Strategic deployment in graphs

Elmar Langetepe

University of Bonn

Oktober 11th 2013

Strategic Deployment

Strategic Deployment

- Starting form a basis camp with a set of agents

Strategic Deployment

- Starting form a basis camp with a set of agents
- Take over and control some settlements

Strategic Deployment

- Starting form a basis camp with a set of agents
- Take over and control some settlements
- Resistance in the outback

Strategic Deployment

- Starting form a basis camp with a set of agents
- Take over and control some settlements
- Resistance in the outback
- Enough agents for the movements between the settlements and for controlling the settlements

Strategic Deployment

- Starting form a basis camp with a set of agents
- Take over and control some settlements
- Resistance in the outback
- Enough agents for the movements between the settlements and for controlling the settlements
- Task: Move *efficiently* around and occupy the settlements

Strategic Deployment

- Starting form a basis camp with a set of agents
- Take over and control some settlements
- Resistance in the outback
- Enough agents for the movements between the settlements and for controlling the settlements
- Task: Move *efficiently* around and occupy the settlements
- Historic examples!
  Gaius Julius Ceasar: Conquer of the Gauls (58 to 51 B.C.)
  Alexander the Great (356 B.C. to 323 B.C.): Alexander's campaign

Strategic Deployment

- Starting form a basis camp with a set of agents
- Take over and control some settlements
- Resistance in the outback
- Enough agents for the movements between the settlements and for controlling the settlements
- Task: Move *efficiently* around and occupy the settlements
- Historic examples!
  Gaius Julius Ceasar: Conquer of the Gauls (58 to 51 B.C.)
  Alexander the Great (356 B.C. to 323 B.C.): Alexander's campaign
- Modeled by an edge an vertex-weigthed graph

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:
  1. Edge e, weight w_e: *Lower bound* on the number of agents required for traversal of e.

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:
  1. Edge e, weight w_e: *Lower bound* on the number of agents required for traversal of e.
  2. Vertex v, weight w_v: Number of agents that have to be placed at the vertex.

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:
  1. Edge e, weight w_e: *Lower bound* on the number of agents required for traversal of e.
  2. Vertex v, weight w_v: Number of agents that have to be placed at the vertex.
  3. First visit of v: Full amount w_v have to be placed, these agents cannot be removed any more.

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:
  1. Edge e, weight w_e: *Lower bound* on the number of agents required for traversal of e.
  2. Vertex v, weight w_v: Number of agents that have to be placed at the vertex.
  3. First visit of v: Full amount w_v have to be placed, these agents cannot be removed any more.
- Visit and occupy the vertices accordingly

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:
  1. Edge e, weight w_e: *Lower bound* on the number of agents required for traversal of e.
  2. Vertex v, weight w_v: Number of agents that have to be placed at the vertex.
  3. First visit of v: Full amount w_v have to be placed, these agents cannot be removed any more.
- Visit and occupy the vertices accordingly
- Interesting computational questions:

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:
  1. Edge e, weight w_e: *Lower bound* on the number of agents required for traversal of e.
  2. Vertex v, weight w_v: Number of agents that have to be placed at the vertex.
  3. First visit of v: Full amount w_v have to be placed, these agents cannot be removed any more.
- Visit and occupy the vertices accordingly
- Interesting computational questions:
  How many agents are required in total?

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:
  1. Edge e, weight w_e: *Lower bound* on the number of agents required for traversal of e.
  2. Vertex v, weight w_v: Number of agents that have to be placed at the vertex.
  3. First visit of v: Full amount w_v have to be placed, these agents cannot be removed any more.
- Visit and occupy the vertices accordingly
- Interesting computational questions:
  How many agents are required in total?
  How long does this take?

Model of the Problem

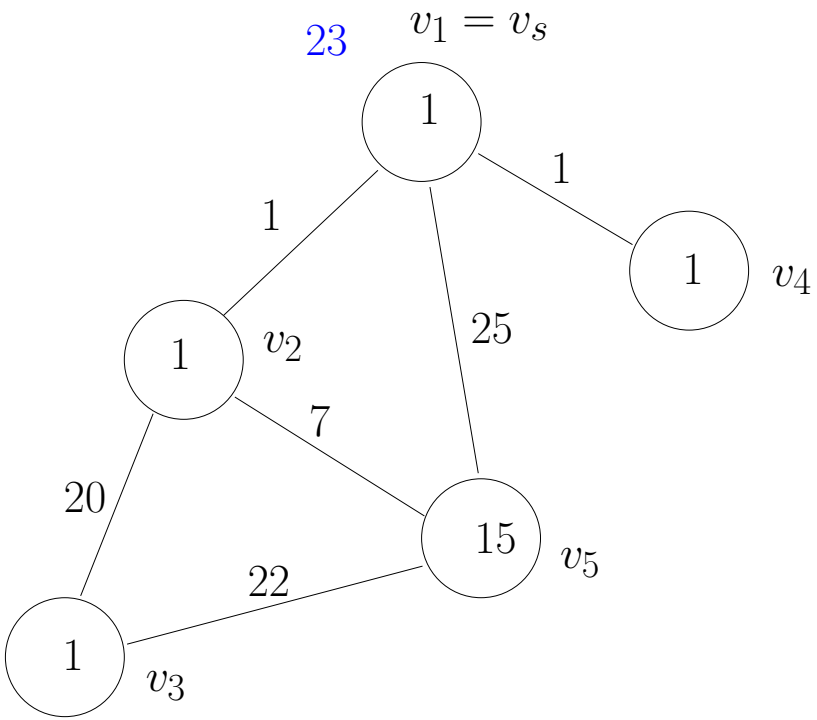
- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:
  1. Edge e, weight w_e: *Lower bound* on the number of agents required for traversal of e.
  2. Vertex v, weight w_v: Number of agents that have to be placed at the vertex.
  3. First visit of v: Full amount w_v have to be placed, these agents cannot be removed any more.
- Visit and occupy the vertices accordingly
- Interesting computational questions:
  How many agents are required in total?
  How long does this take?
  k agents given: How many settlements can we get?

Model of the Problem

- Edge- and vertex-weigthed graph G, Graph is fully known
- Some start vertex v_s
- Rules for the movement:
  1. Edge e, weight w_e: *Lower bound* on the number of agents required for traversal of e.
  2. Vertex v, weight w_v: Number of agents that have to be placed at the vertex.
  3. First visit of v: Full amount w_v have to be placed, these agents cannot be removed any more.
- Visit and occupy the vertices accordingly
- Interesting computational questions:
  How many agents are required in total?
  How long does this take?
  k agents given: How many settlements can we get?

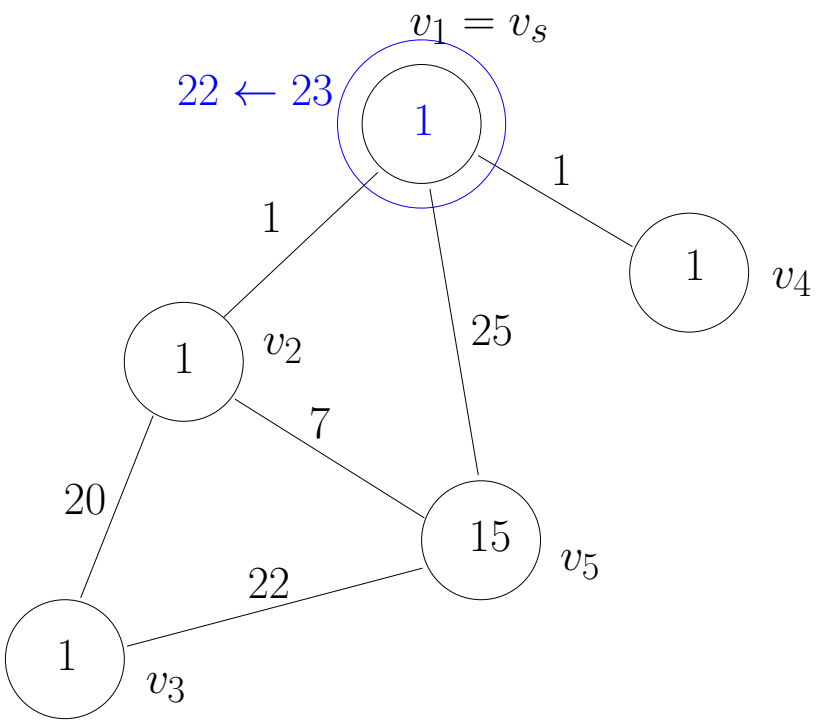
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



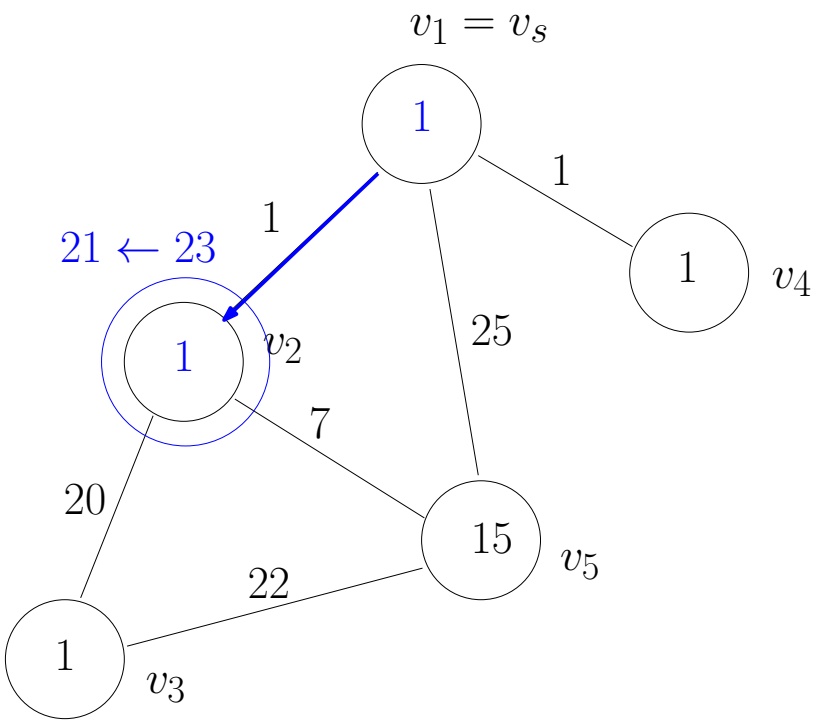
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



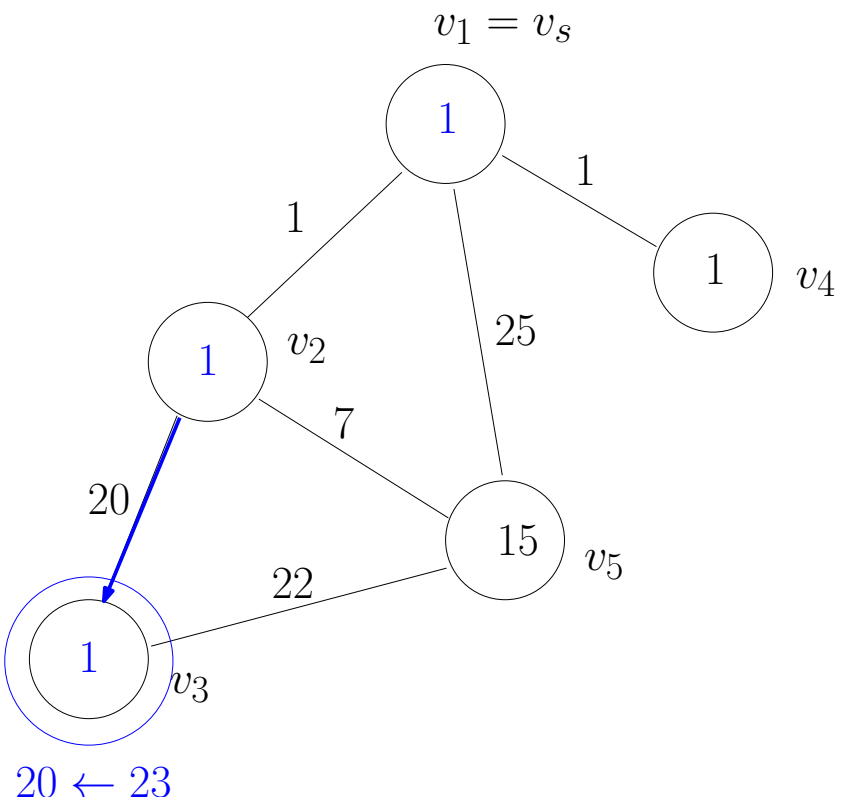
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



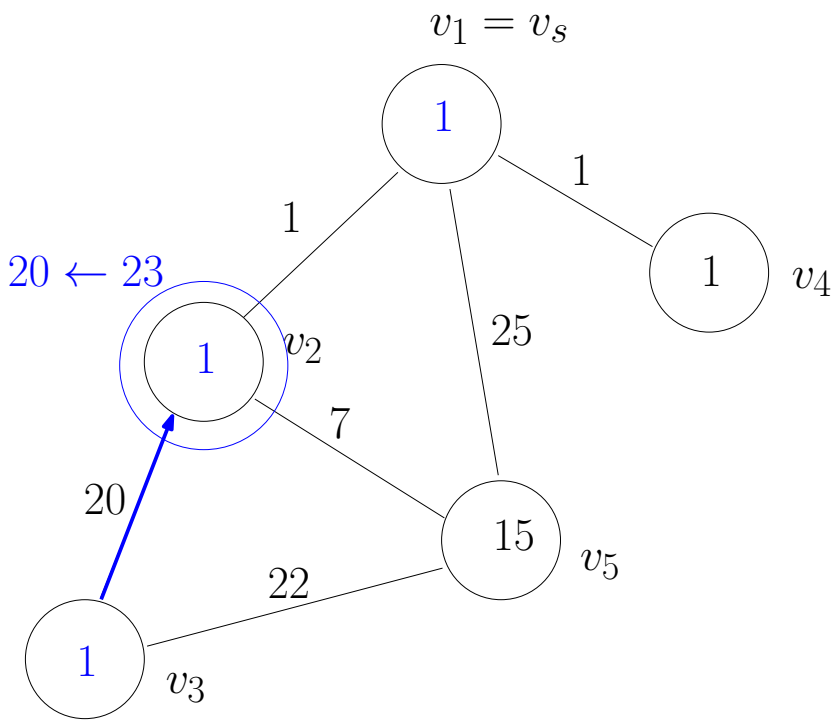
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



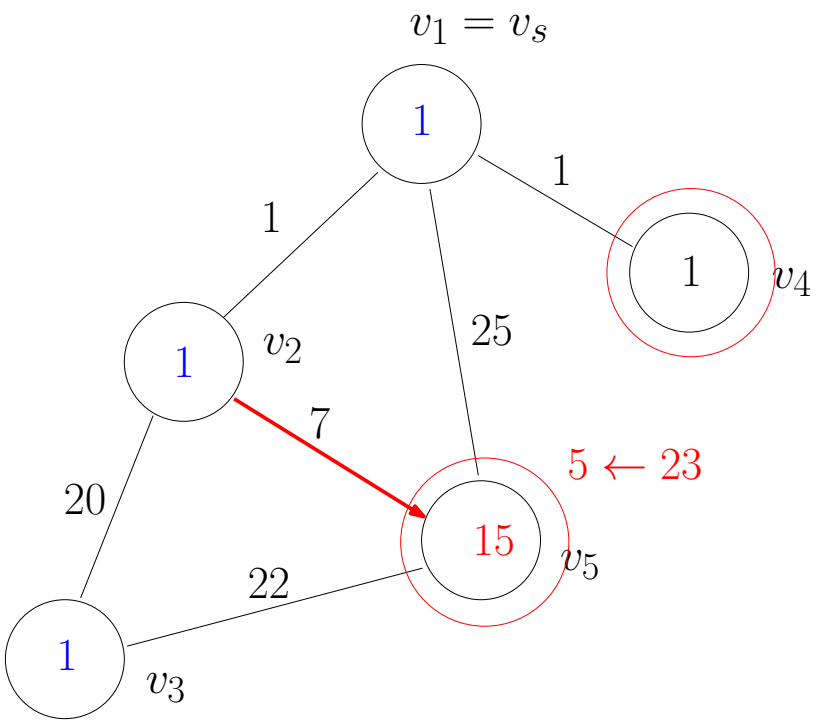
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



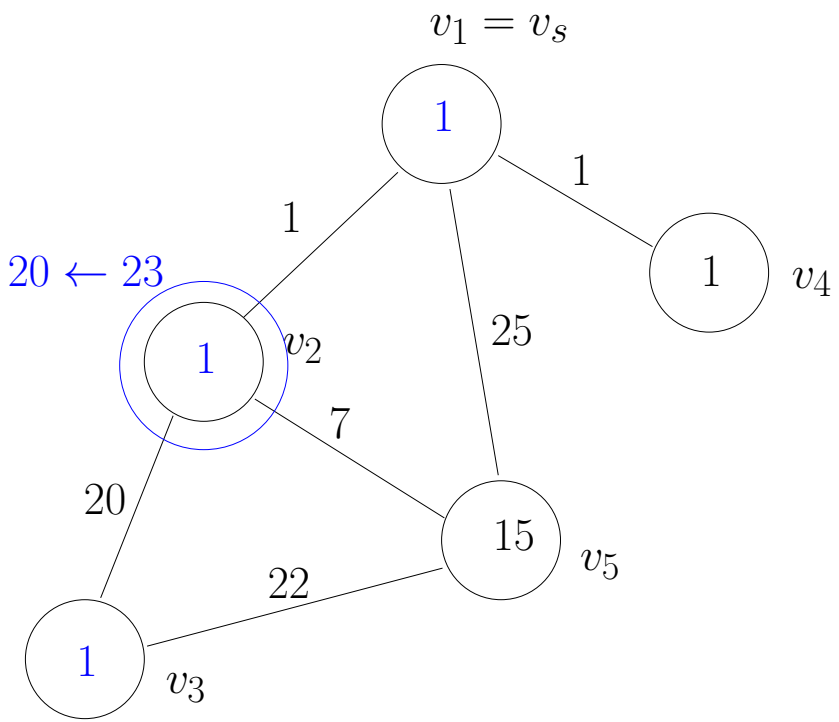
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



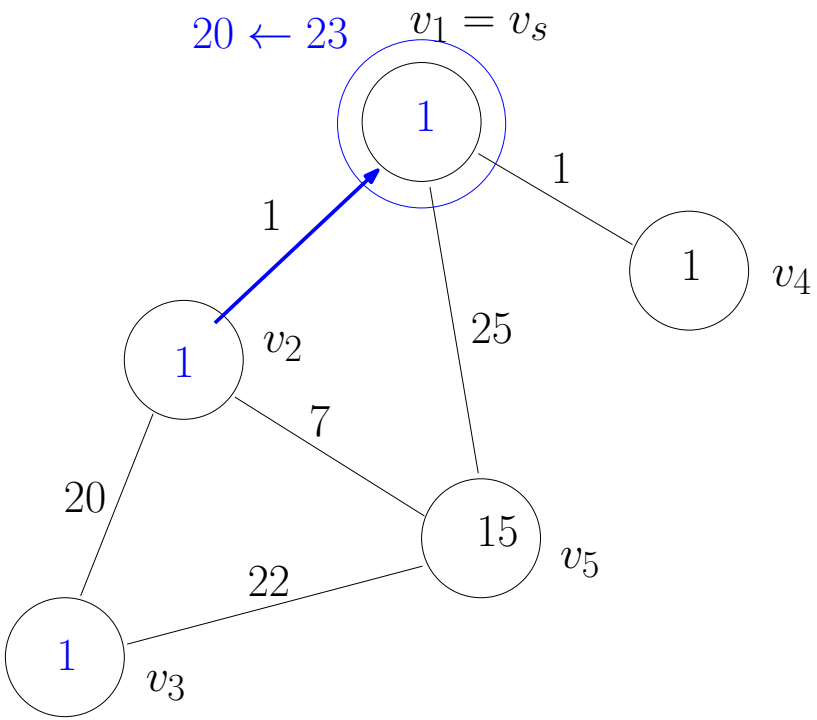
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



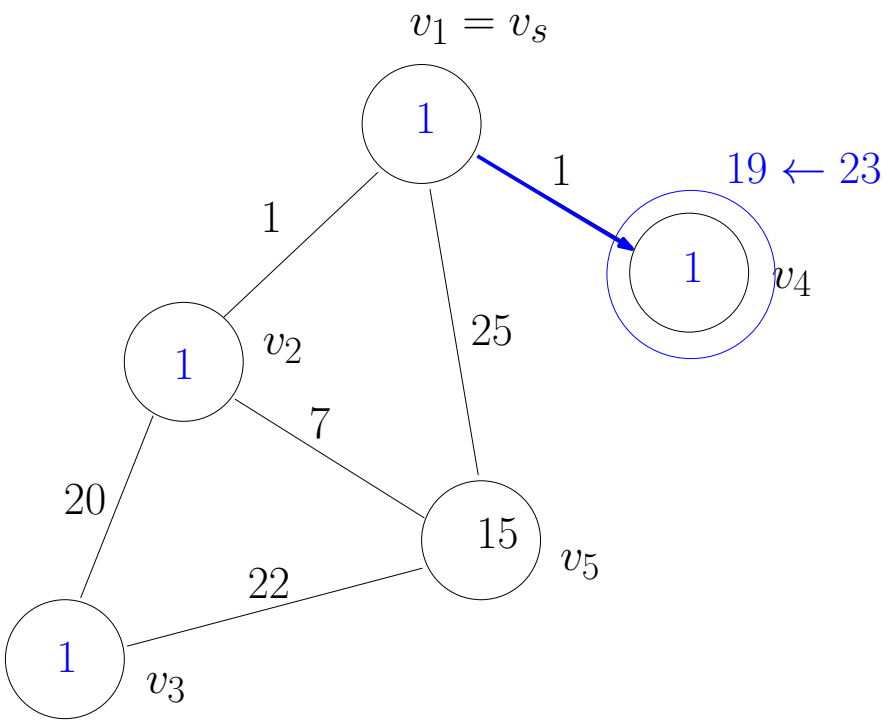
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



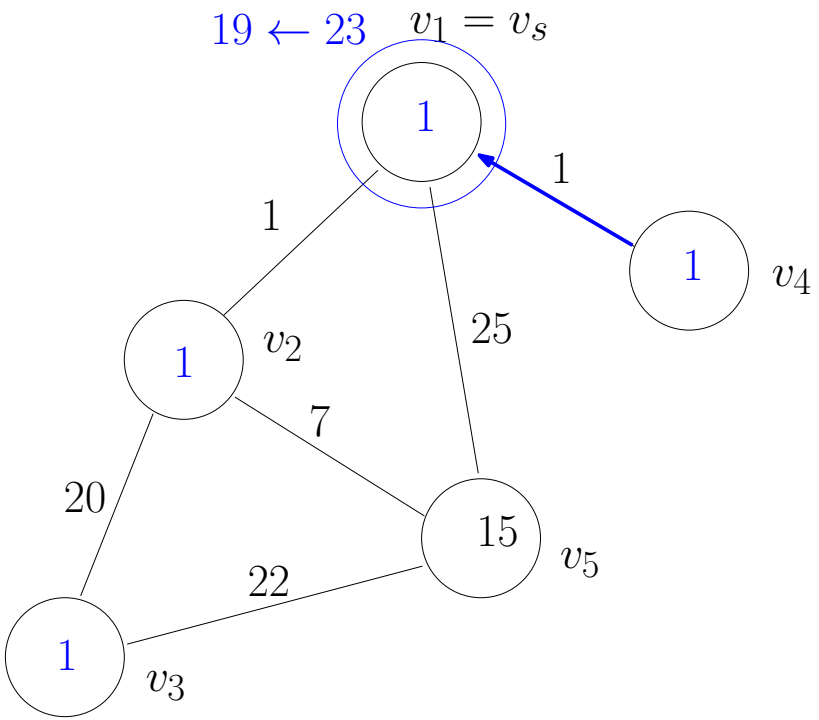
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



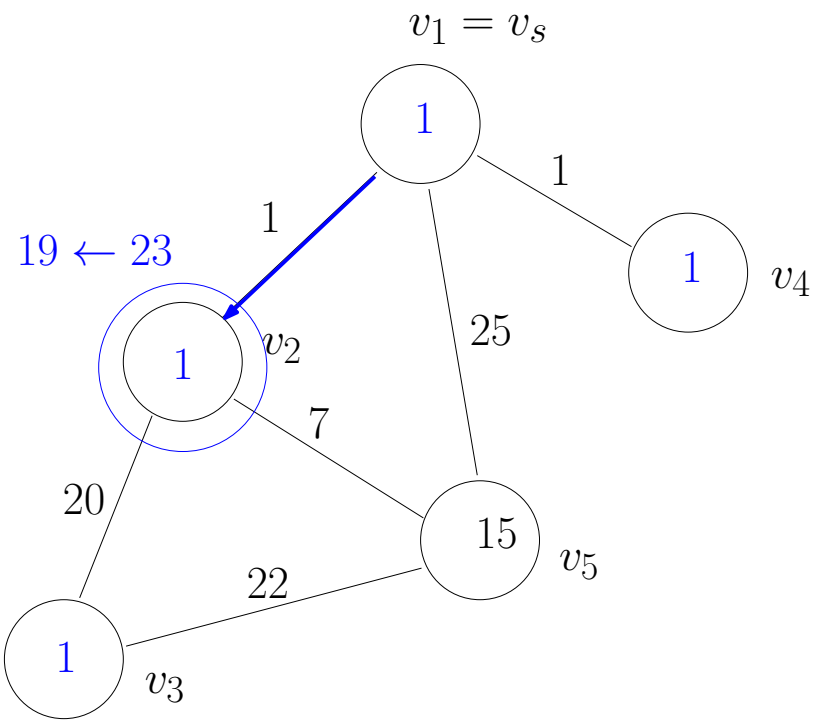
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



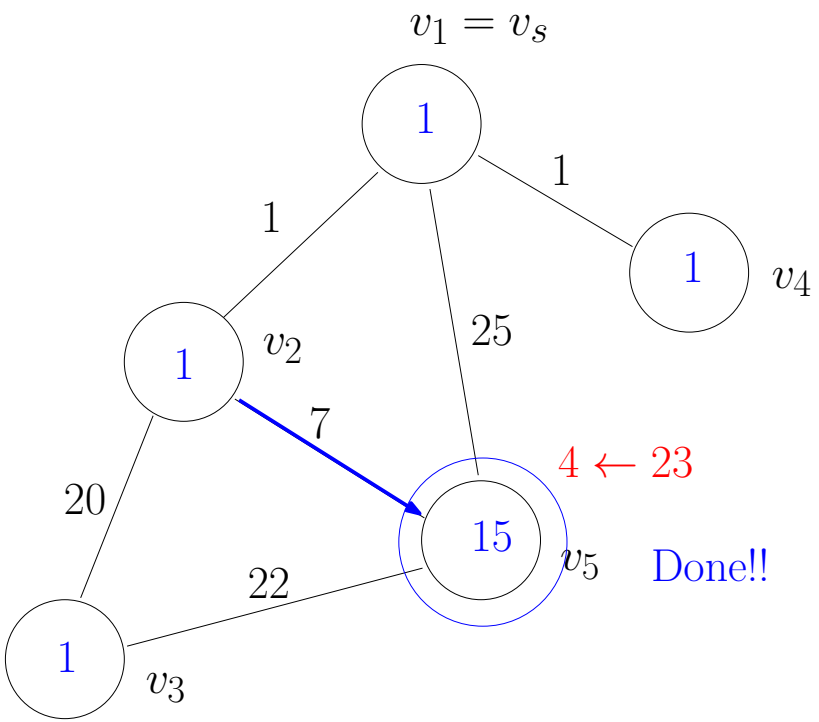
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



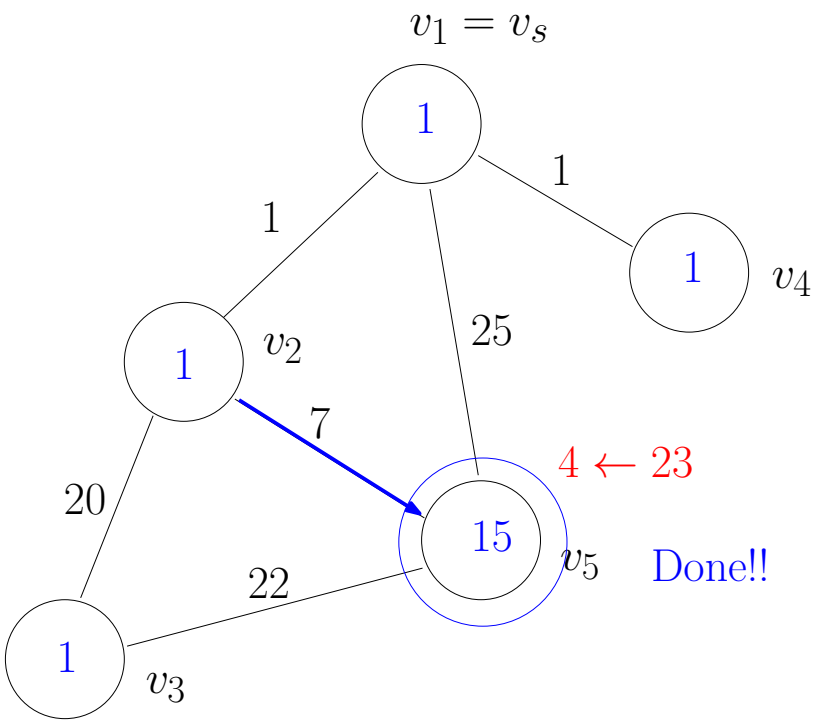
Example: Minimal number of agents required is 23!

1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



Example: Minimal number of agents required is 23!

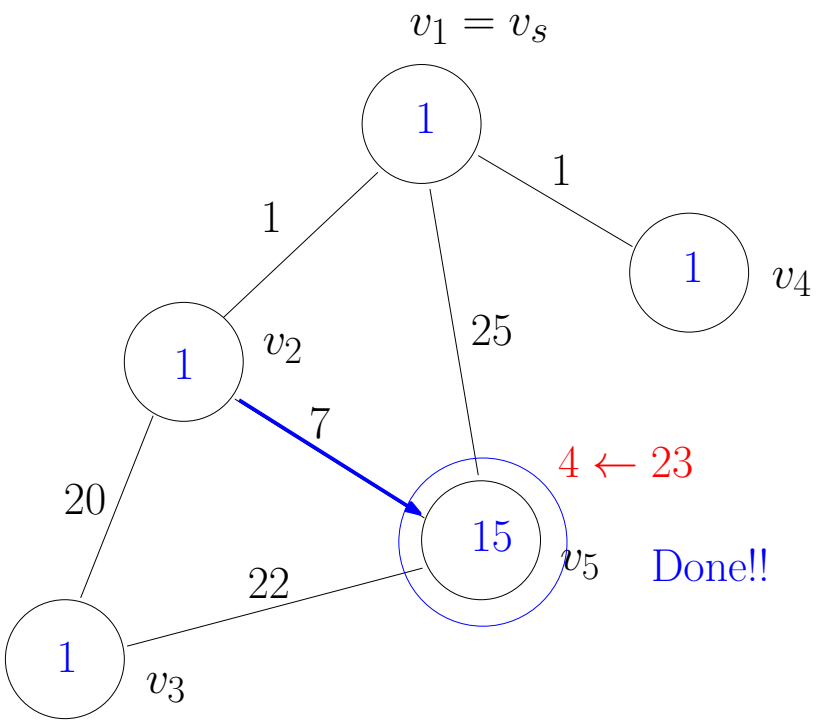
1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



4 agents remain unsettled! No other strategy is better!!

Example: Minimal number of agents required is 23!

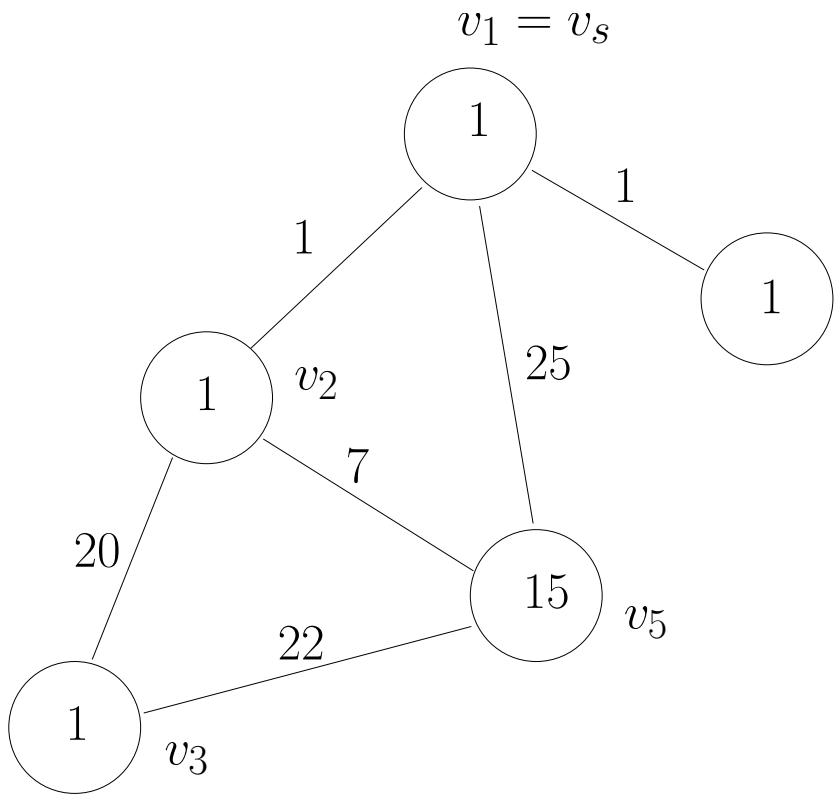
1. At least w_e for traversing edge e are required
2. At the first visit of v exactly w_v have to be placed and will never by removed!



4 agents remain unsettled! No other strategy is better!!
Is the problem clear?

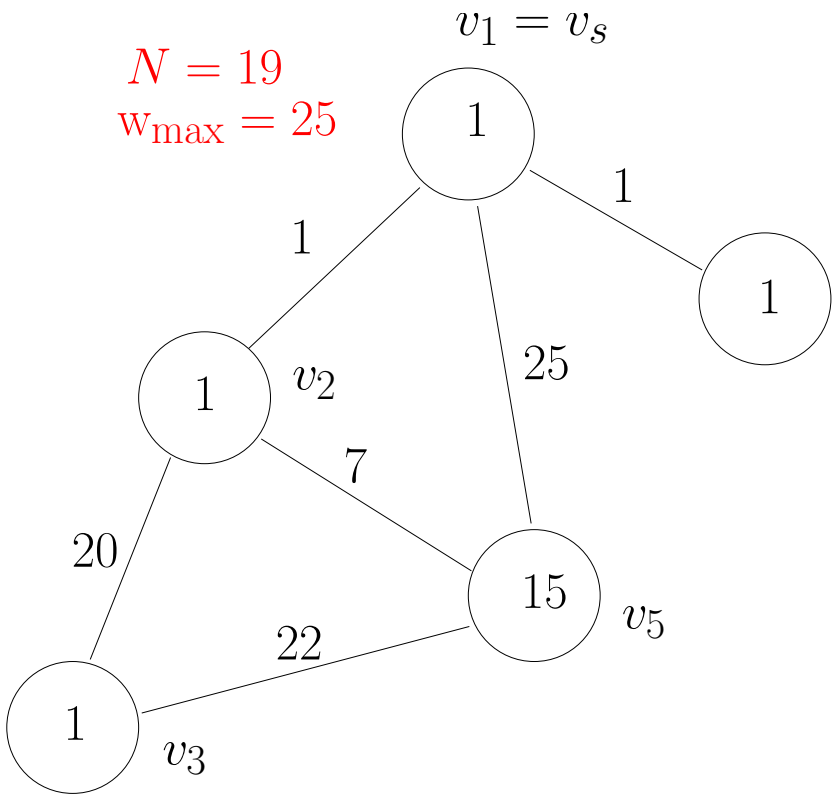
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$



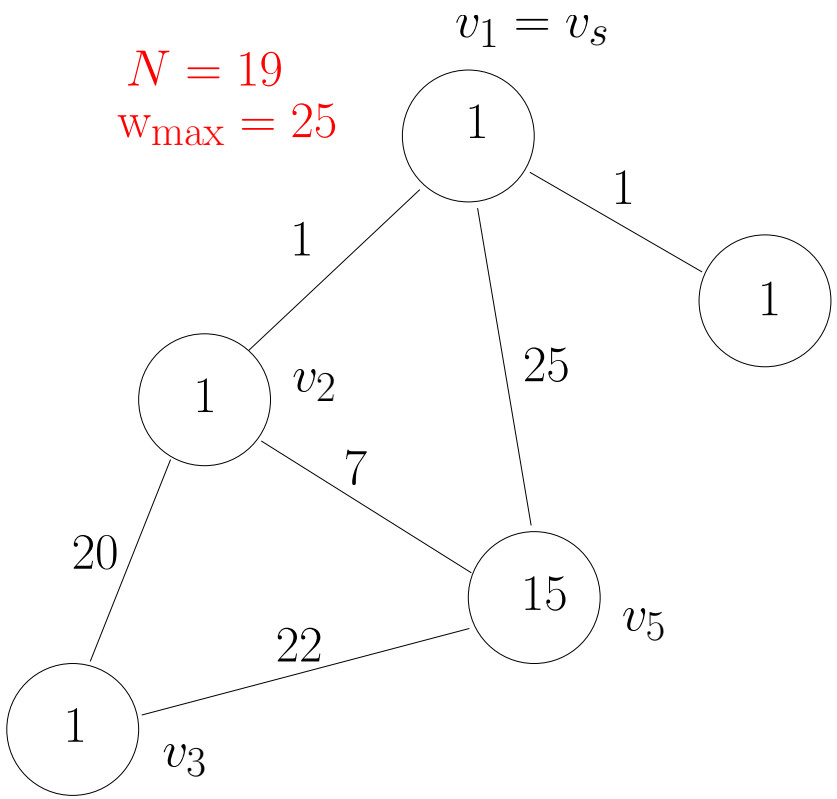
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$



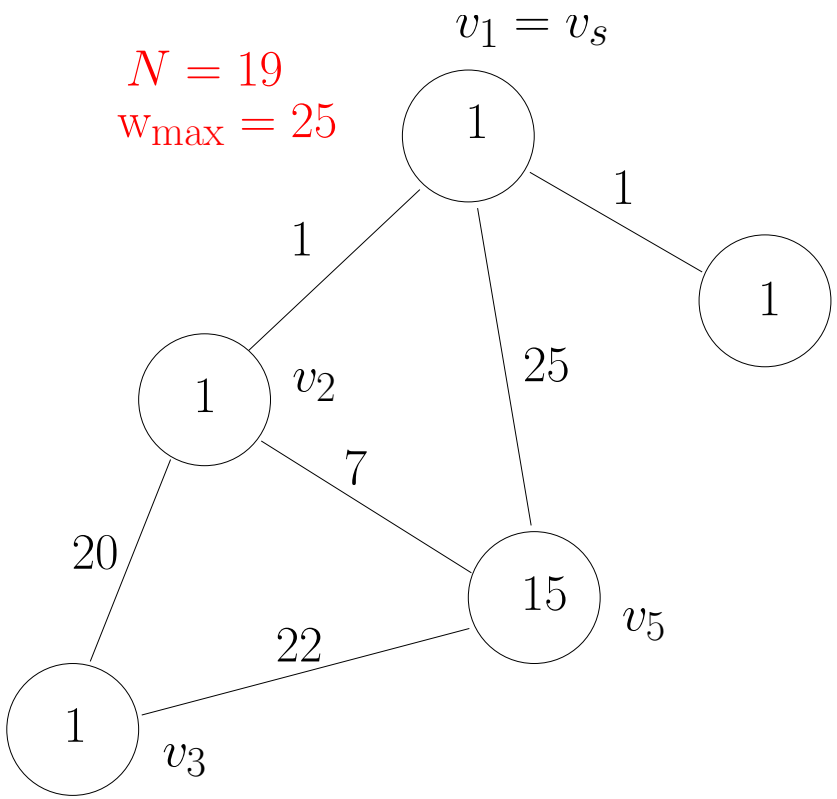
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$
- $N + w_{\max}$ on G is sufficient!



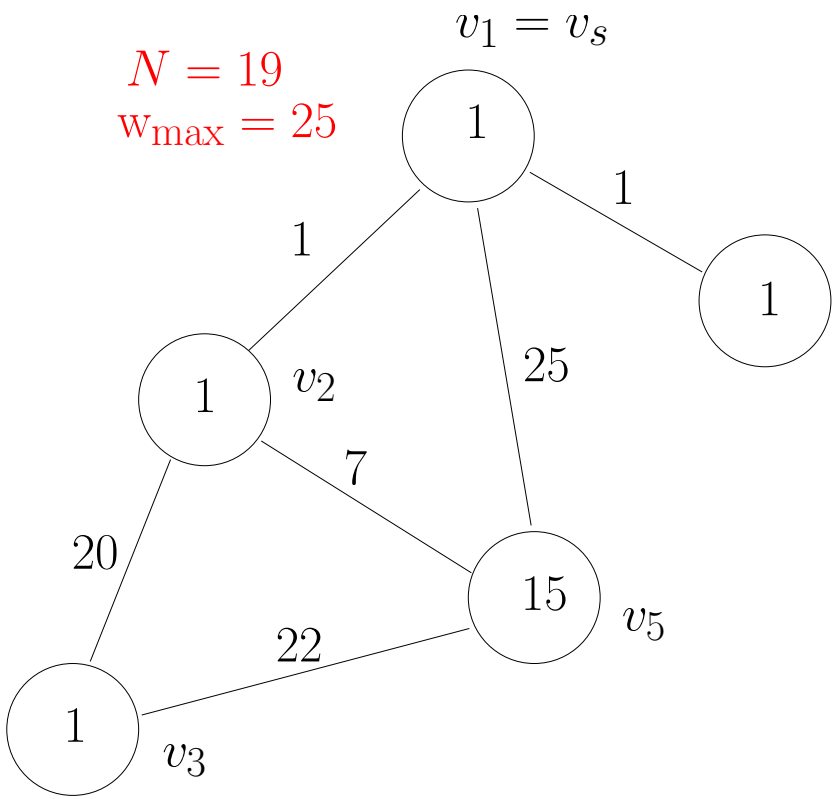
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$
- $N + w_{\max}$ on G is sufficient!
- Strategy S:



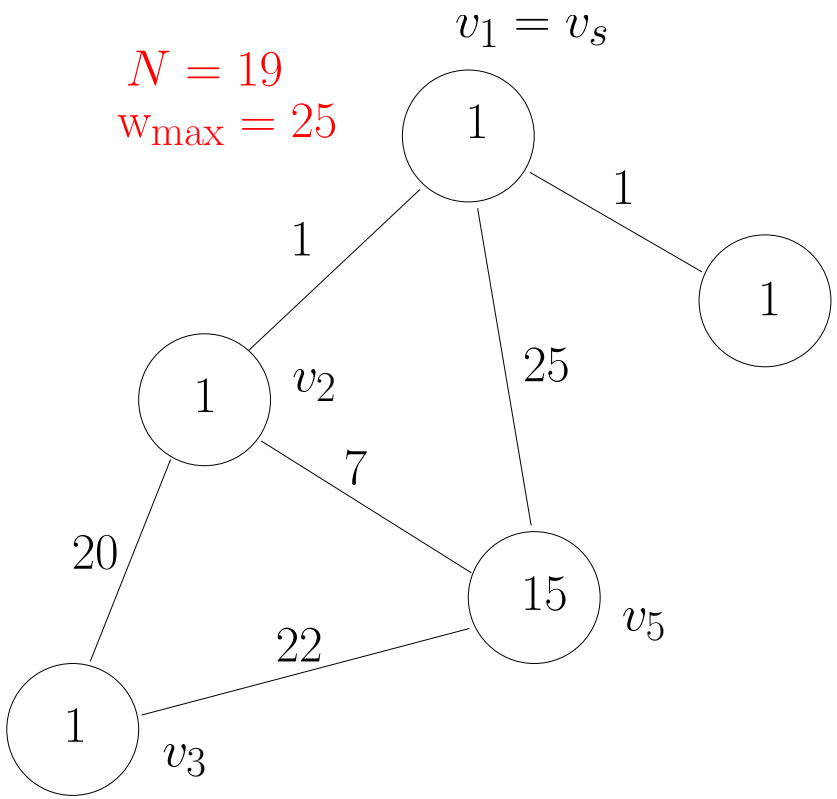
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$
- $N + w_{\max}$ on G is sufficient!
- Strategy S:
  $w_S := \max\{w_e | e \text{ was visited by } S\}$



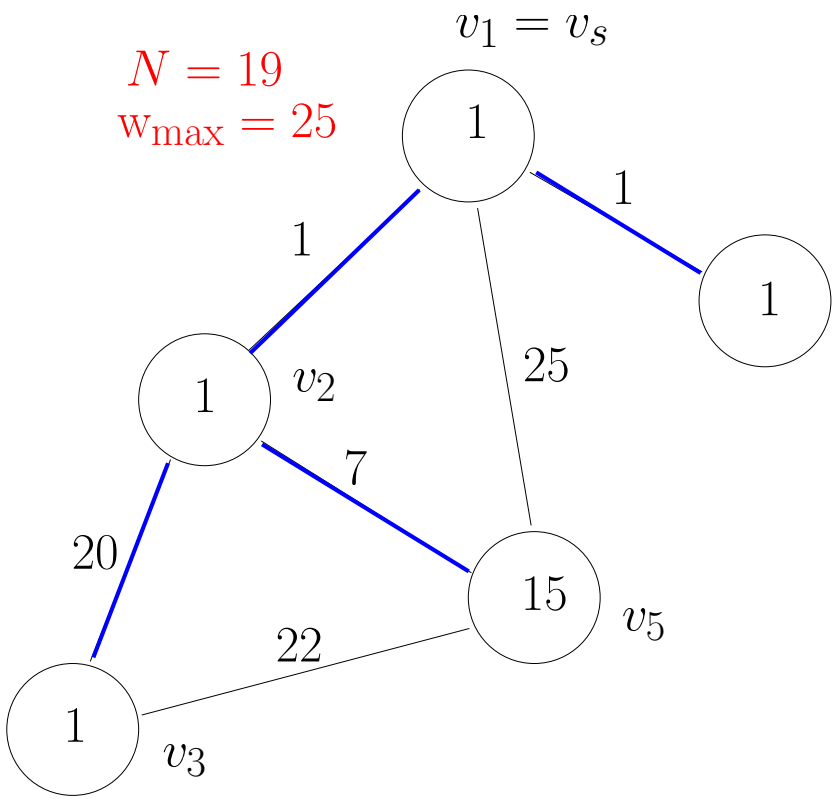
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$
- $N + w_{\max}$ on G is sufficient!
- Strategy S:
  $w_S := \max\{w_e | e \text{ was visited by } S\}$
  S requires at most $N + w_S$
  S requires at least $\max\{N, w_S\}$



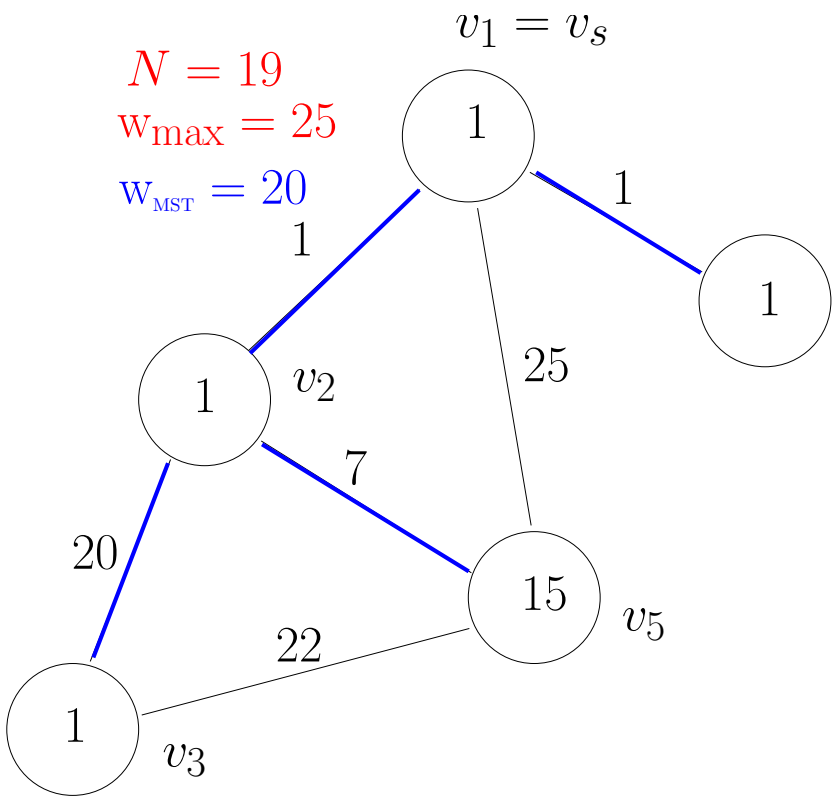
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$
- $N + w_{\max}$ on G is sufficient!
- Strategy S:
  $w_S := \max\{w_e | e \text{ was visited by } S\}$
  S requires at most $N + w_S$
  S requires at least $\max\{N, w_S\}$
- Minimum Spanning Tree (MST),
  $w_{\text{MST}} := \max\{w_e | e \in \text{ MST}\}$



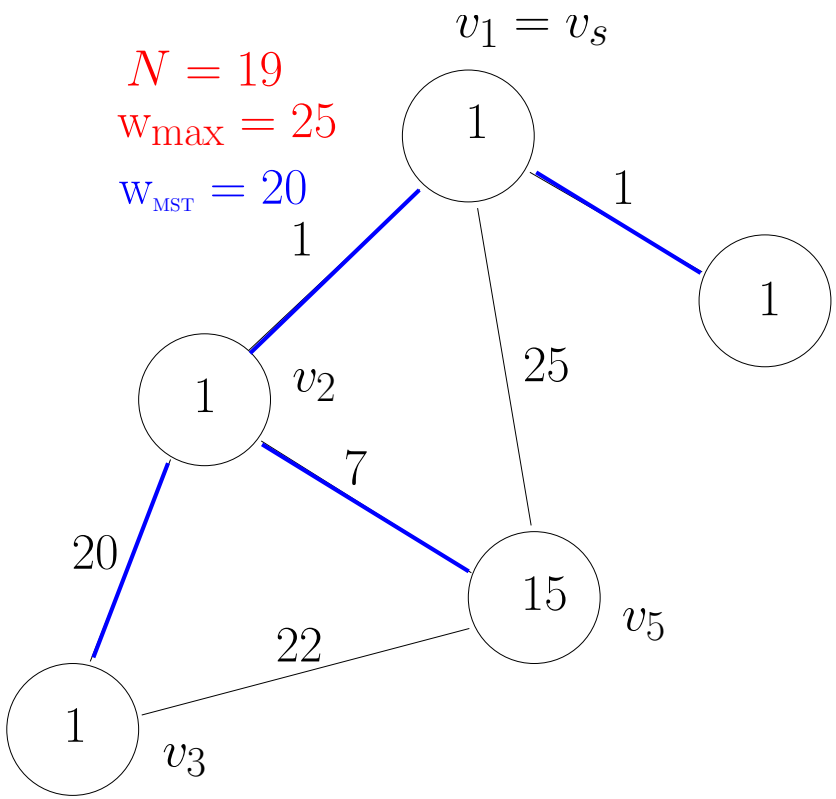
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$
- $N + w_{\max}$ on G is sufficient!
- Strategy S:
  $w_S := \max\{w_e | e \text{ was visited by } S\}$
  S requires at most $N + w_S$
  S requires at least $\max\{N, w_S\}$
- Minimum Spanning Tree (MST),
  $w_{\text{MST}} := \max\{w_e | e \in \text{MST}\}$



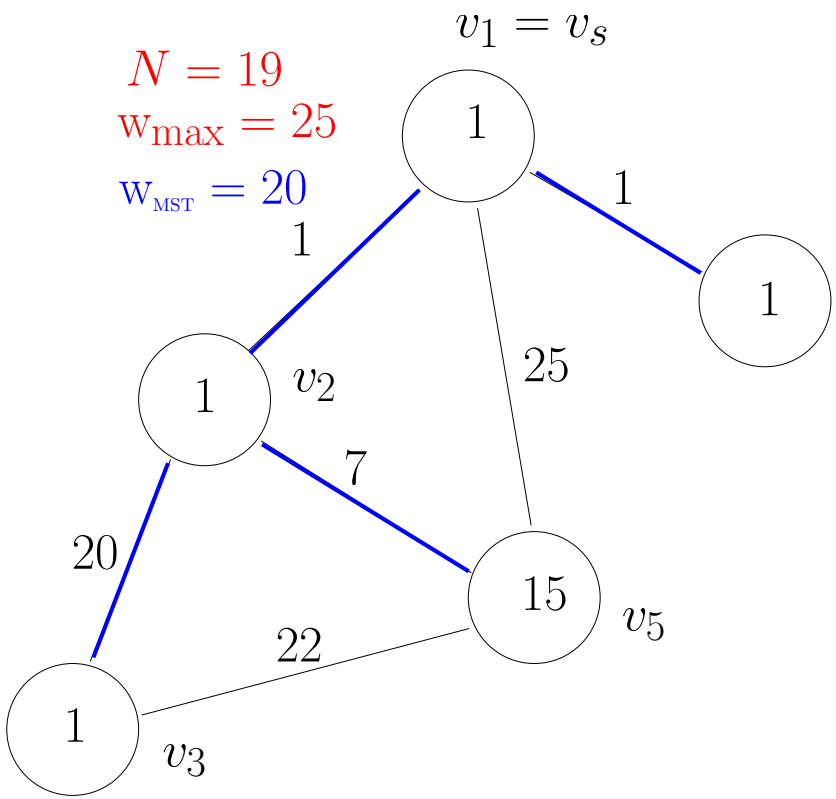
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$
- $N + w_{\max}$ on G is sufficient!
- Strategy S:
  $w_S := \max\{w_e | e \text{ was visited by } S\}$
  S requires at most $N + w_S$
  S requires at least $\max\{N, w_S\}$
- Minimum Spanning Tree (MST),
  $w_{\text{MST}} := \max\{w_e | e \in \text{ MST}\}$
- $N + w_{\text{MST}}$ on MST is sufficient



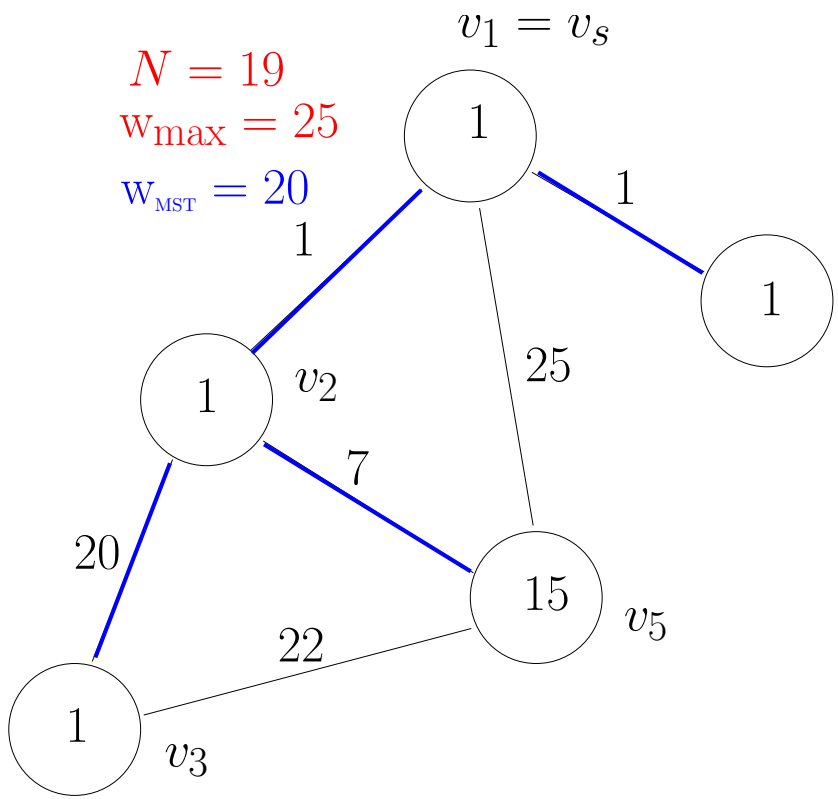
Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$
- $N + w_{\max}$ on G is sufficient!
- Strategy S:
  $w_S := \max\{w_e | e \text{ was visited by } S\}$
  S requires at most $N + w_S$
  S requires at least $\max\{N, w_S\}$
- Minimum Spanning Tree (MST),
  $w_{\text{MST}} := \max\{w_e | e \in \text{ MST}\}$
- $N + w_{\text{MST}}$ on MST is sufficient
  Any Strategy S on G requires
  at least $\max\{N, w_{\text{MST}}\}$



Number of agents required: Simple bounds!

- $G = (V, E)$, $N := \sum_{v \in V} w_v$
  $w_{\max} := \max\{w_e | e \in E\}$
- $N + w_{\max}$ on G is sufficient!
- Strategy S:
  $w_S := \max\{w_e | e \text{ was visited by } S\}$
  S requires at most $N + w_S$
  S requires at least $\max\{N, w_S\}$
- Minimum Spanning Tree (MST),
  $w_{\text{MST}} := \max\{w_e | e \in \text{MST}\}$
- $N + w_{\text{MST}}$ on MST is sufficient
  Any Strategy S on G requires
  at least $\max\{N, w_{\text{MST}}\}$
- **Lemma:** Optimal Strategy for
  MST gives 2-Approximation for G



Variants: Return or No-Return!

(No-return) It suffices to fill the vertices as required, no agents have to return to the start vertex.

Variants: Return or No-Return!

(No-return) It suffices to fill the vertices as required, no agents have to return to the start vertex.

(Return) Finally some agents have to return to the start vertex and report the success of the whole operation.

Variants: Return or No-Return!

(No-return) It suffices to fill the vertices as required, no agents have to return to the start vertex.

(Return) Finally some agents have to return to the start vertex and report the success of the whole operation.

Comparable to *routes* (round-trips) and *tours* (open paths) in TSP

Variants: Return or No-Return!

(No-return) It suffices to fill the vertices as required, no agents have to return to the start vertex.

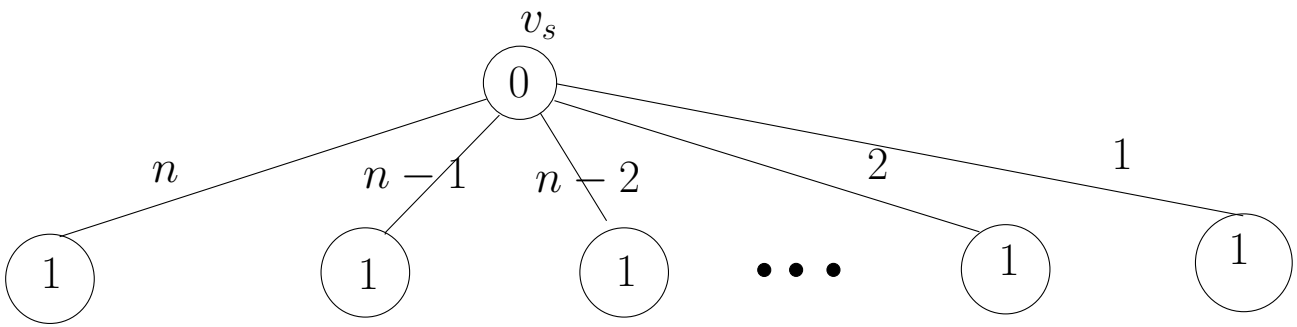
(Return) Finally some agents have to return to the start vertex and report the success of the whole operation.

Comparable to *routes* (round-trips) and *tours* (open paths) in TSP

Reporting the success formally means:
Set, M, of agents return to v_s, the union of *all* vertices visited by the members of M equals V.

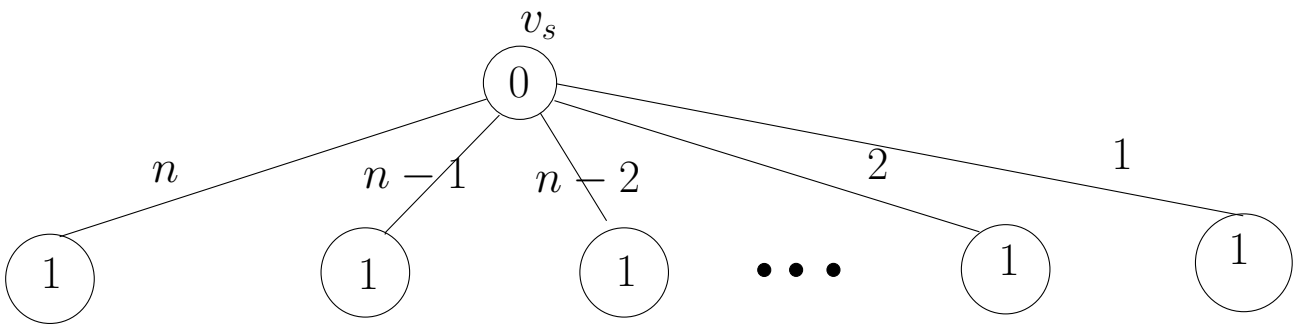
Optimal Algorithm for Trees: Return Variant

Computational lower bound and algorithmic idea! Example!



Optimal Algorithm for Trees: Return Variant

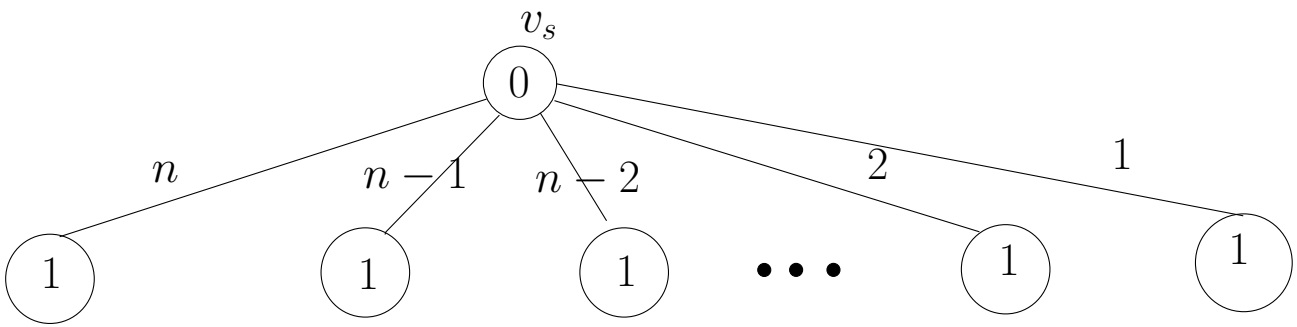
Computational lower bound and algorithmic idea! Example!



Optimal strategy: $n+1$ agents, visit vertices in order of decreasing edge weights: n,$n-1$,$n-2$,...,2,1

Optimal Algorithm for Trees: Return Variant

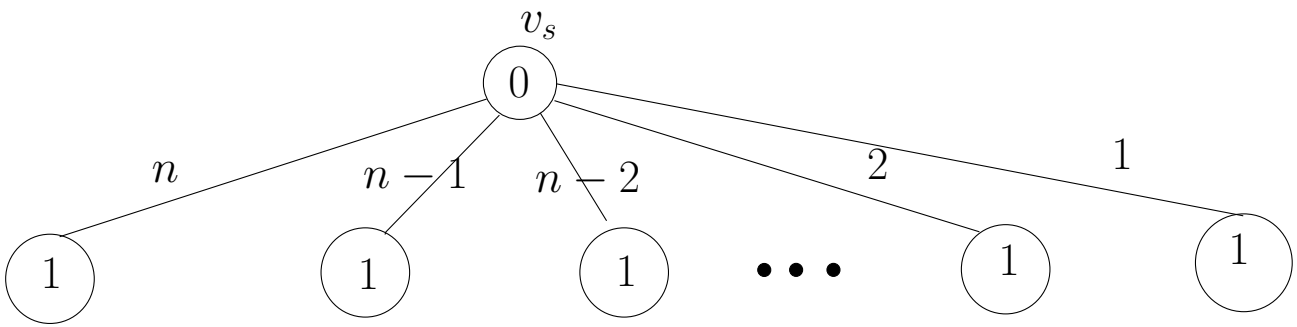
Computational lower bound and algorithmic idea! Example!



Optimal strategy: $n+1$ agents, visit vertices in order of decreasing edge weights: n,$n-1$,$n-2$,...,2,1
Any other order will increase the number!

Optimal Algorithm for Trees: Return Variant

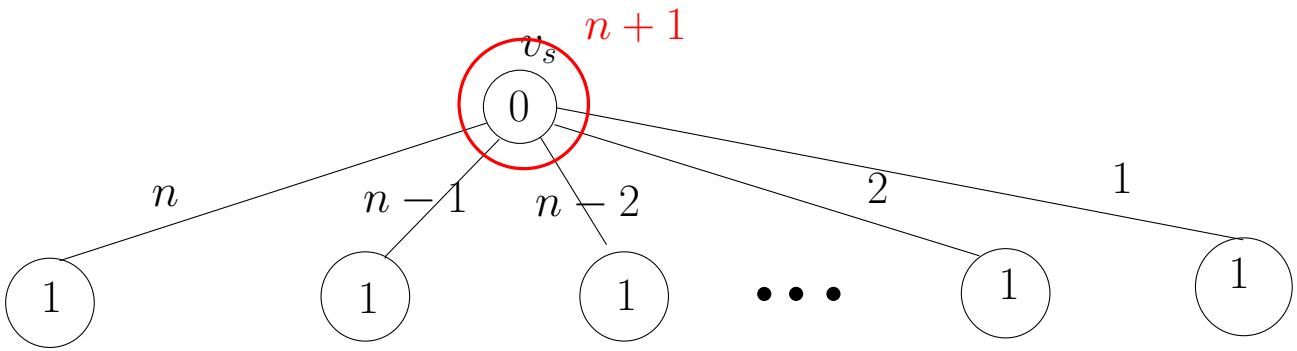
Computational lower bound and algorithmic idea! Example!



Optimal strategy: $n+1$ agents, visit vertices in order of decreasing edge weights: n,$n-1$,$n-2$,...,2,1
Any other order will increase the number!
Example: Visit $n-2$ before n

Optimal Algorithm for Trees: Return Variant

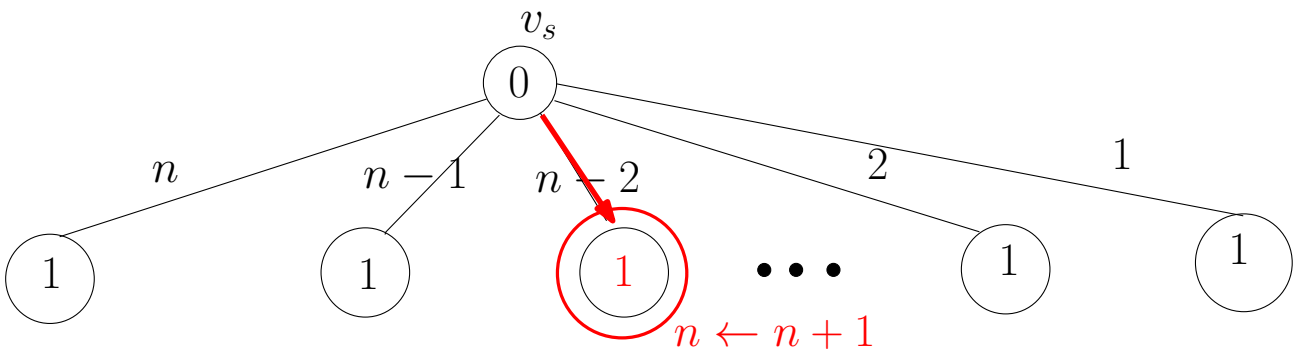
Computational lower bound and algorithmic idea! Example!



Optimal strategy: $n+1$ agents, visit vertices in order of decreasing edge weights: $n,n-1,n-2,\ldots,2,1$
Any other order will increase the number!
Example: Visit $n-2$ before n

Optimal Algorithm for Trees: Return Variant

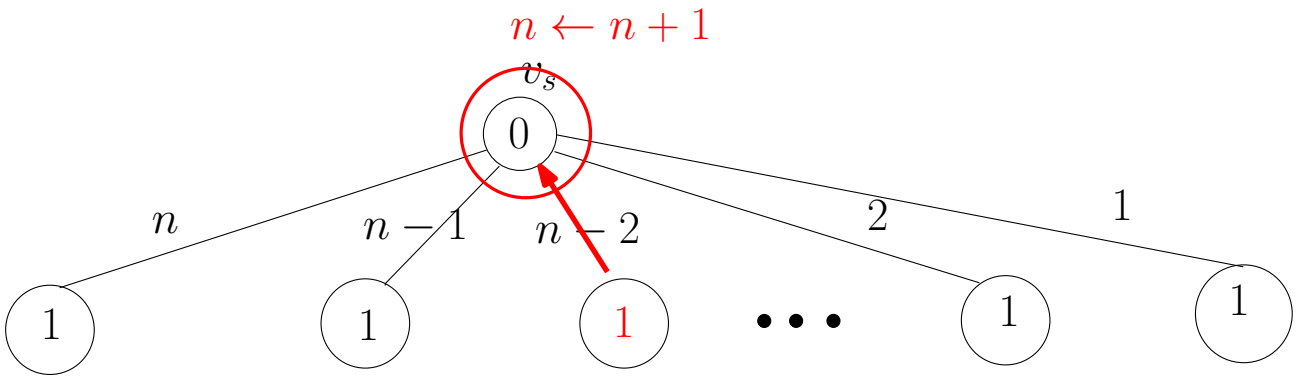
Computational lower bound and algorithmic idea! Example!



Optimal strategy: $n+1$ agents, visit vertices in order of decreasing edge weights: $n,n-1,n-2,\ldots,2,1$
Any other order will increase the number!
Example: Visit $n-2$ before n

Optimal Algorithm for Trees: Return Variant

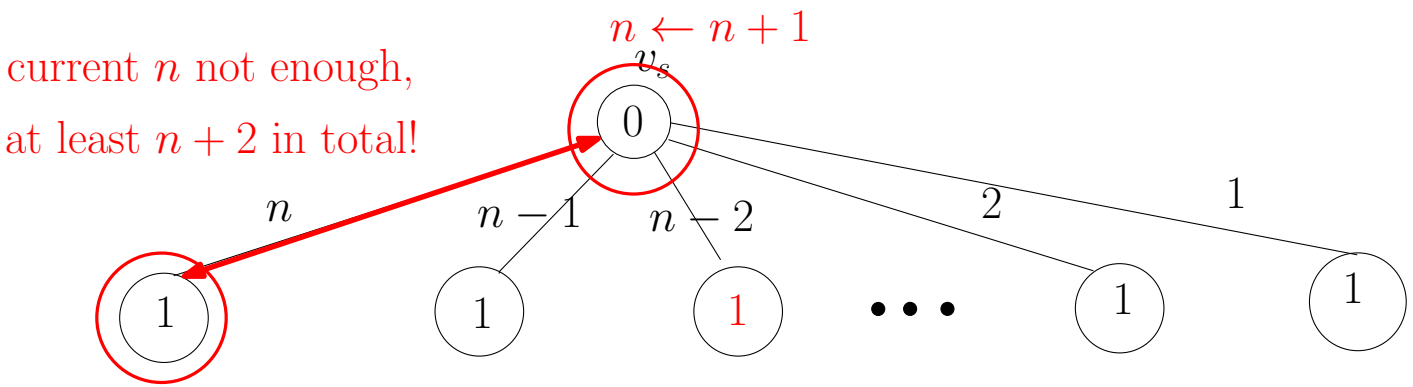
Computational lower bound and algorithmic idea! Example!



Optimal strategy: $n+1$ agents, visit vertices in order of decreasing edge weights: $n,n-1,n-2,\ldots,2,1$
Any other order will increase the number!
Example: Visit $n-2$ before n

Optimal Algorithm for Trees: Return Variant

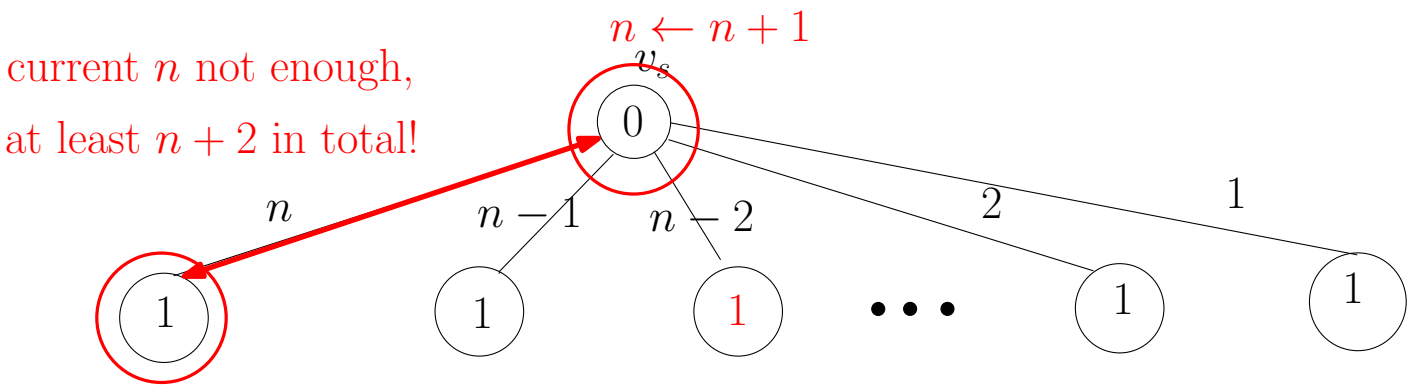
Computational lower bound and algorithmic idea! Example!



Optimal strategy: $n+1$ agents, visit vertices in order of decreasing edge weights: $n,n-1,n-2,\ldots,2,1$
Any other order will increase the number!
Example: Visit $n-2$ before n

Optimal Algorithm for Trees: Return Variant

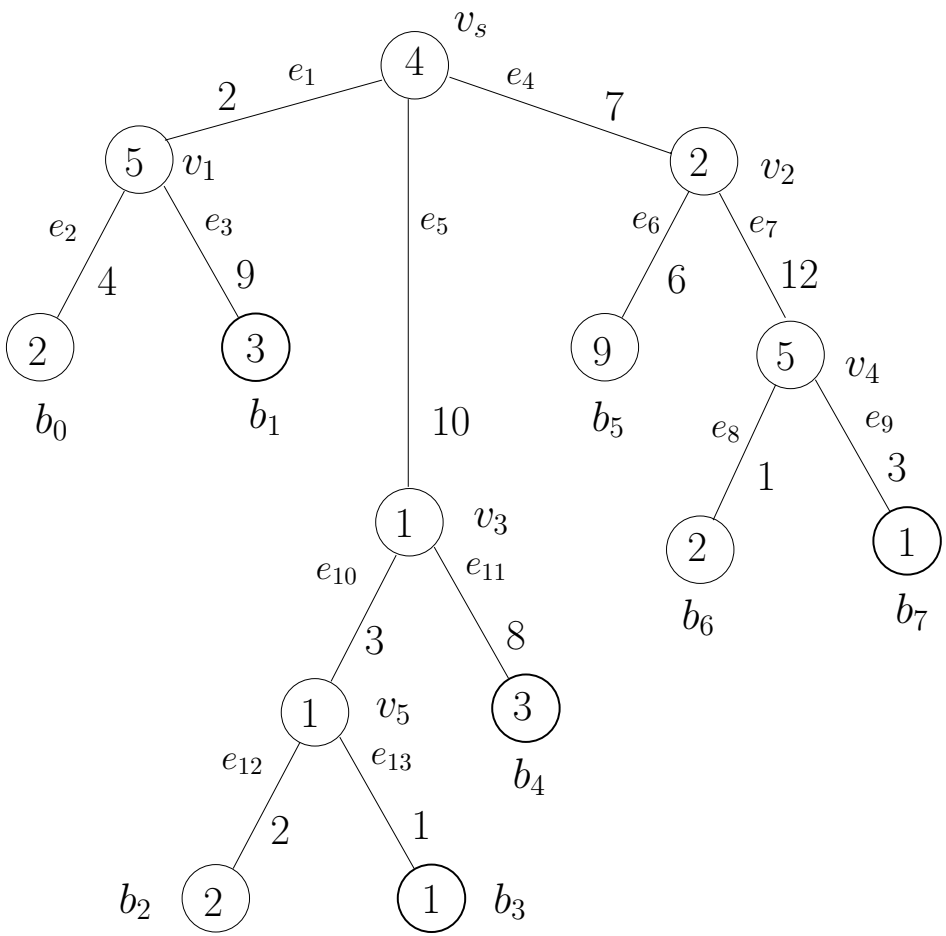
Computational lower bound and algorithmic idea! Example!



Optimal strategy: $n+1$ agents, visit vertices in order of decreasing edge weights: $n,n-1,n-2,\ldots,2,1$
Any other order will increase the number!
Example: Visit $n-2$ before n
Lemma: Computational lower bound $O(n \log n)$ by sorting (both variants, but real weights)!

$O(n \log n)$ Algorithm for Trees: Return Variant!

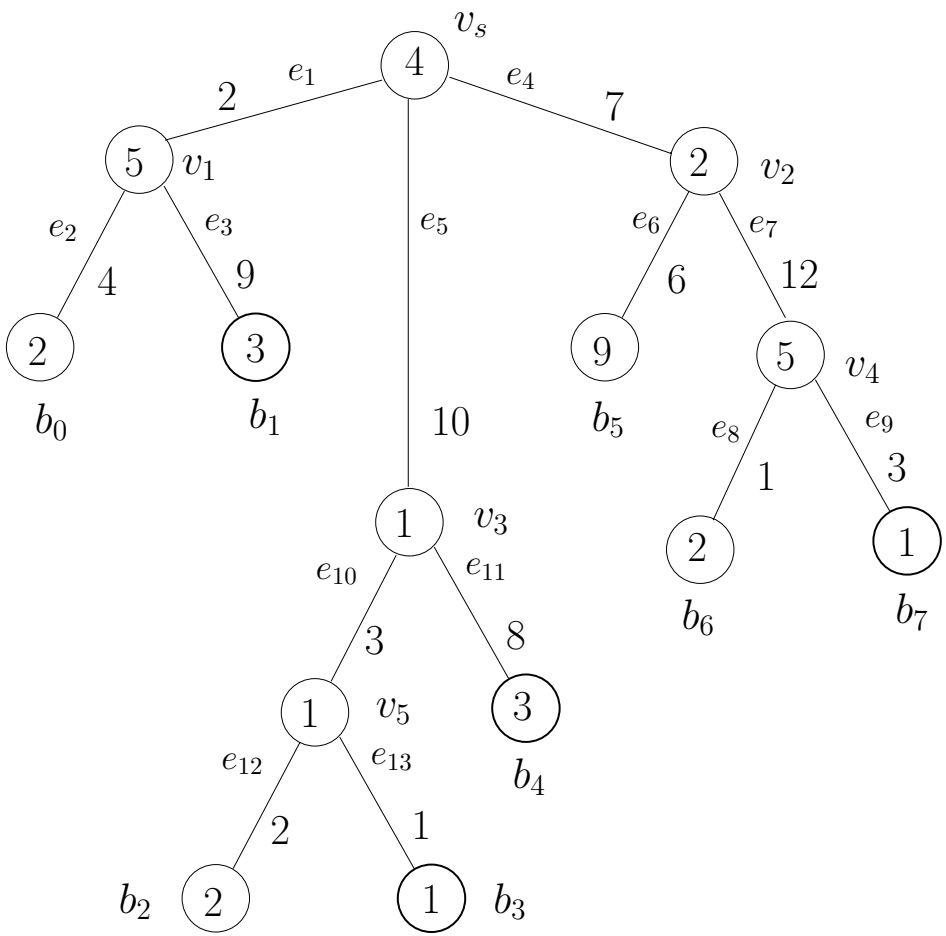
Have to visit all leafs and return!



$O(n \log n)$ Algorithm for Trees: Return Variant!

Have to visit all leafs and return!

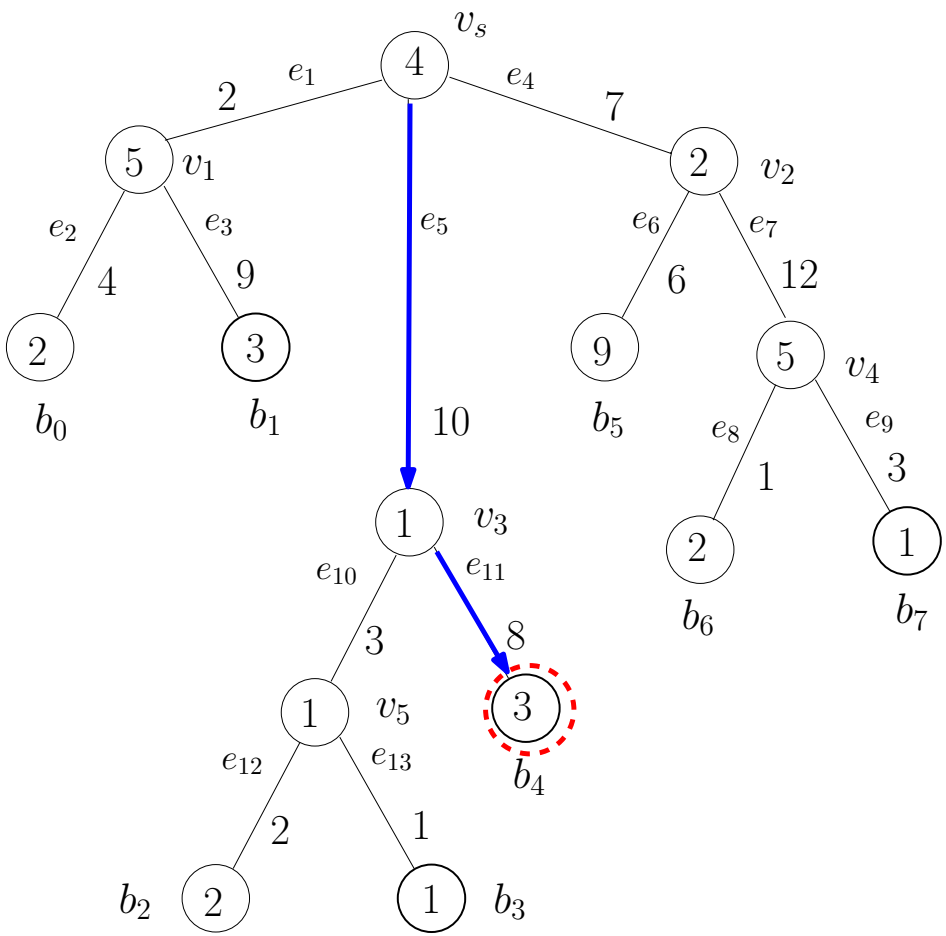
Collect the leafs in subtrees w.r.t. dominating edge along path!



$O(n \log n)$ Algorithm for Trees: Return Variant!

Have to visit all leafs and return!

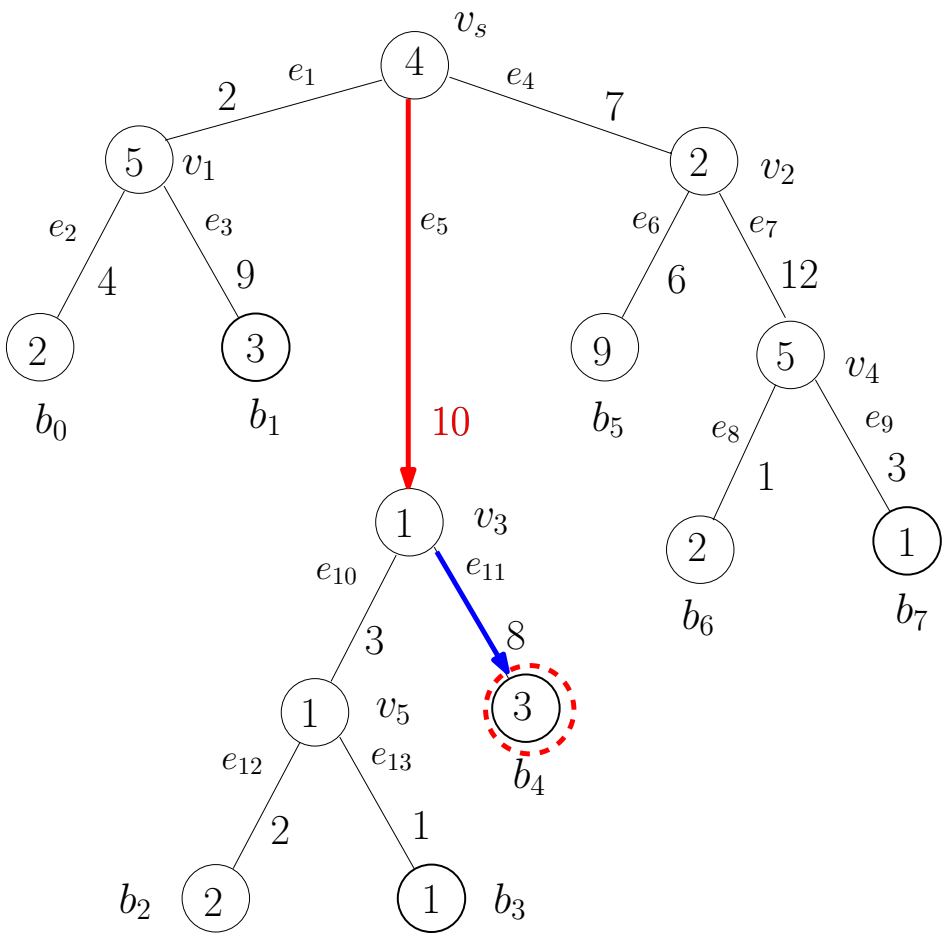
Collect the leafs in subtrees w.r.t. dominating edge along path!



$O(n \log n)$ Algorithm for Trees: Return Variant!

Have to visit all leafs and return!

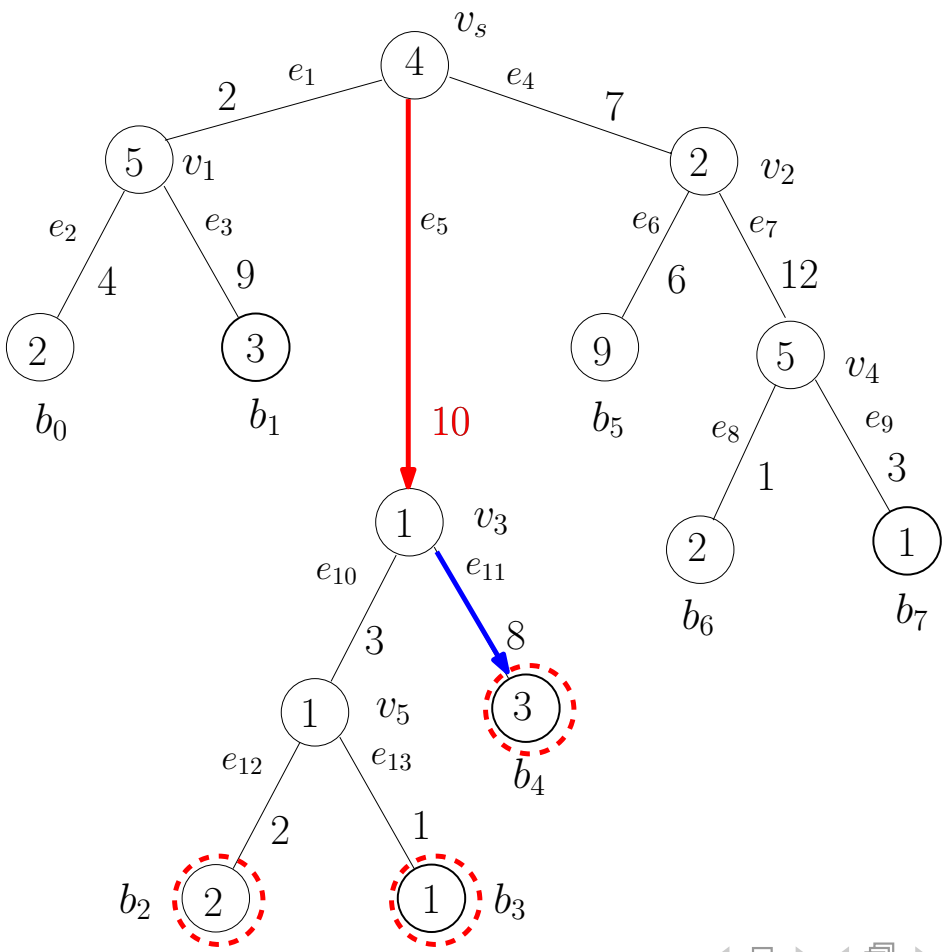
Collect the leafs in subtrees w.r.t. dominating edge along path!



$O(n \log n)$ Algorithm for Trees: Return Variant!

Have to visit all leafs and return!

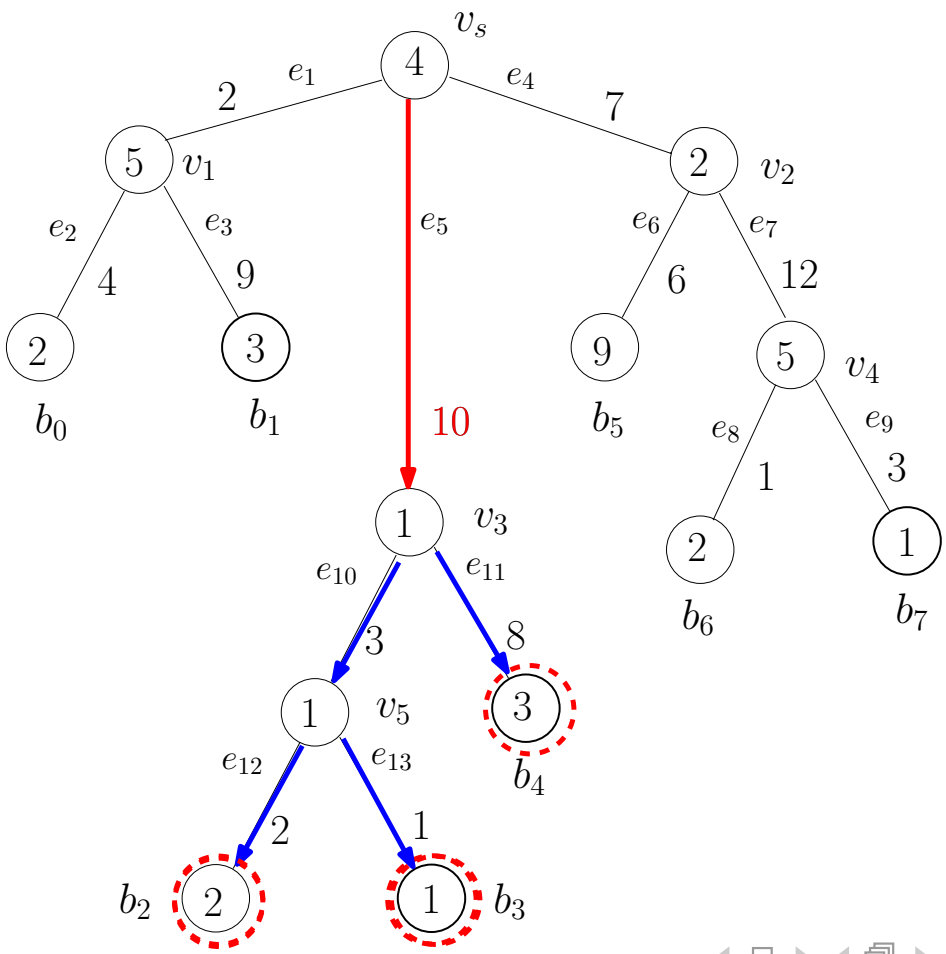
Collect the leafs in subtrees w.r.t. dominating edge along path!



$O(n \log n)$ Algorithm for Trees: Return Variant!

Have to visit all leafs and return!

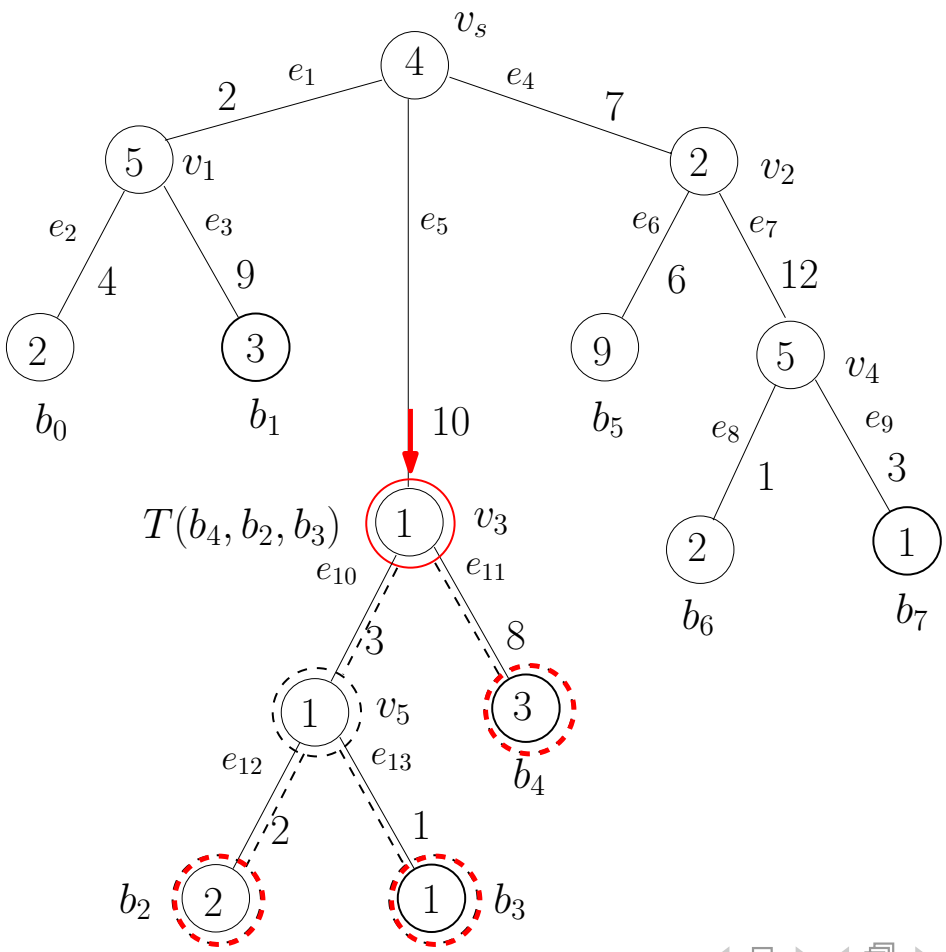
Collect the leafs in subtrees w.r.t. dominating edge along path!



$O(n \log n)$ Algorithm for Trees: Return Variant!

Have to visit all leafs and return!

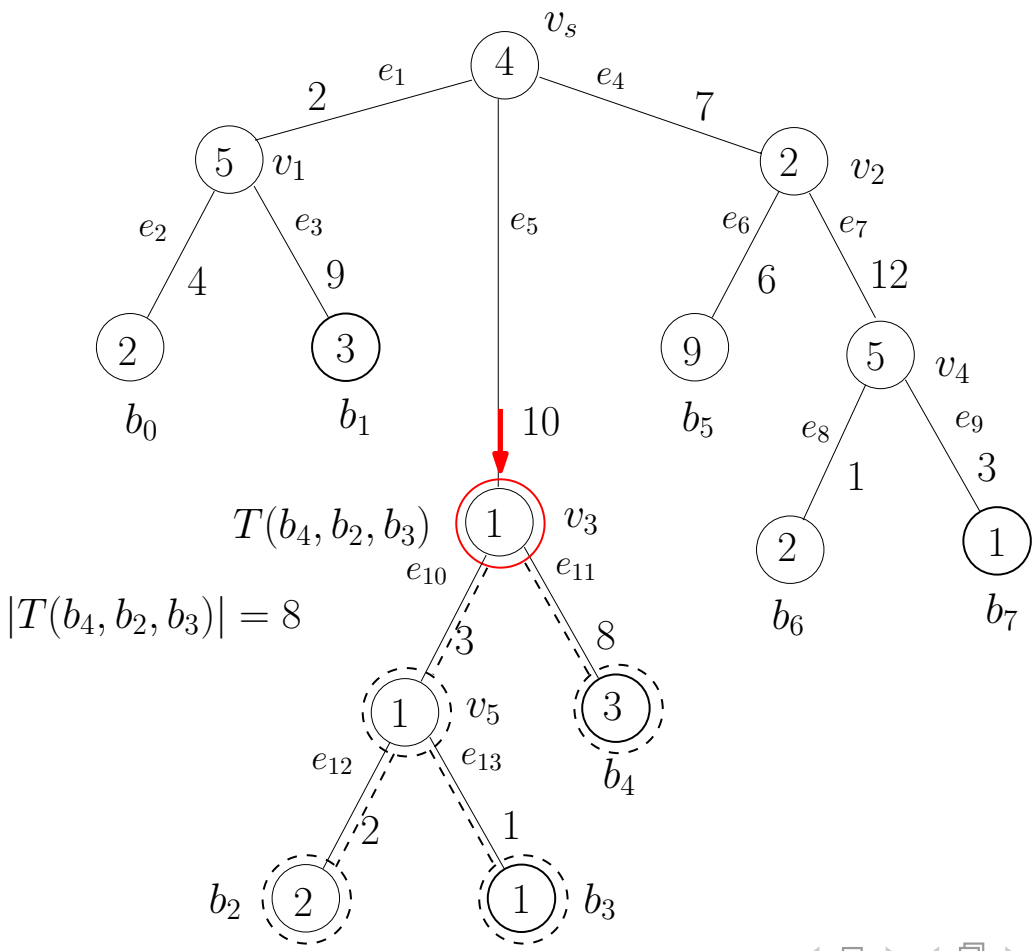
Collect the leafs in subtrees w.r.t. dominating edge along path!



$O(n \log n)$ Algorithm for Trees: Return Variant!

Have to visit all leafs and return!

Collect the leafs in subtrees w.r.t. dominating edge along path!

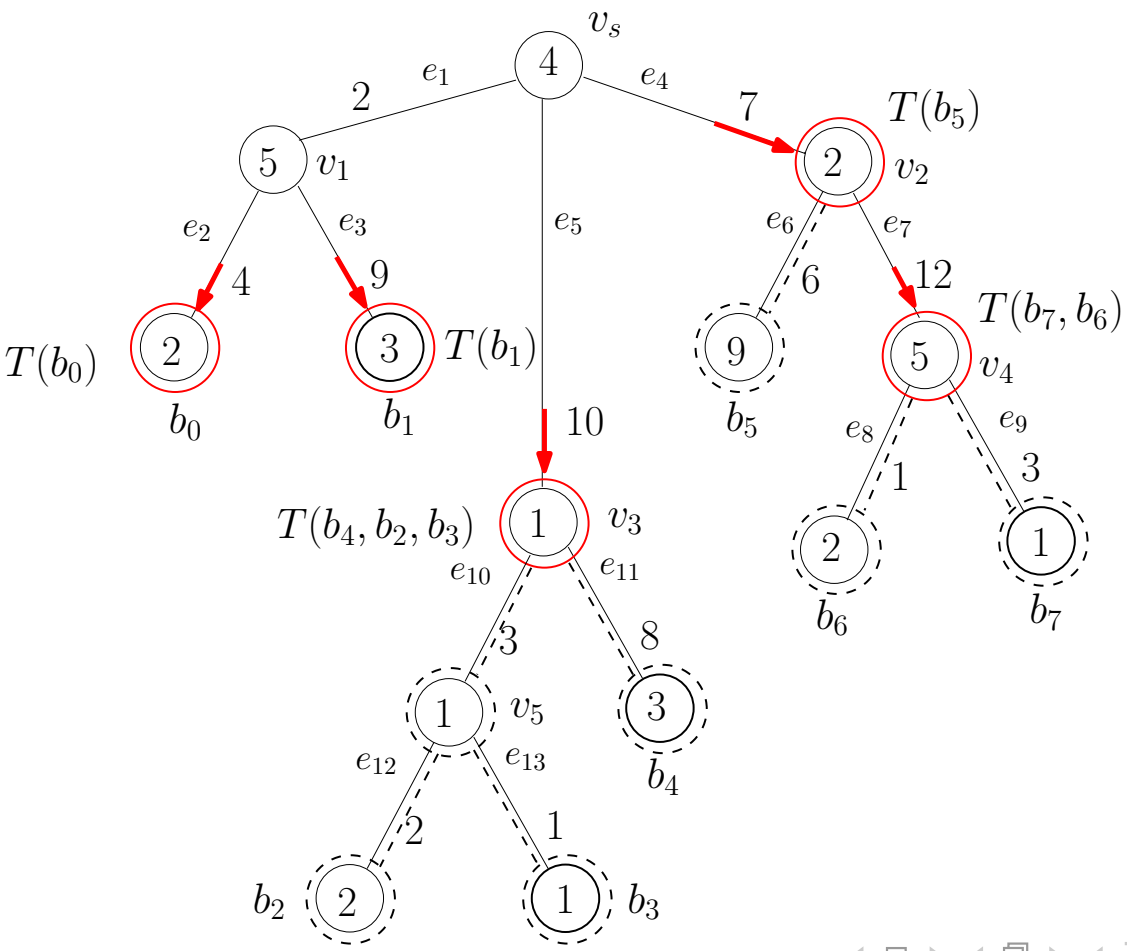


$O(n \log n)$ Algorithm for Trees: Return Variant!

Have to visit all leafs and return!

Collect the leafs in subtrees w.r.t. dominating edge along path!

Optimal strategy: Collected subtrees in decreasing of dominating edges. Simple DFS inside the coll. subtrees!

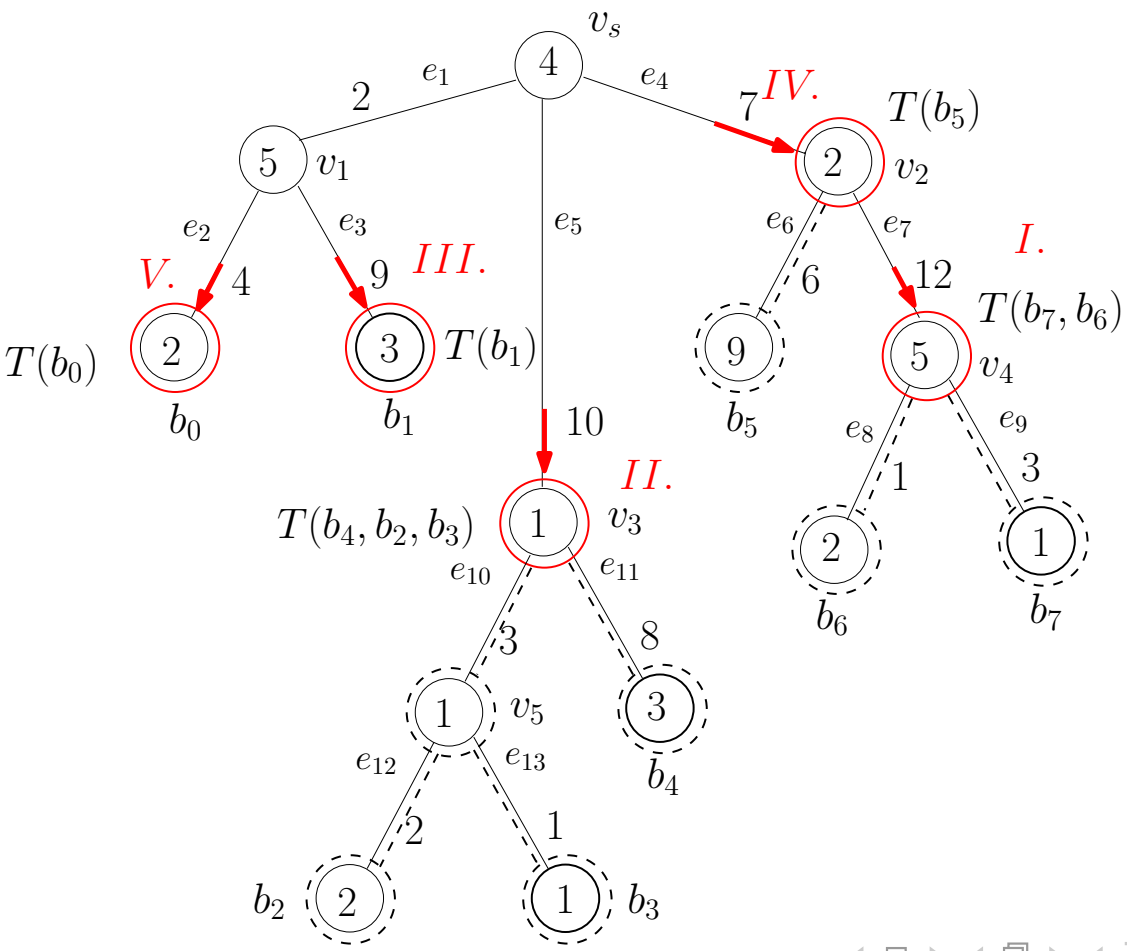


$O(n \log n)$ Algorithm for Trees: Return Variant!

Have to visit all leafs and return!

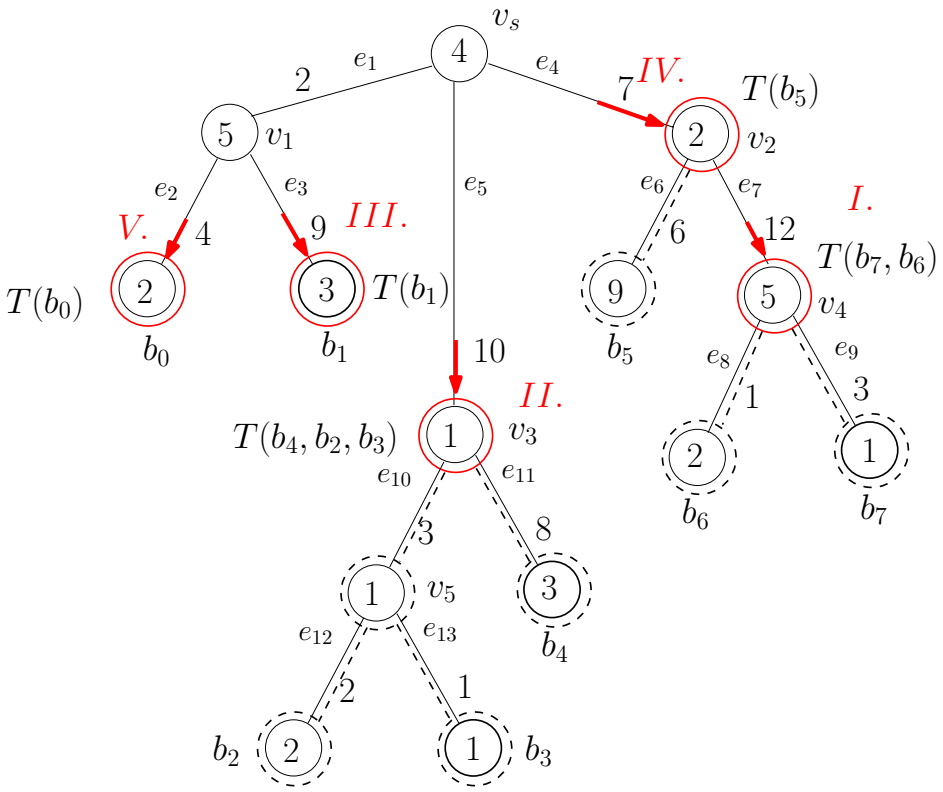
Collect the leafs in subtrees w.r.t. dominating edge along path!

Optimal strategy: Collected subtrees in decreasing of dominating edges. Simple DFS inside the coll. subtrees!



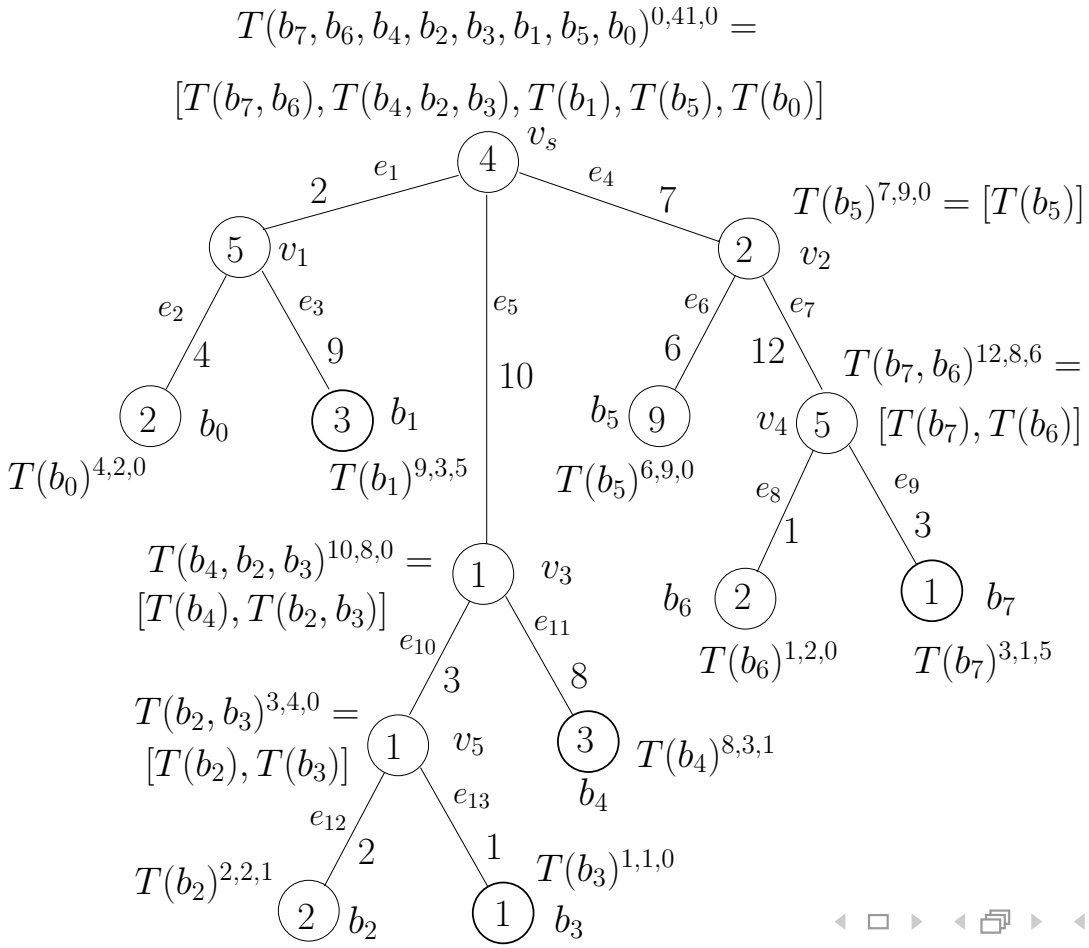
$O(n \log n)$ Algorithm for Trees: Return Variant!

Theorem: The optimal strategy for the return variant in a tree visits (and then fully explores (by DFS)) the collected subtrees in the order of the dominating edges weights. The number of required agents can be computed in $O(n \log n)$ time.



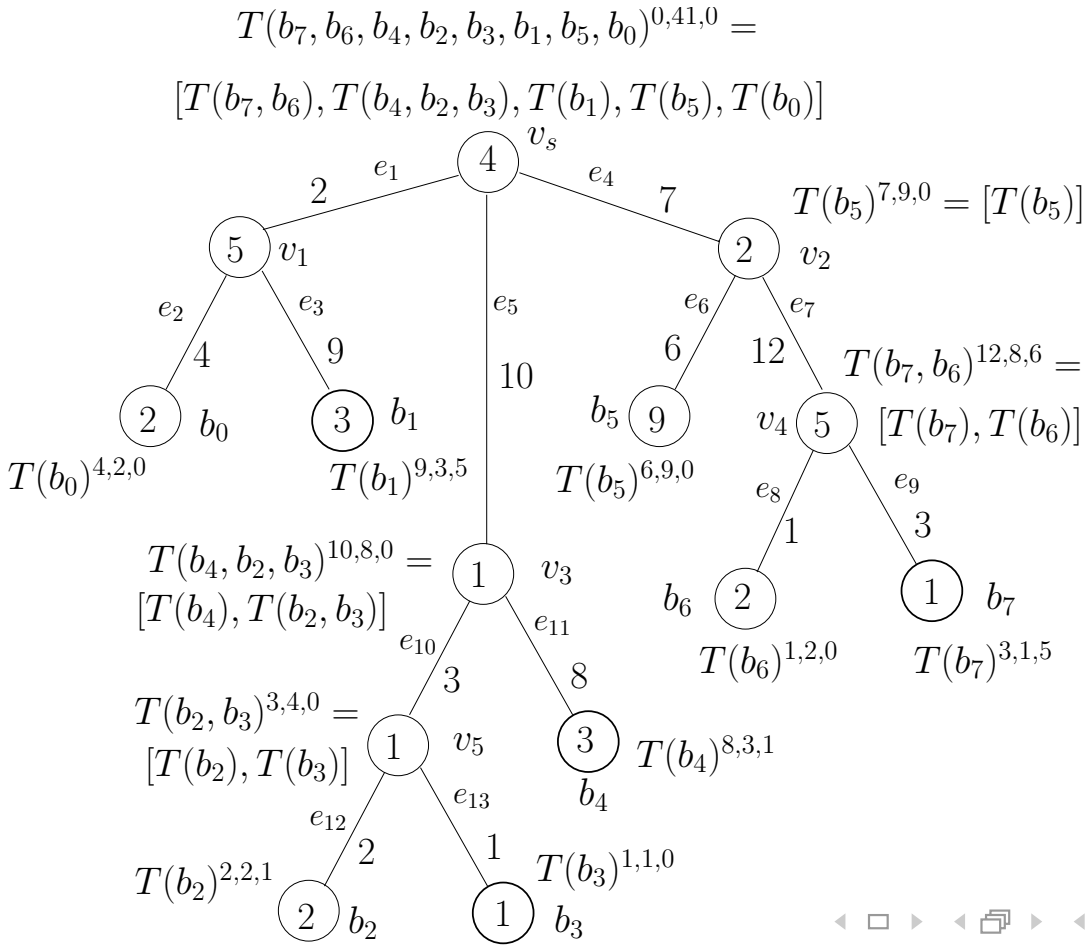
Amortized $O(n \log n)$ Algorithm for No-Return Variant!

- Important question: In which leaf b should we finally end?



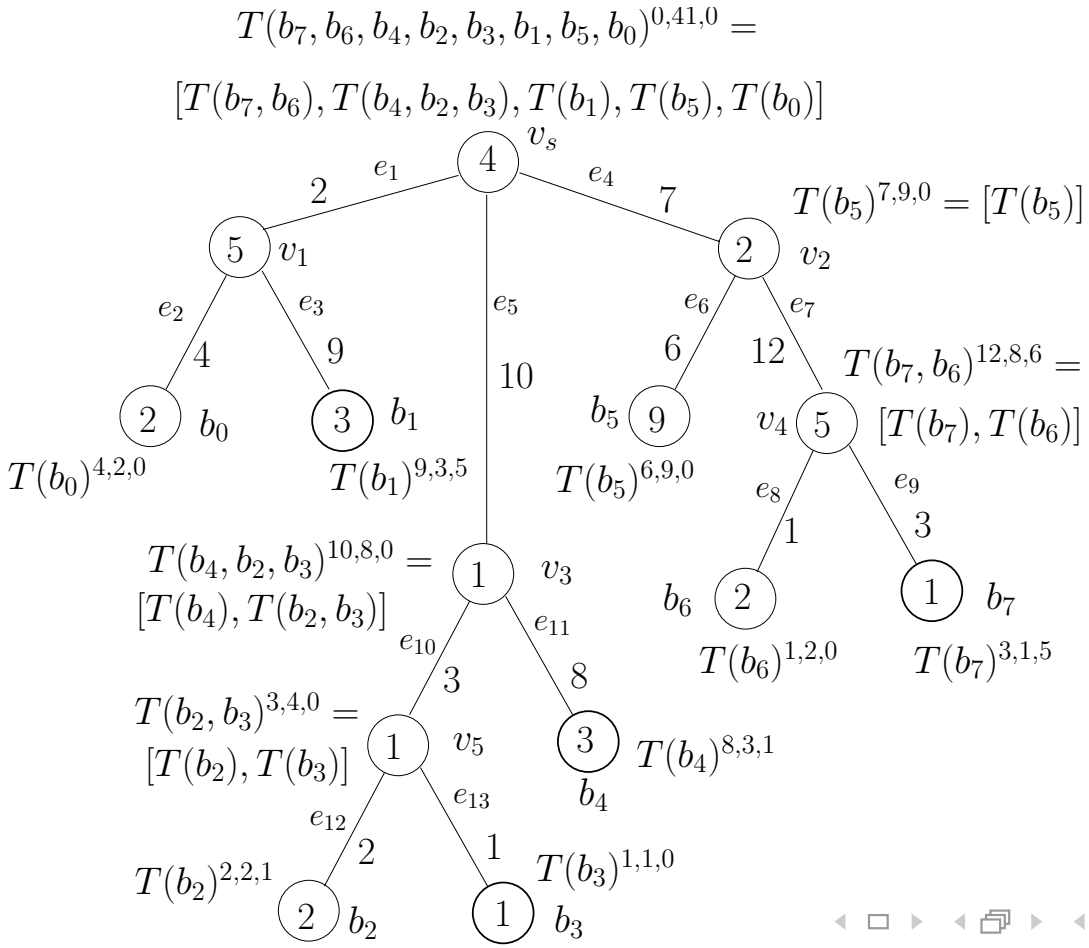
Amortized $O(n \log n)$ Algorithm for No-Return Variant!

- Important question: In which leaf b should we finally end?
- Recursively: Choose collected subtree visited last, remaining subtrees in decreasing order.



Amortized $O(n \log n)$ Algorithm for No-Return Variant!

- Important question: In which leaf b should we finally end?
- Recursively: Choose collected subtree visited last, remaining subtrees in decreasing order.
- Datastructure comparing all alternatives, amortized $O(n \log n)$



Structural Theorem! Move in a single group!

Theorem: For any strategy of the strategic deployment problem on a graph G and the minimum number of agents required there is always an optimal strategy that let the optimal number of agents run in a single group!

NP-hard for Graphs: 3-ExactCover Reduction

Given: Finite set X of $3n$ items and a collection $F = \{F_1, F_2, \ldots, F_m\}$ of subsets m of X with $|F_i| = 3$.
Question: Is there a subset $F_c \subseteq F$ with $X = \bigcup_{F_i \in F_c} F_i$ and $|F_c| = n$?

NP-hard for Graphs: 3-ExactCover Reduction

Given: Finite set X of $3n$ items and a collection $F = \{F_1, F_2, \ldots, F_m\}$ of subsets m of X with $|F_i| = 3$.

Question: Is there a subset $F_c \subseteq F$ with $X = \bigcup_{F_i \in F_c} F_i$ and $|F_c| = n$?

Example: $X = \{a_1, a_2, \ldots, a_{12}\}$

$F_1 = \{a_1, a_2, a_3\}, F_2 = \{a_1, a_2, a_4\}, \ldots, F_6 = \{a_9, a_{11}, a_{12}\}$

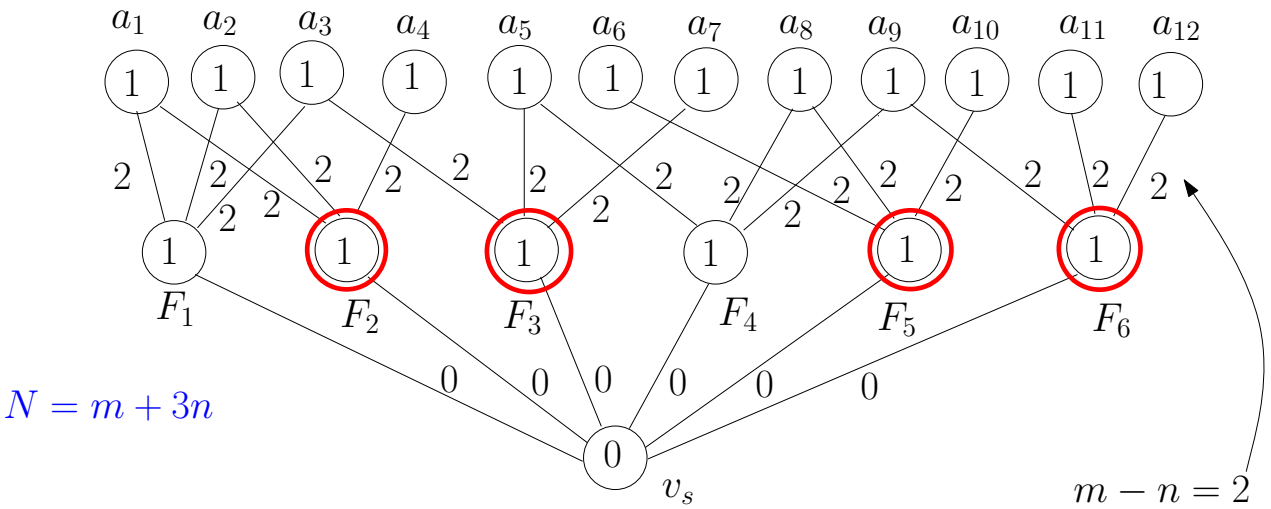
NP-hard for Graphs: 3-ExactCover Reduction

Given: Finite set X of $3n$ items and a collection $F = \{F_1, F_2, \ldots, F_m\}$ of subsets m of X with $|F_i| = 3$.

Question: Is there a subset $F_c \subseteq F$ with $X = \bigcup_{F_i \in F_c} F_i$ and $|F_c| = n$?

Example: $X = \{a_1, a_2, \ldots, a_{12}\}$

$F_1 = \{a_1, a_2, a_3\}, F_2 = \{a_1, a_2, a_4\}, \ldots, F_6 = \{a_9, a_{11}, a_{12}\}$



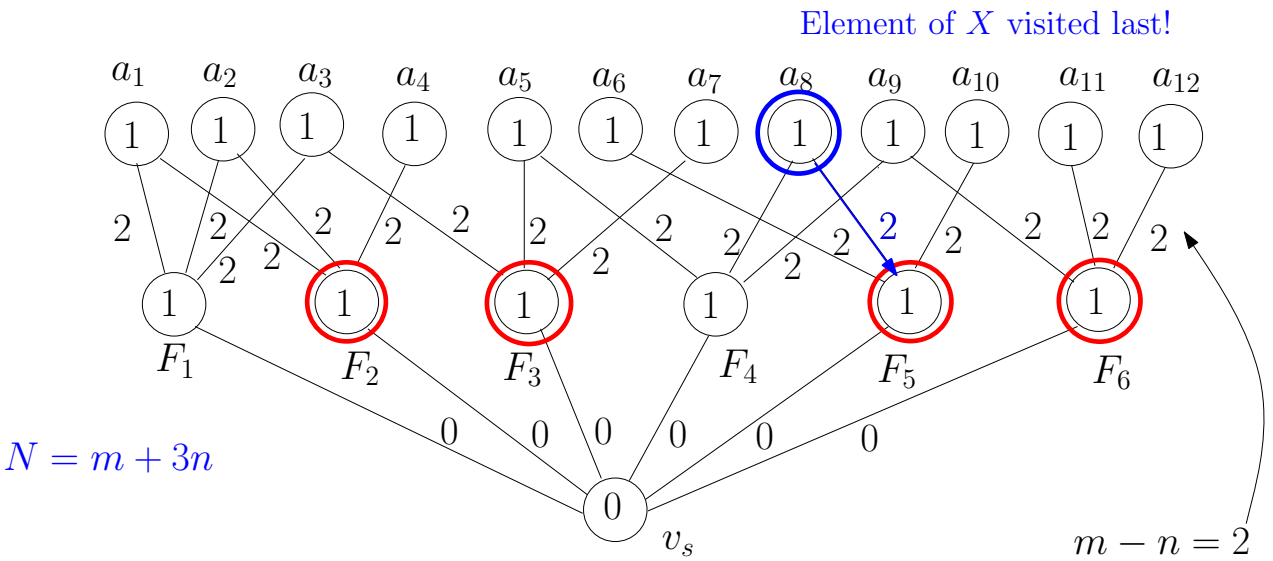
NP-hard for Graphs: 3-ExactCover Reduction

Given: Finite set X of $3n$ items and a collection $F = \{F_1, F_2, \ldots, F_m\}$ of subsets m of X with $|F_i| = 3$.

Question: Is there a subset $F_c \subseteq F$ with $X = \bigcup_{F_i \in F_c} F_i$ and $|F_c| = n$?

Example: $X = \{a_1, a_2, \ldots, a_{12}\}$

$F_1 = \{a_1, a_2, a_3\}, F_2 = \{a_1, a_2, a_4\}, \ldots, F_6 = \{a_9, a_{11}, a_{12}\}$



Conclusion

- Novel deployment problem on graphs with security constraints
- Many interesting variants and questions
- Optimal number of agents
  - NP-hard in general
  - 2-Approximation by MST
  - Optimal solution for trees in $\Theta(n \log n)$ (both variants!)
- Open questions: Combined measures (steps/number), Better approximations
- Joint work with Bernd Brüggemann (FKIE) and Andreas Lenerz (Univers. Bonn)