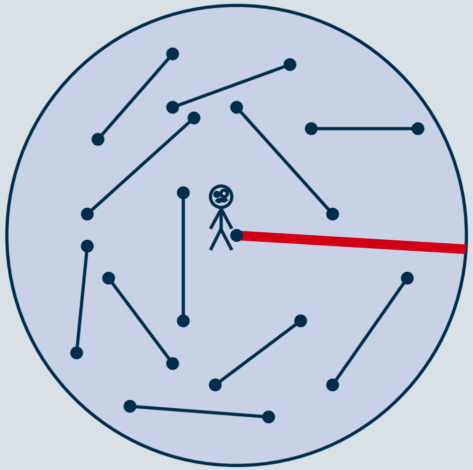
Computational Geometry

Exercise 5 *Assignment 4, 5 and Voronoi diagrams of segments*

Jean-Pierre, Marcus, Wendy

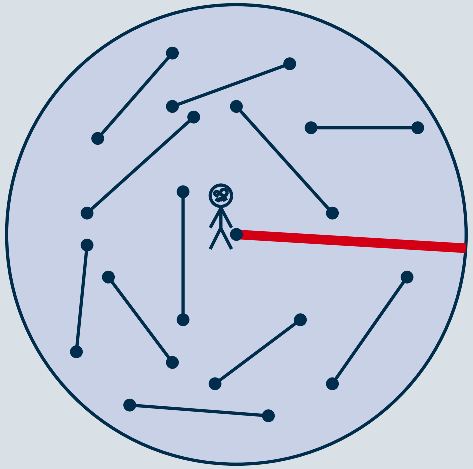
Mission Impossible + 1.5 Range Queries



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line



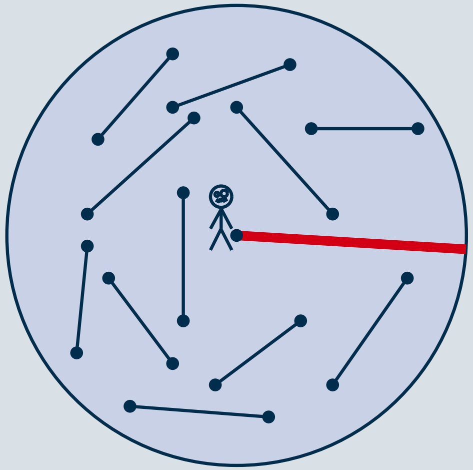
query: $O(\log(n) + k)$

preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

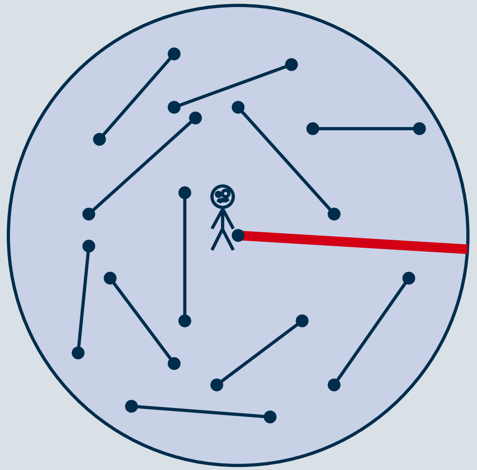
Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

timestamp?

- data structure: persistent binary search tree



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

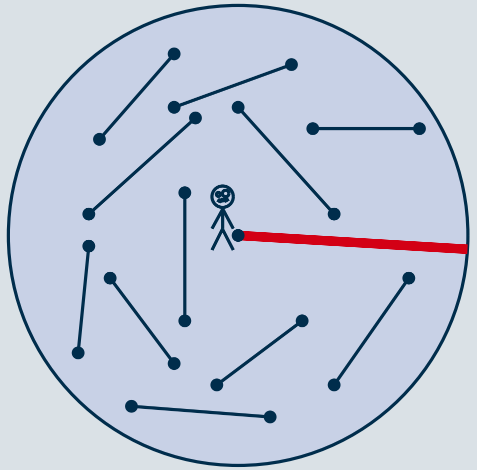
Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

timestamp?

- data structure: persistent binary search tree

at $\alpha = 0$?



query: $O(\log(n) + k)$

preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

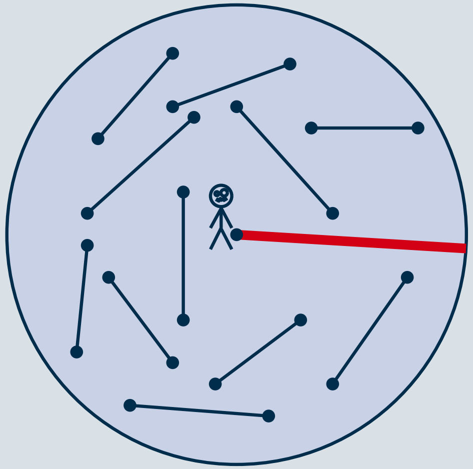
- walls that intersect sweep line
- sort by distance to center

timestamp?

- data structure: persistent binary search tree

at $\alpha = 0$?

Events



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

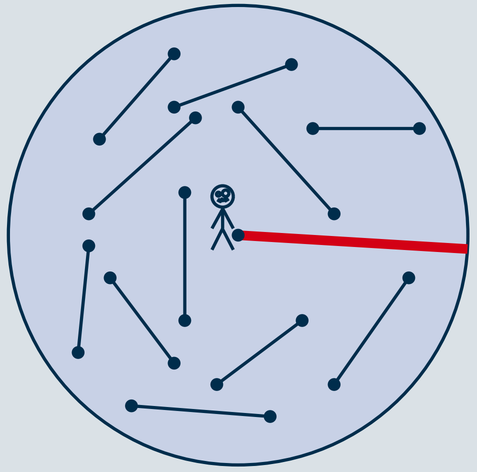
timestamp?

- data structure: persistent binary search tree

at $\alpha = 0$?

Events

- start point: add wall
- end point: remove wall



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

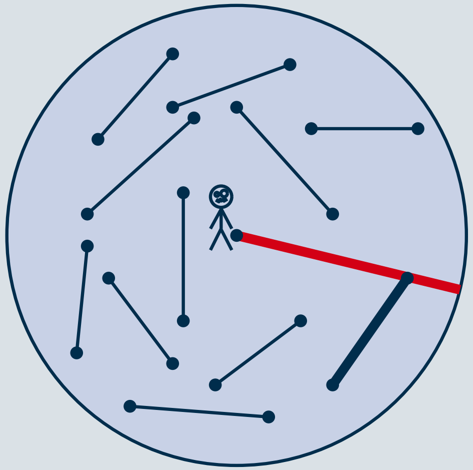
timestamp?

- data structure: persistent binary search tree

at $\alpha = 0$?

Events

- start point: add wall
- end point: remove wall



query: $O(\log(n) + k)$

preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

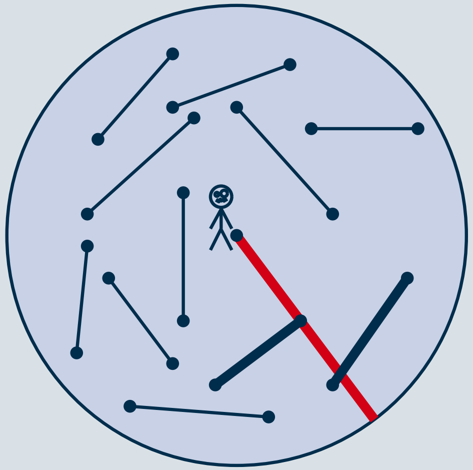
timestamp?

- data structure: persistent binary search tree

at $\alpha = 0$?

Events

- start point: add wall
- end point: remove wall



query: $O(\log(n) + k)$

preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

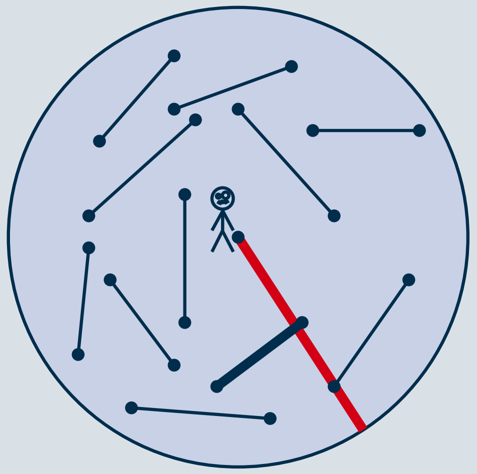
timestamp?

- data structure: persistent binary search tree

at $\alpha = 0$?

Events

- start point: add wall
- end point: remove wall



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

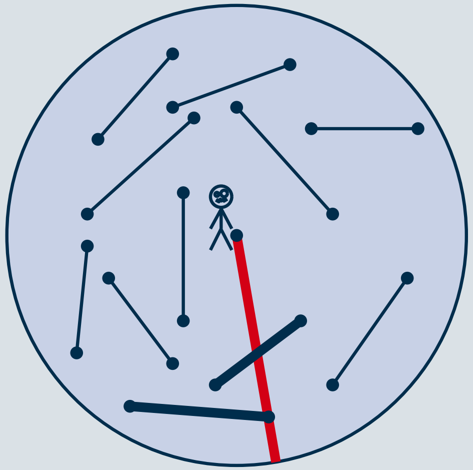
timestamp?

- data structure: persistent binary search tree

at $\alpha = 0$?

Events

- start point: add wall
- end point: remove wall



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

timestamp?

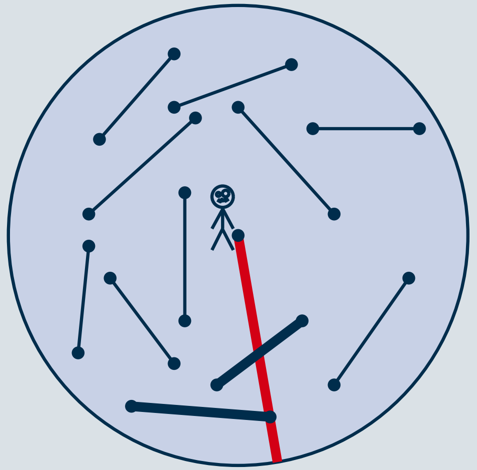
- data structure: persistent binary search tree

at $\alpha = 0$?

Events

- start point: add wall
- end point: remove wall

Queries



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

timestamp?

- data structure: persistent binary search tree

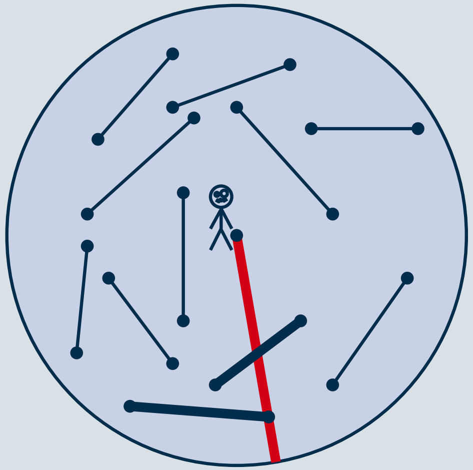
at $\alpha = 0$?

Events

- start point: add wall
- end point: remove wall

Queries

- ask for time $= \alpha$
- output all walls currently in sweep line state



query: $O(\log(n) + k)$

preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

timestamp?

- data structure: persistent binary search tree

at $\alpha = 0$?

Events

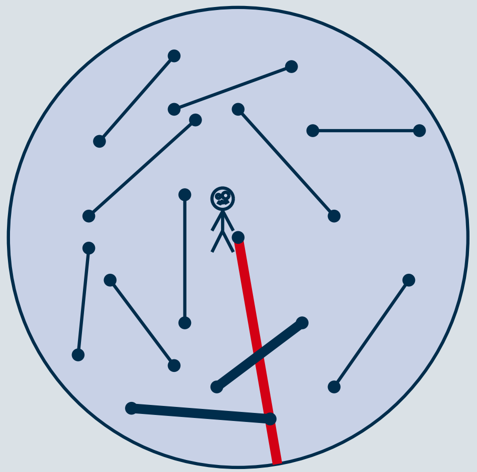
- start point: add wall
- end point: remove wall

Queries

- ask for time $= \alpha$

Why $O(\log(n) + k)$?

- output all walls currently in sweep line state



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center
- data structure: persistent binary search tree

timestamp?

at $\alpha = 0$?

Events

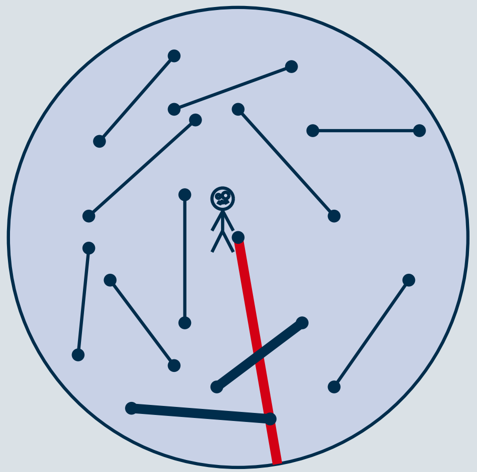
- start point: add wall
- end point: remove wall

But wait! What about space?

Queries

- ask for time $= \alpha$
- output all walls currently in sweep line state

Why $O(\log(n) + k)$?



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$

Mission Impossible + 1.5 Range Queries

Persistent Rotating Sweep Line

- walls that intersect sweep line
- sort by distance to center

timestamp?

- data structure: persistent binary search tree

Events

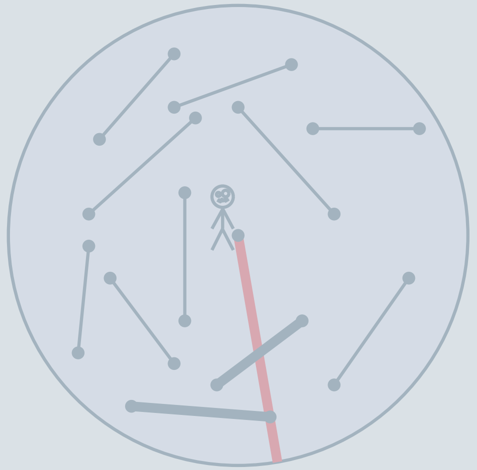
- start point: add wall
- end point: remove wall

Queries

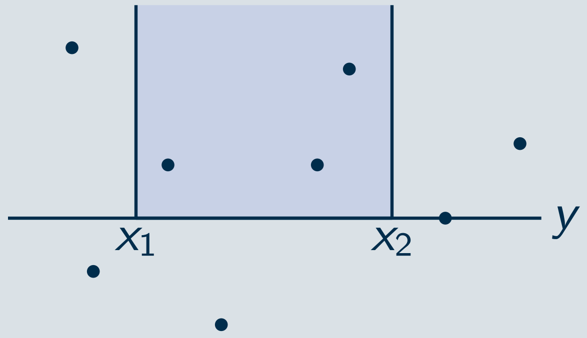
- ask for time $= \alpha$

Why $O(\log(n) + k)$?

- output all walls currently in sweep line state



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$



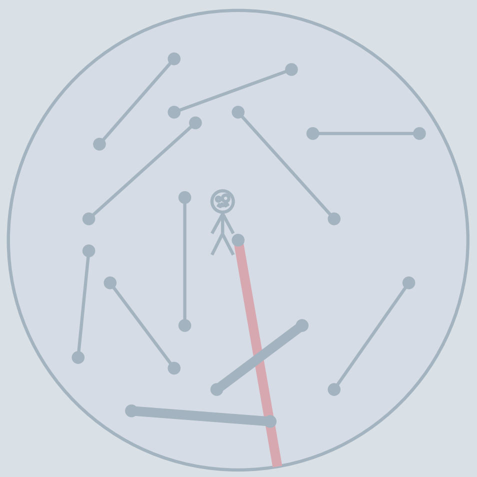
Mission Impossible + 1.5 Range Queries

Persistent ~~Rotating~~ Sweep Line

- ~~walls~~ points that ~~intersect~~ are above sweep line
- sort by distance to center

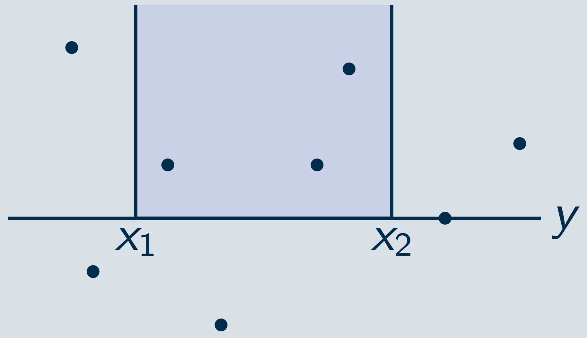
timestamp?

- data structure: persistent binary search tree



query: $O(\log(n) + k)$

preprocessing: $O(n \log(n))$



Events

- start point: add wall
- end point: remove wall

Queries

- ask for time $= \alpha$

Why $O(\log(n) + k)$?

- output all walls currently in sweep line state

Mission Impossible + 1.5 Range Queries

Persistent ~~Rotating~~ Sweep Line

- ~~walls~~ points that ~~intersect~~ are above sweep line
- sort by ~~distance to center~~ x-coordinate
- data structure: persistent binary search tree

timestamp?

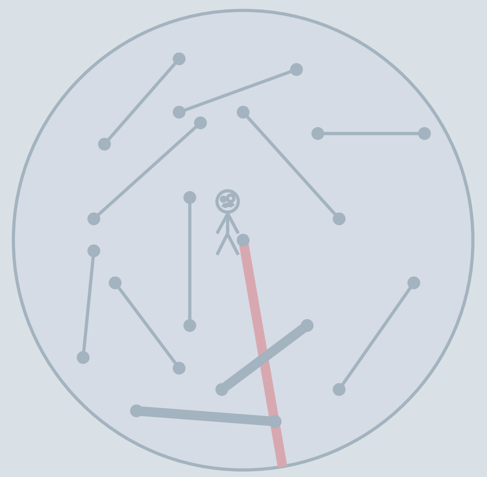
Events

- start point: add wall
- end point: remove wall

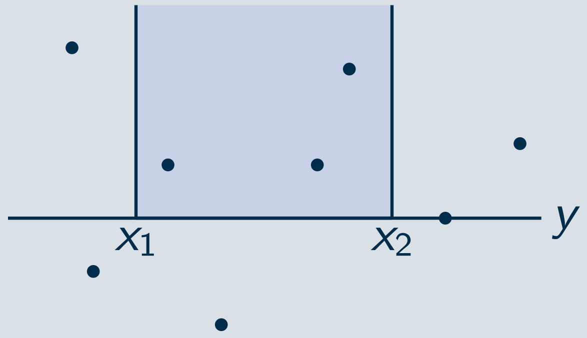
Queries

- ask for time $= \alpha$
- output all walls currently in sweep line state

Why $O(\log(n) + k)$?



query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$



Mission Impossible + 1.5 Range Queries

Persistent ~~Rotating~~ Sweep Line

- ~~walls~~ points that ~~intersect~~ are above sweep line
- sort by ~~distance to center~~ x-coordinate
- data structure: persistent binary search tree

timestamp?

Events

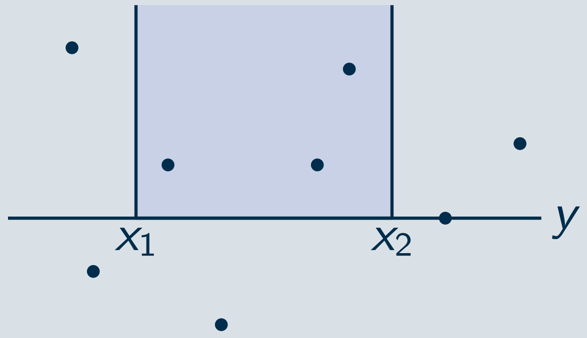
- start point: add ~~wall~~ point
- ~~end point: remove wall~~

Queries

- ask for time $= \alpha$
- output all walls currently in sweep line state

Why $O(\log(n) + k)$?

query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$



Mission Impossible + 1.5 Range Queries

Persistent ~~Rotating~~ Sweep Line

- ~~walls~~ points that ~~intersect~~ are above sweep line
- sort by ~~distance to center~~ x-coordinate
- data structure: persistent binary search tree

timestamp?

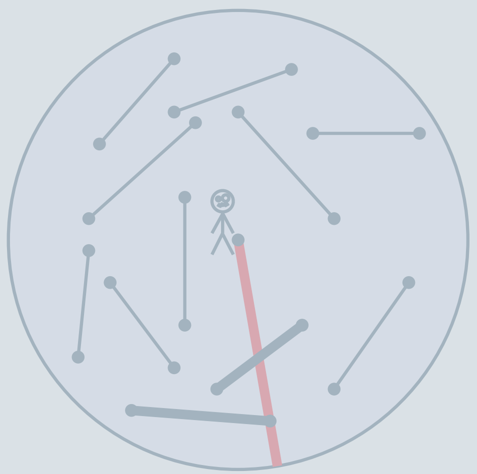
Events

- start point: add ~~wall~~ point
- ~~end point: remove wall~~

Queries

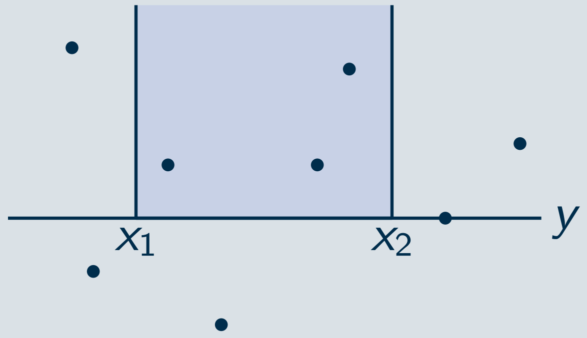
- ask for time $= \sout{\alpha}$ y
- output all walls currently in sweep line state

Why $O(\log(n) + k)$?



query: $O(\log(n) + k)$

preprocessing: $O(n \log(n))$



Mission Impossible + 1.5 Range Queries

Persistent ~~Rotating~~ Sweep Line

- ~~walls~~ points that ~~intersect~~ are above sweep line
- sort by ~~distance to center~~ x-coordinate
- data structure: persistent binary search tree

timestamp?

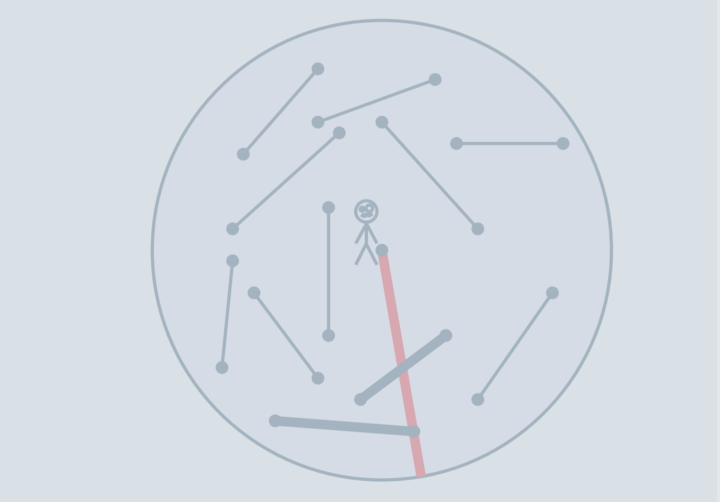
Events

- start point: add ~~wall~~ point
- ~~end point: remove wall~~

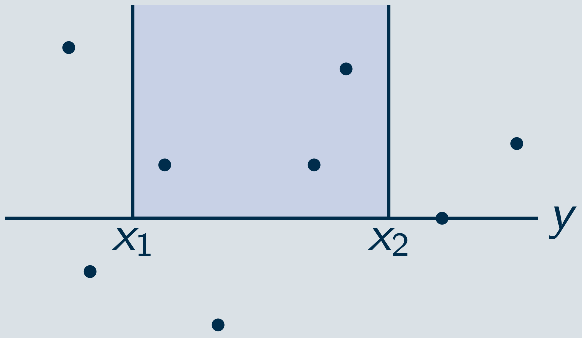
Queries

- ask for time $=$ ~~α~~ y
- ~~output all walls currently in sweep line state~~ find x_1, output all between x_1 and x_2

Why $O(\log(n) + k)$?

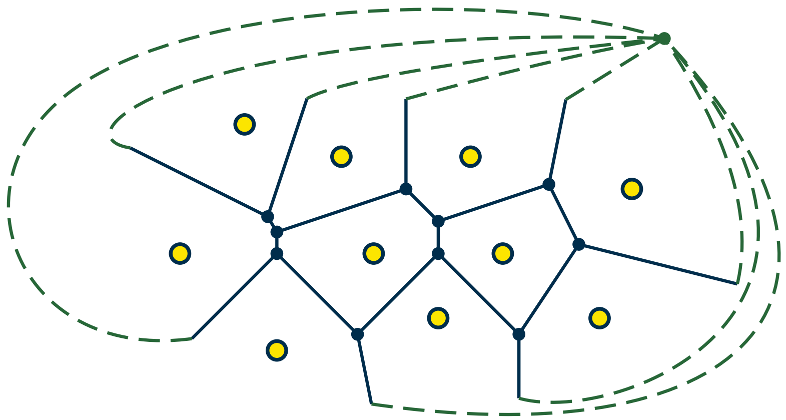


query: $O(\log(n) + k)$
preprocessing: $O(n \log(n))$



Voronoi Diagram

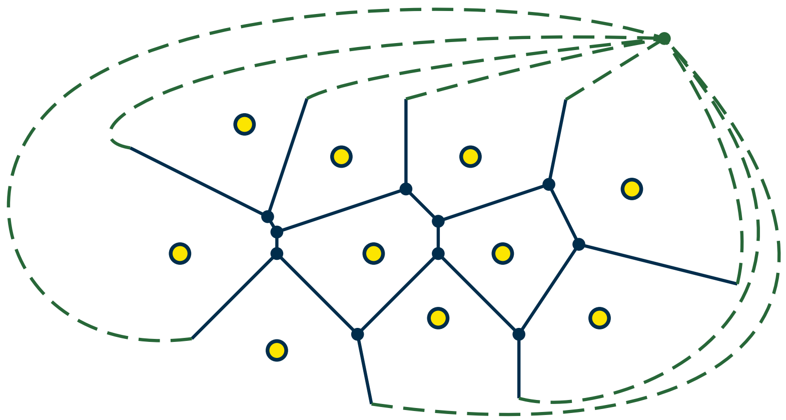
Size of Voronoi Diagram



- add vertex to collect loose edges

Voronoi Diagram

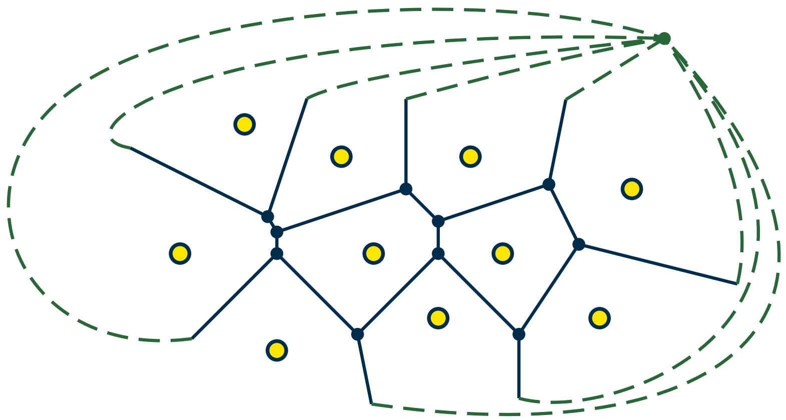
Size of Voronoi Diagram



- add vertex to collect loose edges
- n faces

Voronoi Diagram

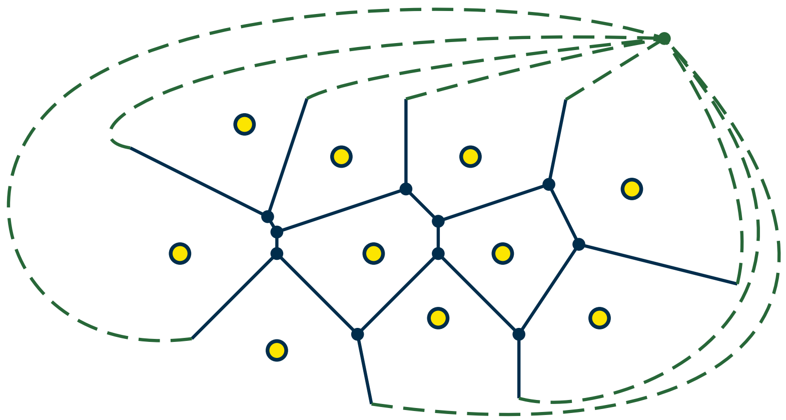
Size of Voronoi Diagram



- add vertex to collect loose edges
- n faces
- $v + 1$ vertices, with degree ≥ 3

Voronoi Diagram

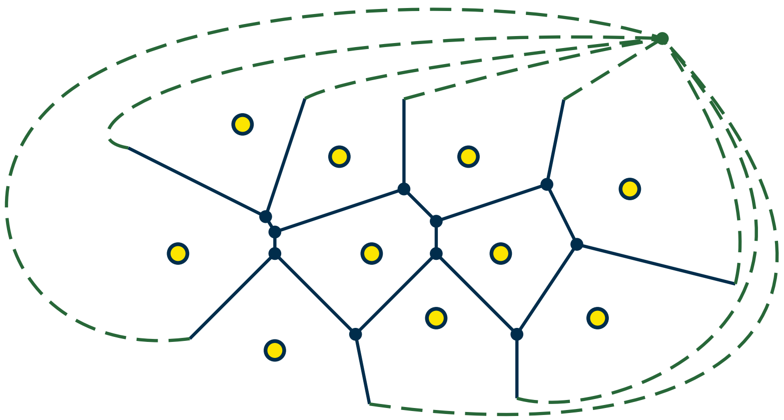
Size of Voronoi Diagram



- add vertex to collect loose edges
- n faces
- $v + 1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v + 1)$

Voronoi Diagram

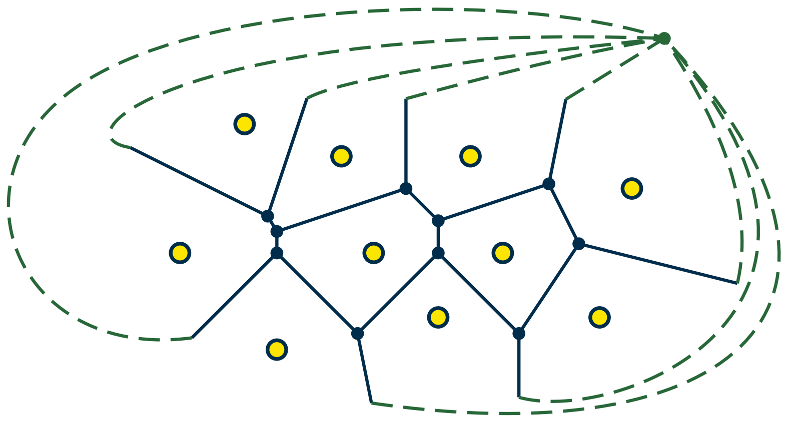
Size of Voronoi Diagram



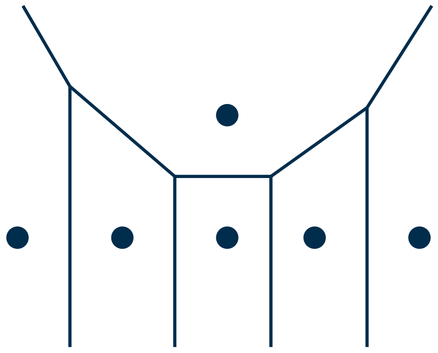
- add vertex to collect loose edges
- n faces
- $v + 1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v + 1)$
- use Euler's formula

Voronoi Diagram

Size of Voronoi Diagram



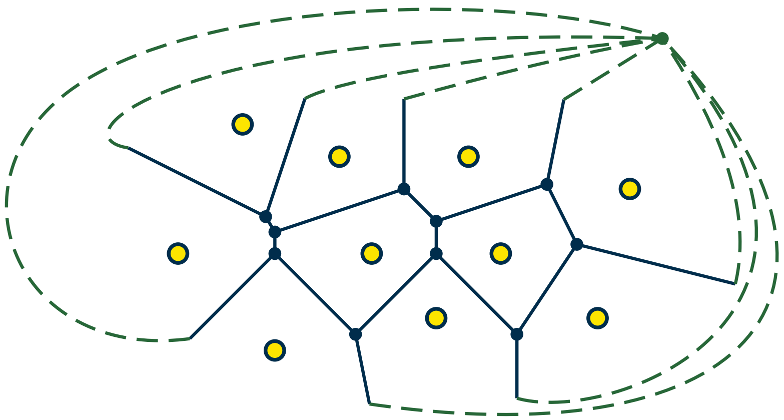
$\Omega(n \log(n))$ lower bound



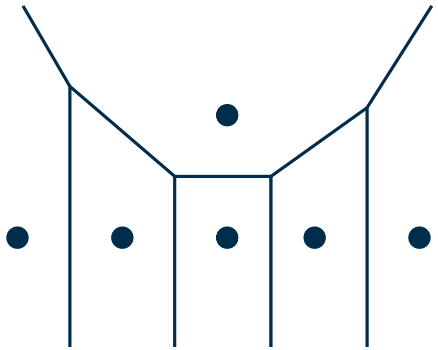
- add vertex to collect loose edges
- n faces
- $v + 1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v + 1)$
- use Euler's formula

Voronoi Diagram

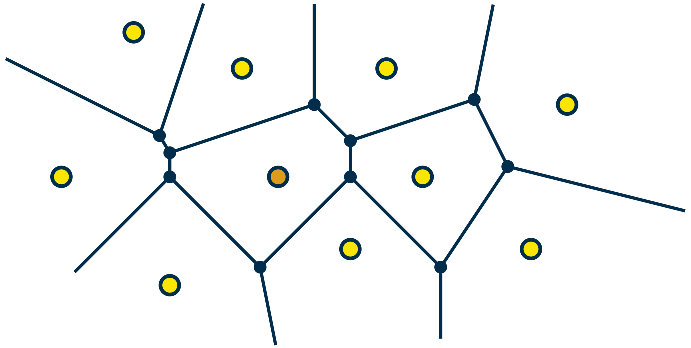
Size of Voronoi Diagram



$\Omega(n\log(n))$ lower bound



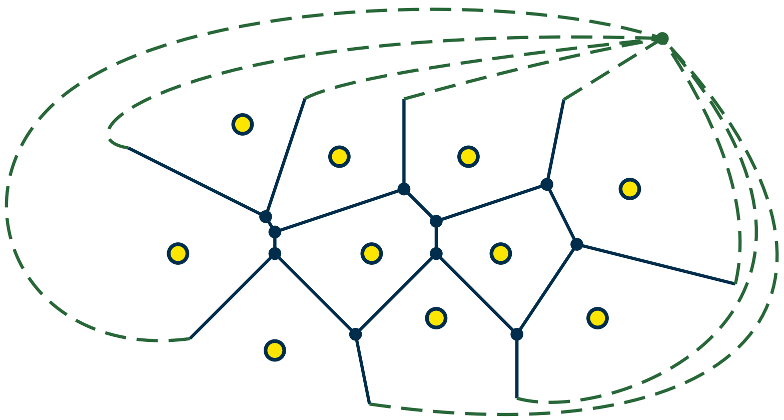
Find closest point



- add vertex to collect loose edges
- n faces
- $v+1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v+1)$
- use Euler's formula

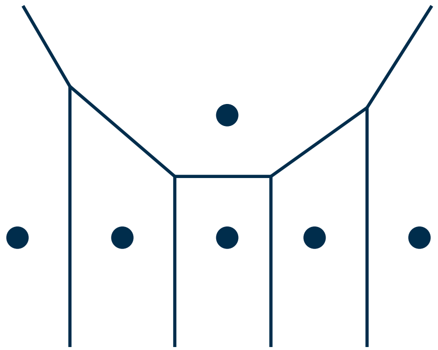
Voronoi Diagram

Size of Voronoi Diagram

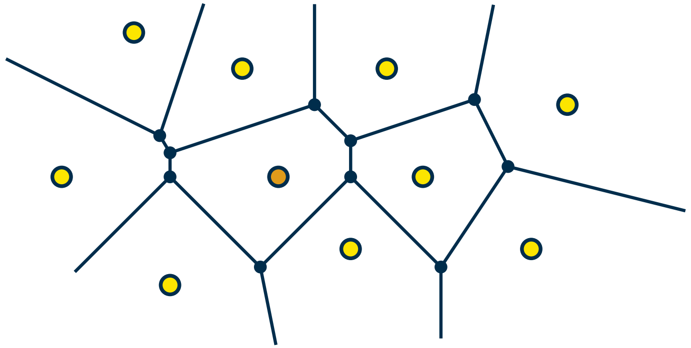


- add vertex to collect loose edges
- n faces
- $v + 1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v + 1)$
- use Euler's formula

$\Omega(n \log(n))$ lower bound



Find closest point

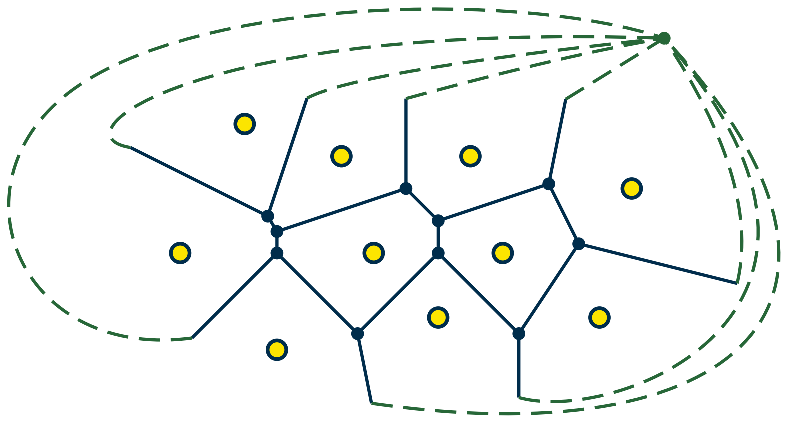


why?

- closest point is in neighbor cell

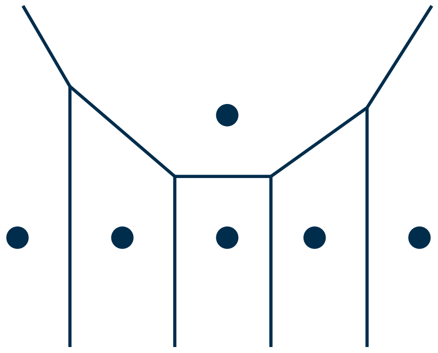
Voronoi Diagram

Size of Voronoi Diagram

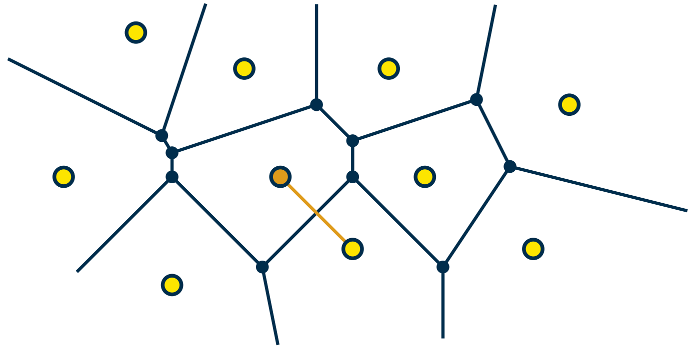


- add vertex to collect loose edges
- n faces
- $v + 1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v + 1)$
- use Euler's formula

$\Omega(n \log(n))$ lower bound



Find closest point

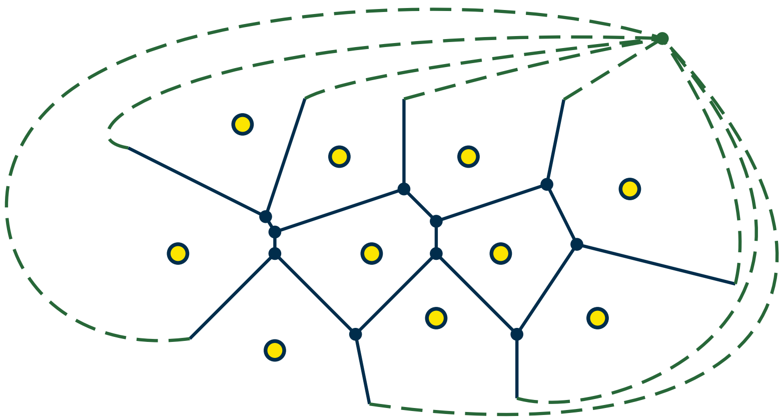


why?

- closest point is in neighbor cell

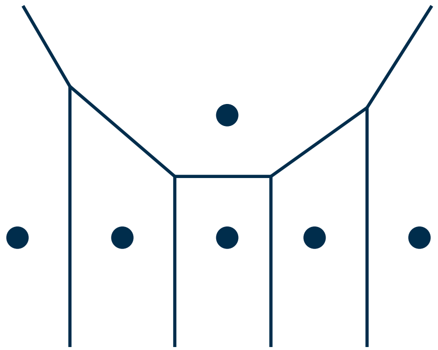
Voronoi Diagram

Size of Voronoi Diagram

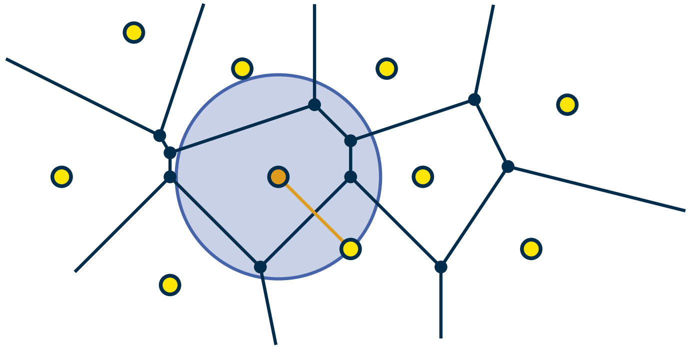


- add vertex to collect loose edges
- n faces
- $v+1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v+1)$
- use Euler's formula

$\Omega(n \log(n))$ lower bound



Find closest point

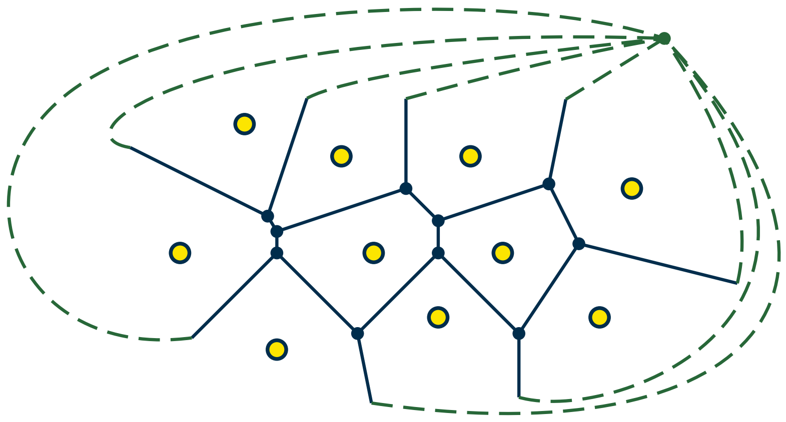


why?

- closest point is in neighbor cell

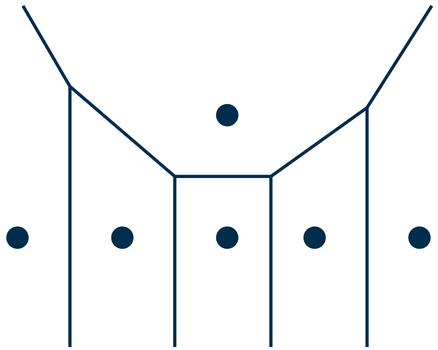
Voronoi Diagram

Size of Voronoi Diagram

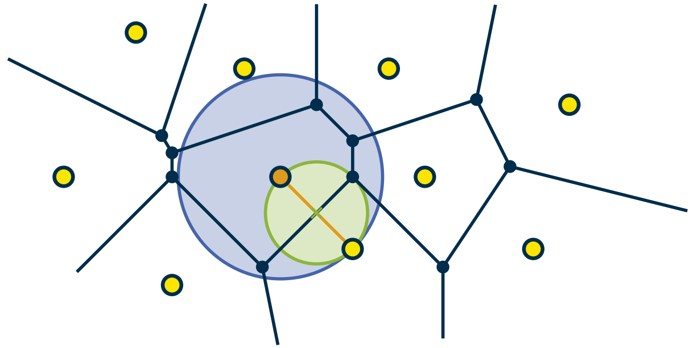


- add vertex to collect loose edges
- n faces
- $v + 1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v + 1)$
- use Euler's formula

$\Omega(n \log(n))$ lower bound



Find closest point

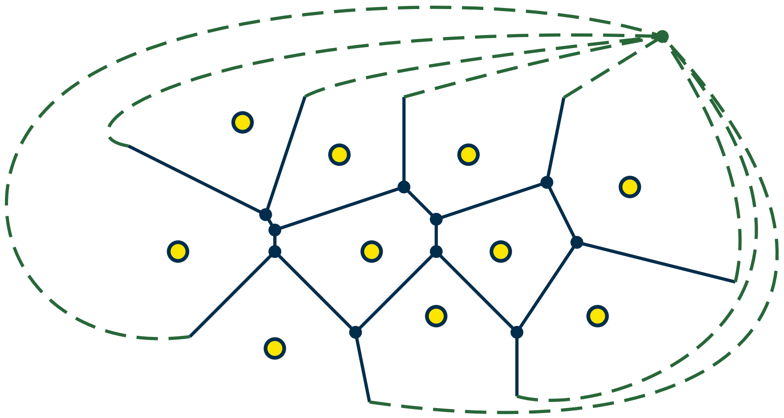


why?

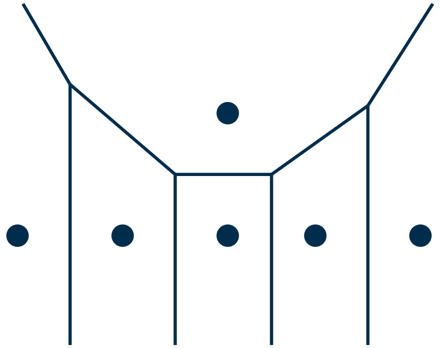
- closest point is in neighbor cell

Voronoi Diagram

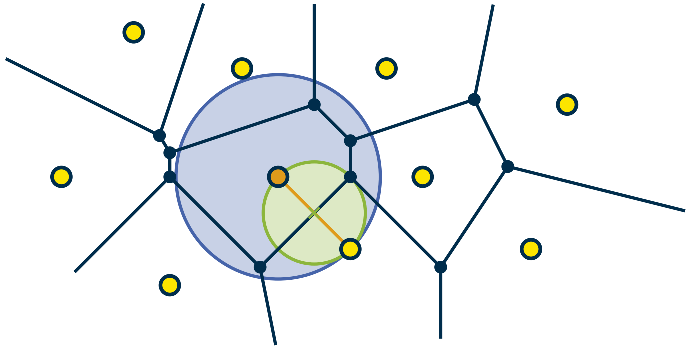
Size of Voronoi Diagram



$\Omega(n \log(n))$ lower bound



Find closest point



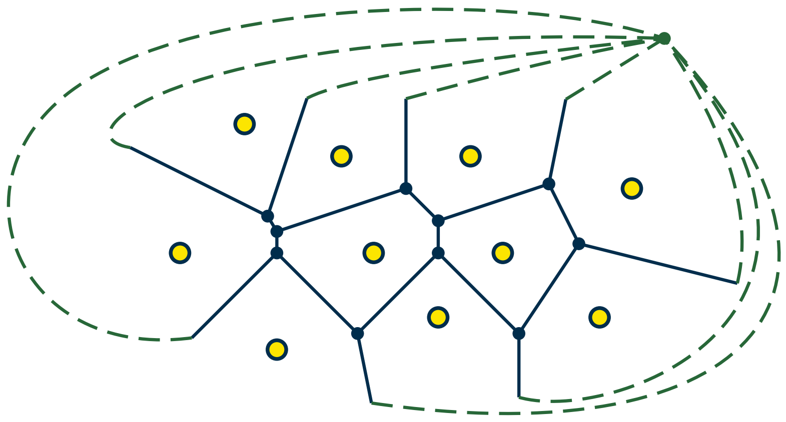
- add vertex to collect loose edges
- n faces
- $v + 1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v + 1)$
- use Euler's formula

why?

- closest point is in neighbor cell
- mid point lies on edge of cell

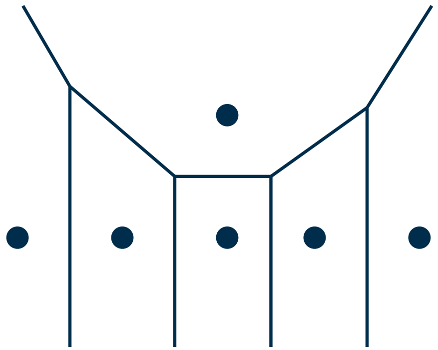
Voronoi Diagram

Size of Voronoi Diagram

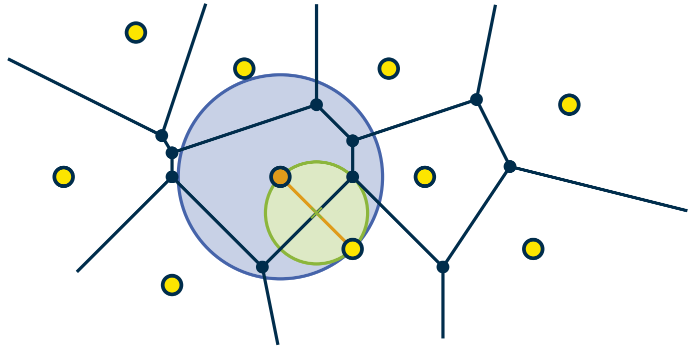


- add vertex to collect loose edges
- n faces
- $v + 1$ vertices, with degree ≥ 3
- m edges with $2m \geq 3(v + 1)$
- use Euler's formula

$\Omega(n \log(n))$ lower bound



Find closest point



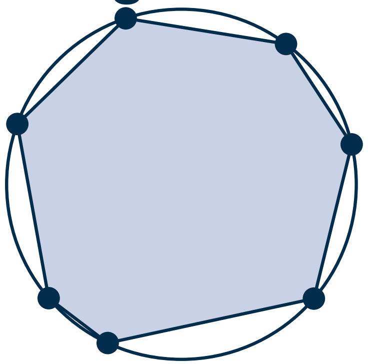
why?

- closest point is in neighbor cell
- mid point lies on edge of cell
- for each point: try all neighbors in VD

Running time?

Assignment 5

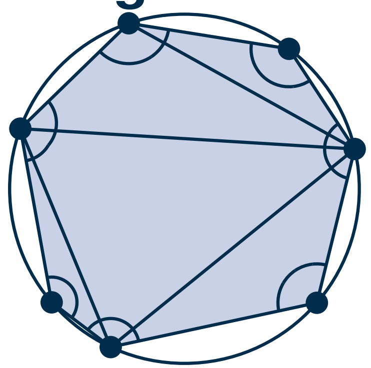
Triangulation of Concentric Points



find optimal triangulation
(smallest angle vector)

Assignment 5

Triangulation of Concentric Points

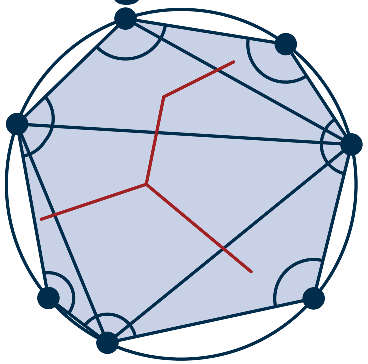


find optimal triangulation
(smallest angle vector)

- maximum angle vector ⇔ maximum length vector

Assignment 5

Triangulation of Concentric Points

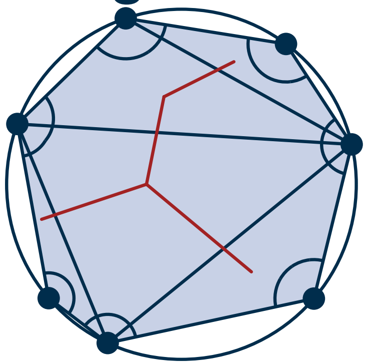


find optimal triangulation
(smallest angle vector)

- maximum angle vector ⇔ maximum length vector
- weak dual graph is a path in optimal triangulation

Assignment 5

Triangulation of Concentric Points



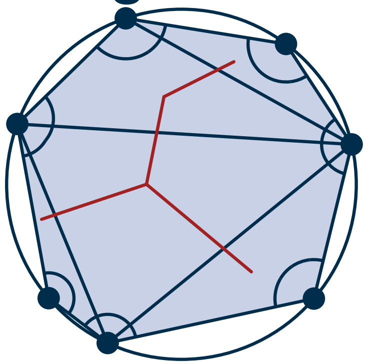
find optimal triangulation
(smallest angle vector)

- maximum angle vector ⇔ maximum length vector
- weak dual graph is a path in optimal triangulation
- algorithm

try to maximize leaves of path

Assignment 5

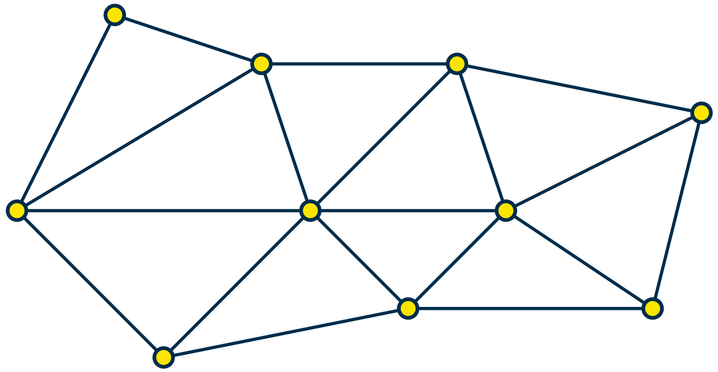
Triangulation of Concentric Points



find optimal triangulation
(smallest angle vector)

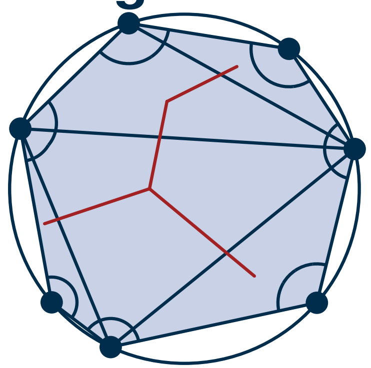
- maximum angle vector ⇔ maximum length vector
- weak dual graph is a path in optimal triangulation
- algorithm try to maximize leaves of path

MST ⊆ Delaunay



Assignment 5

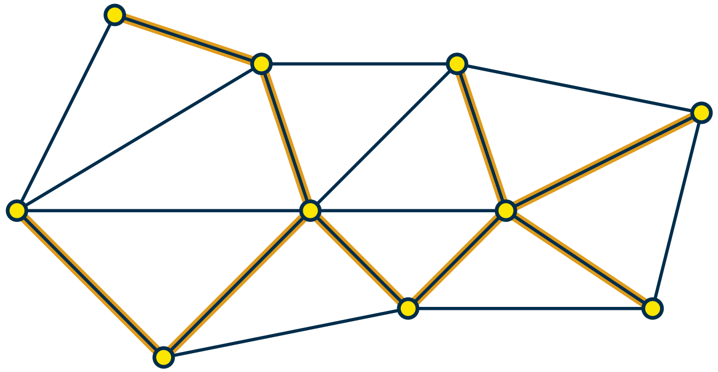
Triangulation of Concentric Points



find optimal triangulation
(smallest angle vector)

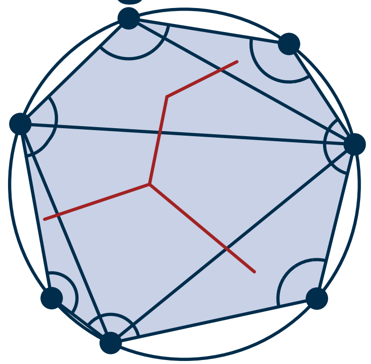
- maximum angle vector ⇔ maximum length vector
- weak dual graph is a path in optimal triangulation
- algorithm try to maximize leaves of path

MST ⊆ Delaunay



Assignment 5

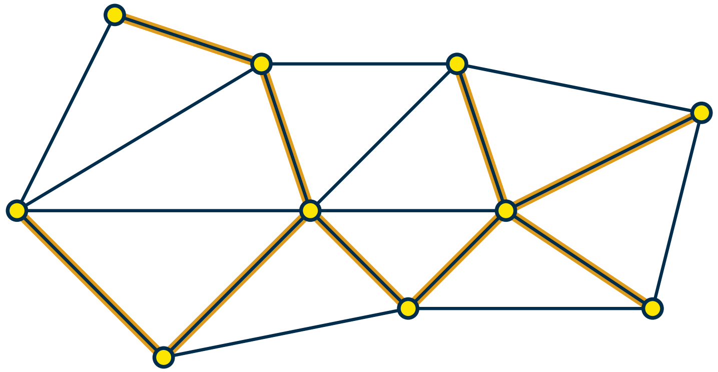
Triangulation of Concentric Points



find optimal triangulation
(smallest angle vector)

- maximum angle vector ⇔ maximum length vector
- weak dual graph is a path in optimal triangulation
- algorithm try to maximize leaves of path

MST ⊆ Delaunay

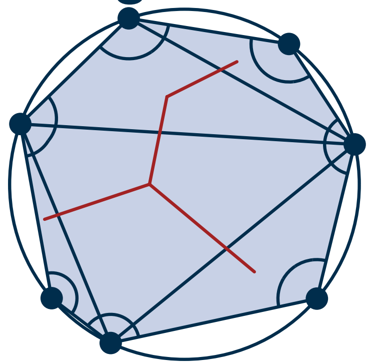


Foldability of Mountain/Valley Patterns

- test foldability in $O(n)$

Assignment 5

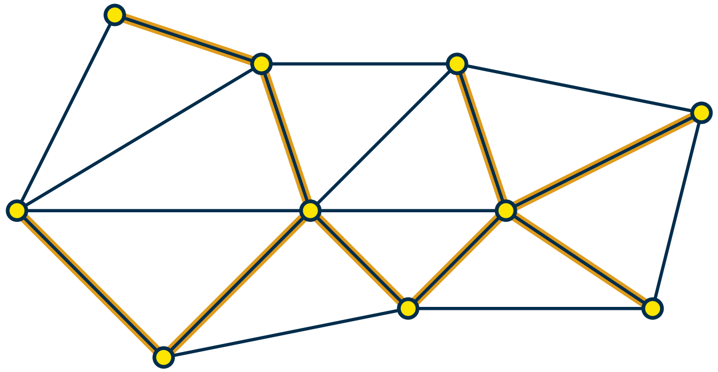
Triangulation of Concentric Points



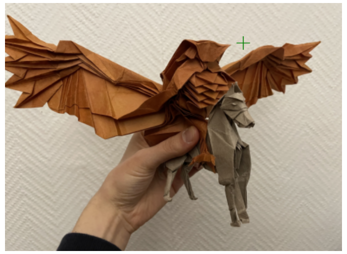
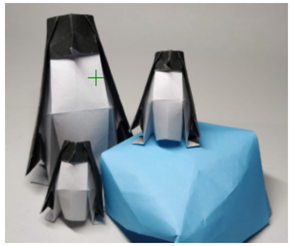
find optimal triangulation
(smallest angle vector)

- maximum angle vector ⇔ maximum length vector
- weak dual graph is a path in optimal triangulation
- algorithm try to maximize leaves of path

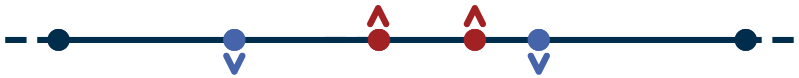
MST ⊆ Delaunay



Creative Outlet


Foldability of Mountain/Valley Patterns



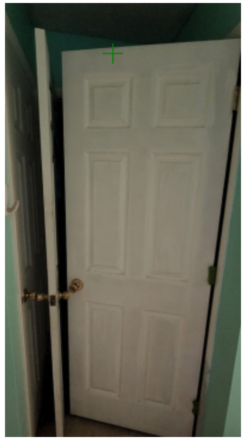
- test foldability in $O(n)$

Blocking Doors

- doors are weird and can block each other

Blocking Doors

- doors are weird and can block each other

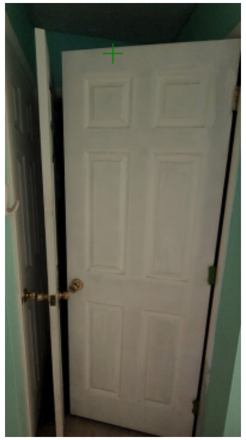


Blocking Doors

- doors are weird and can block each other

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

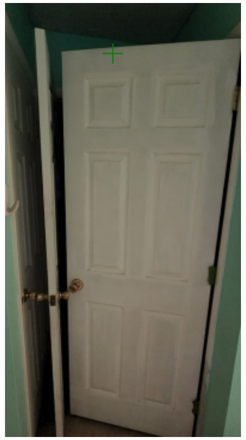


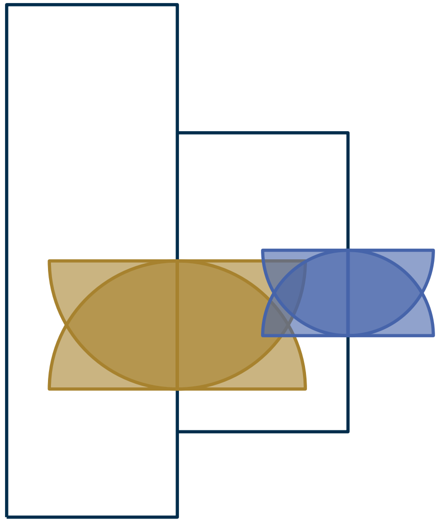
Blocking Doors

- doors are weird and can block each other

Problem: BlockingDoors (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?



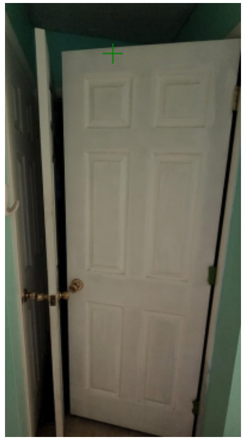


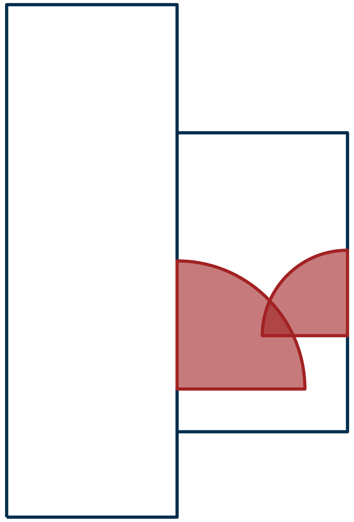
Blocking Doors

- doors are weird and can block each other

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

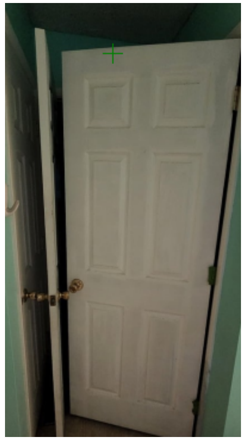
Is is possible to assign every door a hinge and an opening direction such that no doors block each other?



Blocking Doors

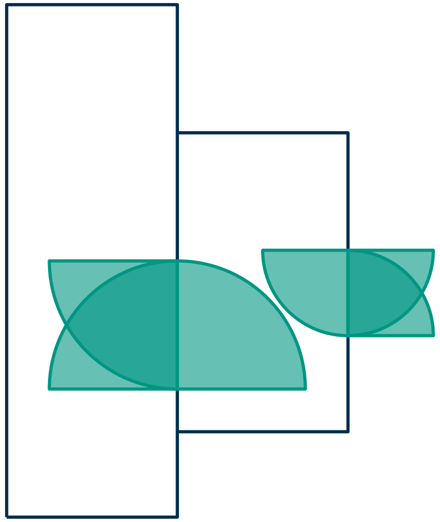
- doors are weird and can block each other



Problem: BLOCKINGDOORS (BA Oğuz)

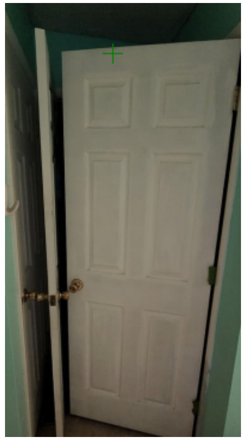
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?



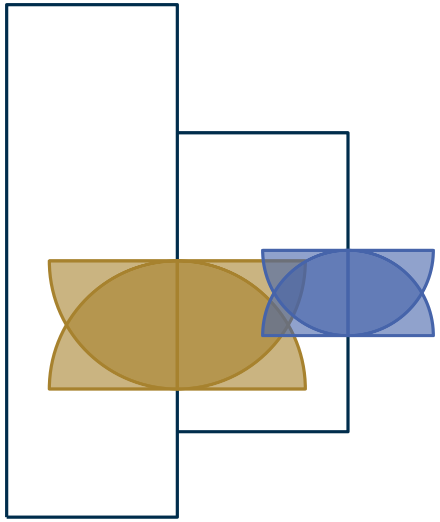
Blocking Doors

- doors are weird and can block each other



Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

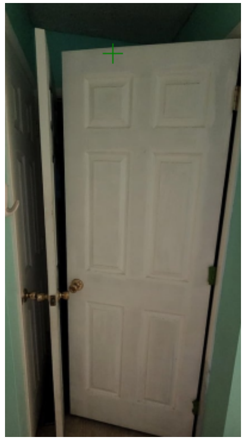


- Show that BLOCKINGDOORS is NP-hard.

Build gadgets!

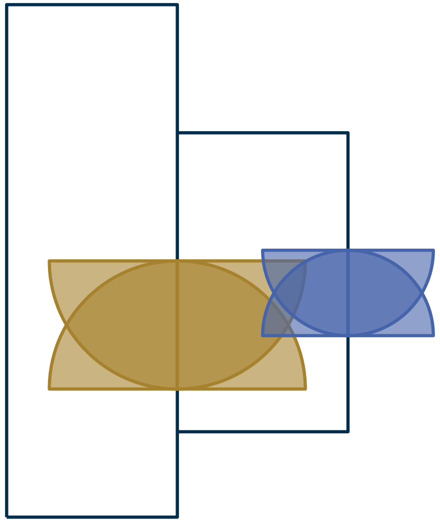
Blocking Doors

- doors are weird and can block each other



Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?



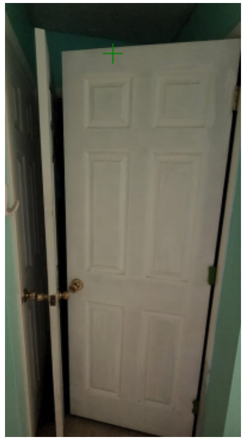
- Show that BLOCKINGDOORS is NP-hard.

Build gadgets!

- Is it NP-complete?

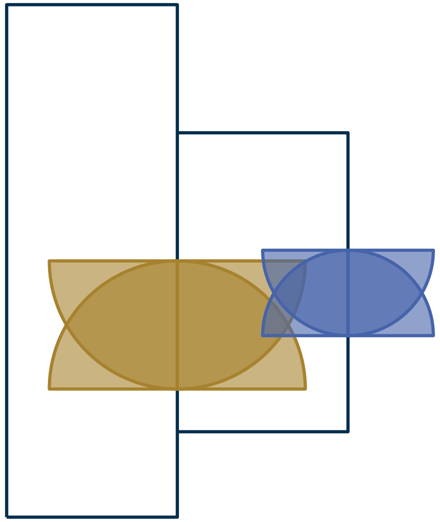
Blocking Doors

- doors are weird and can block each other



Problem: BlockingDoors (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?



- Show that BlockingDoors is NP-hard.

Build gadgets!

- Is it NP-complete?
- Complexity if the side of the hinge is given for each door?

Blocking Doors

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

- Show that BLOCKING DOORS is NP-hard by reduction from PLANAR MONOTONE RECTILINEAR 3-SAT.

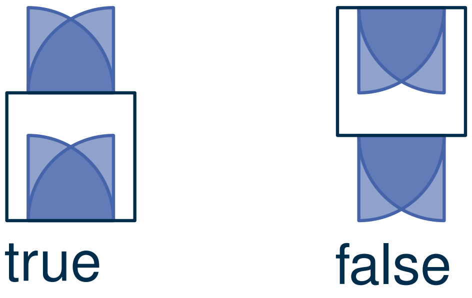
Blocking Doors

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

- Show that BLOCKING DOORS is NP-hard by reduction from PLANAR MONOTONE RECTILINEAR 3-SAT.

Variables



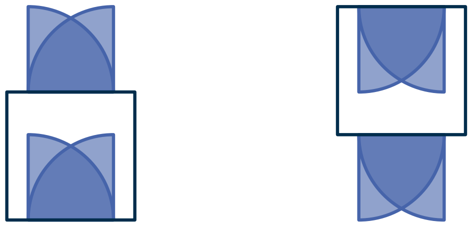
Blocking Doors

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

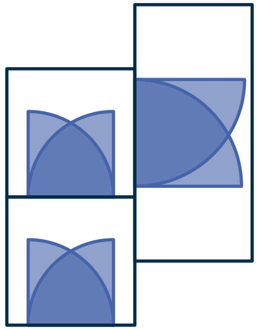
- Show that BLOCKING DOORS is NP-hard by reduction from PLANAR MONOTONE RECTILINEAR 3-SAT.

Variables



true false

Transport

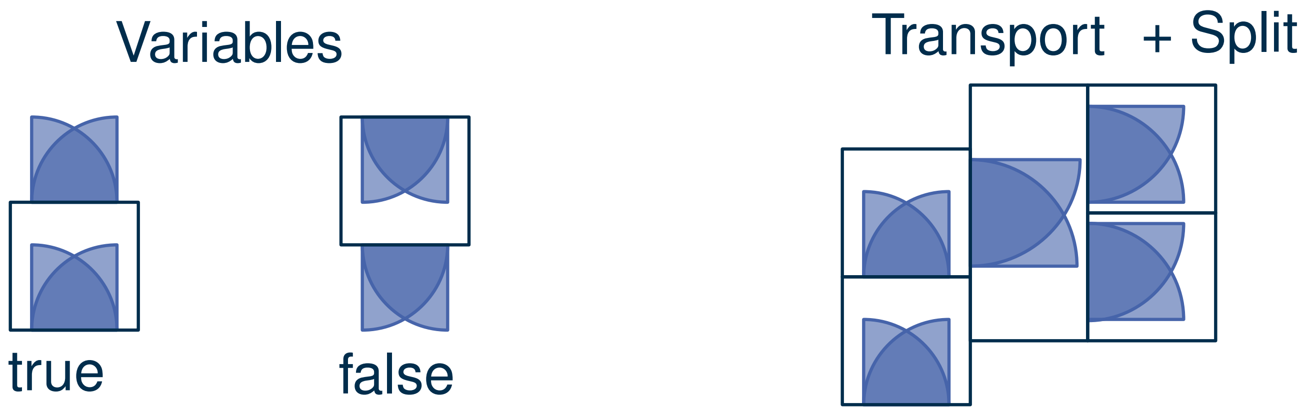


Blocking Doors

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

- Show that BLOCKING DOORS is NP-hard by reduction from PLANAR MONOTONE RECTILINEAR 3-SAT.



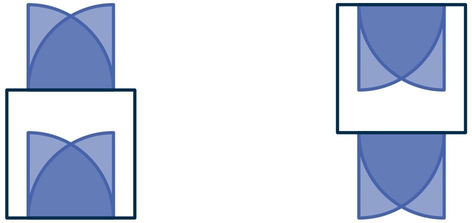
Blocking Doors

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

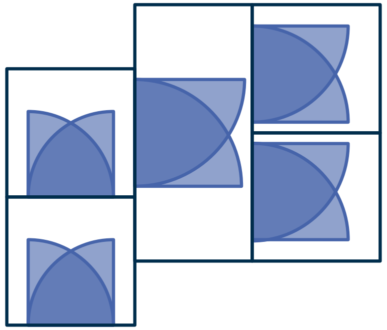
- Show that BLOCKING DOORS is NP-hard by reduction from PLANAR MONOTONE RECTILINEAR 3-SAT.

Variables



true false

Transport + Split



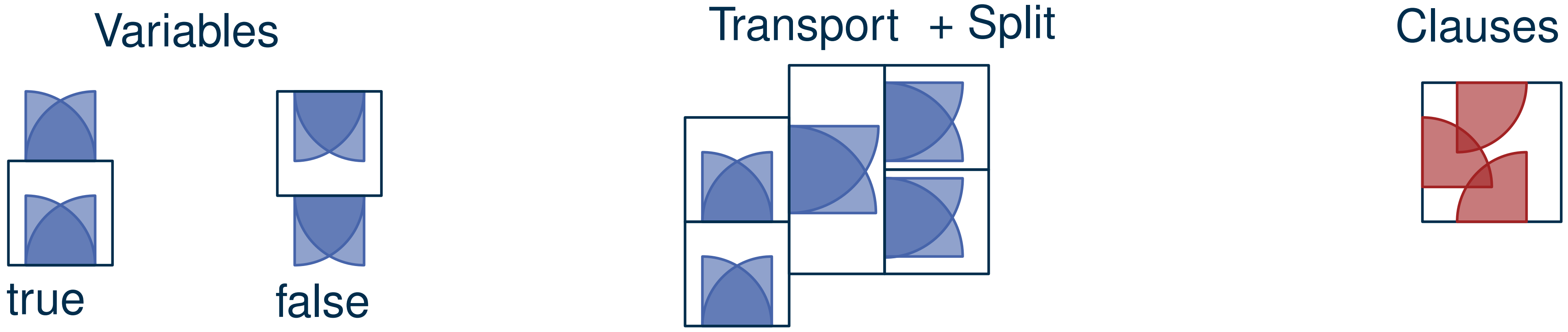
Clauses

Blocking Doors

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

- Show that BLOCKING DOORS is NP-hard by reduction from PLANAR MONOTONE RECTILINEAR 3-SAT.



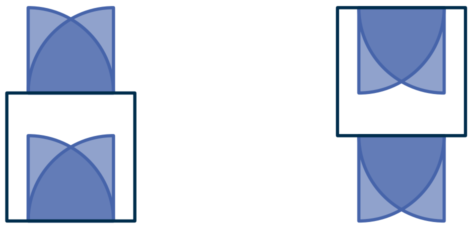
Blocking Doors

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

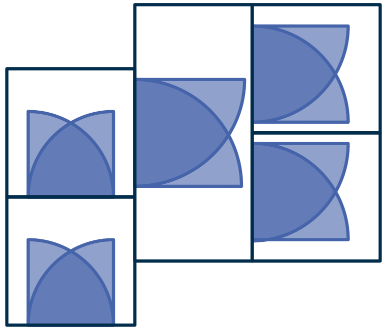
- Show that BLOCKING DOORS is NP-hard by reduction from PLANAR MONOTONE RECTILINEAR 3-SAT.

Variables



true false

Transport + Split



Clauses

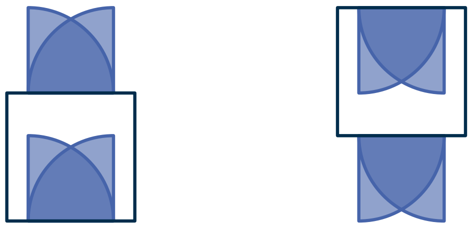
Blocking Doors

Problem: BLOCKINGDOORS (BA Oğuz)
Given: Set of rectangular rooms with a set of doors

Is is possible to assign every door a hinge and an opening direction such that no doors block each other?

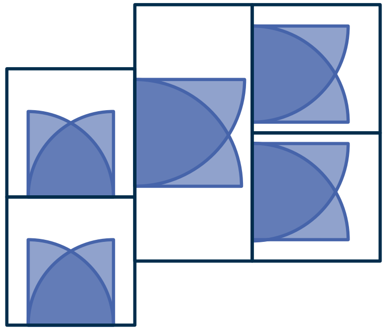
- Show that BLOCKING DOORS is NP-hard by reduction from PLANAR MONOTONE RECTILINEAR 3-SAT.

Variables



true false

Transport + Split



Clauses