescence rooted at r that has exactly one edge entering C.

- Proof: Let T be a min-cost arborescence rooted at r.
- Case 0. T has no edges entering C.
- Since T is an arborescence, there is an r $\rightarrow$ v path for each node v that $\Longrightarrow$ at least one edge enters C.

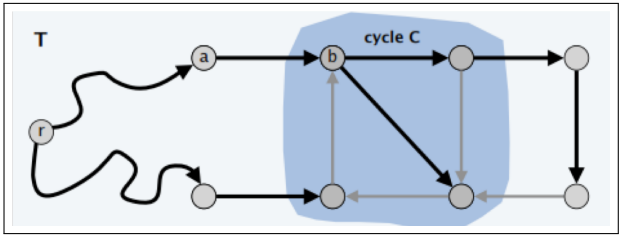
Key Lemma in Edmonds Algorithm

Claim: Let C be a cycle in G consisting of 0-cost edges. There exists a mincost arborescence rooted at r that has exactly one edge entering C.

- Proof: Let T be a min-cost arborescence rooted at r.
- Case 0. T has no edges entering C.
- Since T is an arborescence, there is an r $\rightarrow$ v path for each node v that $\Longrightarrow$ at least one edge enters C.
- Case 1. T has exactly one edge entering C. T satisfies the lemma.

Key Lemma in Edmonds Algorithm...

- Case 2. T has more than one edge that enters C.
- We construct another min-cost arborescence T' that has exactly one edge entering C as follows:
  - Let (a, b) be an edge in T entering C that lies on a shortest [1] path from r.
  - We delete all edges of T that enter a node in C except (a, b).
  - We add in all edges of C except the one that enters b

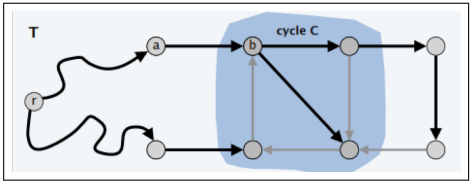


[1] path from r to C uses only one node in C

Key Lemma in Edmonds Algorithm...

Claim: T' is a min-cost arborescence.

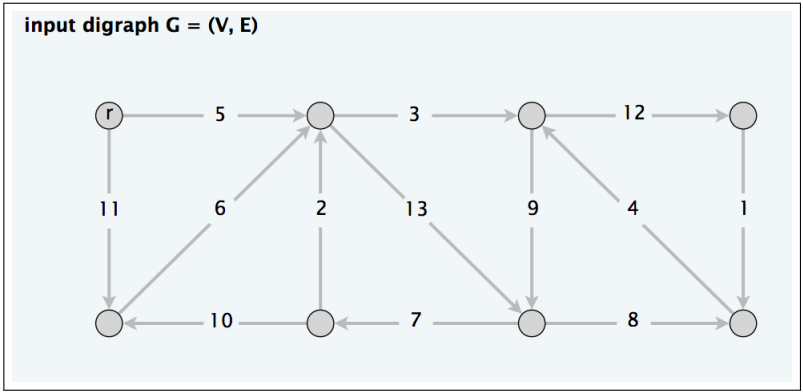
- The cost of T' is at most that of T since we add only 0-cost edges.
- T' has exactly one edge entering each node $v \neq r$. [2]
- T' has no directed cycles.
- T had no cycles before; no cycles within C; now only (a, b) enters C) [3]



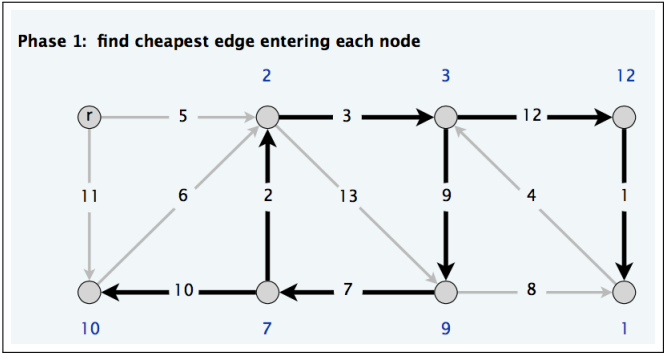
[2] T is an arborescence rooted at r

[3] and the only path in T' to a is the path from r to a (since any path must follow unique entering edge back to r)

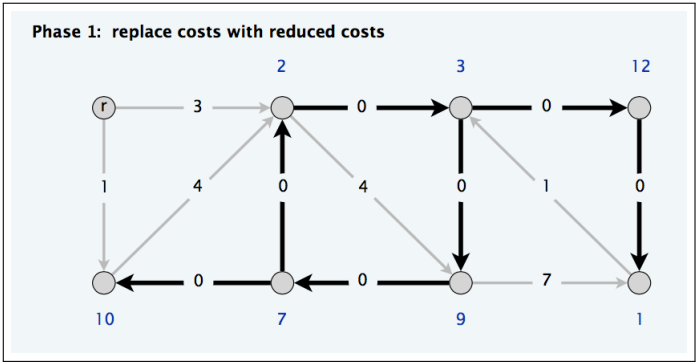
Demo: Edmonds Algorithm



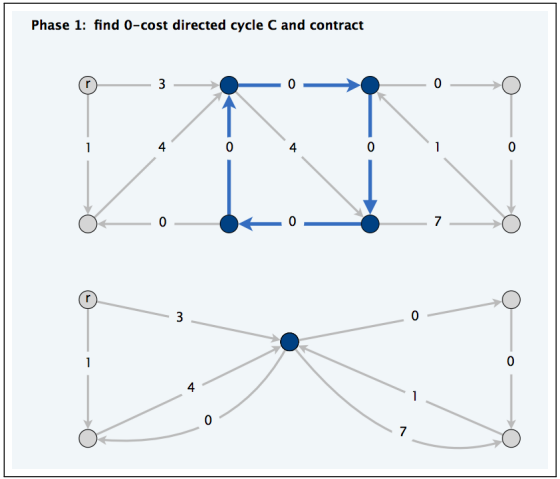
Demo: Edmonds Algorithm...



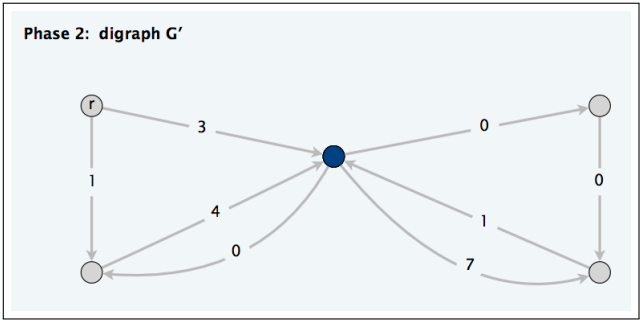
Demo: Edmonds Algorithm...



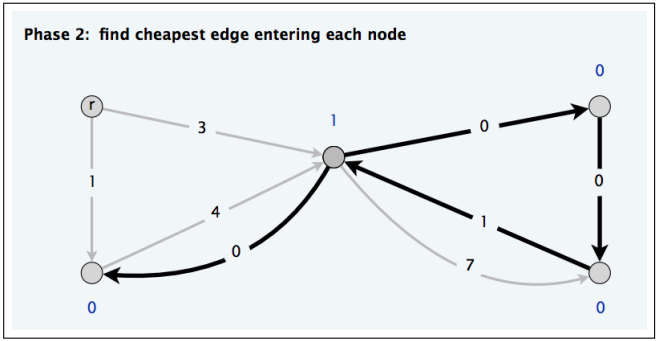
Demo: Edmonds Algorithm...



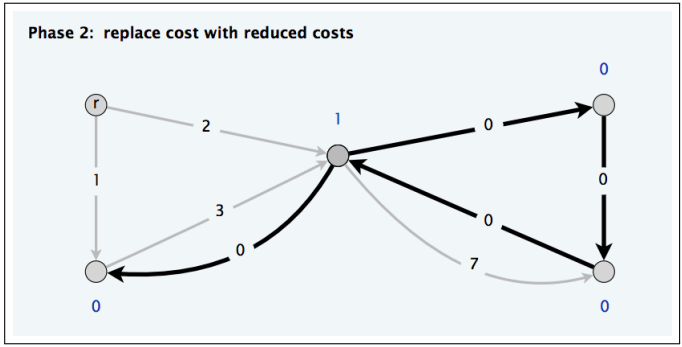
Demo: Edmonds Algorithm...



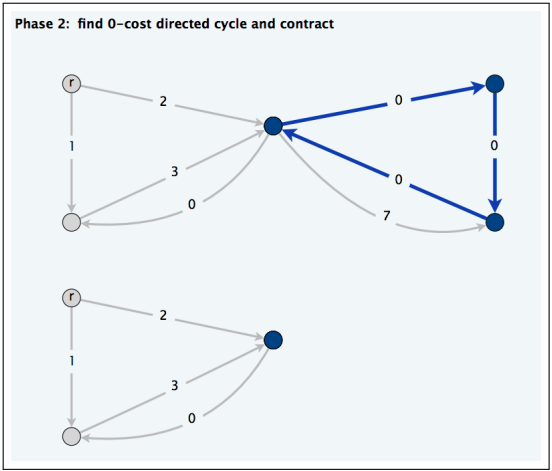
Demo: Edmonds Algorithm...



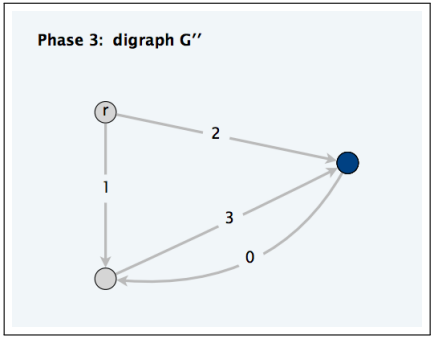
Demo: Edmonds Algorithm...



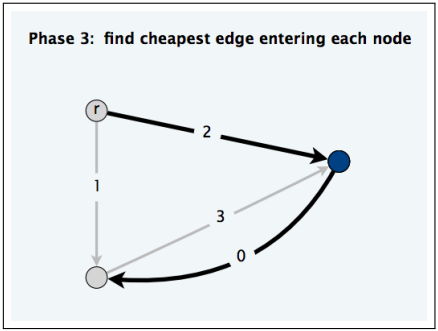
Demo: Edmonds Algorithm...



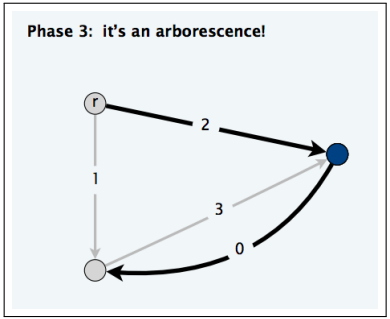
Demo: Edmonds Algorithm...



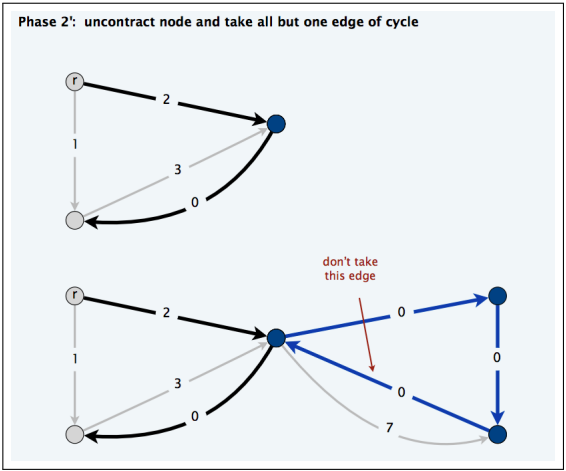
Demo: Edmonds Algorithm...



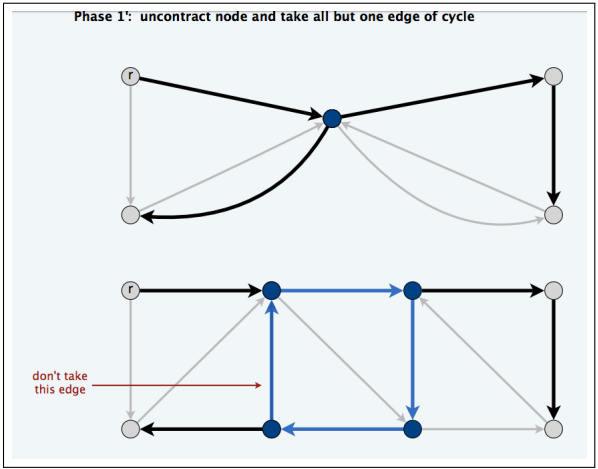
Demo: Edmonds Algorithm...



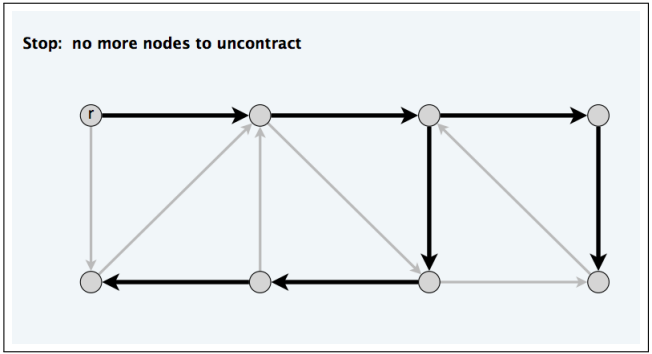
Demo: Edmonds Algorithm...



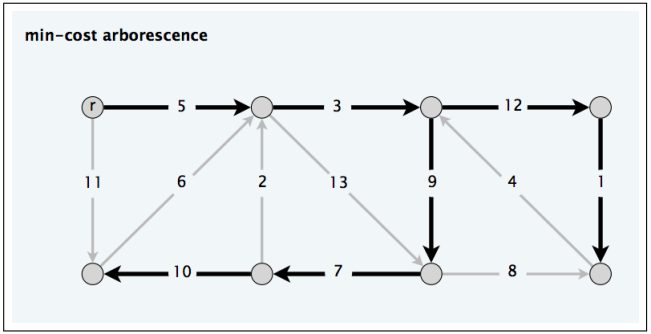
Demo: Edmonds Algorithm...



Demo: Edmonds Algorithm...



Demo: Edmonds Algorithm...



Other Algorithms

Algorithms due to

- Chu and Liu, Edmonds and Bock
- independently have given efficient algorithms for finding the MST on a directed graph.
- The Chu-Liu and Edmonds algorithms are virtually identical, the Bock algorithm is similar but stated on matrices instead of on graphs.

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight
- If no cycle formed, then G(N,S) is a MST, otherwise

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight
- If no cycle formed, then G(N,S) is a MST, otherwise
- For each cycle that is formed:

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight
- If no cycle formed, then G(N,S) is a MST, otherwise
- For each cycle that is formed:
  - edge *m* is the edge in this cycle with minimum weight

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight
- If no cycle formed, then G(N,S) is a MST, otherwise
- For each cycle that is formed:
  - edge *m* is the edge in this cycle with minimum weight
  - Combine all the cycle's nodes into one pseudo-node *k*

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight
- If no cycle formed, then G(N,S) is a MST, otherwise
- For each cycle that is formed:
  - edge m is the edge in this cycle with minimum weight
  - Combine all the cycle's nodes into one pseudo-node k
  - For each edge e entering a node in k in the original graph:

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight
- If no cycle formed, then G(N,S) is a MST, otherwise
- For each cycle that is formed:
  - edge m is the edge in this cycle with minimum weight
  - Combine all the cycle's nodes into one pseudo-node k
  - For each edge e entering a node in k in the original graph:
    - edge n is the edge currently entering this node in the cycle

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight
- If no cycle formed, then G(N,S) is a MST, otherwise
- For each cycle that is formed:
  - edge m is the edge in this cycle with minimum weight
  - Combine all the cycle's nodes into one pseudo-node k
  - For each edge e entering a node in k in the original graph:
    - edge n is the edge currently entering this node in the cycle
    - track the minimum modified edge weight between each e based on the following:

Chu-Liu/Edmonds Algorithm

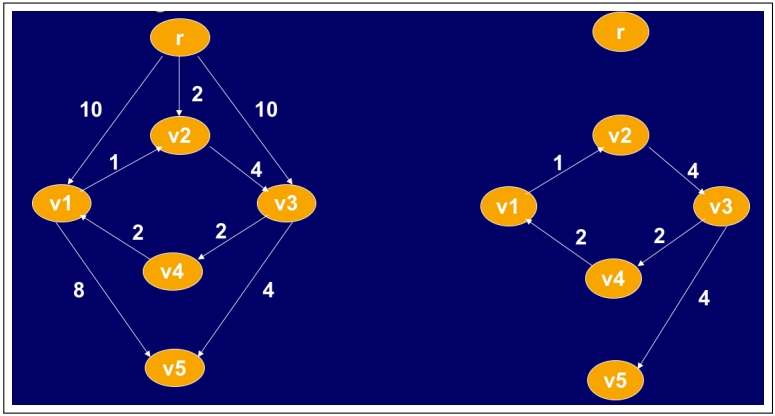
- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight
- If no cycle formed, then G(N,S) is a MST, otherwise
- For each cycle that is formed:
  - edge m is the edge in this cycle with minimum weight
  - Combine all the cycle's nodes into one pseudo-node k
  - For each edge e entering a node in k in the original graph:
    - edge n is the edge currently entering this node in the cycle
    - track the minimum modified edge weight between each e based on the following:
    - modWeight = weight(e) - (weight(n) - weight(m))

Chu-Liu/Edmonds Algorithm

- Remove all edges going into the root node
- For each node, select only the incoming edge with smallest weight
- If no cycle formed, then G(N,S) is a MST, otherwise
- For each cycle that is formed:
  - edge m is the edge in this cycle with minimum weight
  - Combine all the cycle's nodes into one pseudo-node k
  - For each edge e entering a node in k in the original graph:
    - edge n is the edge currently entering this node in the cycle
    - track the minimum modified edge weight between each e based on the following:
    - modWeight = weight(e) - (weight(n) - weight(m))
  - On edge e with minimum modified weight, add edge e and remove edge n

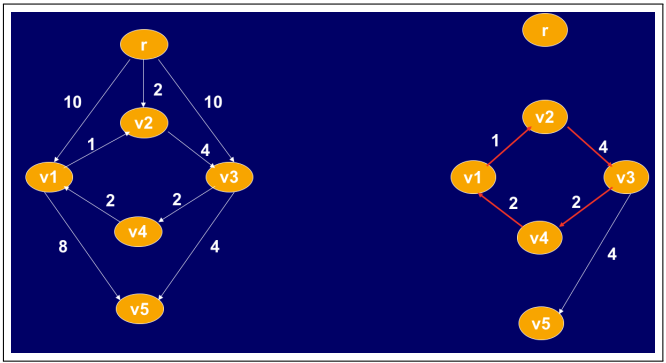
An illustration

Step 1:- For each node, select only the incoming edge with smallest weight



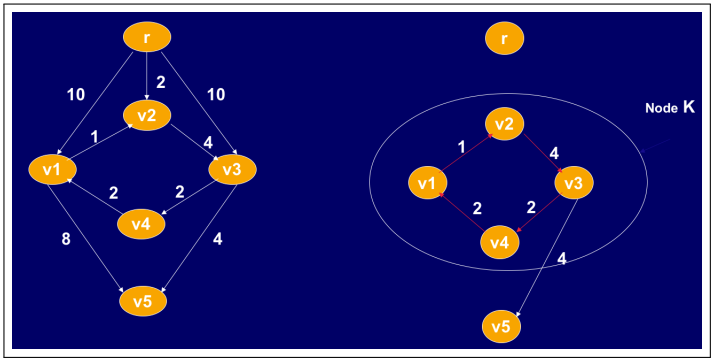
An illustration...

Step 2:- Verify if the cycle is formed OR not



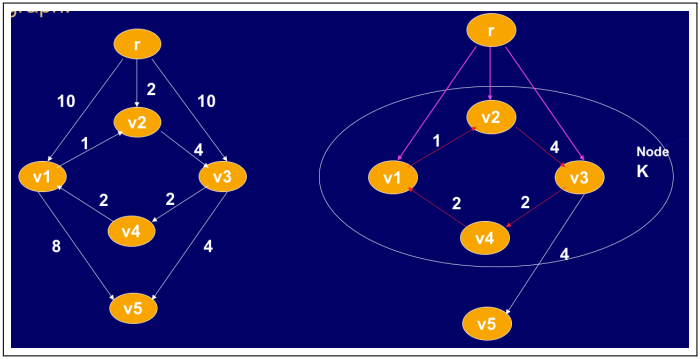
An illustration...

Step 3:- Combine all the cycle's nodes into one pseudo-node "k"



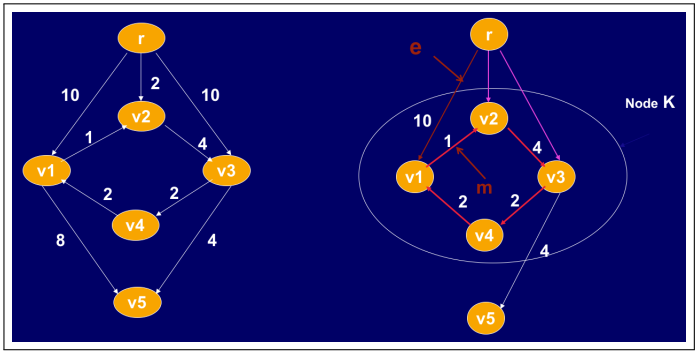
An illustration...

Step 4:- Select edges which are entering the node "k" in the original graph.



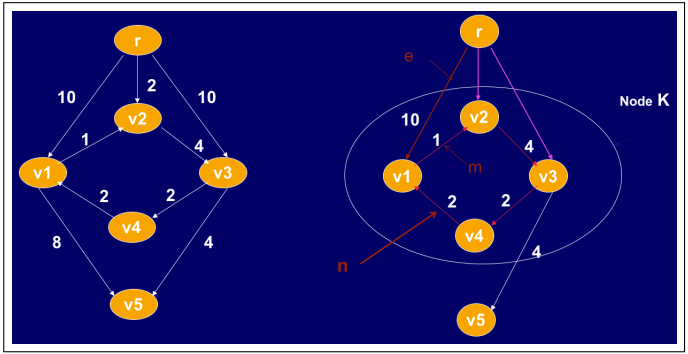
An illustration...

Step 5:- Select Edge "e" (r,v1) edge "m" in this cycle with minimum weight is (v1,v2,1).



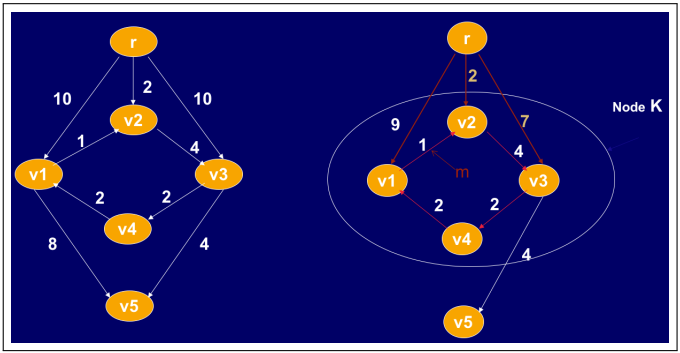
An illustration...

Step 6:- edge "n" in this cycle currently entering this node is (v4,v1,2).



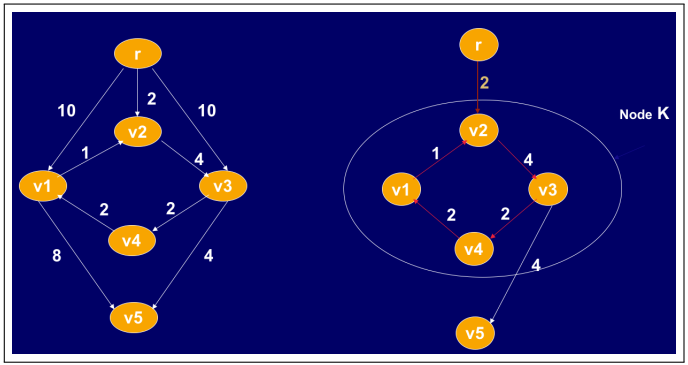
An illustration...

Step 7:- $modWeight = weight("e") = weight("n") - weight("m") = 10 - (2 - 1) = 9$



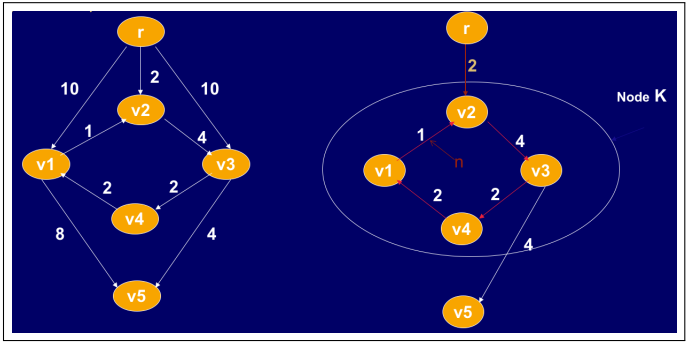
An illustration...

Step 8:- :- All Modified edges.



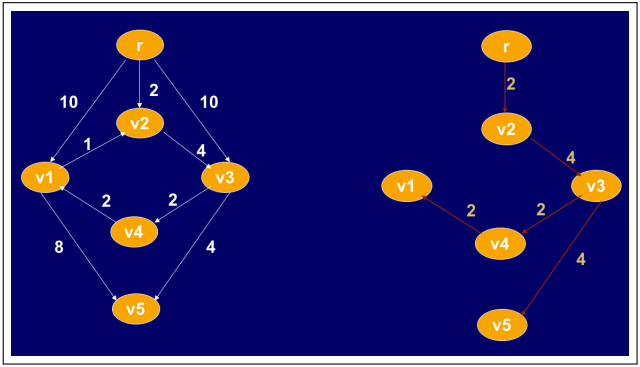
An illustration...

Step 9:- Select minimum modified weight edge "e" is (r,v2,2).



An illustration...

Step 10:-Add edge "e" (r,v2,2) and remove edge "n". Edge "n" is (v1,v2,1).



Tutorial Problem 1

Consider the 0/1 knapsack problem with n objects whose profits and weights are v_i, w_i $1 \leq i \leq n$, respectively. The capacity of the knapsack is m. IT is also given that all the objects have the same weight. Present an $O(m)$ algorithm to solve this problem. Also, give its proof of correctness.

Tutorial Problem 2

Suppose that in a 0-1 knapsack problem, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value. Give an efficient algorithm to find an optimal solution to this variant of the knapsack problem and argue that your answer is correct

Tutorial Problem 3

Let F(I) be the value of the solution generated on knapsack problem instance I by GreedyKnapsack when the objects are processed in nonincreasing order of their profit values. Let F*(I) be the value of an optimal solution of this instance. How large can the ratio F*(I)/F(I) get ?

Tutorial Problem 4

Consider an input array a[1..n] of arbitrary numbers. It is given that the array has only O(1) distinct elements. Present an O(n) time algorithm to sort this array.

Tutorial Problem 5

Suppose that we have a set of activities to schedule amongst a large number of lecture halls. We wish to schedule all the activities using as few lecture halls as possible. Give an efficient greedy algorithm to determine which activity should use which lecture hall.

Tutorial Problem 6

Prof Midas drives an automobile from Newark to Reno along Interstate 80. His car's gas tank, when full, holds enough gas to travel n miles and his map gives the distances between gas stations on his route. The professors wishes to make as few gas stops as possible along the way. Give an efficient method by which Professor Midas can determine at which gas stations should he stop and prove that your strategy yields an optimal solution.

Tutorial Problem 7

Let's consider a long, quiet country road with houses scattered very sparsely along it. That is, picture the road as a line segment, with an eastern endpoint and a western endpoint). Further, let's suppose that despite the bucolic setting, the residents of all these houses are avid cell phone users. You want to place cell phone base stations ('towers ' as we say here) at certain points along the road, so that every house is within four miles of one of the base stations.
Give an efficient algorithm that achieves the goal, using as few base stations as is possible. Prove your algorithm , too.

Tutorial Problem 8

Suppose a Consulting firm that does security consultancy needs to obtain licenses for n different pieces of cryptographic software. However, due to regulations, they can only buy these licenses at the rate of at most one license per month. Each license is currently selling at the rate of Rs 1000. However, the cost of the licenses are designed to increase following the monthly exponential growth, at the defined rates i.e. the cost of license j increases at the rate r_j (> 1) every month, that grows exponentially. For example, if license j is purchased t months from now, it will cost 100 * r_{jt}.We assume that all the price growth rates are distinct i.e. r_j != r_j for i != j. Given as the input, the n rates of price growth viz. r_1, r_2, r_3, ...r_n, determine the order in which the licenses must be bought so that the total amount of money spent is minimized.

Tutorial Problem 9

Tutorial Problem 10

Tutorial Problem 11

Tutorial Problem 12