Leader Election in d-dimensional CA in time $o(diam^{1+\varepsilon})$

Michael Stratmann Thomas Worsch
LIIN, Fakultät für Informatik, Universität Karlsruhe

Leader Election in d-dimensional CA in time $O(diam \cdot (\log diam)^{d+1})$

Michael Stratmann Thomas Worsch
LIIN, Fakultät für Informatik, Universität Karlsruhe

Leader Election in d-dimensional CA in time $O(diam \cdot \log diam)$

Michael Stratmann Thomas Worsch
LIIN, Fakultät für Informatik, Universität Karlsruhe

Overview

- Problem
- History
- Basic Idea
- Algorithm: selected aspects
  - overview
  - counters
  - graph
  - time complexity

Overview

- Problem
- History
- Basic Idea
- Algorithm: selected aspects
  - overview
  - counters
  - graph
  - time complexity

Overview

- Problem
- History
- Basic Idea
- Algorithm: selected aspects
  - overview
  - counters
  - graph
  - time complexity

Overview

- Problem
- History
- Basic Idea
- Algorithm: selected aspects
  - overview
  - counters
  - graph
  - time complexity

Problem

Given: d-dimensional grid, a subset of states {yellow, black, red}

Problem instance: initial configuration with

- a finite subset of yellow cells,
  connected with respect to von Neumann neighborhood
- surrounded by black cells
  not taking part in the computation

Task: Find a set of states and a local rule for a CA such that always exactly one cell is designated as *leader* or *queen bee*.

Problem

Given: d-dimensional grid, a subset of states {yellow, black, red}

Problem instance: initial configuration with

- a finite subset of yellow cells,
  connected with respect to von Neumann neighborhood
- surrounded by black cells
  not taking part in the computation

Task: Find a set of states and a local rule for a CA such that always exactly one cell is designated as *leader* or *queen bee*.

Problem

Given: d-dimensional grid, a subset of states {yellow, black, red}

Problem instance: initial configuration with

- a finite subset of yellow cells,
  connected with respect to von Neumann neighborhood
- surrounded by black cells
  not taking part in the computation

Task: Find a set of states and a local rule for a CA such that always exactly one cell is designated as *leader* or *queen bee*.

History

- Smith (1971): *Queen Bee Problem*,
  2-dim., $\mathrm{O}\left(peri\right) \subseteq \mathrm{O}\left(vol\right) \subseteq \mathrm{O}\left(diam^2\right)$ (correct?)
- Vollmar (1979): simplified problem, 2-dim., $\mathrm{O}\left(peri\right)$
- Hemmerling (1979): d-dim., $\mathrm{O}\left(vol^5\right)$
- Beckers (1995): 2-dim., $\mathrm{O}\left(peri\right)$
- Mazoyer/Rémila (1998): 2-dim., $\mathrm{O}\left(peri\right)$
- Nichitiu/Rémila (1999): d-dim., $\mathrm{O}\left(diam^2\right)$
- Stratmann/Worsch (2000): d-dim., $\mathrm{O}\left(diam \cdot \log diam\right)$


History

- Smith (1971): *Queen Bee Problem*,
  2-dim., $\mathrm{O}(peri) \subseteq \mathrm{O}(vol) \subseteq \mathrm{O}(diam^2)$ (correct?)
- Vollmar (1979): simplified problem, 2-dim., $\mathrm{O}(peri)$
- Hemmerling (1979): d-dim., $\mathrm{O}(vol^5)$
- Beckers (1995): 2-dim., $\mathrm{O}(peri)$
- Mazoyer/Rémila (1998): 2-dim., $\mathrm{O}(peri)$
- Nichitiu/Rémila (1999): d-dim., $\mathrm{O}(diam^2)$
- Stratmann/Worsch (2000): d-dim., $\mathrm{O}(diam \cdot \log diam)$


History

- Smith (1971): *Queen Bee Problem*,
  2-dim., $\mathrm{O}\left(peri\right) \subseteq \mathrm{O}\left(vol\right) \subseteq \mathrm{O}\left(diam^2\right)$ (correct?)
- Vollmar (1979): simplified problem, 2-dim., $\mathrm{O}\left(peri\right)$
- Hemmerling (1979): d-dim., $\mathrm{O}\left(vol^5\right)$
- Beckers (1995): 2-dim., $\mathrm{O}\left(peri\right)$
- Mazoyer/Rémila (1998): 2-dim., $\mathrm{O}\left(peri\right)$
- Nichitiu/Rémila (1999): d-dim., $\mathrm{O}\left(diam^2\right)$
- Stratmann/Worsch (2000): d-dim., $\mathrm{O}\left(diam \cdot \log diam\right)$


History

- Smith (1971): *Queen Bee Problem*,
  2-dim., $\mathrm{O}(peri) \subseteq \mathrm{O}(vol) \subseteq \mathrm{O}(diam^2)$ (correct?)
- Vollmar (1979): simplified problem, 2-dim., $\mathrm{O}(peri)$
- Hemmerling (1979): d-dim., $\mathrm{O}(vol^5)$
- Beckers (1995): 2-dim., $\mathrm{O}(peri)$
- Mazoyer/Rémila (1998): 2-dim., $\mathrm{O}(peri)$
- Nichitiu/Rémila (1999): d-dim., $\mathrm{O}(diam^2)$
- Stratmann/Worsch (2000): d-dim., $\mathrm{O}(diam \cdot \log diam)$


History

- Smith (1971): *Queen Bee Problem*,
  2-dim., $\mathrm{O}\left(peri\right) \subseteq \mathrm{O}\left(vol\right) \subseteq \mathrm{O}\left(diam^2\right)$ (correct?)
- Vollmar (1979): simplified problem, 2-dim., $\mathrm{O}\left(peri\right)$
- Hemmerling (1979): d-dim., $\mathrm{O}\left(vol^5\right)$
- Beckers (1995): 2-dim., $\mathrm{O}\left(peri\right)$
- Mazoyer/Rémila (1998): 2-dim., $\mathrm{O}\left(peri\right)$
- Nichitiu/Rémila (1999): d-dim., $\mathrm{O}\left(diam^2\right)$
- Stratmann/Worsch (2000): d-dim., $\mathrm{O}\left(diam \cdot \log diam\right)$


History

- Smith (1971): *Queen Bee Problem*,
  2-dim., $\mathrm{O}(peri) \subseteq \mathrm{O}(vol) \subseteq \mathrm{O}(diam^2)$ (correct?)
- Vollmar (1979): simplified problem, 2-dim., $\mathrm{O}(peri)$
- Hemmerling (1979): d-dim., $\mathrm{O}(vol^5)$
- Beckers (1995): 2-dim., $\mathrm{O}(peri)$
- Mazoyer/Rémila (1998): 2-dim., $\mathrm{O}(peri)$
- Nichitiu/Rémila (1999): d-dim., $\mathrm{O}(diam^2)$
- Stratmann/Worsch (2000): d-dim., $\mathrm{O}(diam \cdot \log diam)$


Basic Idea

- Select some candidate cells in the first step.
- Use a total order on the candidates.
- Compare pairs of candidates, eliminating the "weaker" one.
- Elect the only remaining candidate as leader.

Most algorithms: lexicographic order on the coordinates,

$$\text{i.e.} \quad (x,y,z) \sqsubset (x',y',z') \quad \text{iff}$$
$$x < x' \quad \text{or} \quad x = x' \wedge y < y' \quad \text{or} \quad x = x' \wedge y = y' \wedge z < z'$$

Candidates: cells which are the strongest locally (von Neumann)



Basic Idea

- Select some candidate cells in the first step.
- Use a total order on the candidates.
- Compare pairs of candidates, eliminating the "weaker" one.
- Elect the only remaining candidate as leader.

Most algorithms: lexicographic order on the coordinates,

$$\text{i.e.} \quad (x,y,z) \sqsubset (x',y',z') \quad \text{iff}$$
$$x < x' \quad \text{or} \quad x = x' \wedge y < y' \quad \text{or} \quad x = x' \wedge y = y' \wedge z < z'$$

Candidates: cells which are the strongest locally (von Neumann)

Basic Idea

- Select some candidate cells in the first step.
- Use a total order on the candidates.
- Compare pairs of candidates, eliminating the "weaker" one.
- Elect the only remaining candidate as leader.

Most algorithms: lexicographic order on the coordinates,

$$\text{i.e.} \quad (x,y,z) \sqsubset (x',y',z') \quad \text{iff}$$
$$x < x' \quad \text{or} \quad x = x' \wedge y < y' \quad \text{or} \quad x = x' \wedge y = y' \wedge z < z'$$

Candidates: cells which are the strongest locally (von Neumann)

Basic Idea

- Select some candidate cells in the first step.
- Use a total order on the candidates.
- Compare pairs of candidates, eliminating the "weaker" one.
- Elect the only remaining candidate as leader.

Most algorithms: lexicographic order on the coordinates,

$$\text{i.e.} \quad (x,y,z) \sqsubset (x',y',z') \quad \text{iff}$$
$$x < x' \quad \text{or} \quad x = x' \wedge y < y' \quad \text{or} \quad x = x' \wedge y = y' \wedge z < z'$$

Candidates: cells which are the strongest locally (von Neumann)

Basic Idea

- Select some candidate cells in the first step.
- Use a total order on the candidates.
- Compare pairs of candidates, eliminating the "weaker" one.
- Elect the only remaining candidate as leader.

Most algorithms: lexicographic order on the coordinates,

$$\text{i.e.} \quad (x,y,z) \sqsubset (x',y',z') \quad \text{iff}$$
$$x < x' \quad \text{or} \quad x = x' \wedge y < y' \quad \text{or} \quad x = x' \wedge y = y' \wedge z < z'$$

Candidates: cells which are the strongest locally (von Neumann)

Basic Idea

- Select some candidate cells in the first step.
- Use a total order on the candidates.
- Compare pairs of candidates, eliminating the "weaker" one.
- Elect the only remaining candidate as leader.

Most algorithms: lexicographic order on the coordinates,

$$\text{i.e.} \quad (x,y,z) \sqsubset (x',y',z') \quad \text{iff}$$
$$x < x' \quad \text{or} \quad x = x' \wedge y < y' \quad \text{or} \quad x = x' \wedge y = y' \wedge z < z'$$

Candidates: cells which are the strongest locally (von Neumann)



Basic Idea

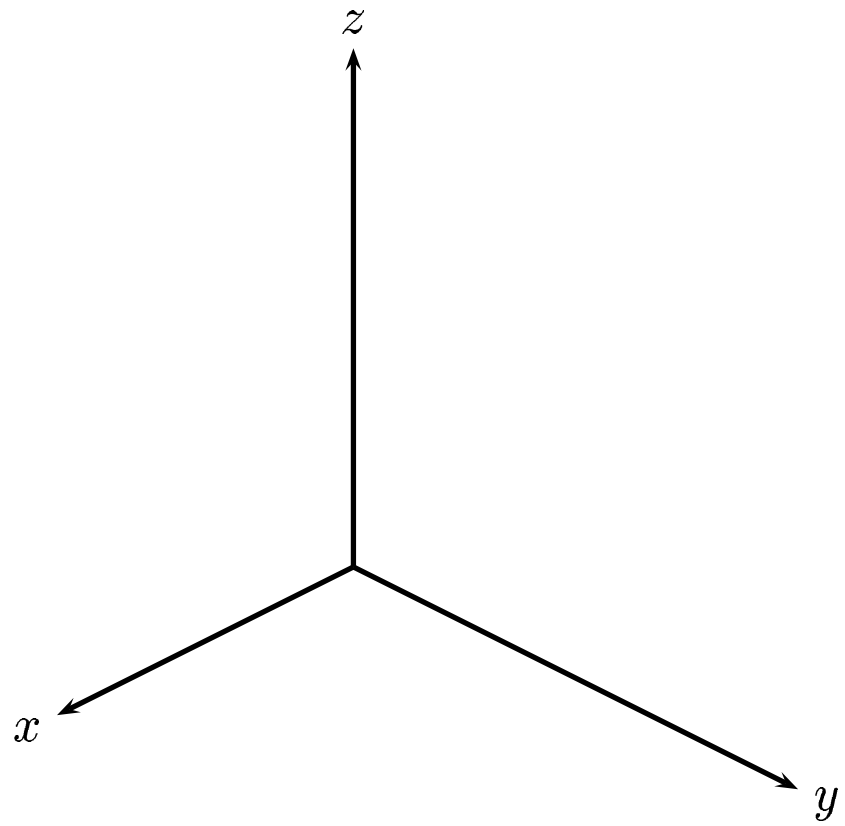
- Select some candidate cells in the first step.
- Use a total order on the candidates.
- Compare pairs of candidates, eliminating the "weaker" one.
- Elect the only remaining candidate as leader.

Most algorithms: lexicographic order on the coordinates,

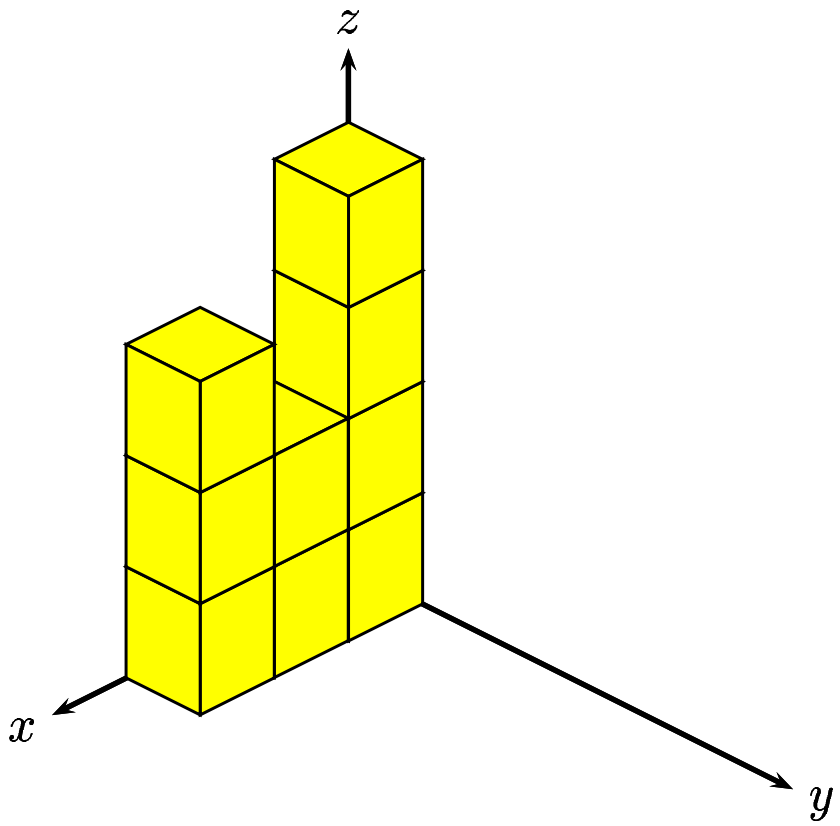
$$\text{i.e.} \quad (x,y,z) \sqsubset (x',y',z') \quad \text{iff}$$
$$x < x' \quad \text{or} \quad x = x' \wedge y < y' \quad \text{or} \quad x = x' \wedge y = y' \wedge z < z'$$

Candidates: cells which are the strongest locally (von Neumann)

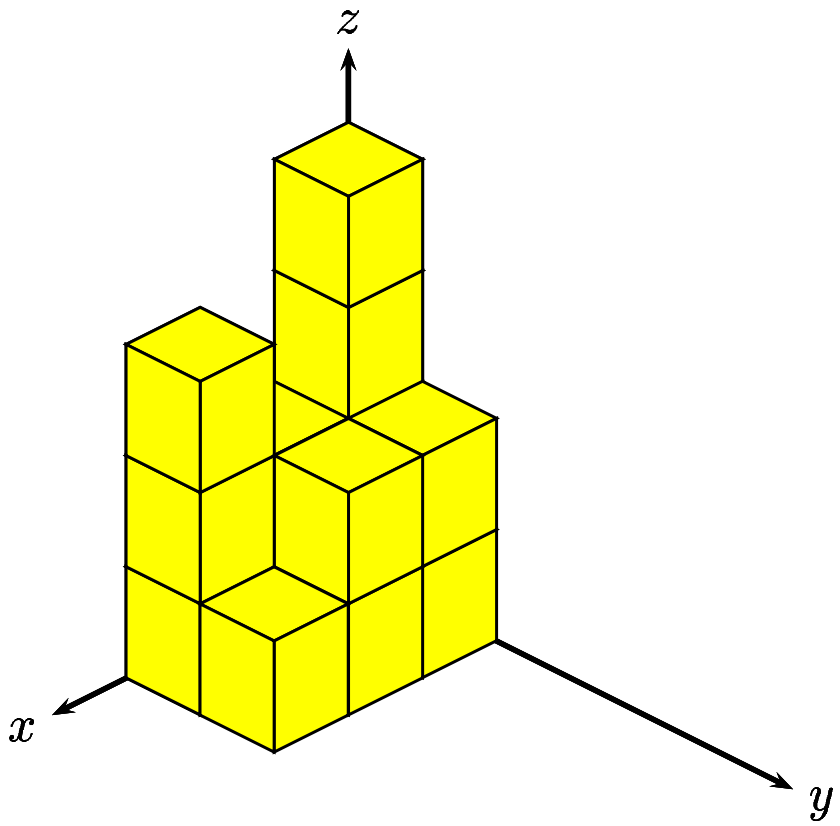
Pattern, Leader, Candidates



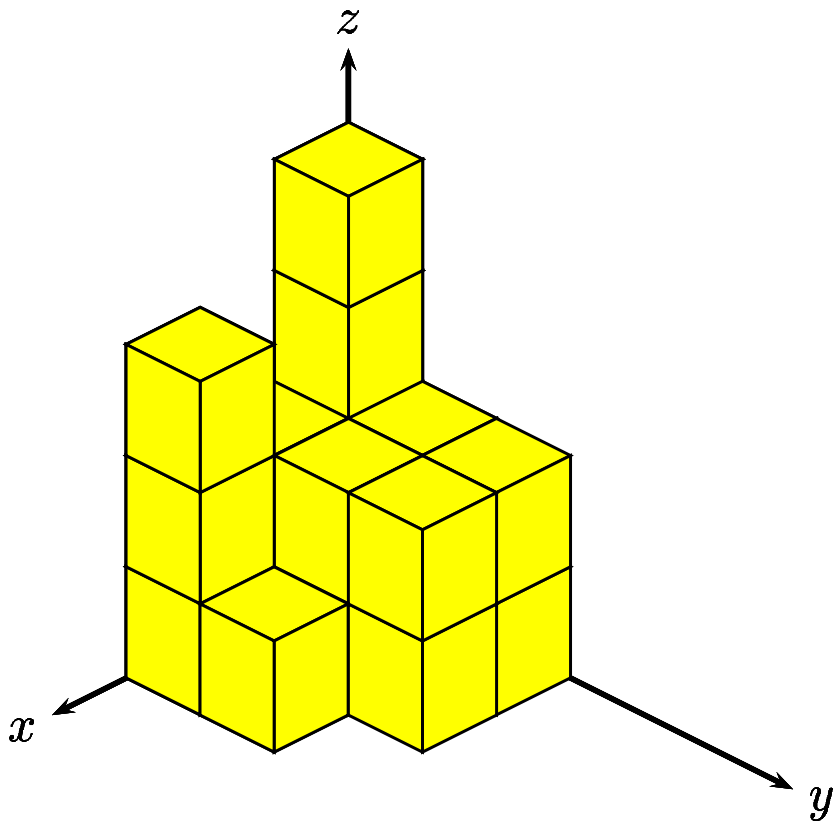
Pattern, Leader, Candidates



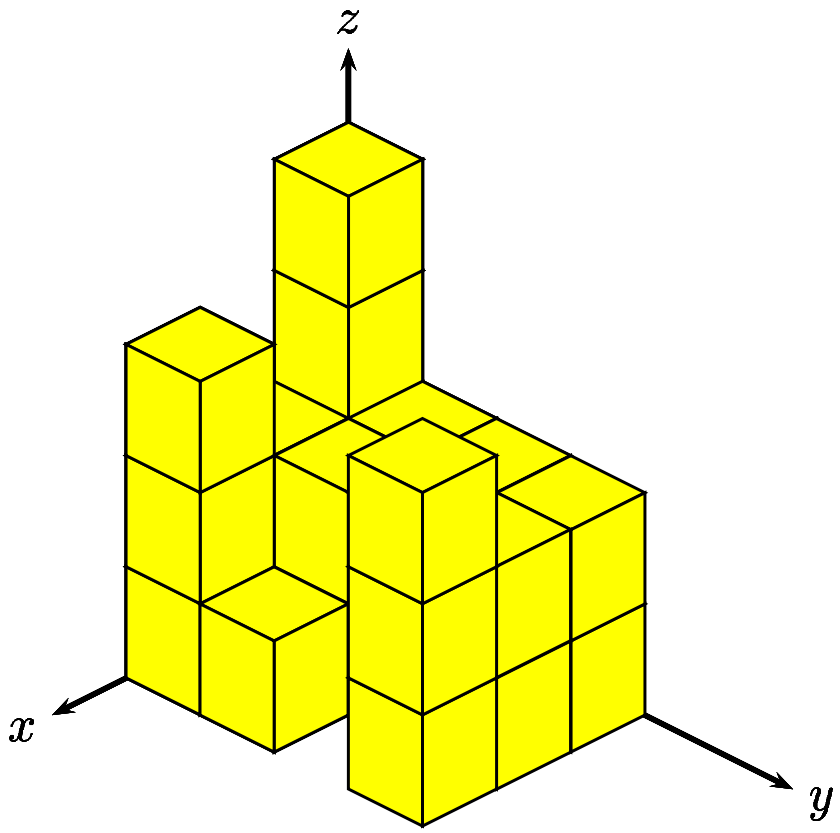
Pattern, Leader, Candidates



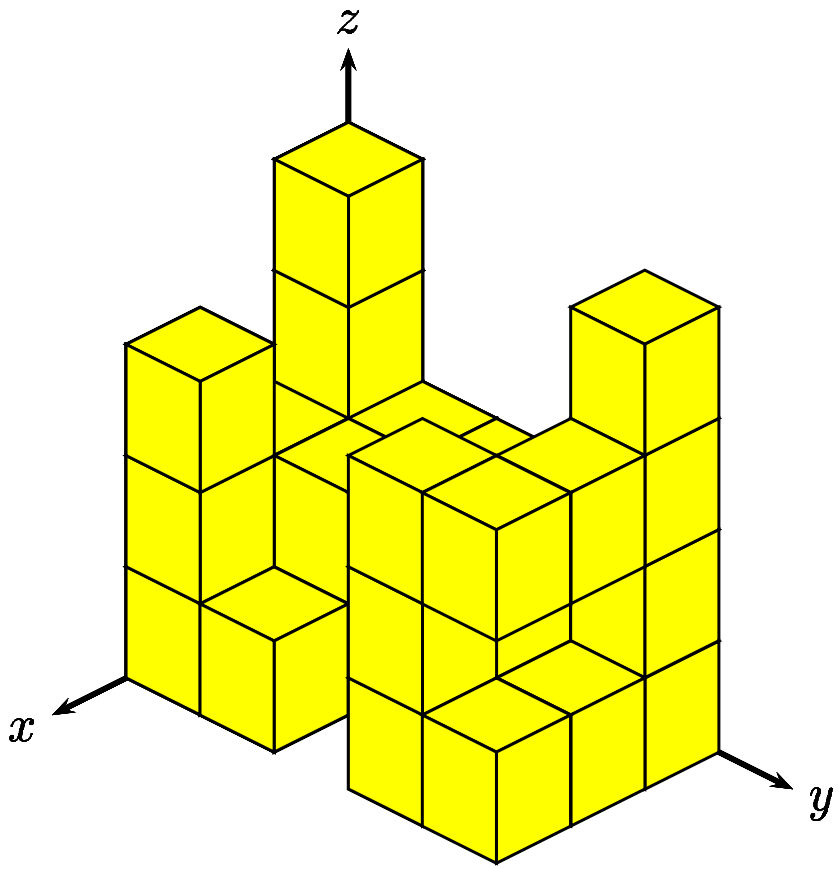
Pattern, Leader, Candidates



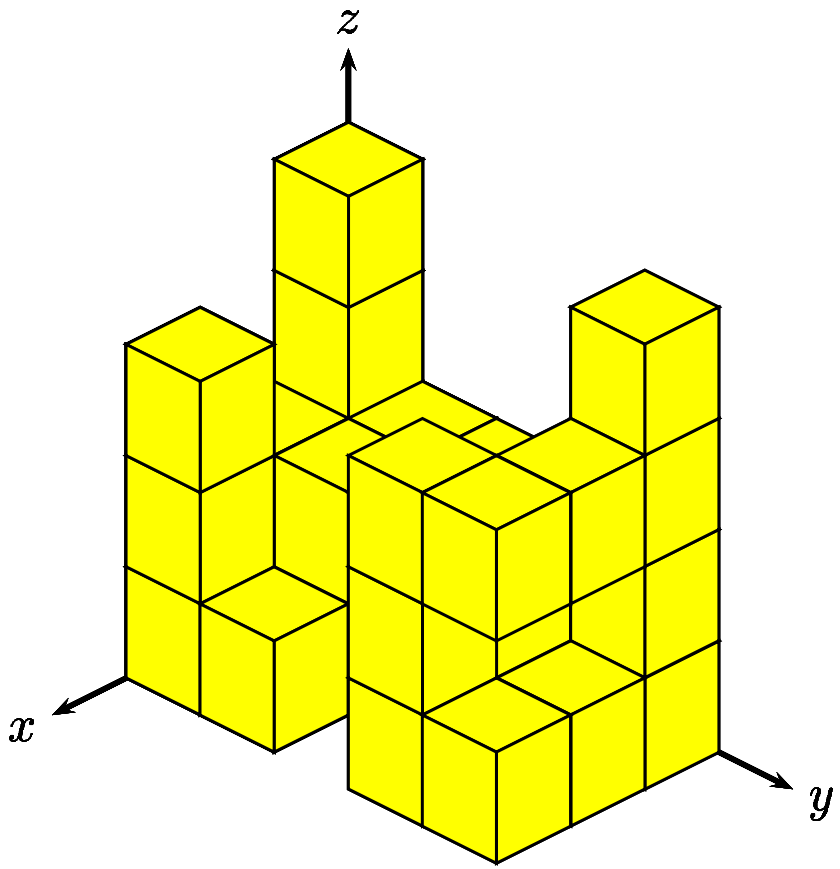
Pattern, Leader, Candidates



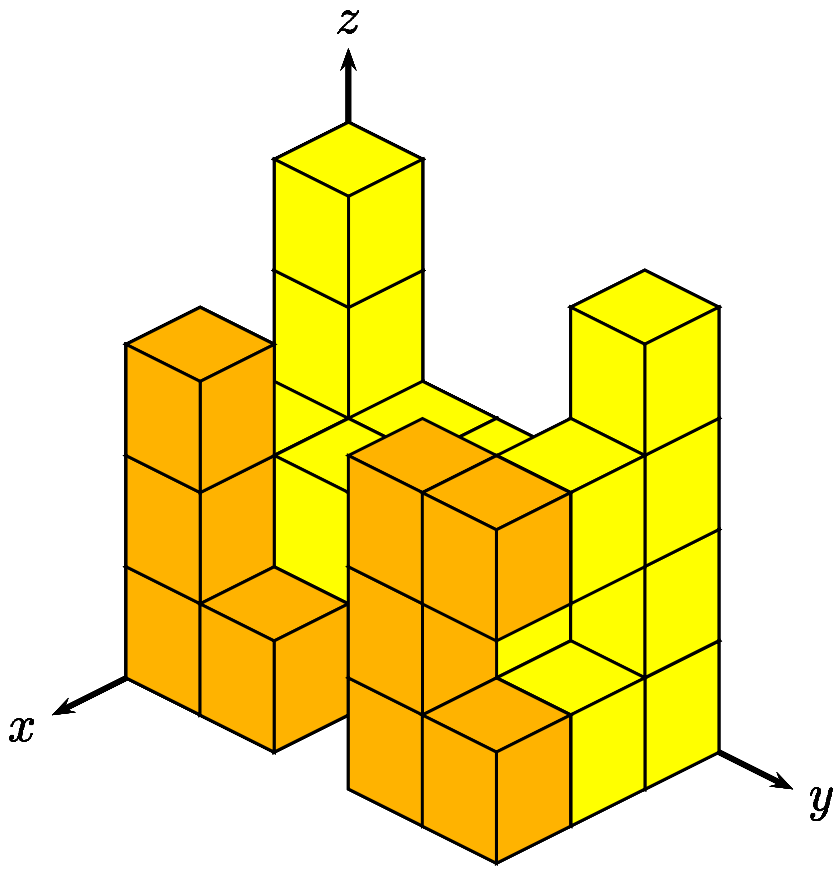
Pattern, Leader, Candidates



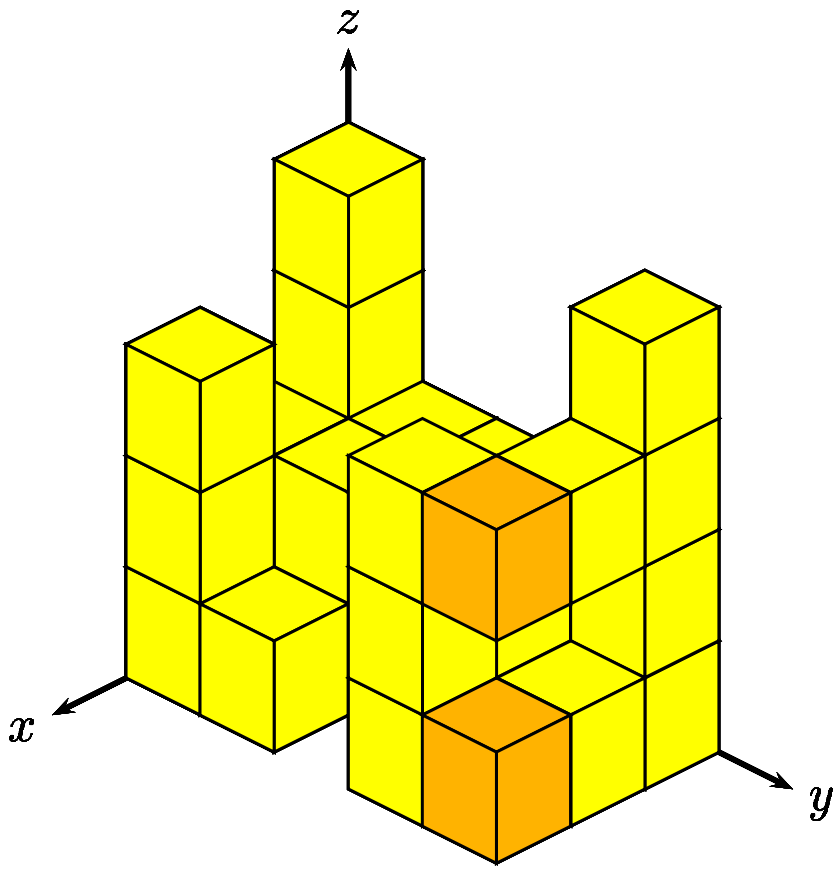
Pattern, **Leader,** Candidates



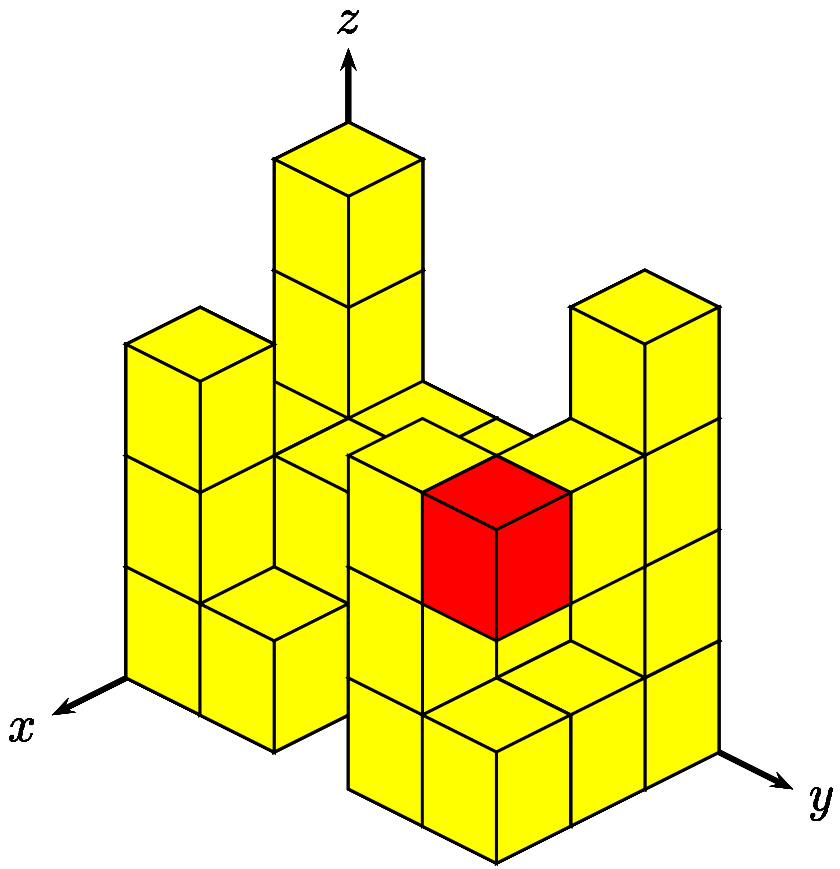
Pattern, **Leader,** Candidates



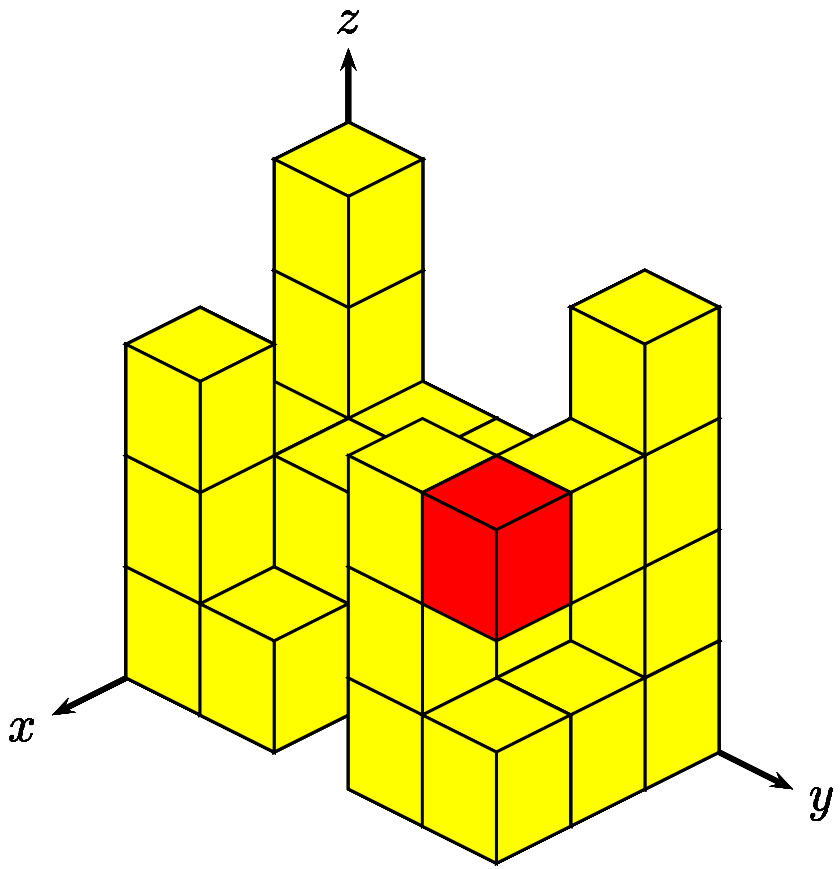
Pattern, **Leader,** Candidates



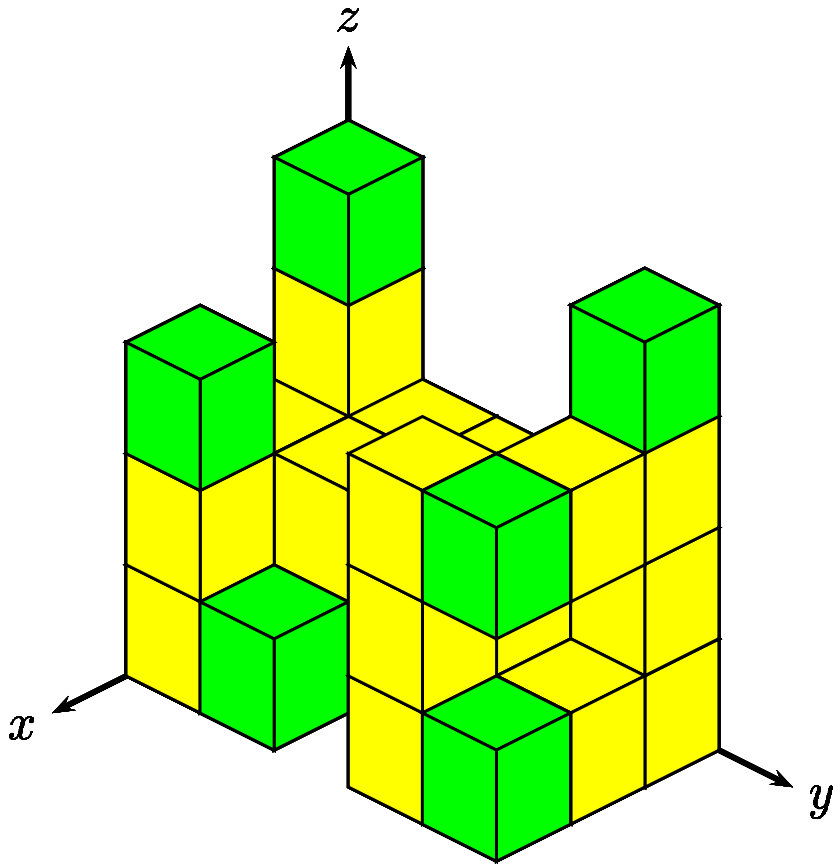
Pattern, **Leader,** Candidates



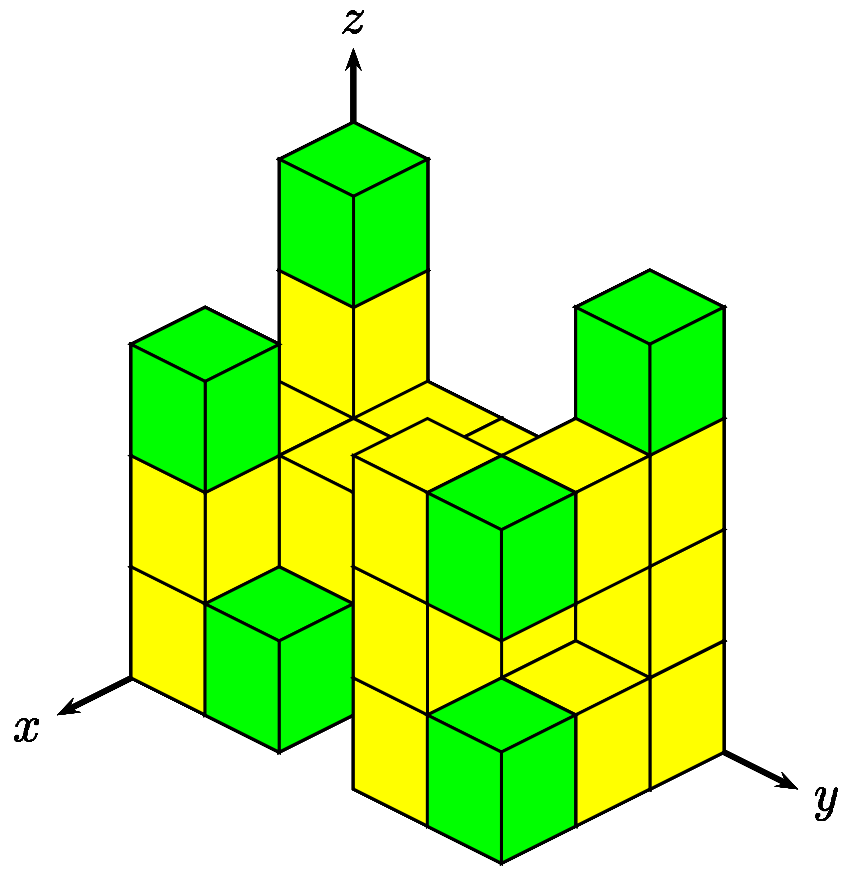
Pattern, Leader, **Candidates**



Pattern, Leader, **Candidates**



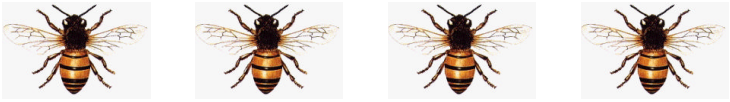
Pattern, Leader, **Candidates**



Problem: comparison of candidates

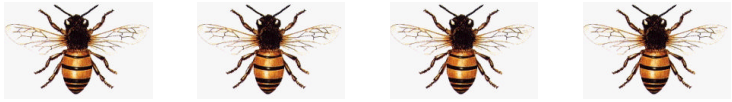
The Algorithm: selected aspects

- overview
- space counter
  - simple moving counter
  - generalization
- graph
  - initialization
  - deletion of edges
  - deletion of vertices
- time complexity



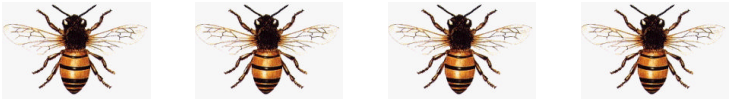
The Algorithm: selected aspects

- overview
- space counter
  - simple moving counter
  - generalization
- graph
  - initialization
  - deletion of edges
  - deletion of vertices
- time complexity



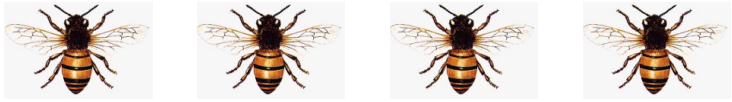
The Algorithm: selected aspects

- overview
- space counter
  - simple moving counter
  - generalization
- graph
  - initialization
  - deletion of edges
  - deletion of vertices
- time complexity



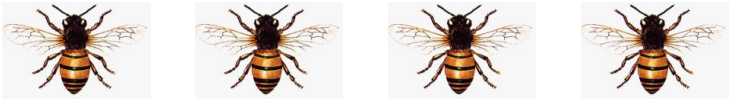
The Algorithm: selected aspects

- overview
- space counter
  - simple moving counter
  - generalization
- graph
  - initialization
  - deletion of edges
  - deletion of vertices
- time complexity



The Algorithm: selected aspects

- overview
- space counter
  - simple moving counter
  - generalization
- graph
  - initialization
  - deletion of edges
  - deletion of vertices
- time complexity



Algorithm: overview

- work on a graph, initially corresponding to the pattern
- shrink the graph until only one vertex is left: the leader

- build spanning trees starting at each candidate
- when two trees hit each other, they duel
  - the one originating at the stronger candidate wins
  - and spreads further into the territory of the weaker candidate
- when a tree hits the border, the branch dies

- each branch is constructed as the trail of a moving counter;
  it contains the relative distance to the candidate it started at

Algorithm: overview

- work on a graph, initially corresponding to the pattern
- shrink the graph until only one vertex is left: the leader

- build spanning trees starting at each candidate
- when two trees hit each other, they duel
  - the one originating at the stronger candidate wins
  - and spreads further into the territory of the weaker candidate
- when a tree hits the border, the branch dies

- each branch is constructed as the trail of a moving counter;
  it contains the relative distance to the candidate it started at

Algorithm: overview

- work on a graph, initially corresponding to the pattern
- shrink the graph until only one vertex is left: the leader

- build spanning trees starting at each candidate
- when two trees hit each other, they duel
  - the one originating at the stronger candidate wins
  - and spreads further into the territory of the weaker candidate
- when a tree hits the border, the branch dies

- each branch is constructed as the trail of a moving counter;
  it contains the relative distance to the candidate it started at

Algorithm: overview

- work on a graph, initially corresponding to the pattern
- shrink the graph until only one vertex is left: the leader

- build spanning trees starting at each candidate
- when two trees hit each other, they duel
  - the one originating at the stronger candidate wins
  - and spreads further into the territory of the weaker candidate
- when a tree hits the border, the branch dies

- each branch is constructed as the trail of a moving counter;
  it contains the relative distance to the candidate it started at

Algorithm: overview

- work on a graph, initially corresponding to the pattern
- shrink the graph until only one vertex is left: the leader

- build spanning trees starting at each candidate
- when two trees hit each other, they duel
  - the one originating at the stronger candidate wins
  - and spreads further into the territory of the weaker candidate
- when a tree hits the border, the branch dies

- each branch is constructed as the trail of a moving counter;
  it contains the relative distance to the candidate it started at

Algorithm: overview

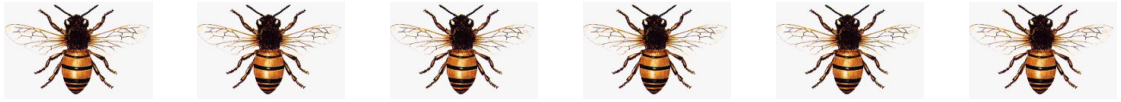
- work on a graph, initially corresponding to the pattern
- shrink the graph until only one vertex is left: the leader

- build spanning trees starting at each candidate
- when two trees hit each other, they duel
  - the one originating at the stronger candidate wins
  - and spreads further into the territory of the weaker candidate
- when a tree hits the border, the branch dies

- each branch is constructed as the trail of a moving counter; it contains the relative distance to the candidate it started at

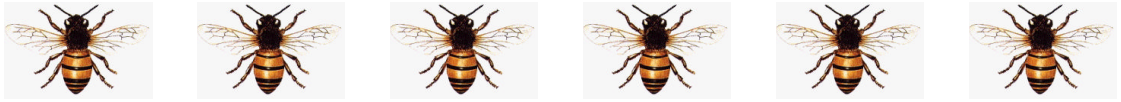
Counters: the simple case

	(1)												
	0												



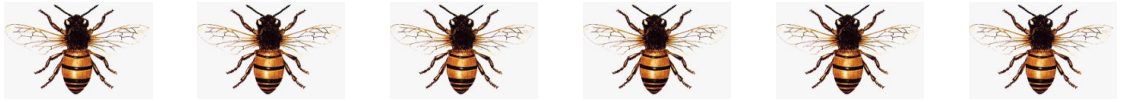
Counters: the simple case

		0)											
		(1											



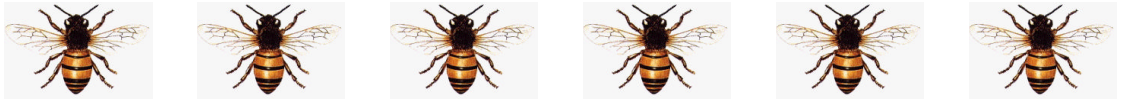
Counters: the simple case

		(1	1)										
		0	0										



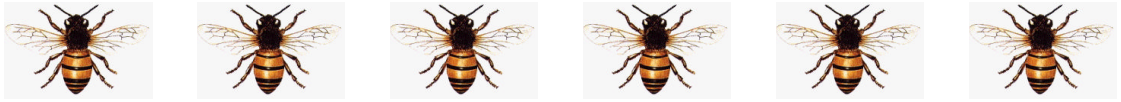
Counters: the simple case

			(1	0)									
			0	1									



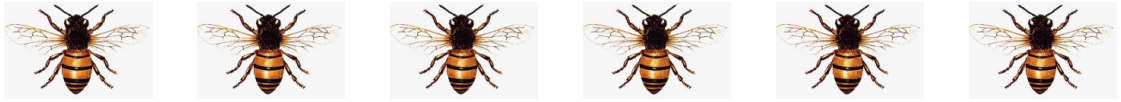
Counters: the simple case

				0	1)								
				(1	0								



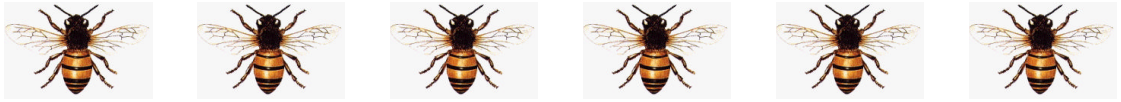
Counters: the simple case

				(1	0	0)							
				0	0	1							



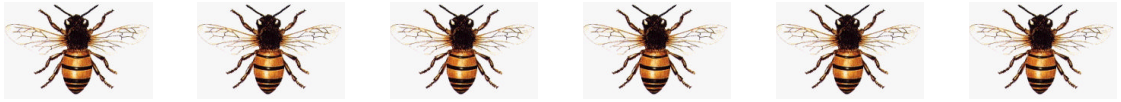
Counters: the simple case

					(1	1	1)						
					0	0	0						



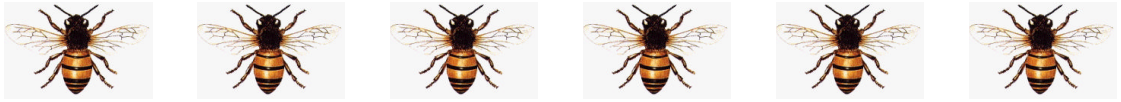
Counters: the simple case

						(1	1	0)					
						0	0	1					



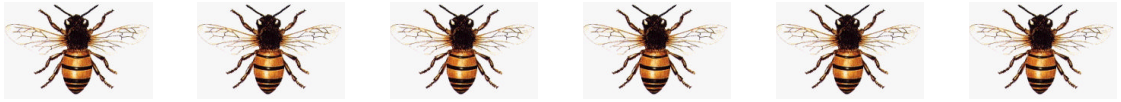
Counters: the simple case

							(1	0	1)				
							0	1	0				



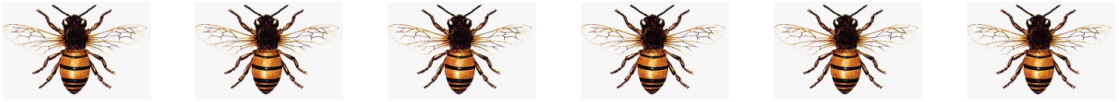
Counters: the simple case

								0	0	0)			
								(1	0	1			



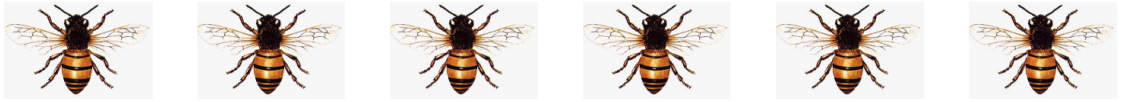
Counters: the simple case

								(1	0	1	1)		
								0	0	0	0		



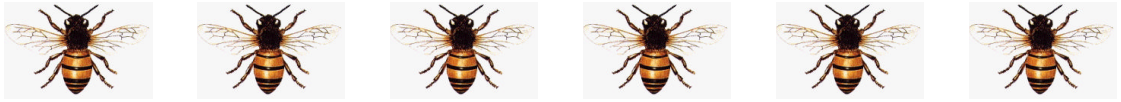
Counters: the simple case

									(1	0	1	0)	
									0	0	0	1	



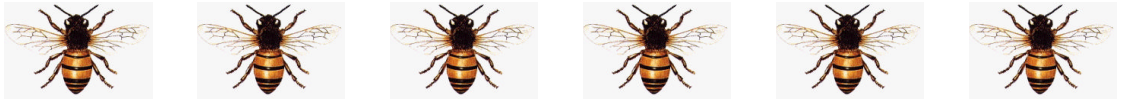
Counters: the simple case

										(1	0	0	
										0	0	1	



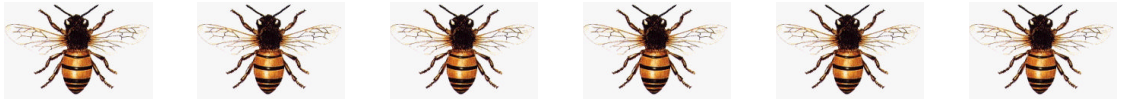
Counters: the simple case

											(1	1	
											0	0	



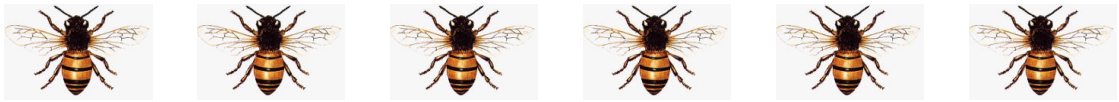
Counters: the simple case

												(1	
												0	



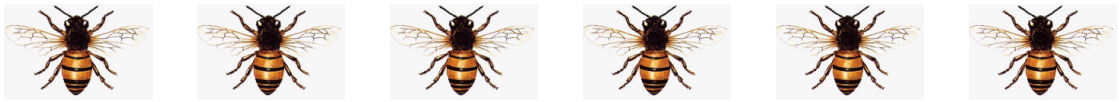
Counters: the simple case

1	2	3	4	5	6	7	8	9	10
(1)									
0									
	0)								
	(1								
	(1	1)							
	0	0							
		(1	0)						
		0	1						
			0	1)					
			(1	0					
			(1	0	0)				
			0	0	1				
				(1	1	1)			
				0	0	0			
					(1	1	0)		
					0	0	1		
						(1	0	1)	
						0	1	0	
							0	0	0)
							(1	0	1



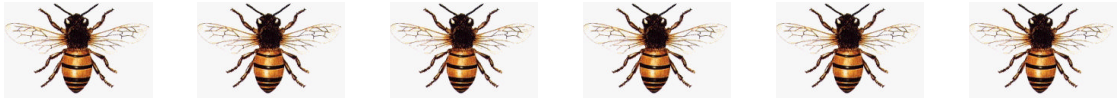
Counters: the simple case

1	2	3	4	5	6	7	8	9	10
(1)									
0									
	0)								
	(1								
	(1	1)							
	0	0							
		(1	0)						
		0	1						
			0	1)					
			(1	0					
			(1	0	0)				
			0	0	1				
				(1	1	1)			
				0	0	0			
					(1	1	0)		
					0	0	1		
						(1	0	1)	
						0	1	0	
							0	0	0)
							(1	0	1



Counters: the simple case

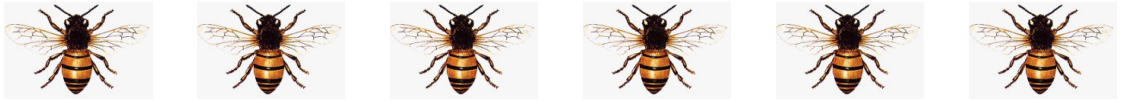
1	2	3	4	5	6	7	8	9	10
(1)									
0									
	0)								
	(1								
	(1	1)							
	0	0							
		(1	0)						
		0	1						
			0	1)					
			(1	0					
			(1	0	0)				
			0	0	1				
				(1	1	1)			
				0	0	0			
					(1	1	0)		
					0	0	1		
						(1	0	1)	
						0	1	0	
							0	0	0)
							(1	0	1



Counters: the simple case

	(1)												
	0												

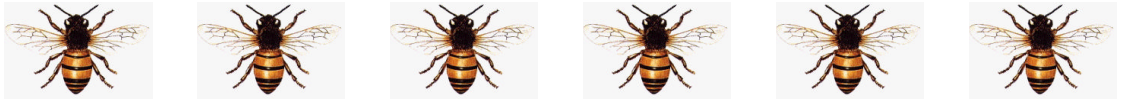
1



Counters: the simple case

		0)											
		(1											

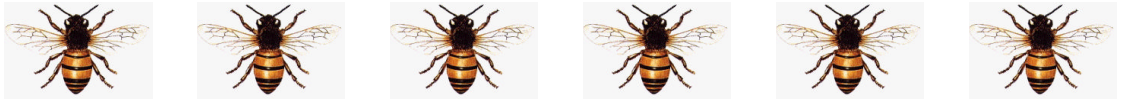
1 0



Counters: the simple case

		(1	1)										
		0	0										

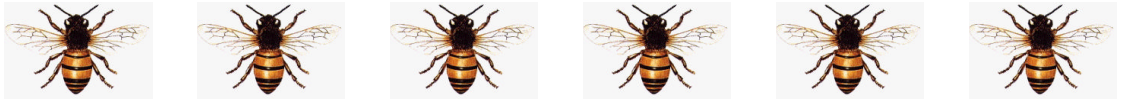
$$\begin{array}{ccc} 1 & 0 & 1 \\ & 1 & \end{array}$$



Counters: the simple case

			(1	0)									
			0	1									

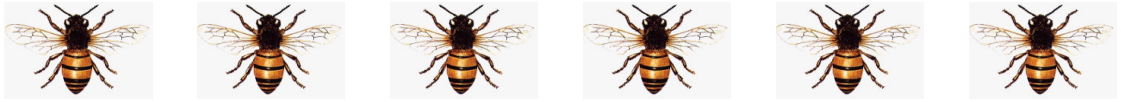
$$\begin{array}{cccc} 1 & 0 & 1 & 0 \\ & 1 & 1 & \end{array}$$



Counters: the simple case

				0	1)								
				(1	0								

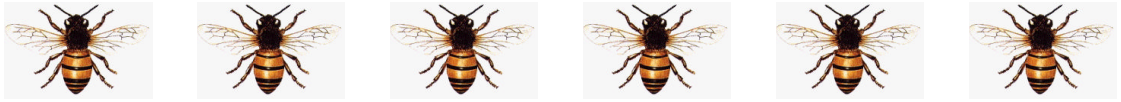
$$\begin{array}{ccccc} 1 & 0 & 1 & 0 & 1 \\ & 1 & 1 & 0 & \end{array}$$



Counters: the simple case

				(1	0	0)							
				0	0	1							

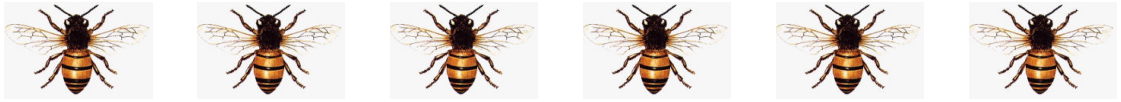
$$
\begin{array}{cccccc}
1 & 0 & 1 & 0 & 1 & 0 \\
 & 1 & 1 & 0 & 0 & \\
 & & & 1 & & \\
\end{array}
$$



Counters: the simple case

					(1	1	1)						
					0	0	0						

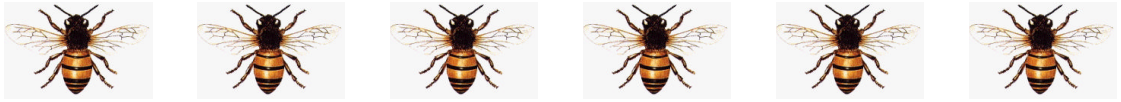
$$
\begin{array}{ccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
 & 1 & 1 & 0 & 0 & 1 & \\
 & & & 1 & 1 & &
\end{array}
$$



Counters: the simple case

						(1	1	0)					
						0	0	1					

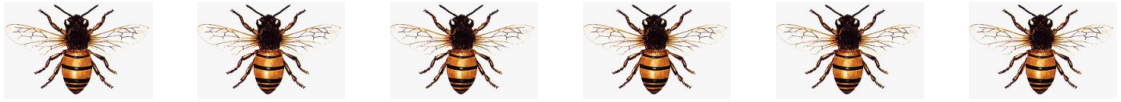
$$
\begin{array}{cccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
 & 1 & 1 & 0 & 0 & 1 & 1 & \\
 & & & 1 & 1 & 1 & & \\
\end{array}
$$



Counters: the simple case

							(1	0	1)				
							0	1	0				

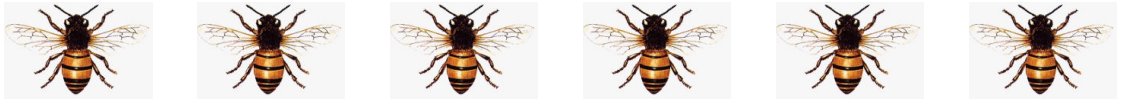
$$
\begin{array}{ccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & \\
 & & & 1 & 1 & 1 & 1 & &
\end{array}
$$



Counters: the simple case

								0	0	0)			
								(1	0	1			

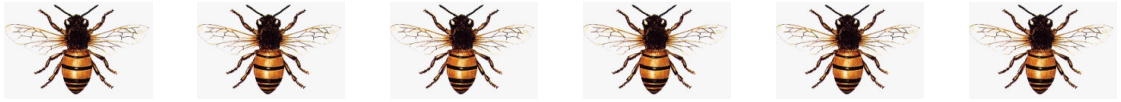
$$
\begin{array}{cccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \\
 & & & 1 & 1 & 1 & 1 & 0 & & \\
\end{array}
$$



Counters: the simple case

								(1	0	1	1)		
								0	0	0	0		

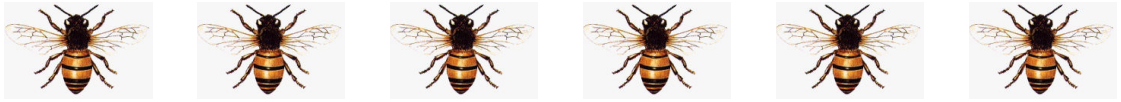
$$
\begin{array}{ccccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \\
 & & & 1 & 1 & 1 & 1 & 0 & 0 & & \\
 & & & & & & & 1 & & &
\end{array}
$$



Counters: the simple case

									(1	0	1	0)	
									0	0	0	1	

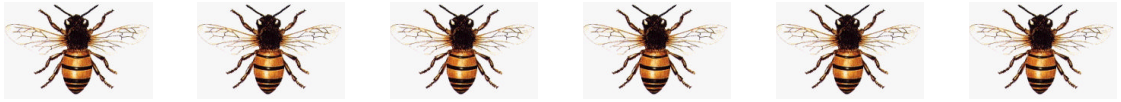
$$
\begin{array}{cccccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \\
 & & & 1 & 1 & 1 & 1 & 0 & 0 & 0 & & \\
 & & & & & & & 1 & 1 & & &
\end{array}
$$



Counters: the simple case

										(1	0	0	
										0	0	1	

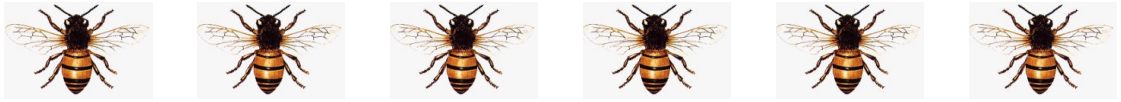
$$
\begin{array}{cccccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
 & & & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \\
 & & & & & & & 1 & 1 & 1 & & \\
\end{array}
$$



Counters: the simple case

											(1	1	
											0	0	

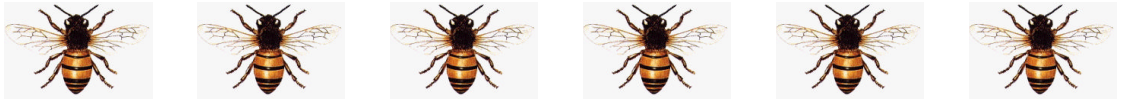
$$
\begin{array}{cccccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
 & & & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
 & & & & & & & 1 & 1 & 1 & 1 &
\end{array}
$$



Counters: the simple case

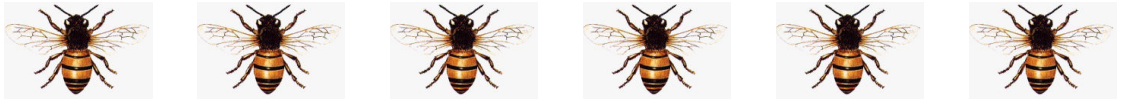
											(1	
											0	

$$
\begin{array}{cccccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
 & & & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
 & & & & & & & 1 & 1 & 1 & 1 & 1
\end{array}
$$



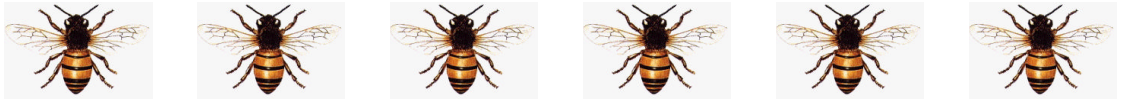
Counters: as used in the algorithm

- counters for *integers*
  increment/decrement when moving in $+/-$ direction
- counters can be stretched and compressed by a factor of 2
- *space counter*: d such counters, one for each dimension
- space counters replicate at nodes with degree > 2: $\Longrightarrow$ bundles
- after moving to the next cell each counter waits for a time proportional to its length
- When two space counters meet, they duel:
  - copies of the contents are compared "bit by bit"
  - the winner moves on, the loser is destroyed



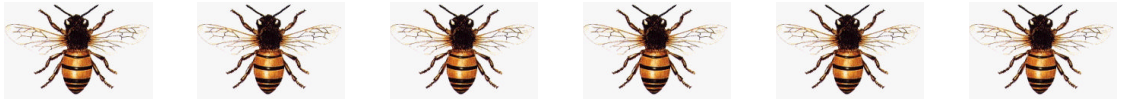
Counters: as used in the algorithm

- counters for *integers*
  increment/decrement when moving in $+/-$ direction
- counters can be stretched and compressed by a factor of 2
- *space counter*: d such counters, one for each dimension
- space counters replicate at nodes with degree > 2: $\Longrightarrow$ bundles
- after moving to the next cell each counter waits for a time proportional to its length
- When two space counters meet, they duel:
  - copies of the contents are compared "bit by bit"
  - the winner moves on, the loser is destroyed



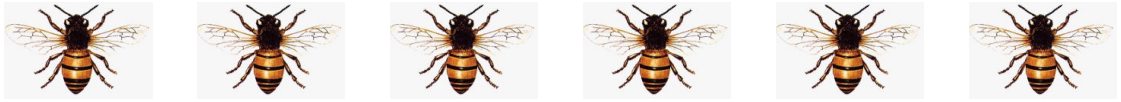
Counters: as used in the algorithm

- counters for *integers*
  increment/decrement when moving in $+/-$ direction
- counters can be stretched and compressed by a factor of 2
- *space counter*: d such counters, one for each dimension
- space counters replicate at nodes with degree > 2: $\Longrightarrow$ bundles
- after moving to the next cell each counter waits for a time proportional to its length
- When two space counters meet, they duel:
  - copies of the contents are compared "bit by bit"
  - the winner moves on, the loser is destroyed



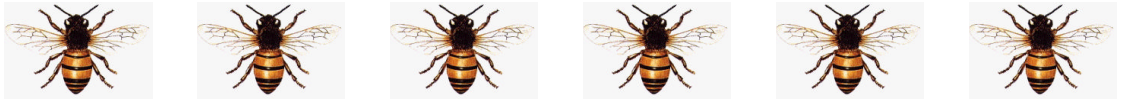
Counters: as used in the algorithm

- counters for *integers*
  increment/decrement when moving in $+/-$ direction
- counters can be stretched and compressed by a factor of 2
- *space counter*: d such counters, one for each dimension
- space counters replicate at nodes with degree > 2: $\Longrightarrow$ bundles
- after moving to the next cell each counter waits for a time proportional to its length
- When two space counters meet, they duel:
  - copies of the contents are compared "bit by bit"
  - the winner moves on, the loser is destroyed



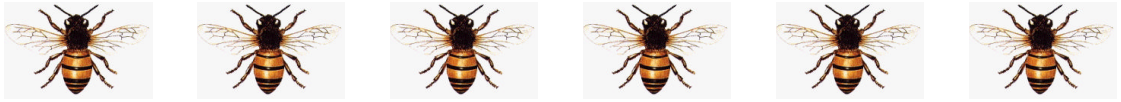
Counters: as used in the algorithm

- counters for *integers*
  increment/decrement when moving in $+/-$ direction
- counters can be stretched and compressed by a factor of 2
- *space counter*: d such counters, one for each dimension
- space counters replicate at nodes with degree > 2: $\Longrightarrow$ bundles
- after moving to the next cell each counter waits for a time proportional to its length
- When two space counters meet, they duel:
  - copies of the contents are compared "bit by bit"
  - the winner moves on, the loser is destroyed



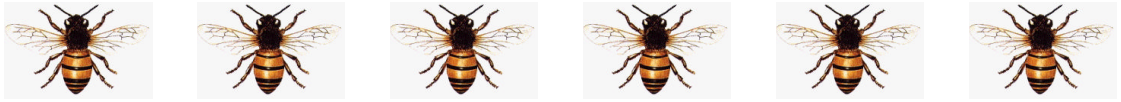
Counters: as used in the algorithm

- counters for *integers*
  increment/decrement when moving in $+/-$ direction
- counters can be stretched and compressed by a factor of 2
- *space counter*: d such counters, one for each dimension
- space counters replicate at nodes with degree > 2: $\Longrightarrow$ bundles
- after moving to the next cell each counter waits for a time proportional to its length
- When two space counters meet, they duel:
  - copies of the contents are compared "bit by bit"
  - the winner moves on, the loser is destroyed



Counters: as used in the algorithm

- counters for *integers*
  increment/decrement when moving in $+/-$ direction
- counters can be stretched and compressed by a factor of 2
- *space counter*: d such counters, one for each dimension
- space counters replicate at nodes with degree > 2: $\Longrightarrow$ bundles
- after moving to the next cell each counter waits for a time proportional to its length
- When two space counters meet, they duel:
  - copies of the contents are compared "bit by bit"
  - the winner moves on, the loser is destroyed



Graph

- each vertex correponds to a cell
- each vertex is stored in that cell
- an edge connects two vertices only if cells are immediate neighbors
- an edge is stored in the "cells" it connects

Initialization:

- a vertex for each cell
- an edge for each pair of neighboring cells

Graph

- each vertex correponds to a cell
- each vertex is stored in that cell
- an edge connects two vertices only if cells are immediate neighbors
- an edge is stored in the "cells" it connects

Initialization:

- a vertex for each cell
- an edge for each pair of neighboring cells

Graph

- each vertex correponds to a cell
- each vertex is stored in that cell
- an edge connects two vertices only if cells are immediate neighbors
- an edge is stored in the "cells" it connects

Initialization:

- a vertex for each cell
- an edge for each pair of neighboring cells

Graph

- each vertex correponds to a cell
- each vertex is stored in that cell
- an edge connects two vertices only if cells are immediate neighbors
- an edge is stored in the "cells" it connects

Initialization:

- a vertex for each cell
- an edge for each pair of neighboring cells

Graph

- each vertex correponds to a cell
- each vertex is stored in that cell
- an edge connects two vertices only if cells are immediate neighbors
- an edge is stored in the "cells" it connects

Initialization:

- a vertex for each cell
- an edge for each pair of neighboring cells

Graph deletions

- a node and its incident edge are deleted
  - if they are the end of a branch which will not extend any more

- an edge is deleted,
  - if a loop is detected (duel of counters with identical contents)

Graph deletions

- a node and its incident edge are deleted
  - if they are the end of a branch which will not extend any more

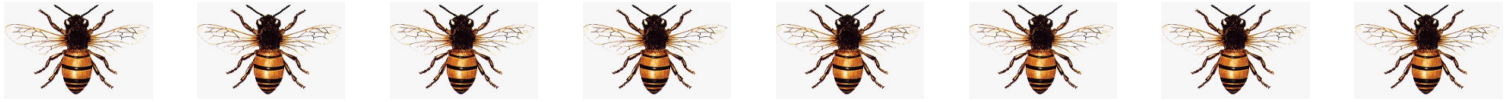
- an edge is deleted,
  - if a loop is detected (duel of counters with identical contents)

Nasty implementation details

much less than in Stratmann's original algorithm
(due to the more synchronized mode of operation of the counters)

Length of counters

The graph does not get disconnected.

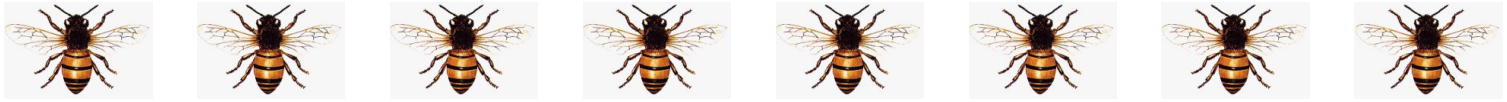


Nasty implementation details

much less than in Stratmann's original algorithm
(due to the more synchronized mode of operation of the counters)

Length of counters

The graph does not get disconnected.

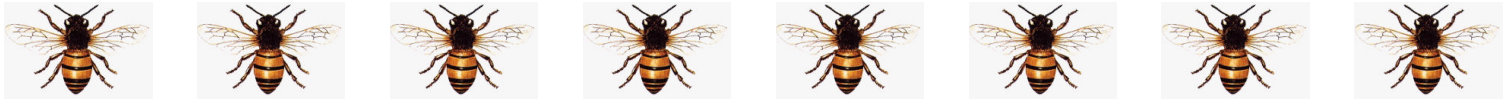


Nasty implementation details

much less than in Stratmann's original algorithm
(due to the more synchronized mode of operation of the counters)

Length of counters

The graph does not get disconnected.



Time complexity

Claim 1:
The length of branches of the spanning trees is $\mathrm{O}(diam)$.

Claim 2:
It takes time $\mathrm{O}(\log diam)$ until a branch is extended by one edge.

Claim 3:
The product is the dominating summand of the running time.

Leader election in d-dimensional CA is possible in time

$$\mathrm{O}(diam \cdot \log diam)$$

Time complexity

Claim 1:
The length of branches of the spanning trees is $\mathrm{O}(diam)$.

Claim 2:
It takes time $\mathrm{O}(\log diam)$ until a branch is extended by one edge.

Claim 3:
The product is the dominating summand of the running time.

Leader election in d-dimensional CA is possible in time

$$\mathrm{O}(diam \cdot \log diam)$$

Time complexity

Claim 1:
The length of branches of the spanning trees is $\mathrm{O}(diam)$.

Claim 2:
It takes time $\mathrm{O}(\log diam)$ until a branch is extended by one edge.

Claim 3:
The product is the dominating summand of the running time.

Leader election in d-dimensional CA is possible in time

$$\mathrm{O}(diam \cdot \log diam)$$

Time complexity

Claim 1:
The length of branches of the spanning trees is $\mathrm{O}(diam)$.

Claim 2:
It takes time $\mathrm{O}(\log diam)$ until a branch is extended by one edge.

Claim 3:
The product is the dominating summand of the running time.

Leader election in d-dimensional CA is possible in time

$$\mathrm{O}(diam \cdot \log diam)$$

Time complexity

Claim 1:
The length of branches of the spanning trees is $\mathrm{O}(diam)$.

Claim 2:
It takes time $\mathrm{O}(\log diam)$ until a branch is extended by one edge.

Claim 3:
The product is the dominating summand of the running time.

Leader election in d-dimensional CA is possible in time

$$\mathbf{O}(diam \cdot \log diam)$$

The end