Input-Thrifty Algorithms
and hyperbolic dovetailing

David Kirkpatrick UBC

CS 420 – Spring 2015

- Rolf Klein
- Robert Tseng
- Sandra Zilles and
- (especially) Raimund Seidel

Acknowledgements

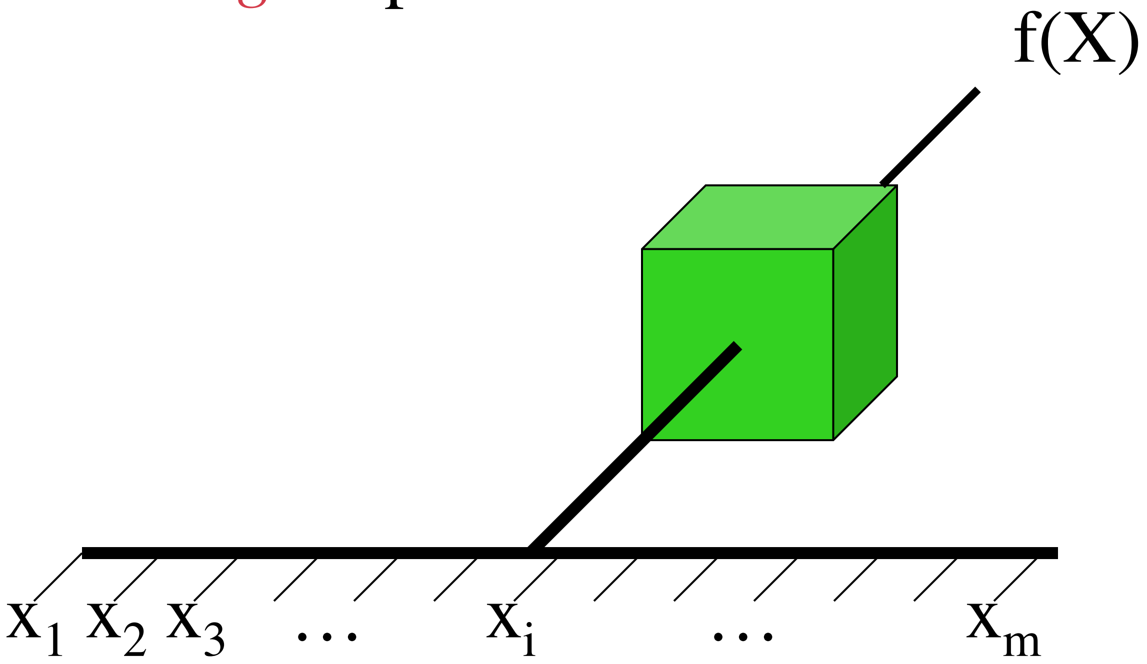
- **Introduction and motivation**
    - **Input-thrifty algorithms**
- **List search**
- **Hyperbolic dovetailing**
- **Applications to input-thrifty algorithms**
- **Extensions & generalizations**

Overview

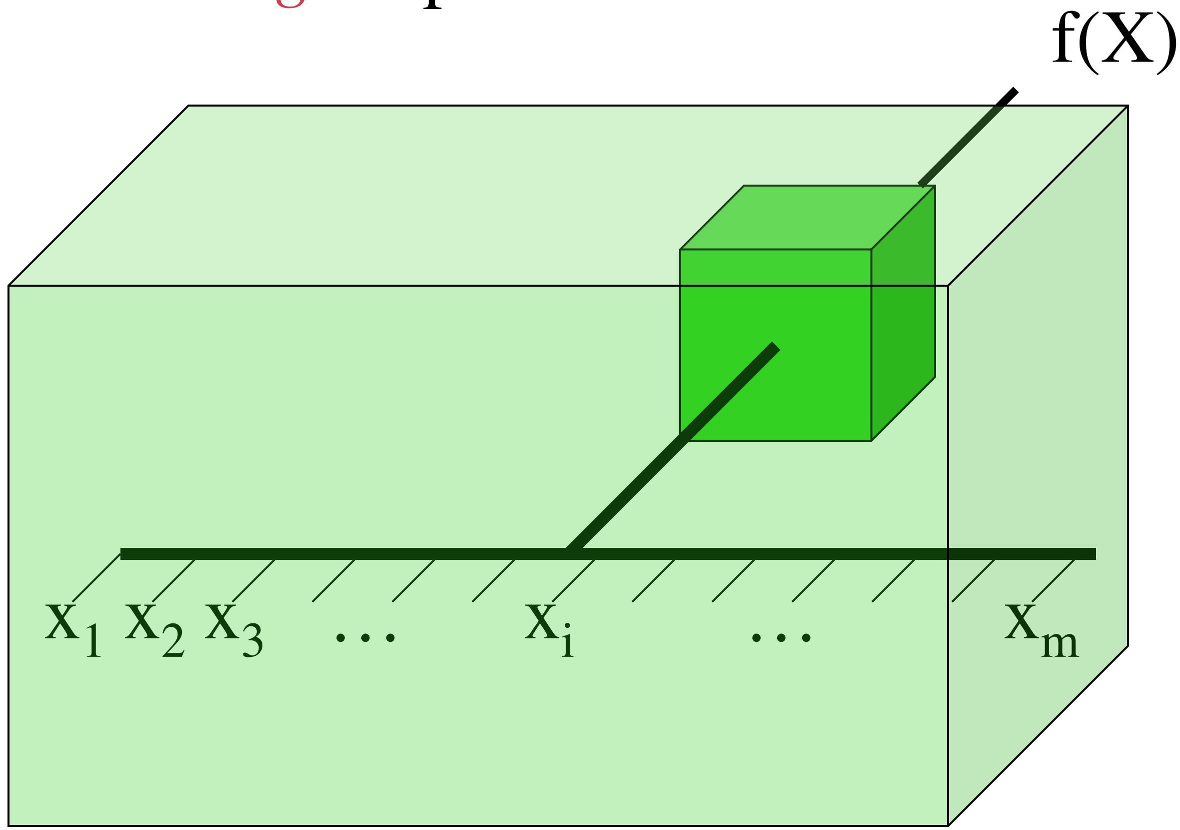
- Introduction and motivation
  - Input-thrifty algorithms
- List search
- Hyperbolic dovetailing
- Applications to input-thrifty algorithms
- Extensions & generalizations

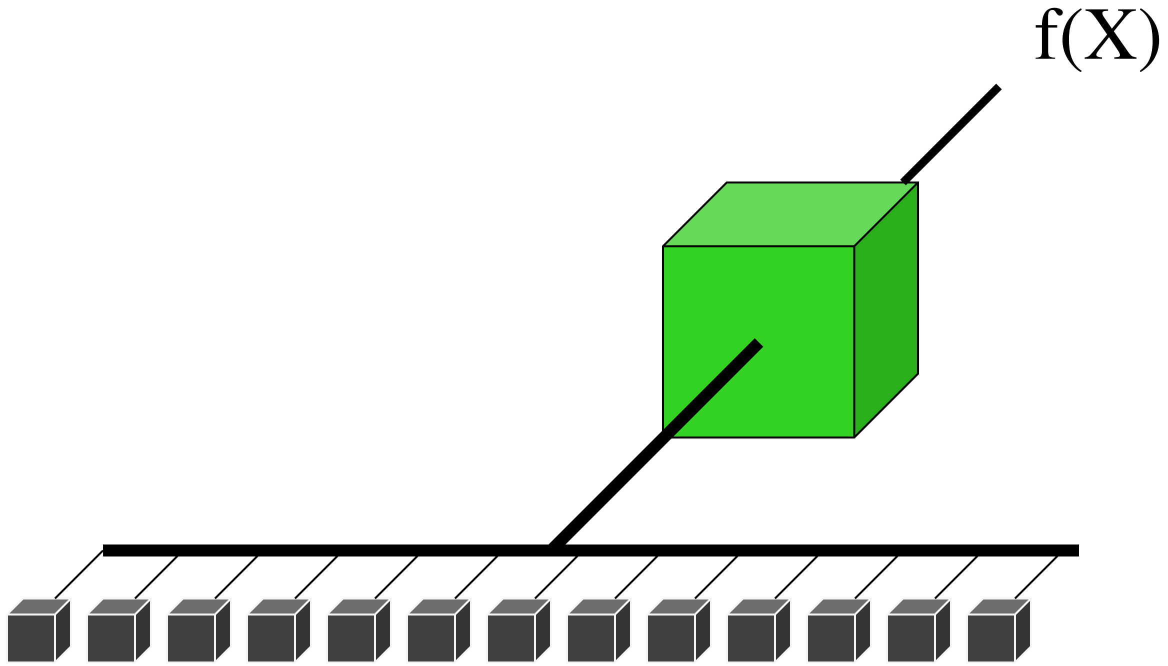
Overview

computing *symmetric* functions
over *large* input/data sets

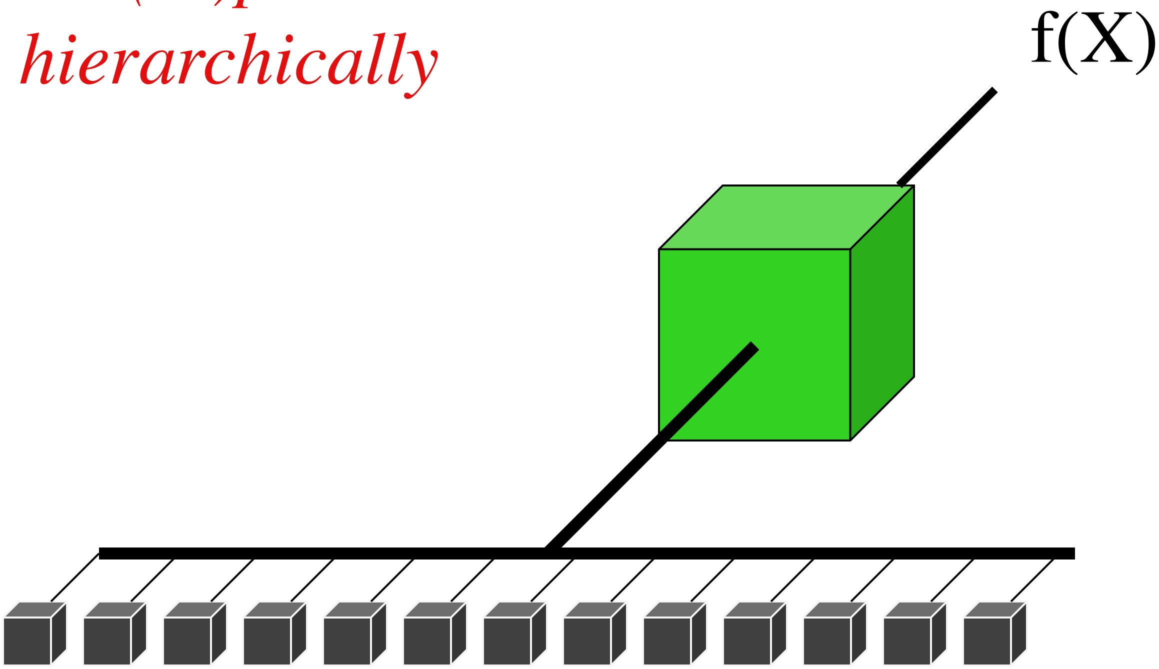


computing *symmetric* functions
over *large* input/data sets

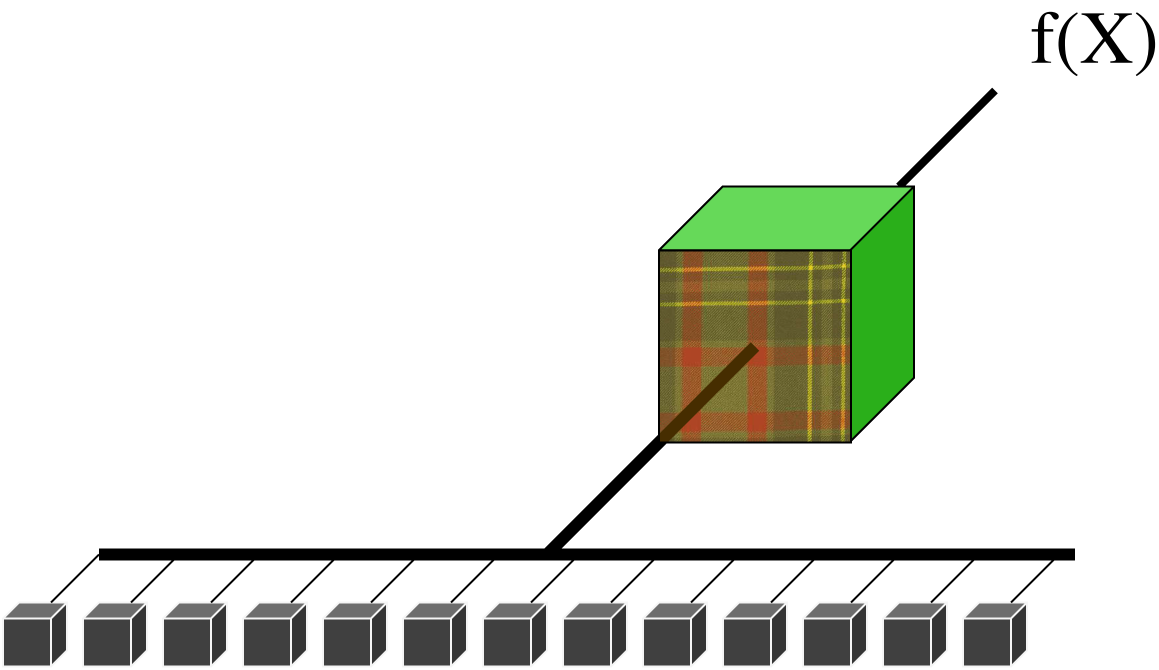



f(X)

individual input/data items
are *(re)presented*
hierarchically

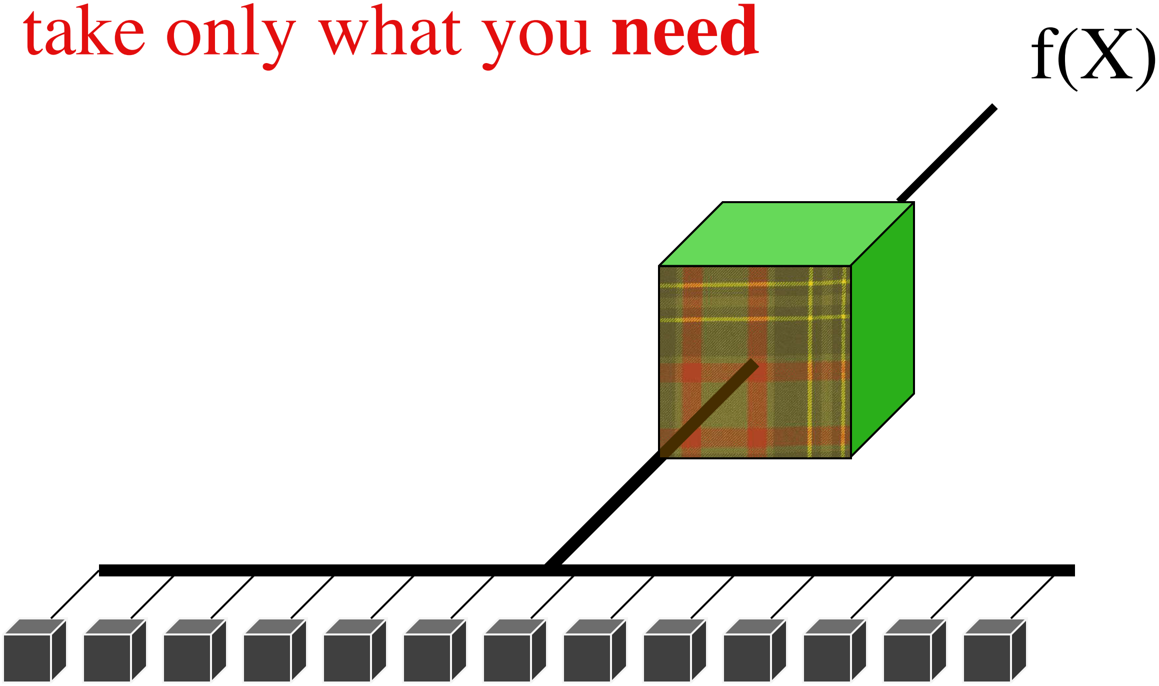


thrifty input/data consumption:



thrifty input/data consumption:

take only what you **need**



Motivation

Some functions can be computed with less than full precision.

Inputs/data may initially be known up to some limited precision/certainty; greater precision is available, but at additional cost.

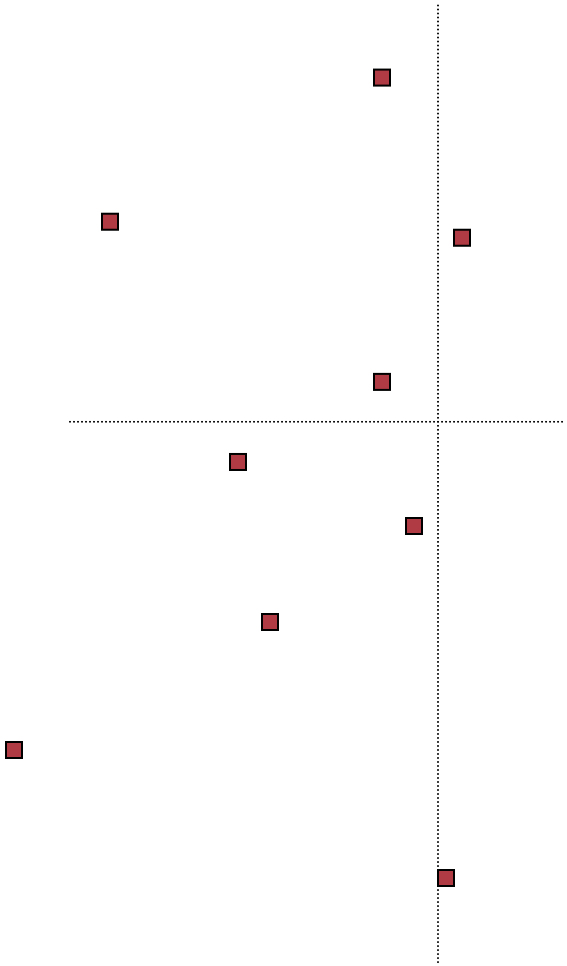
- sensor data
- implicit representation--e.g. root-finding
- hierarchical data structures
- sampling error

Motivation

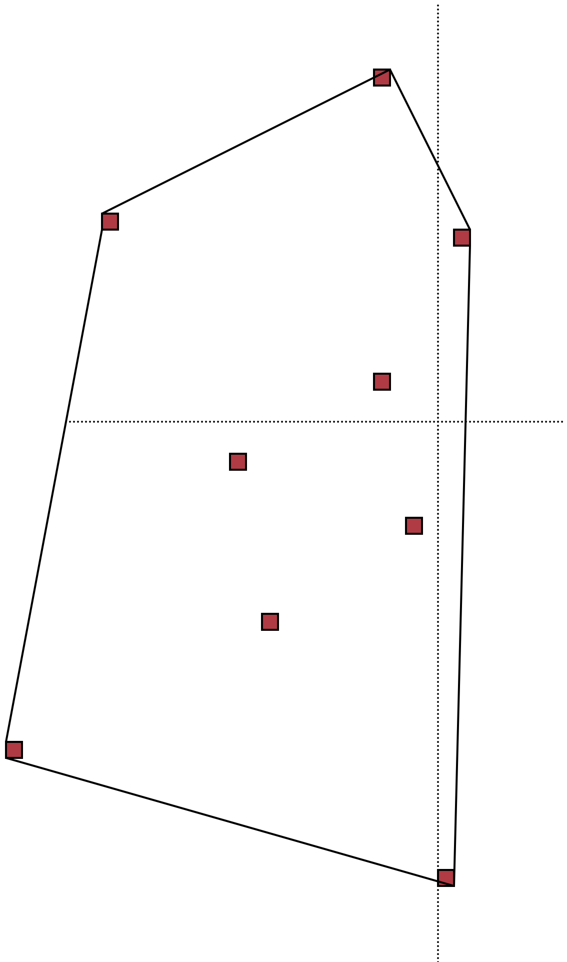
Some functions can be computed with less than full precision.

Inputs/data may initially be known up to some limited precision/certainty; greater precision is available, but at additional cost.

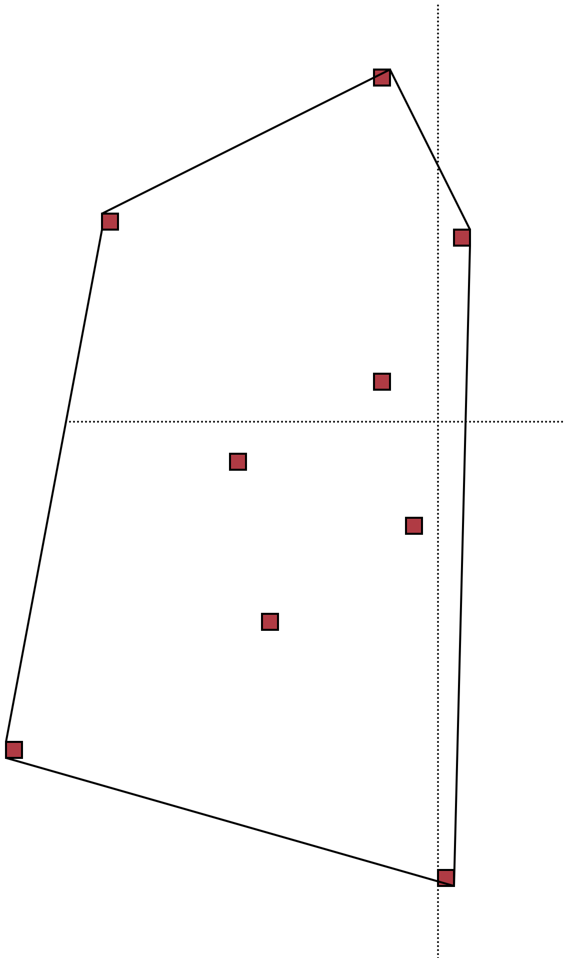
- sensor data
- implicit representation--e.g. root-finding
- hierarchical data structures
- sampling error



Leo Guibas (2003): “Given m points in the plane, does their convex hull contain the origin? Suppose further that the points have (log n)-bit coordinates. How many bits of the input do we need to examine to answer the question, in the worst case? More specifically, if there is a proof using only d bits, can we find it by examining only O(d) bits?”

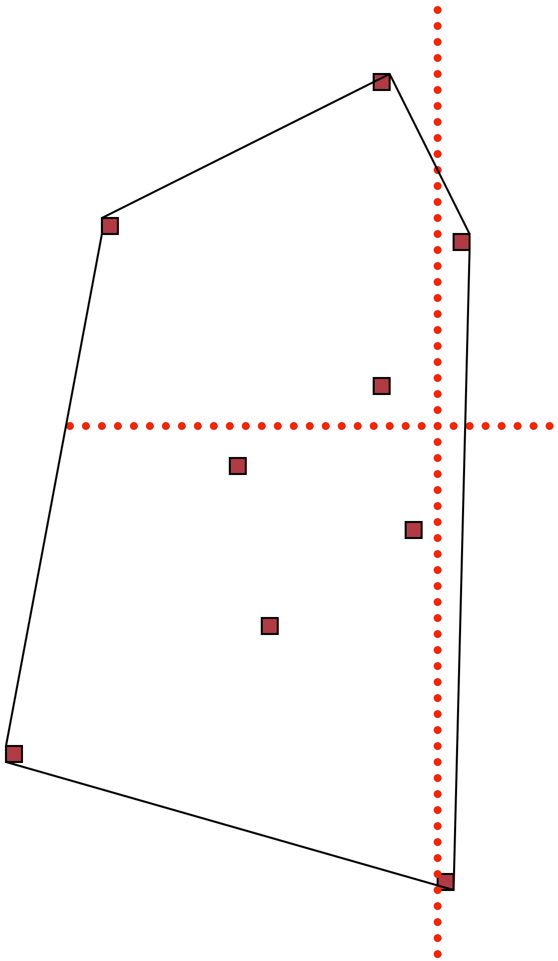


Leo Guibas (2003): "Given m points in the plane, does their convex hull contain the origin? Suppose further that the points have (log n)-bit coordinates. How many bits of the input do we need to examine to answer the question, in the worst case? More specifically, if there is a proof using only d bits, can we find it by examining only O(d) bits?"



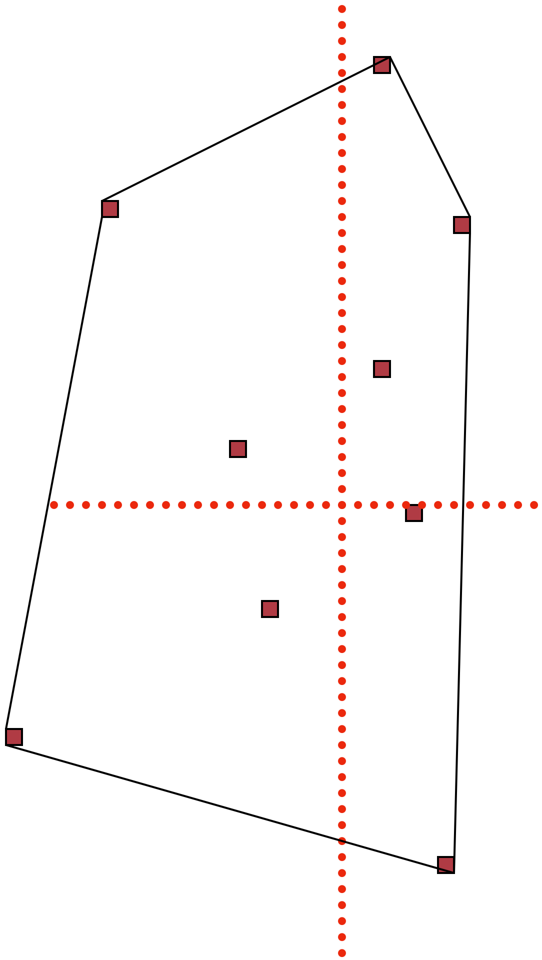
Origin-enclosure

Leo Guibas (2003): "Given m points in the plane, does their convex hull contain the origin? Suppose further that the points have (log n)-bit coordinates. How many bits of the input do we need to examine to answer the question, in the worst case? More specifically, if there is a proof using only d bits, can we find it by examining only O(d) bits?"



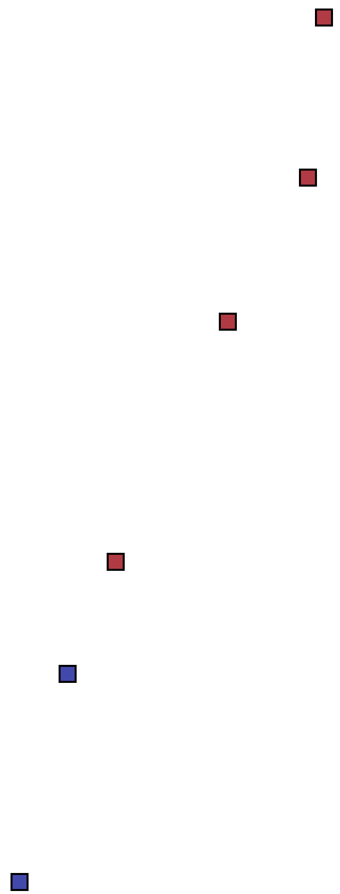
Origin-enclosure

Leo Guibas (2003): "Given m points in the plane, does their convex hull contain the origin? Suppose further that the points have (log n)-bit coordinates. How many bits of the input do we need to examine to answer the question, in the worst case? More specifically, if there is a proof using only d bits, can we find it by examining only O(d) bits?"



Origin-enclosure

Leo Guibas (2003): "Given m points in the plane, does their convex hull contain the origin? Suppose further that the points have (log n)-bit coordinates. How many bits of the input do we need to examine to answer the question, in the worst case? More specifically, if there is a proof using only d bits, can we find it by examining only O(d) bits?"

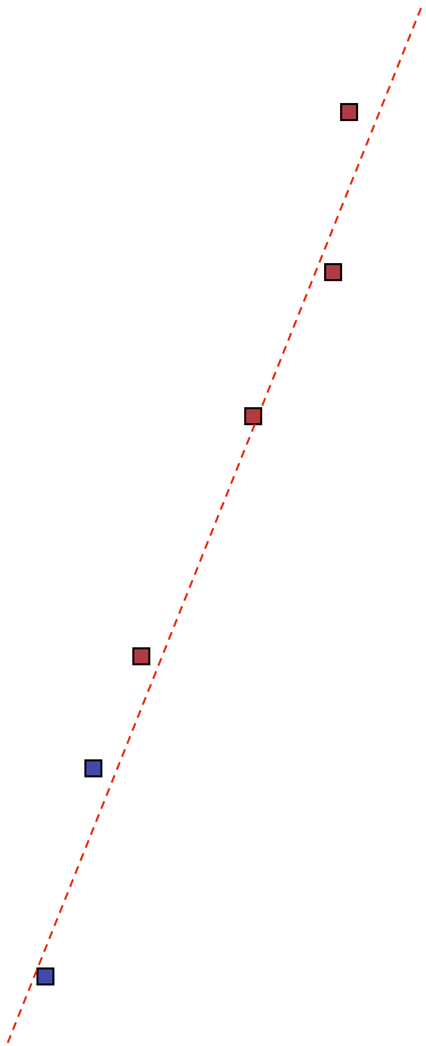


co-linearity (lower dimensionality)

More generally:

If we wish to evaluate some geometric predicate for a set of input points…

How many bits of the input do we need to examine to answer the question, as a function of the number required to certify the result?

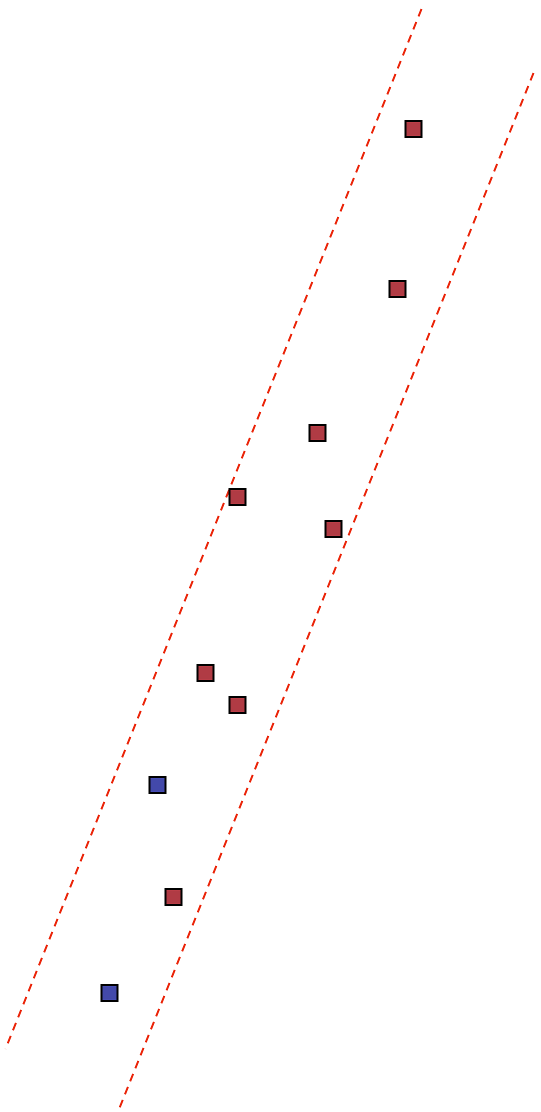


co-linearity (lower dimensionality)

More generally:

If we wish to evaluate some geometric predicate for a set of input points…

How many bits of the input do we need to examine to answer the question, as a function of the number required to certify the result?

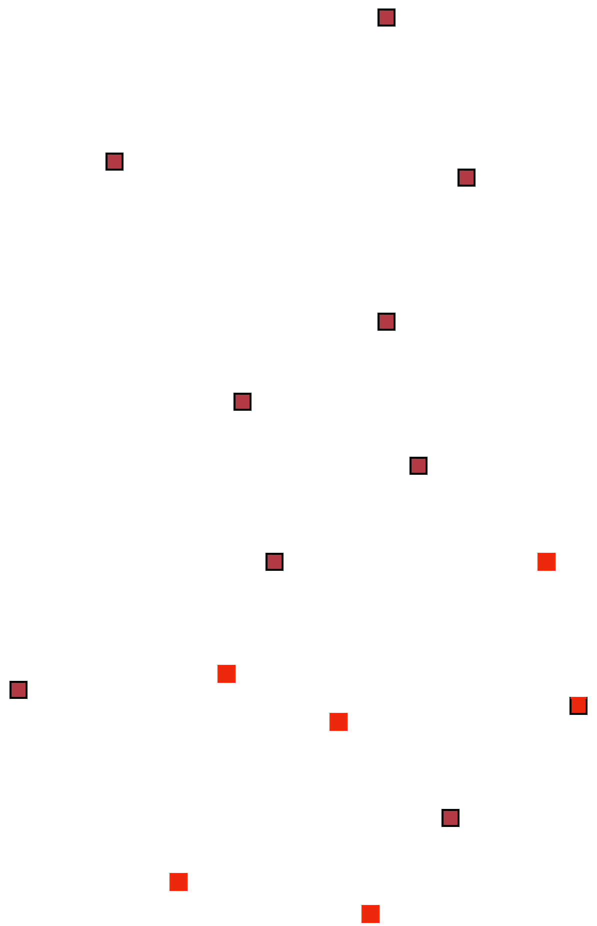


bounded diameter

More generally:

If we wish to evaluate some geometric predicate for a set of input points…

How many bits of the input do we need to examine to answer the question, as a function of the number required to certify the result?

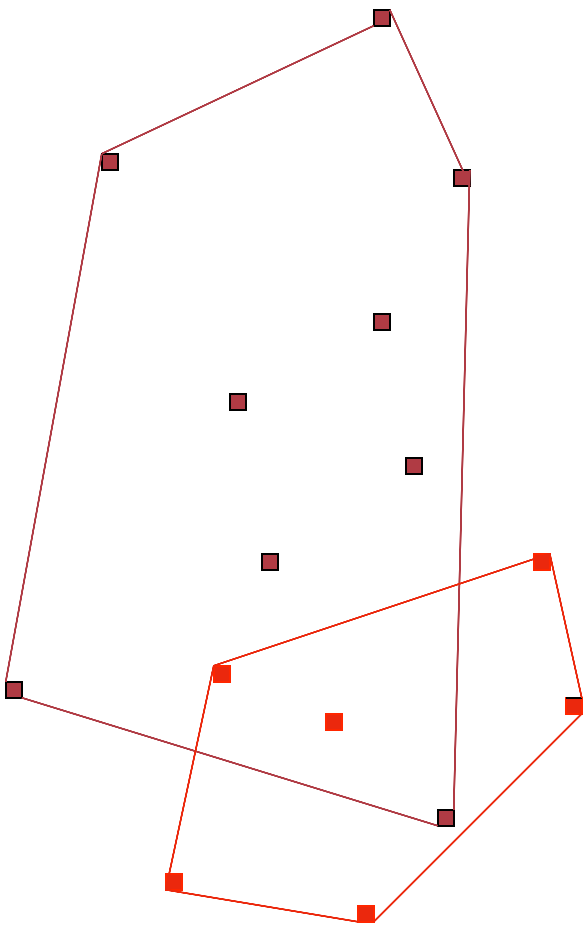


red-blue separability

More generally:

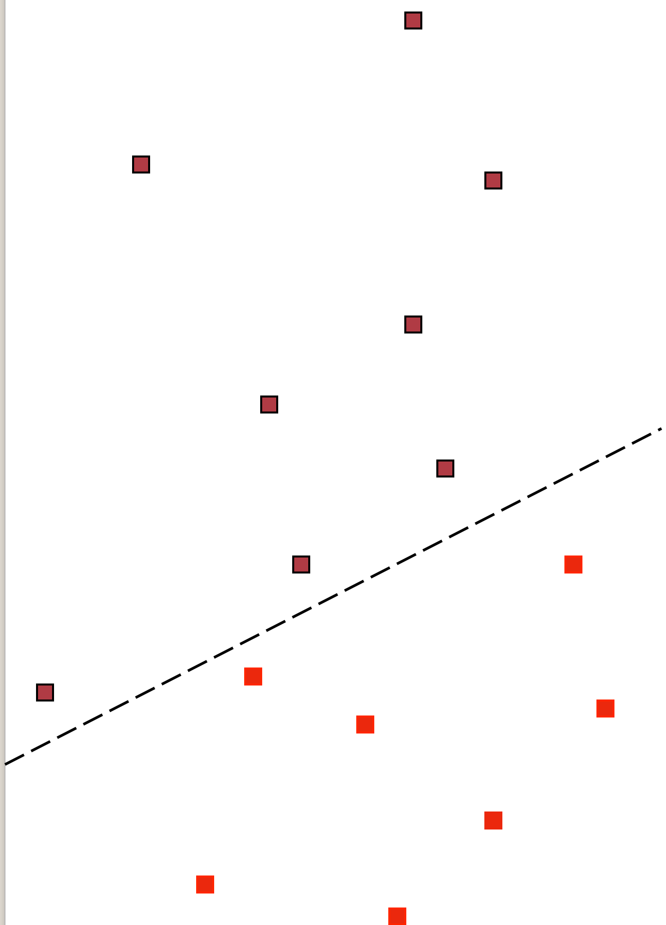
If we wish to evaluate some geometric predicate for a set of input points…

How many bits of the input do we need to examine to answer the question, as a function of the number required to certify the result?



red-blue separability

More generally:
If we wish to evaluate some geometric predicate for a set of input points…
How many bits of the input do we need to examine to answer the question, as a function of the number required to certify the result?



red-blue separability

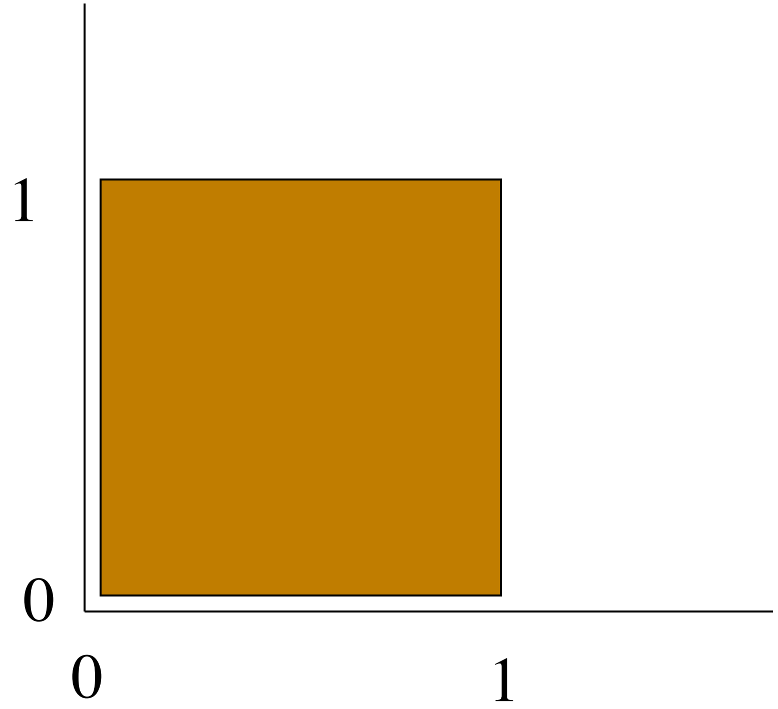
More generally:

If we wish to evaluate some geometric predicate for a set of input points…

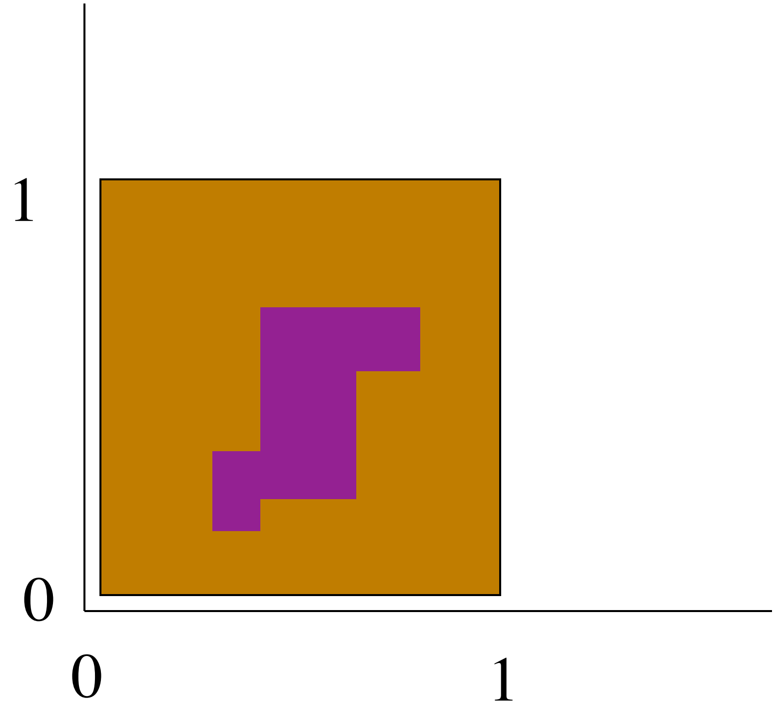
How many bits of the input do we need to examine to answer the question, as a function of the number required to certify the result?

Model (operations)

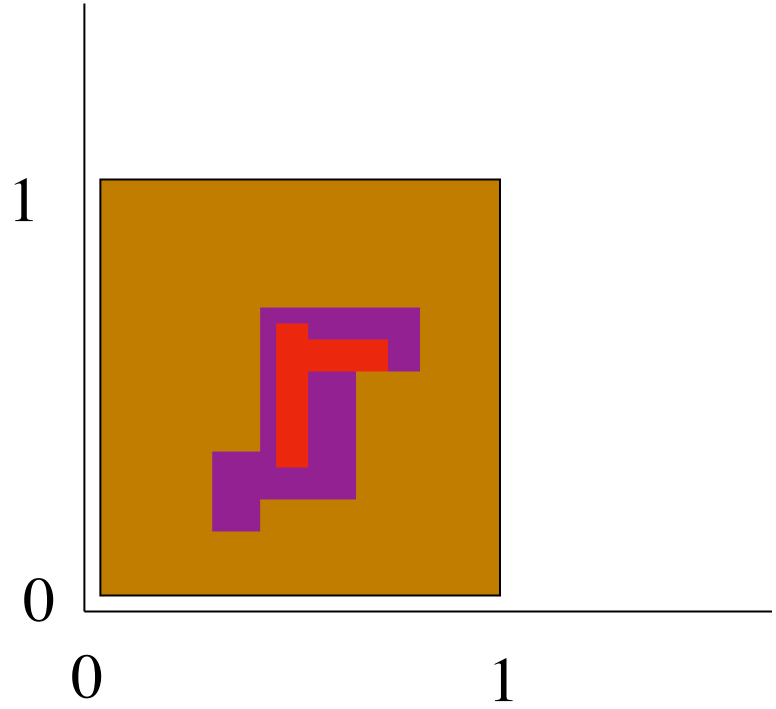
- Arbitrary refinement of uncertainty regions



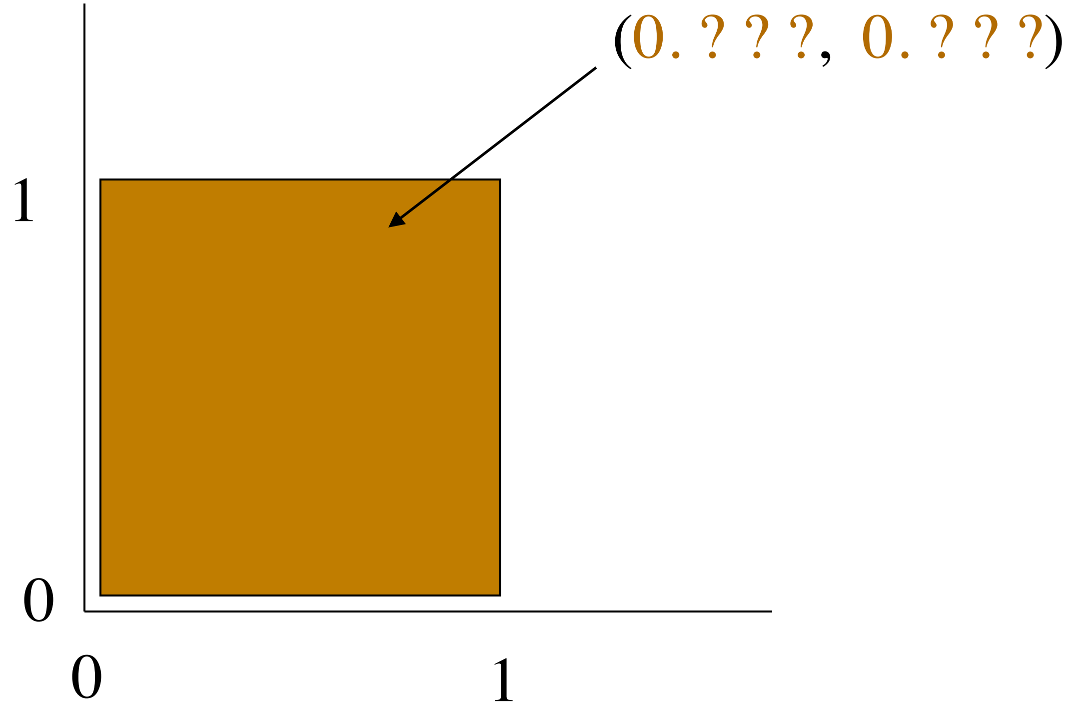
- Arbitrary refinement of uncertainty regions



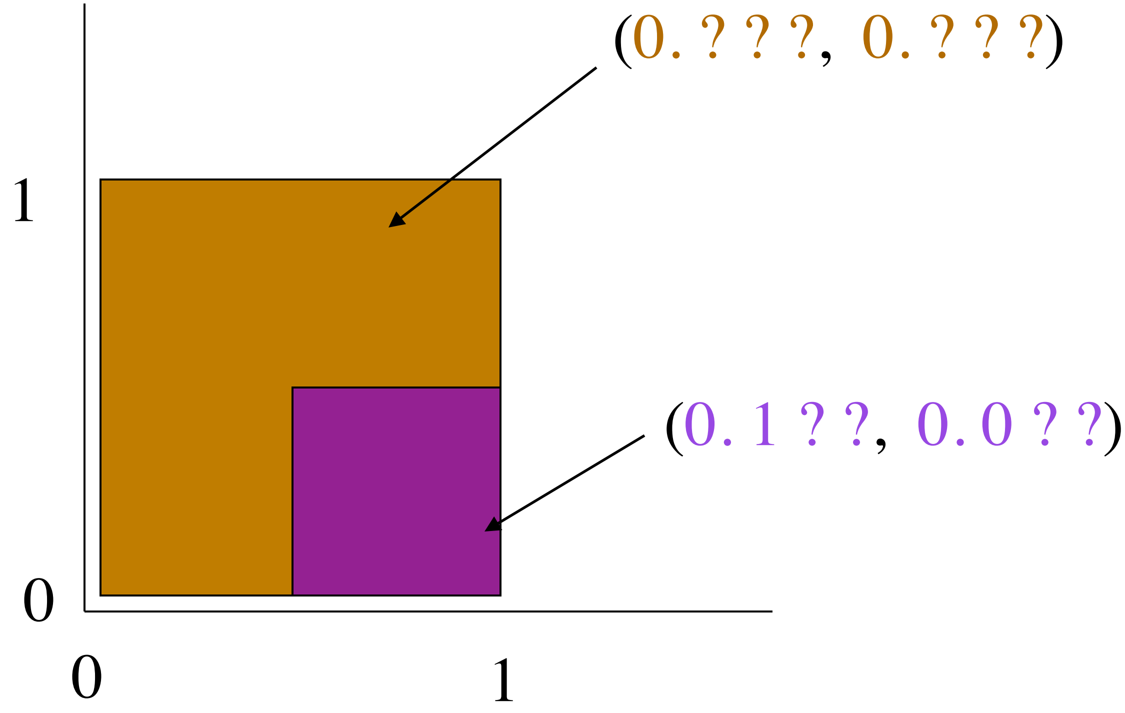
- Arbitrary refinement of uncertainty regions



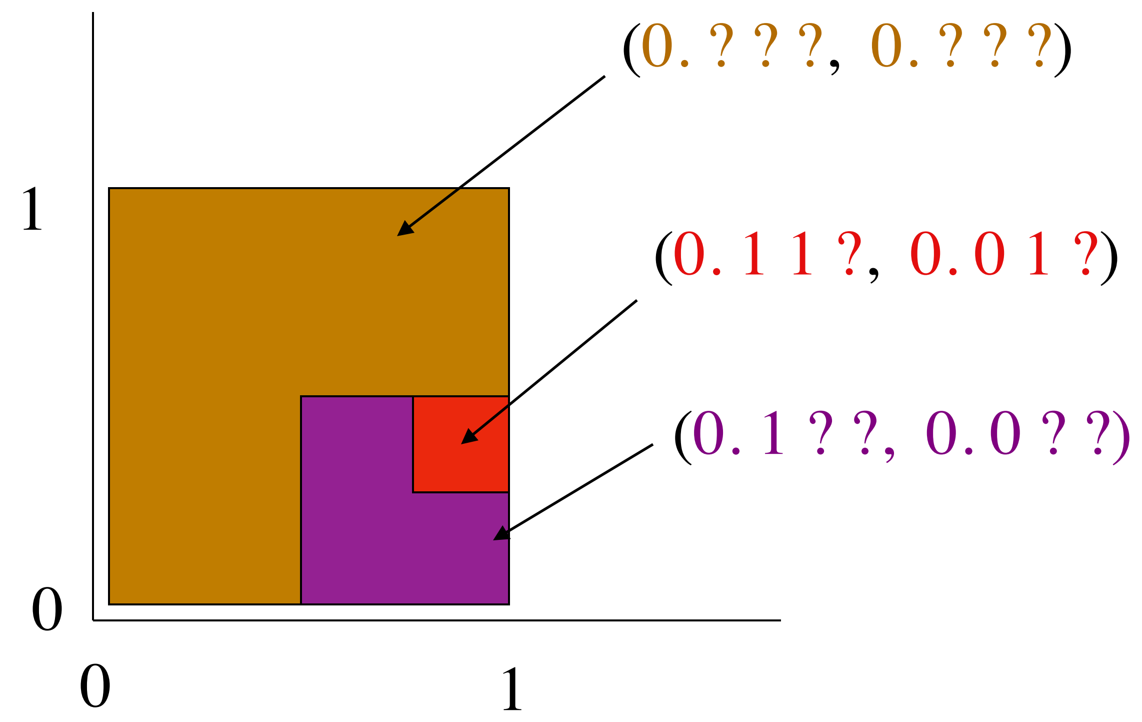
- Sequential bit probes

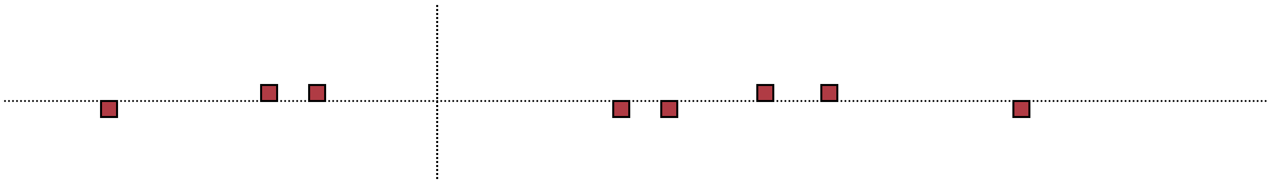


- Sequential bit probes

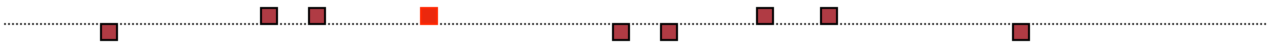


- Sequential bit probes



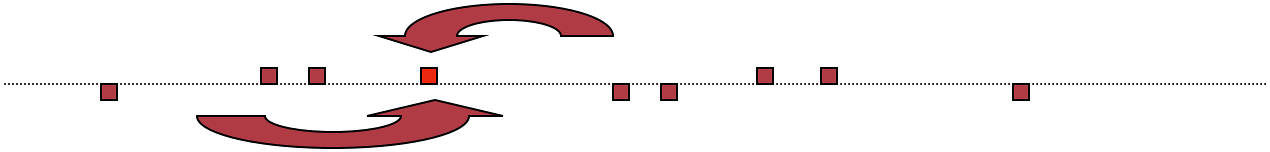


origin enclosure in 1-d



origin enclosure in 1-d

1-d origin enclosure: given n numbers $p^{(1)}, p^{(2)}, \ldots, p^{(n)}$, find a pair $p^{(i)}, p^{(j)}$ that bracket a given number a. (i.e. show $p^{(i)} < a < p^{(j)}$, for some i, j.)



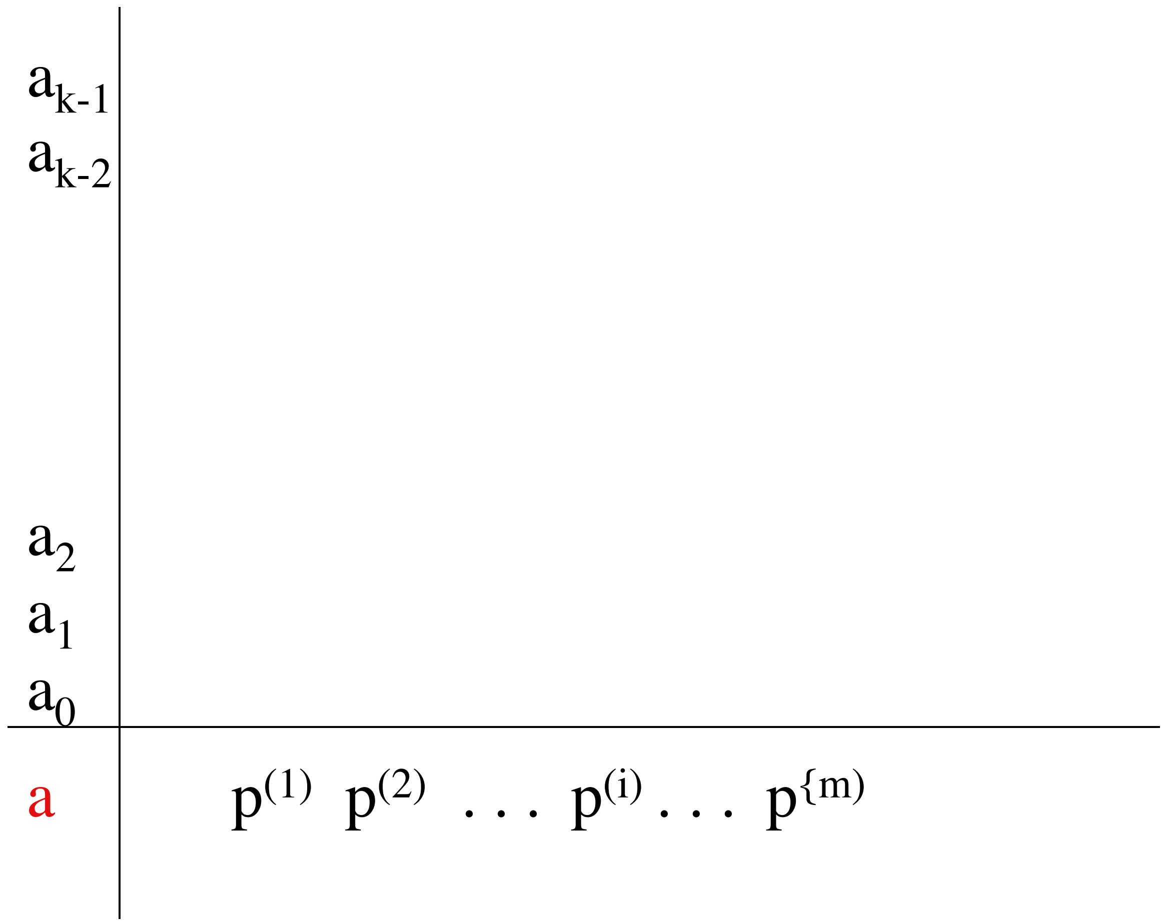
- Introduction and motivation

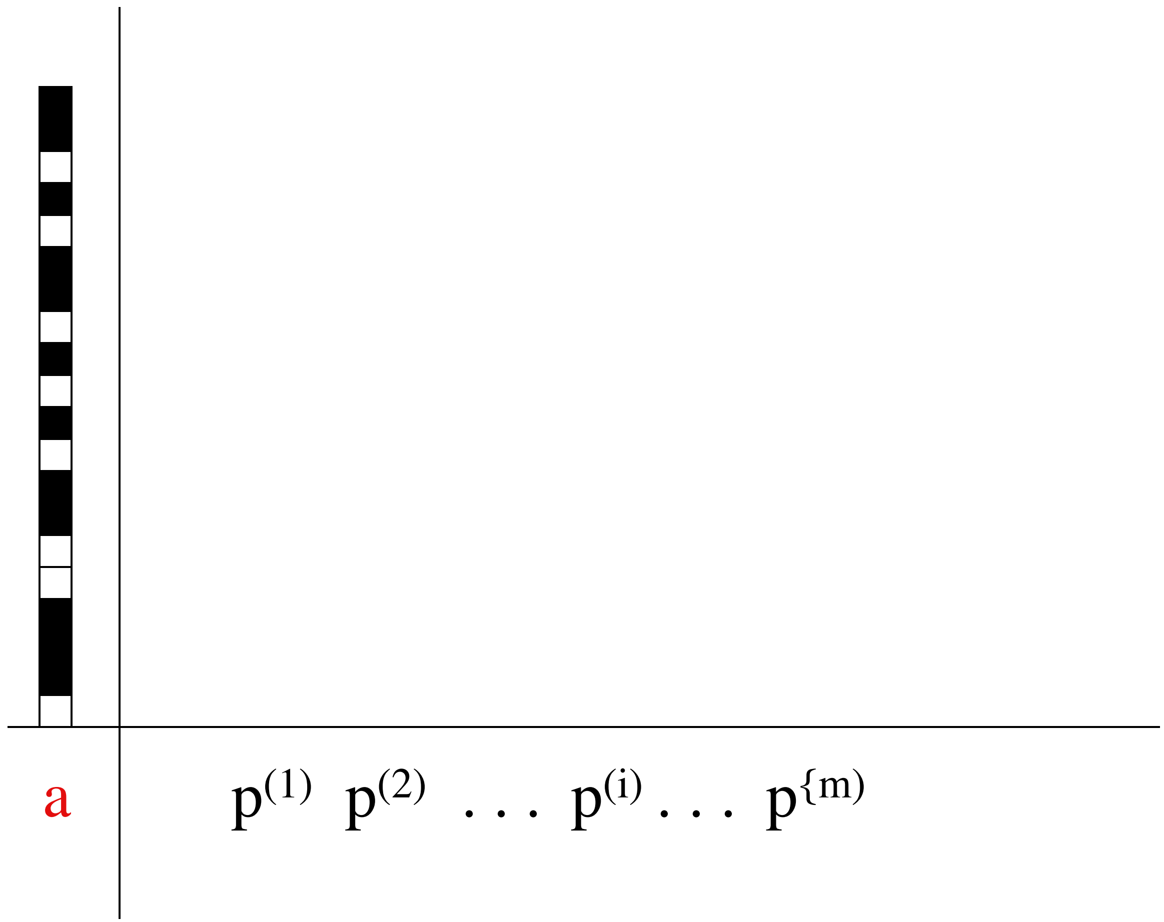
  Input-thrifty algorithms

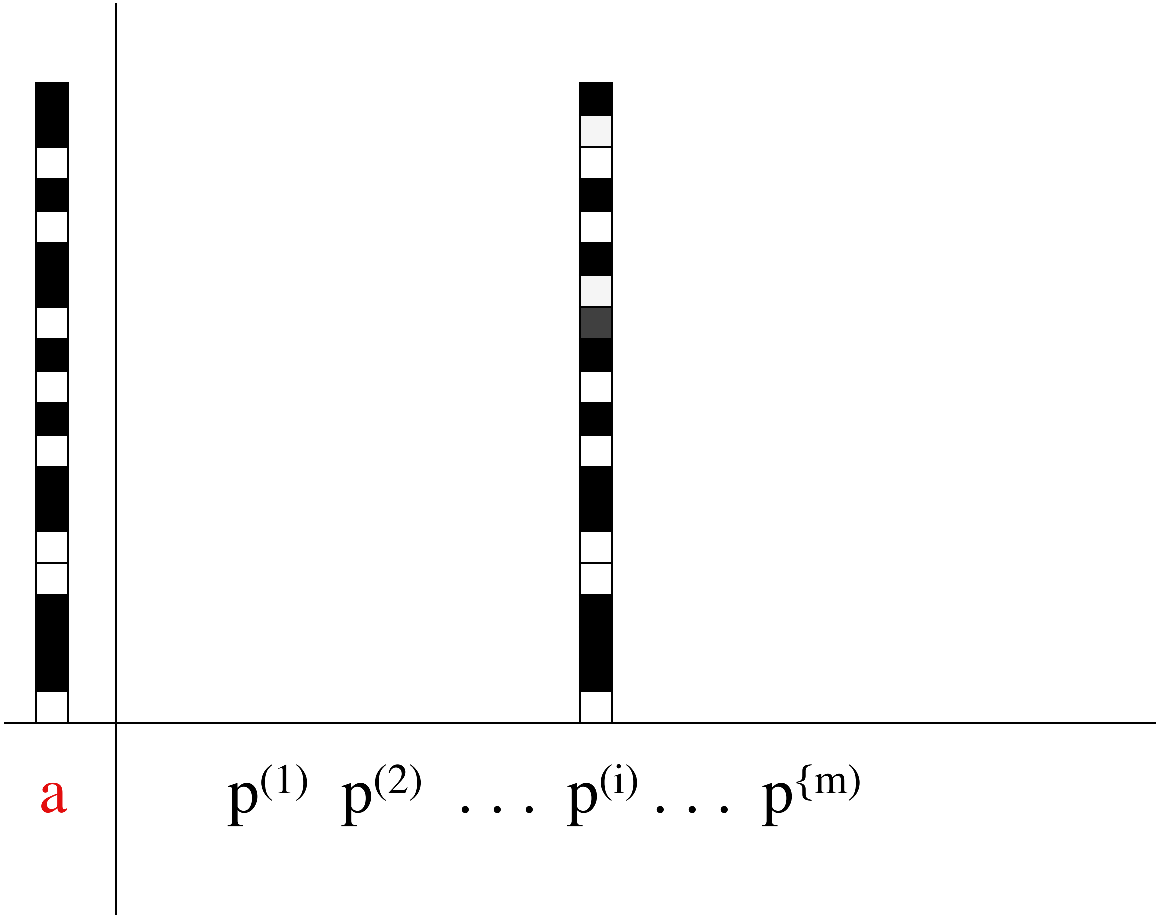
- List search
- Hyperbolic dovetailing
- Applications to input-thrifty algorithms
- Extensions & generalizations

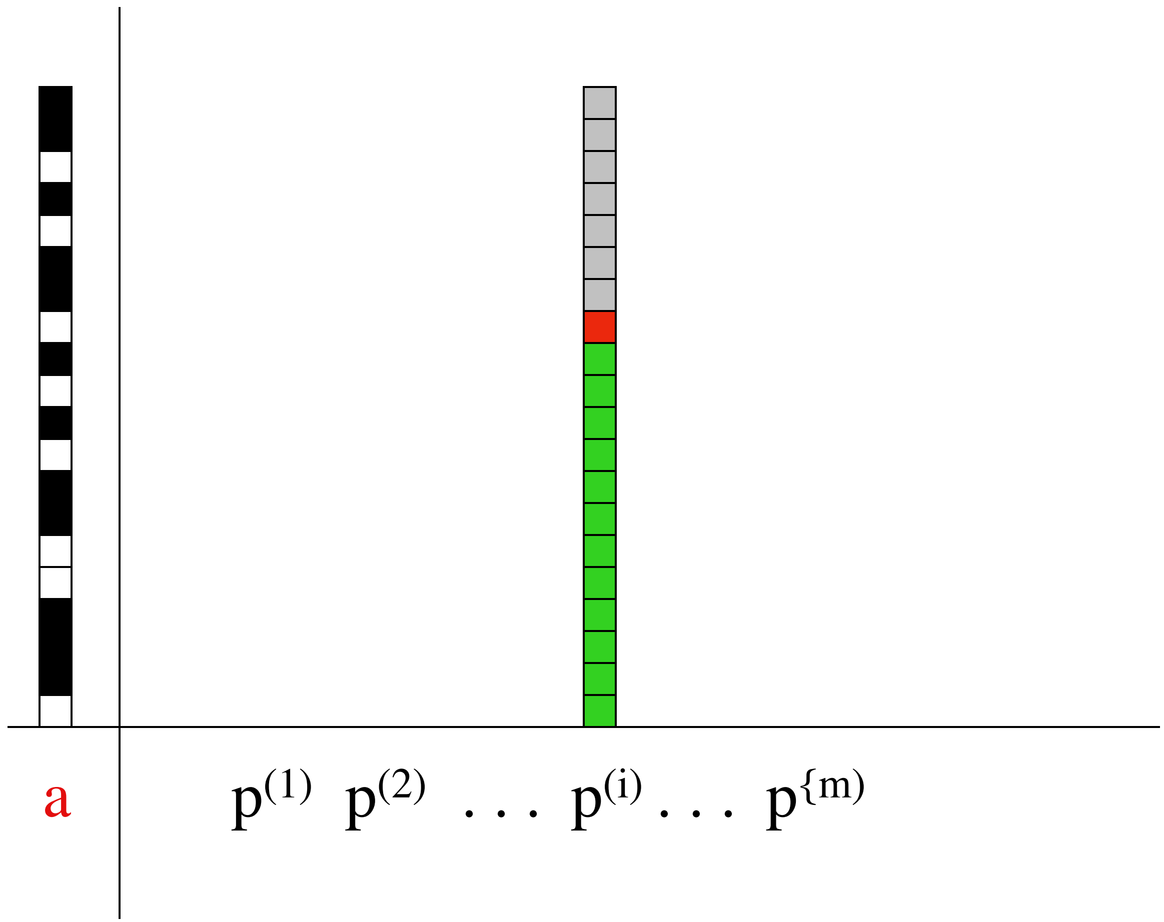
Overview

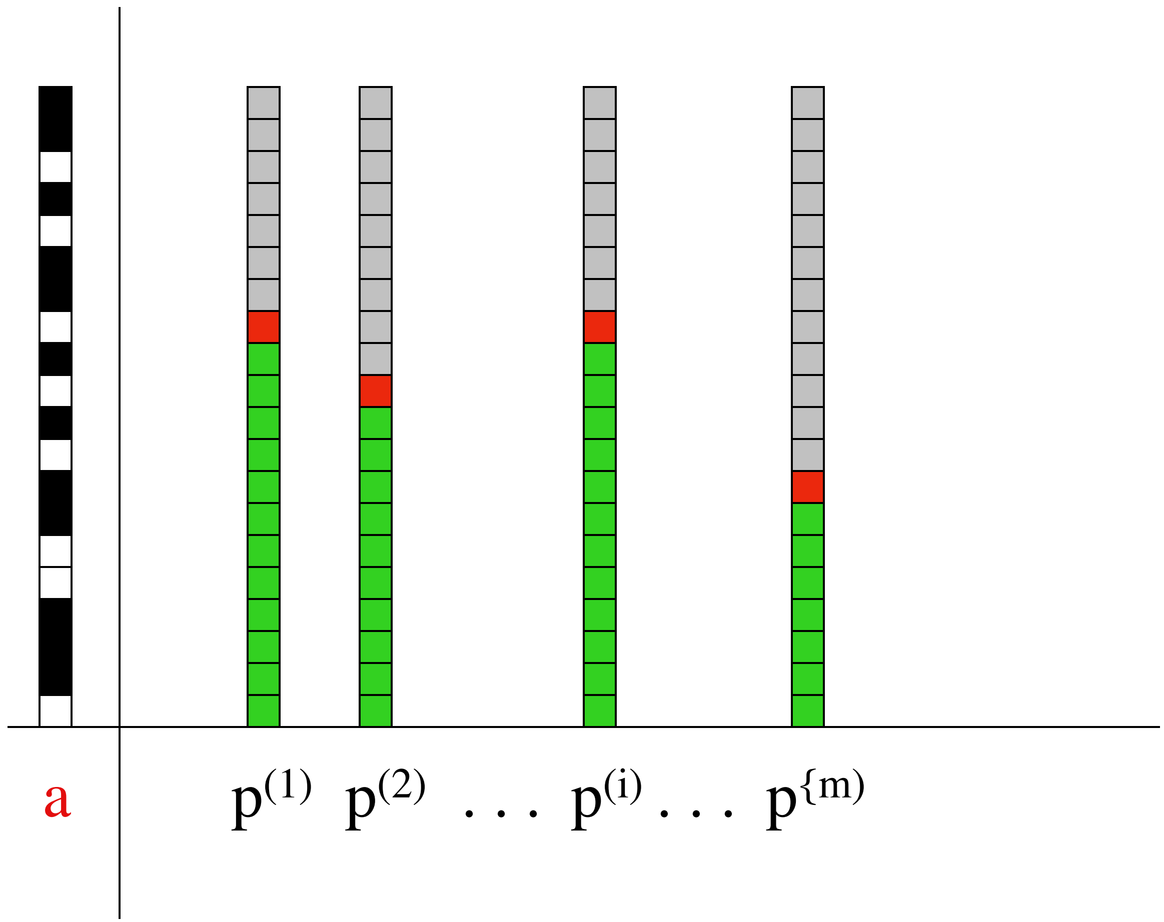
A. Given m numbers $p^{(1)}, p^{(2)}, \ldots, p^{(m)}$, identify at least one that *differs* from a given number a. (i.e. show $p^{(i)} > a$ or $p^{(i)} < a$, for some i.)


a_{k-1}
a_{k-2}
a_2
a_1
a_0
a
$p^{(1)}$ $p^{(2)}$ . . . $p^{(i)}$ . . . $p^{\{m)}$

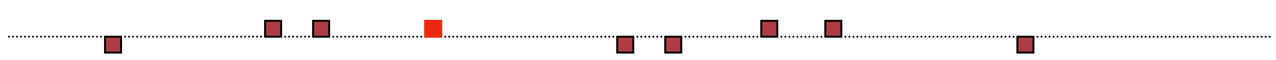

a
$p^{(1)}$ $p^{(2)}$ . . . $p^{(i)}$ . . . $p^{\{m)}$


a
$p^{(1)}$ $p^{(2)}$ . . . $p^{(i)}$ . . . $p^{\{m)}$

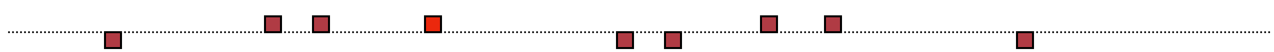

a
p(1) p(2) . . . p(i) . . . p{m)

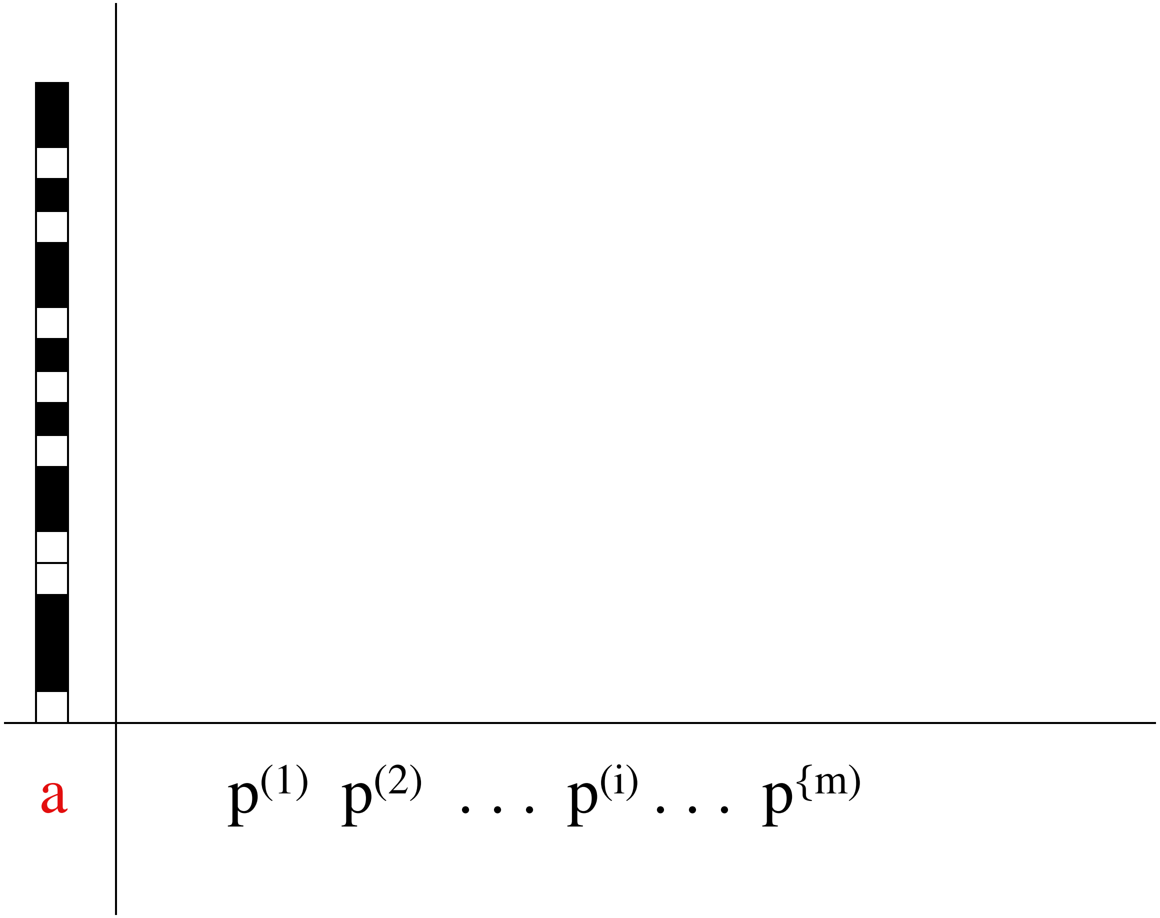

a
p(1) p(2) . . . p(i) . . . p{m)

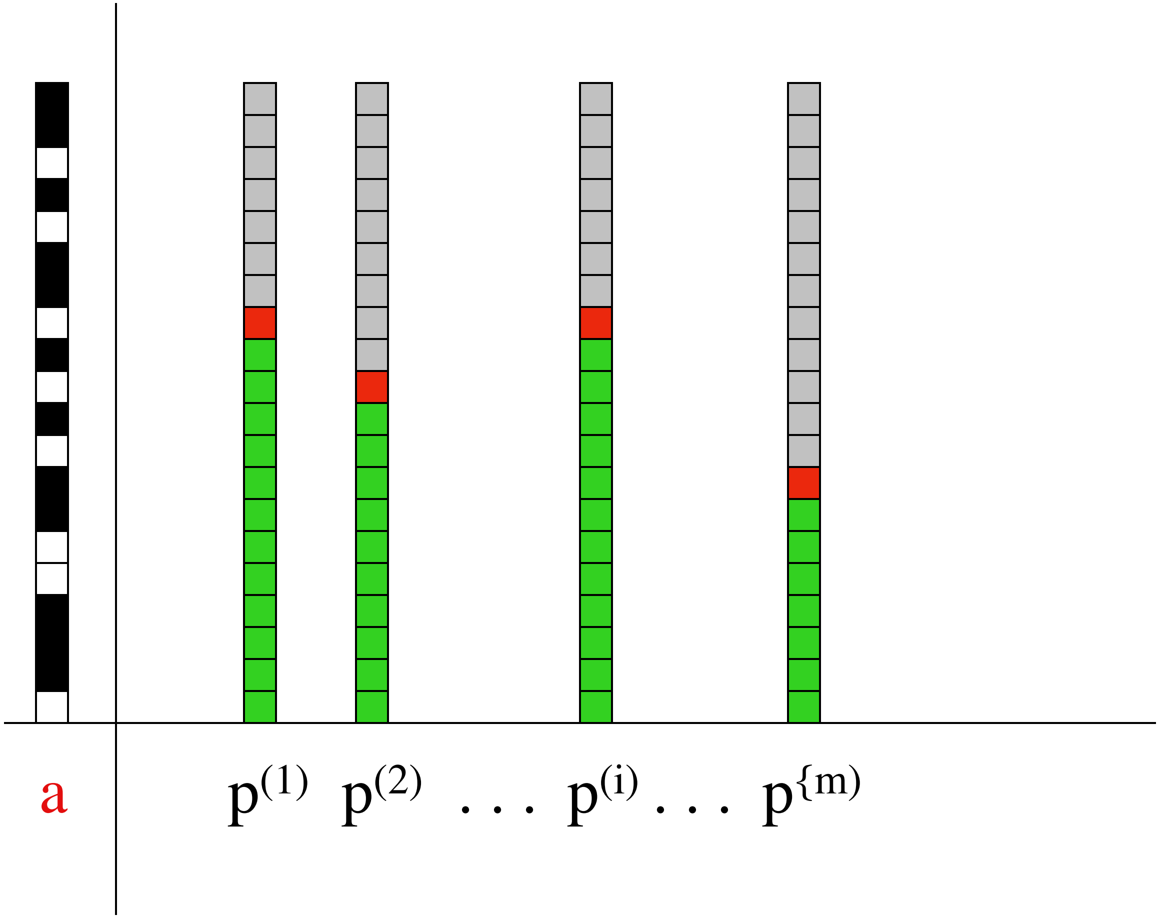
B. Origin enclosure: given n numbers $p^{(1)}, p^{(2)}, \ldots, p^{(m)}$, find a pair $p^{(i)}, p^{(j)}$ that bracket a given number a. (i.e. show $p^{(i)} < a < p^{(j)}$, for some i, j.)



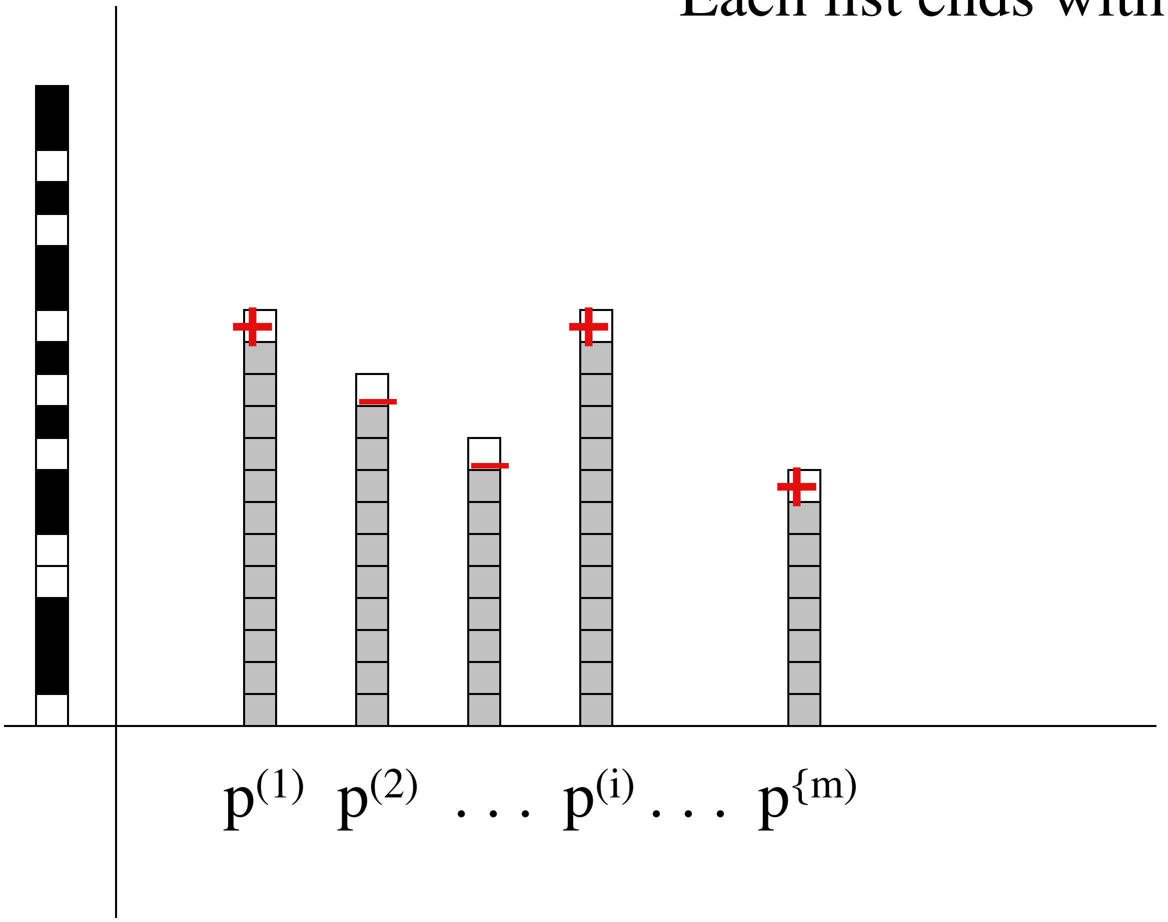
B. Origin enclosure: given n numbers $p^{(1)}, p^{(2)}, \ldots, p^{(m)}$, find a pair $p^{(i)}, p^{(j)}$ that bracket a given number a. (i.e. show $p^{(i)} < a < p^{(j)}$, for some i, j.)




a
$p^{(1)}$ $p^{(2)}$ . . . $p^{(i)}$ . . . $p^{\{m)}$

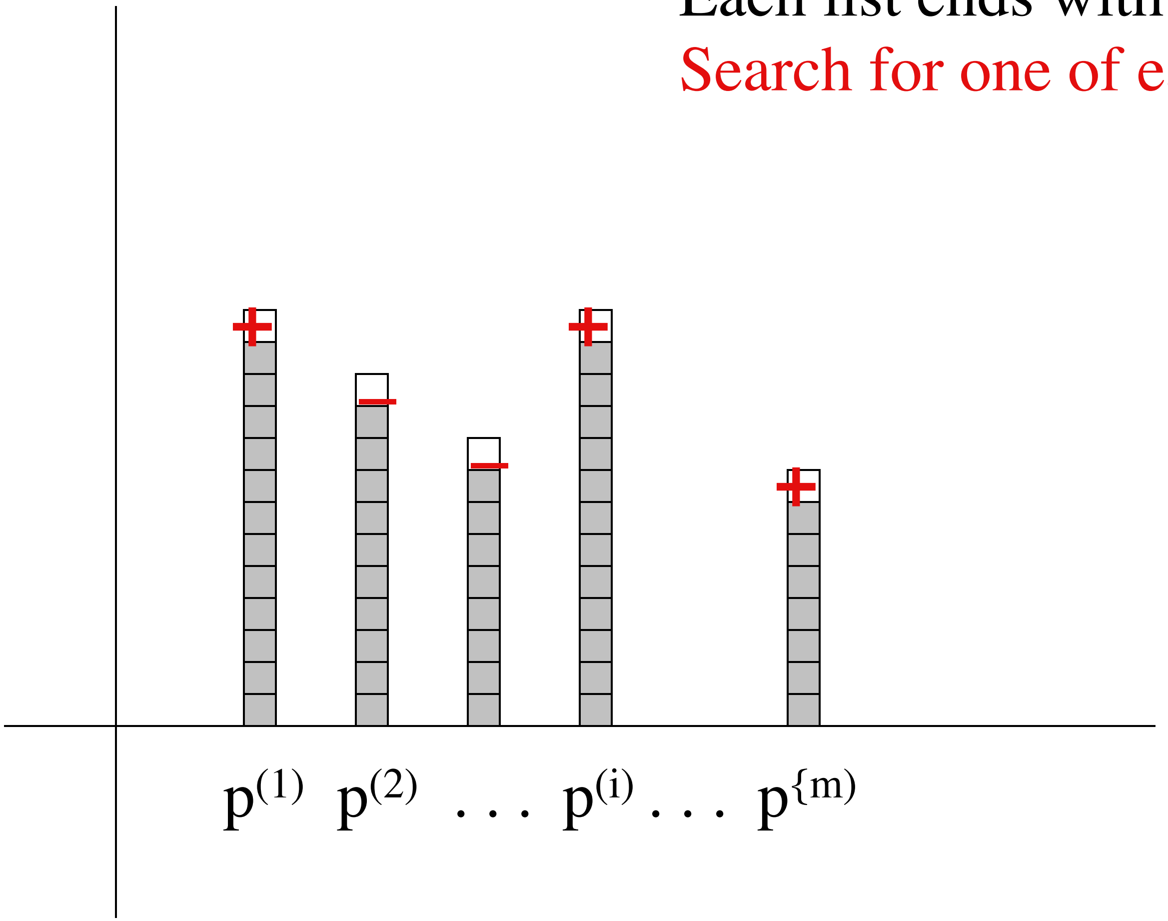

a
p(1) p(2) . . . p(i) . . . p{m)

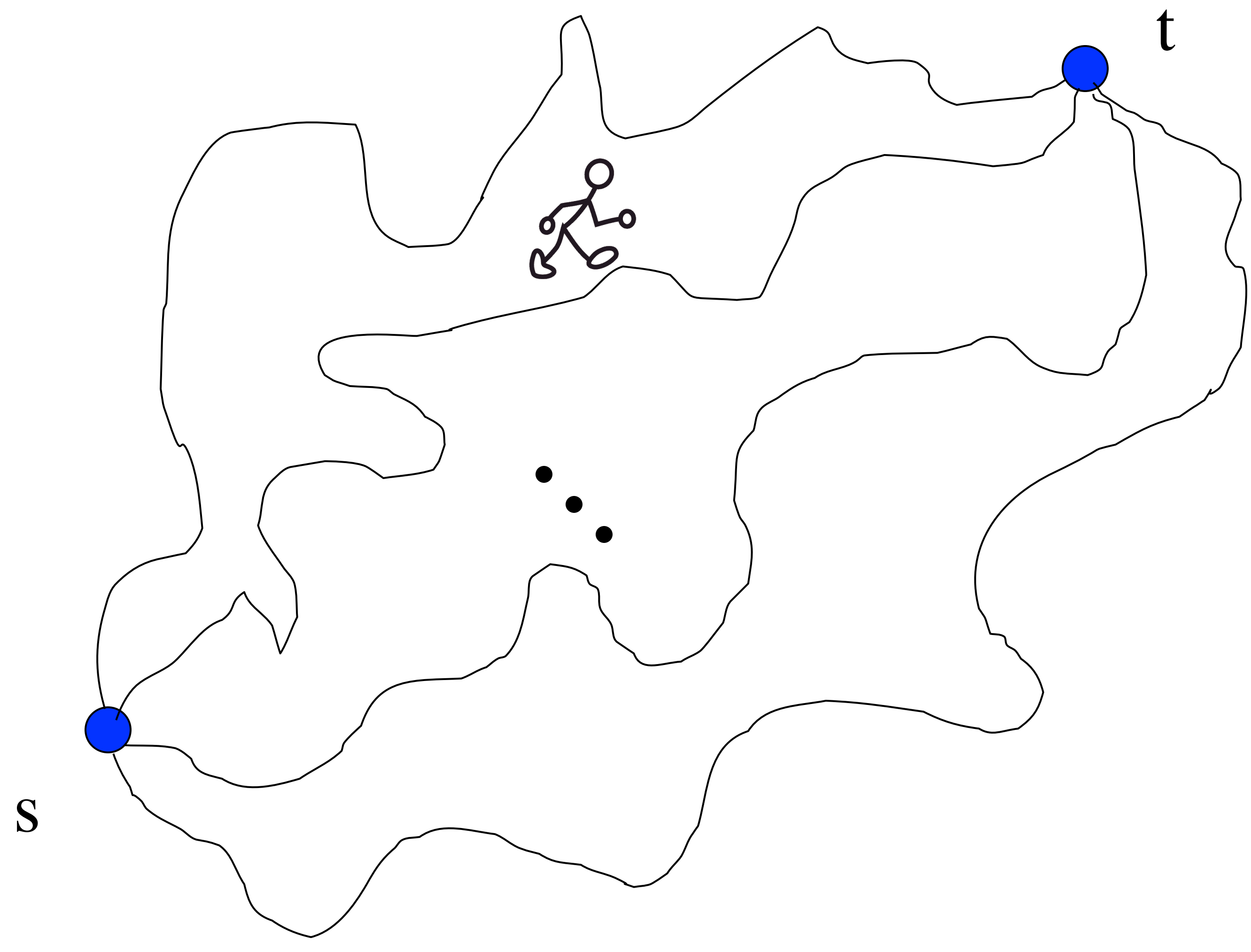
Each list ends with a sign



Each list ends with a sign

Search for one of each type




t
s

Objective:

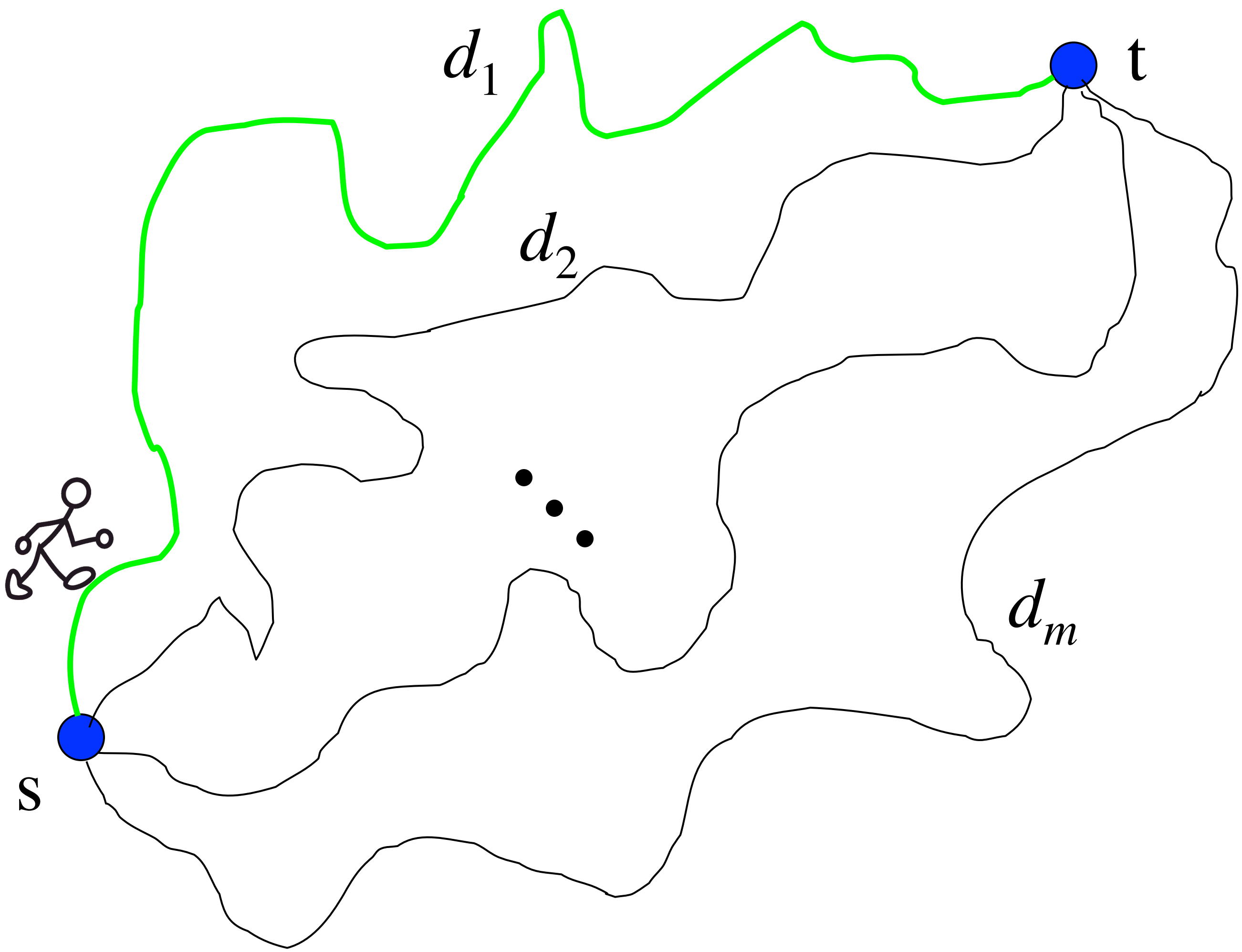
Walk from s to t as efficiently as possible.

Objective:

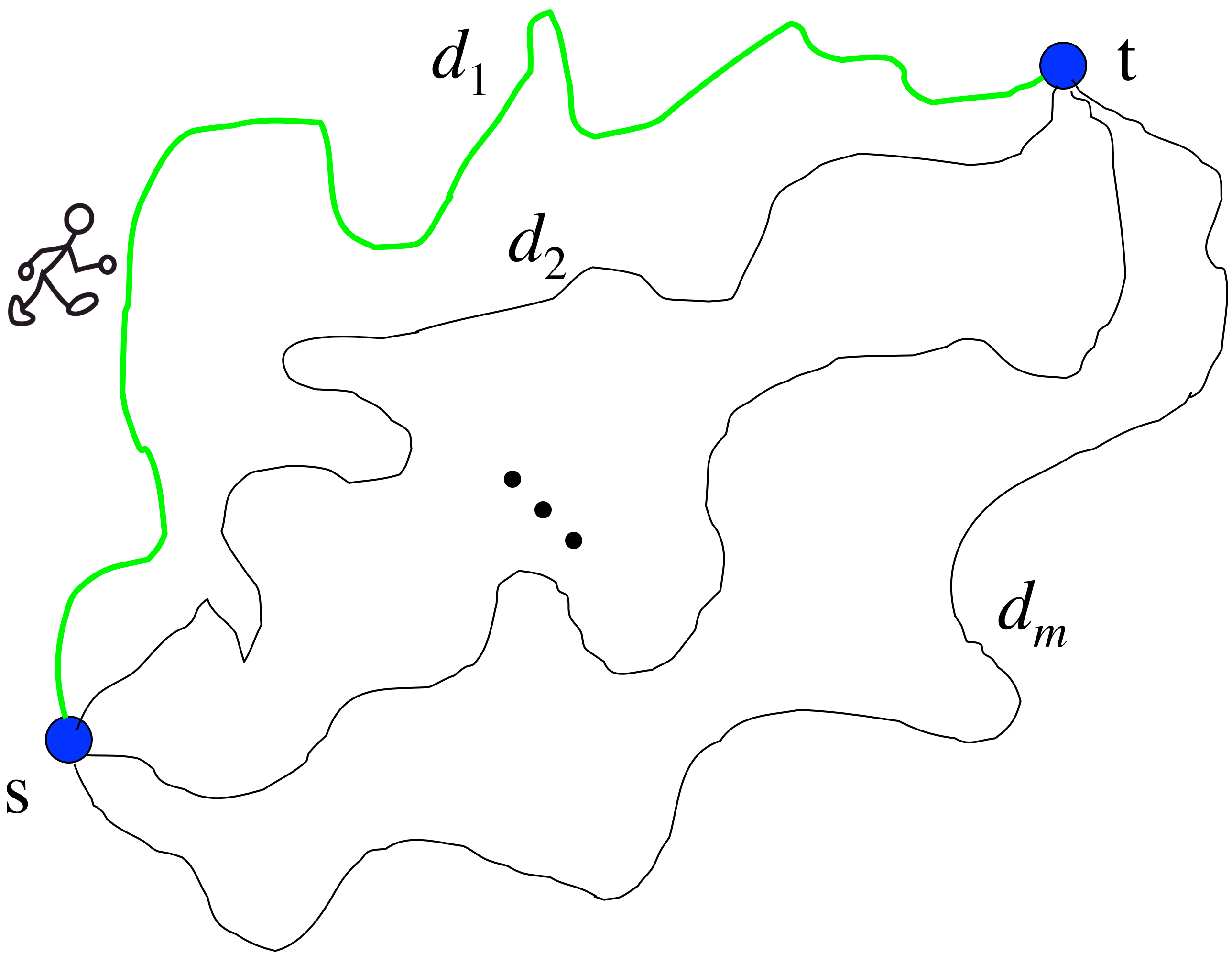
Walk from s to t as efficiently as possible.

Problem:

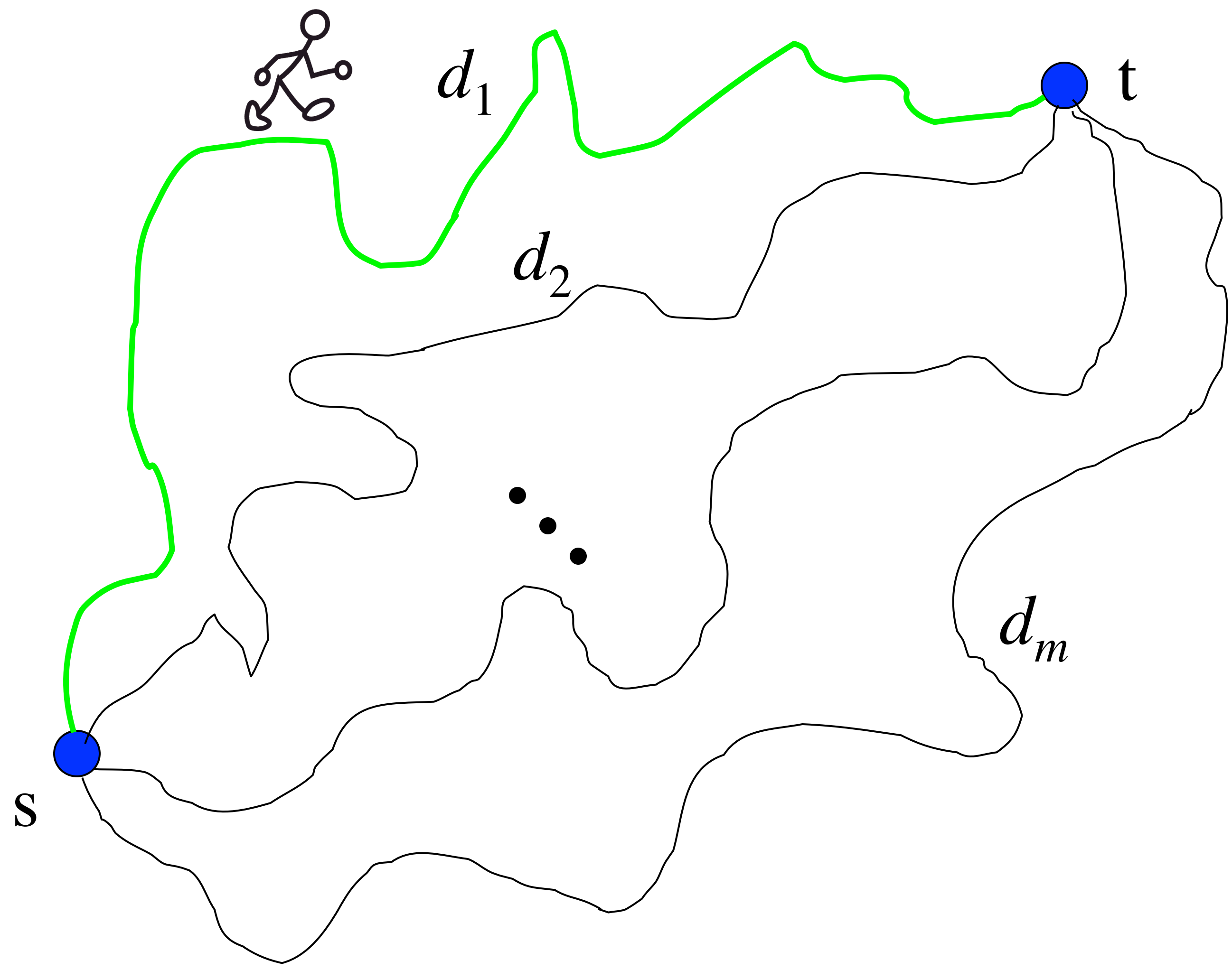
The individual path lengths $d_1, d_2, \ldots, d_m$ are not known!



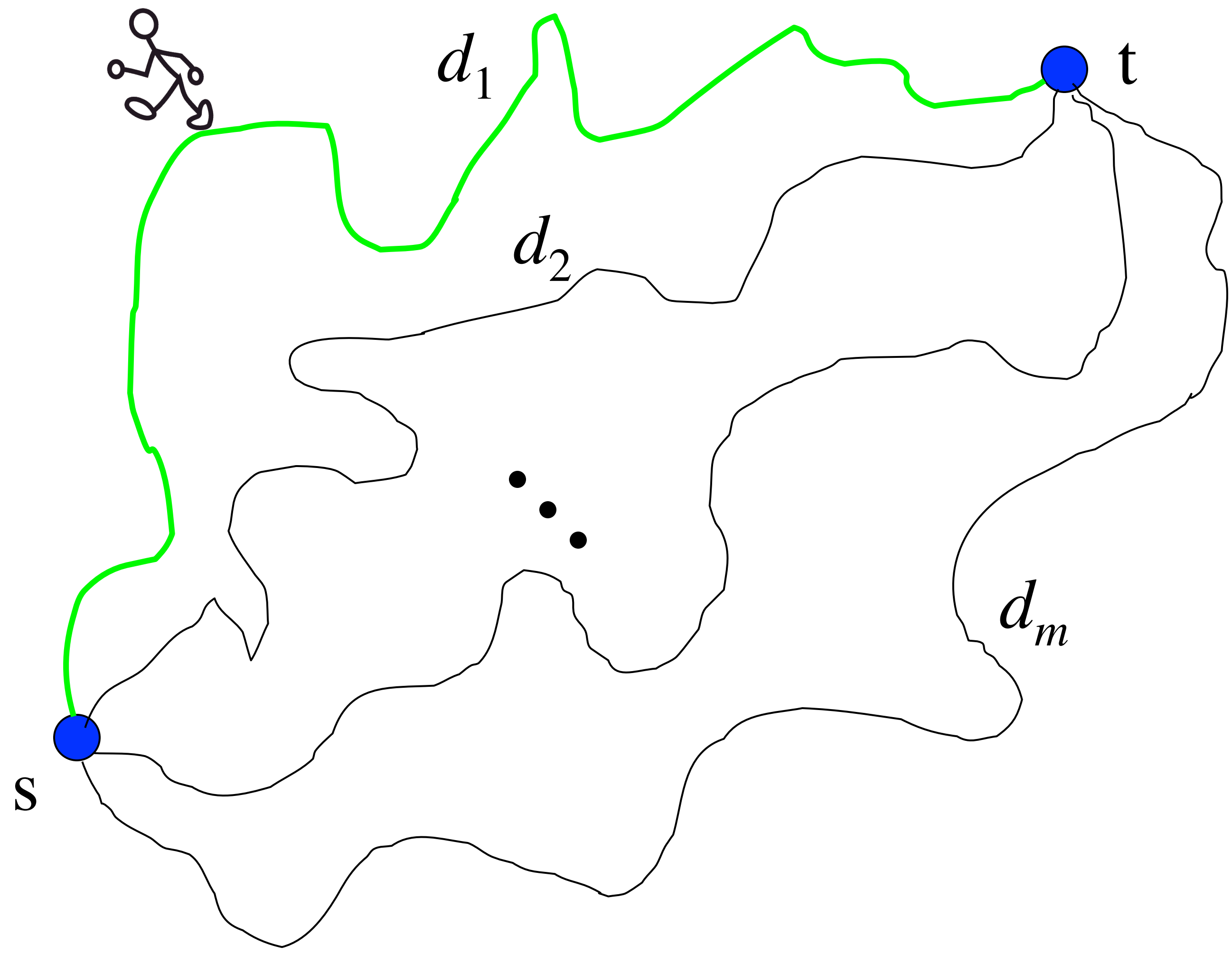
d_1
t
d_2
d_m
s

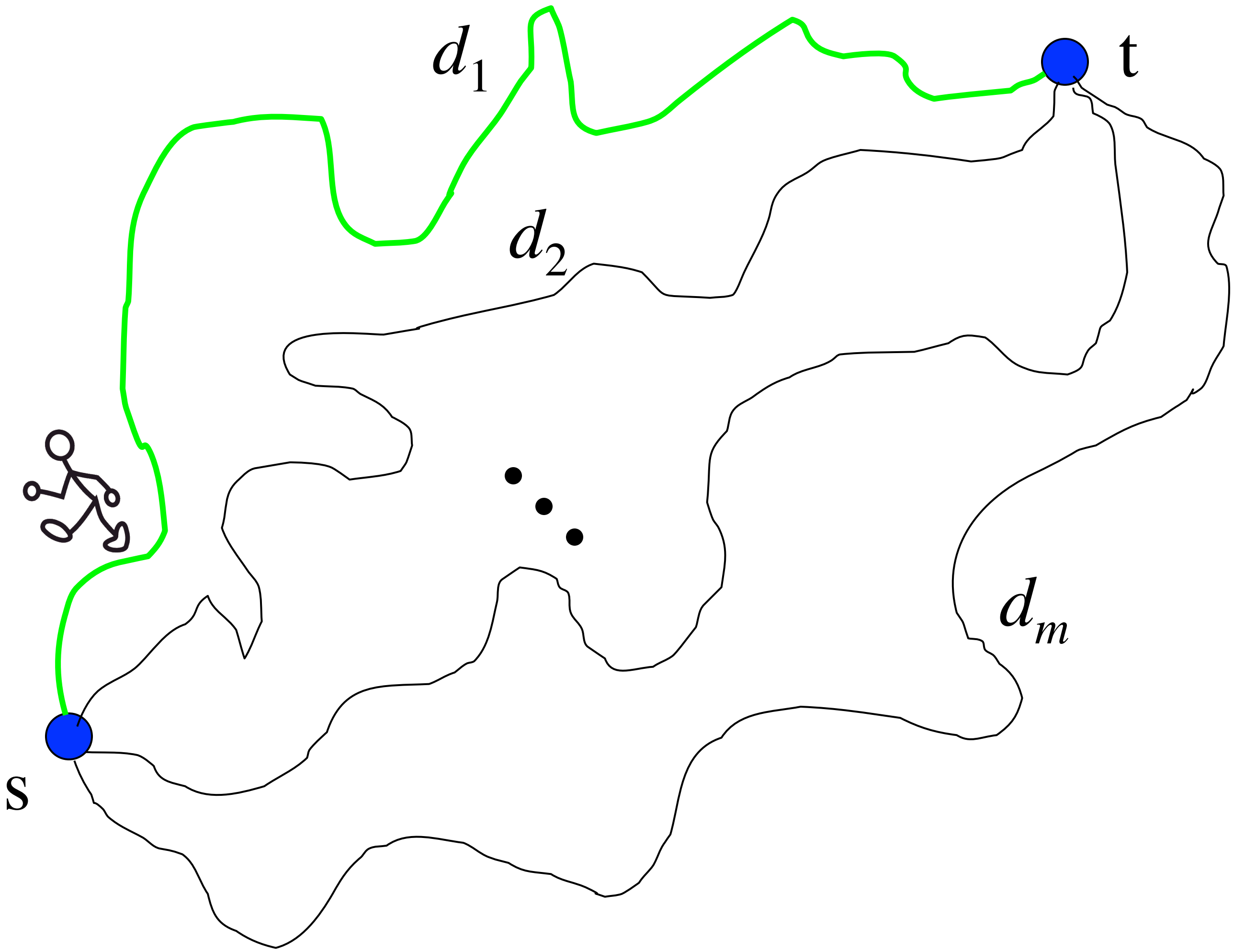


d_1
t
d_2
d_m
s

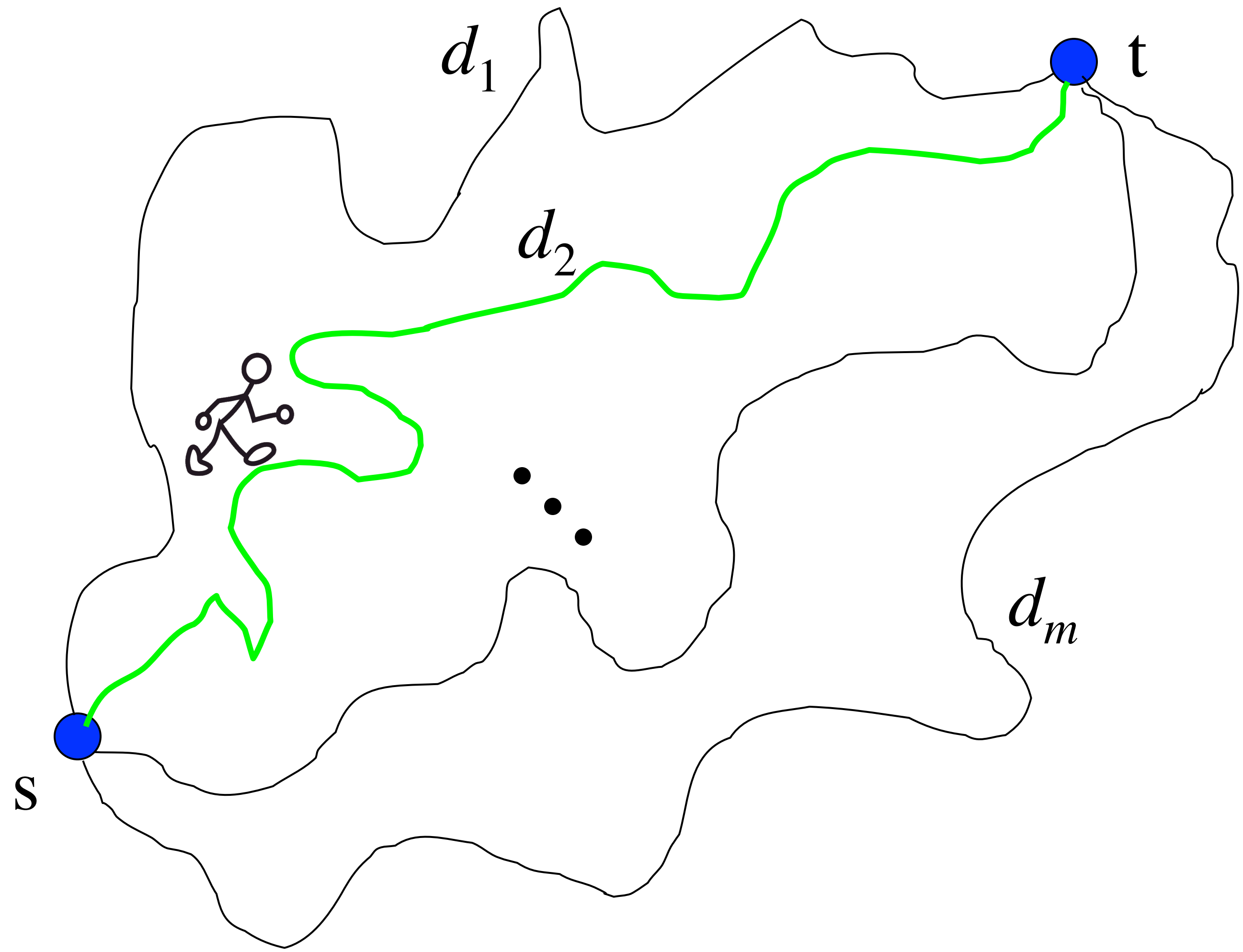


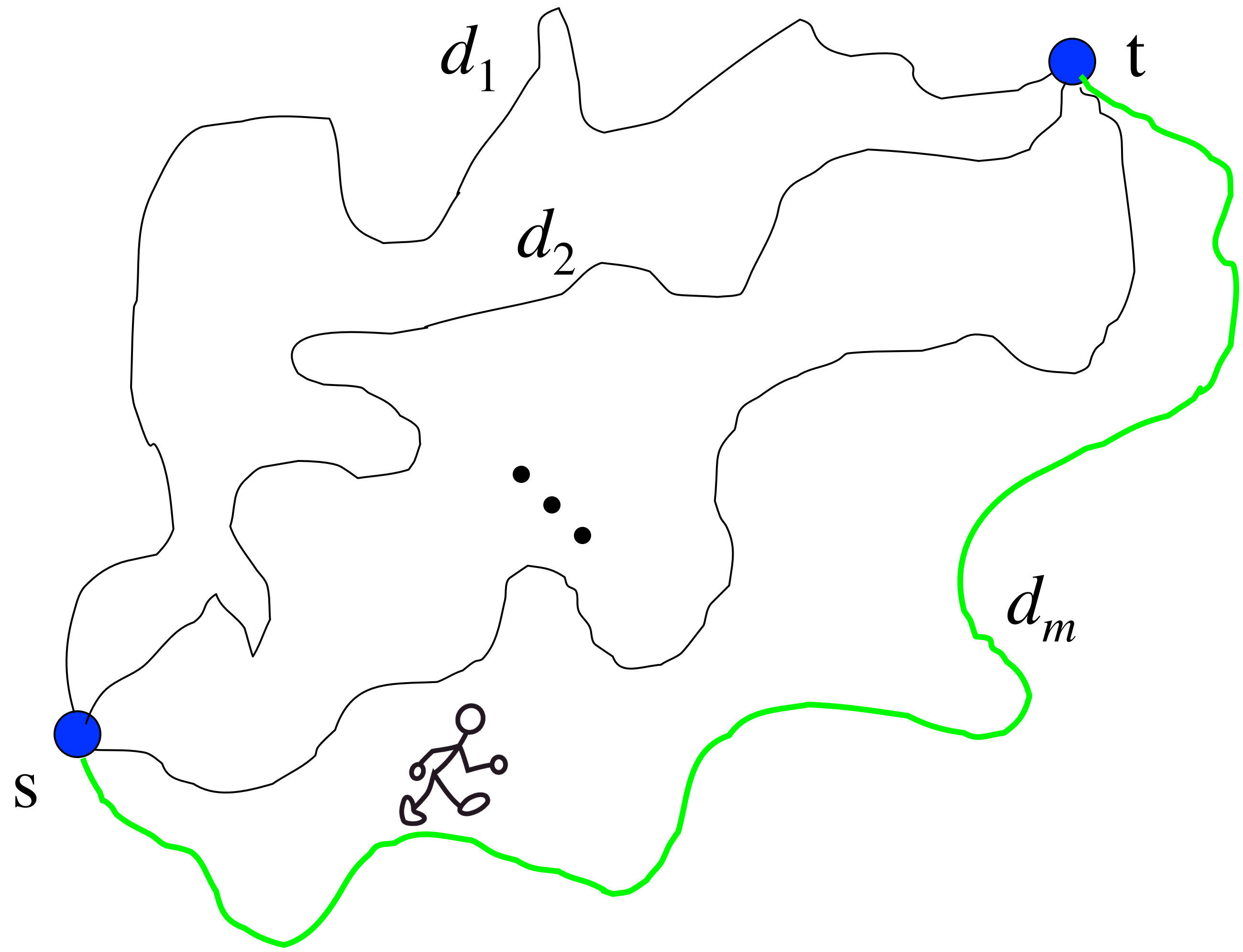
d_1
t
d_2
d_m
s


d_1
d_2
d_m
s
t



d_1
t
d_2
d_m
s


d_1
t
d_2
d_m
s



d_1
t
d_2
d_m
s

How do we decide …

* when to turn around?

How do we decide …

* when to turn around?

* which path to explore next?

How do we evaluate a strategy?

* worst case…

How do we evaluate a strategy?

* worst case…
  all strategies are horrible!

How do we evaluate a strategy?

* worst case…
  all strategies are horrible!

* competitive analysis
  behaviour should reflect *intrinsic complexity* of input

Why should I (you) be interested?

* search games [Gal '80]

* geometric search in unknown environments
  [Papadimitriou et al. '89, Fleischer et al. '04]

* randomized/heuristic algorithm design
  [Luby et al. '93, Kao et al. '98]

* playing slot machines...conducting research...life

Why should I (you) be interested?

* search games [Gal '80]

* geometric search in unknown environments
  [Papadimitriou et al. '89, Fleischer et al. '04]

* randomized/heuristic algorithm design
  [Luby et al. '93, Kao et al. '98]

* playing slot machines...conducting research...life

Why should I (you) be interested?

* search games [Gal ’80]

* geometric search in unknown environments
  [Papadimitriou et al. ‘89, Fleischer et al. ’04]

* randomized/heuristic algorithm design
  [Luby et al. ‘93, Kao et al. ’98]

* playing slot machines...conducting research...life

Why should I (you) be interested?

* search games [Gal ’80]

* geometric search in unknown environments
  [Papadimitriou et al. ‘89, Fleischer et al. ’04]

* randomized/heuristic algorithm design
  [Luby et al. ‘93, Kao et al. ’98]

* playing slot machines...conducting research...life

Why should I (you) be interested?

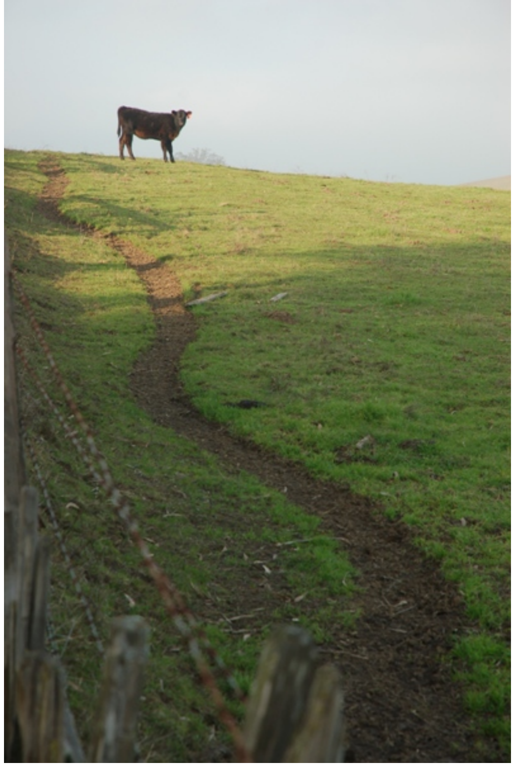
* search games [Gal '80]

* geometric search in unknown environments
  [Papadimitriou et al. '89, Fleischer et al. '04]

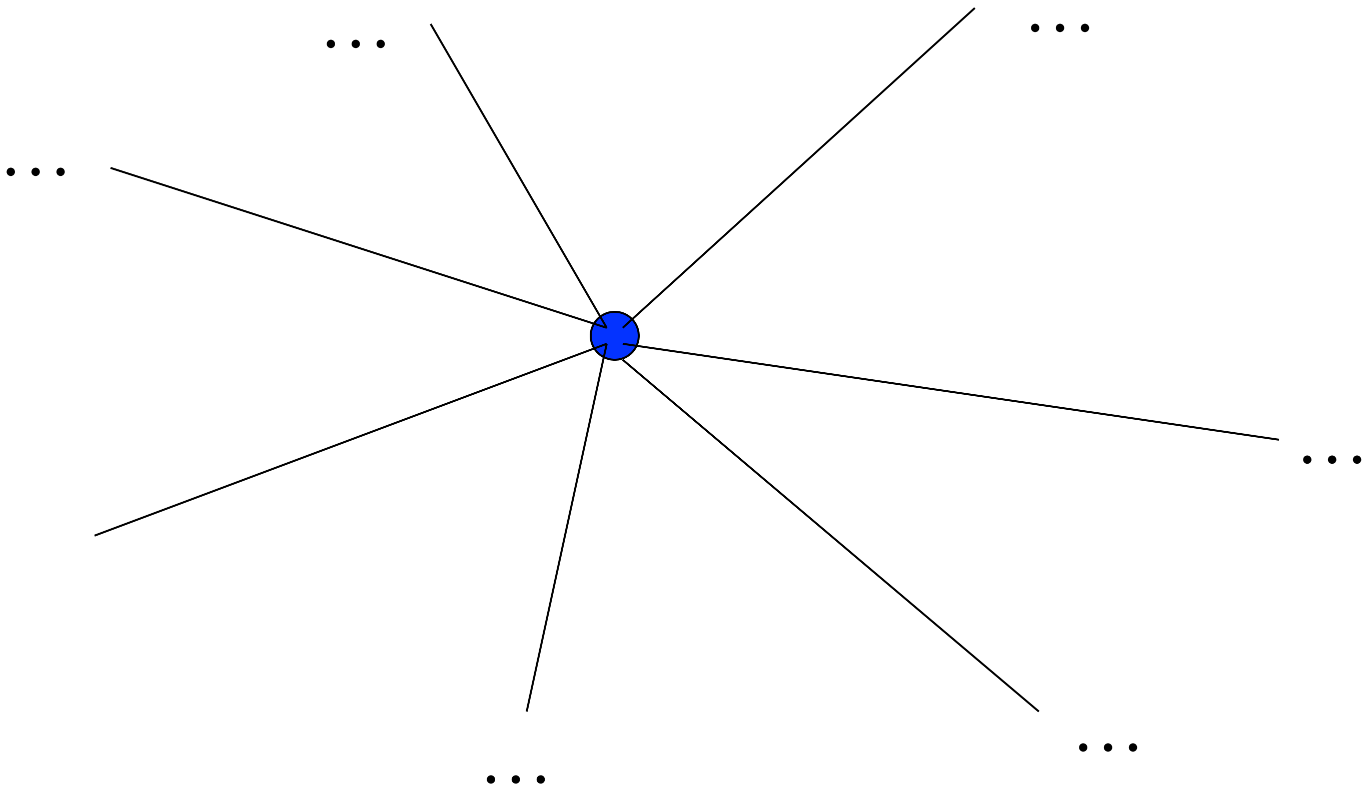
* randomized/heuristic algorithm design
  [Luby et al. '93, Kao et al. '98]

* playing slot machines...conducting research...life

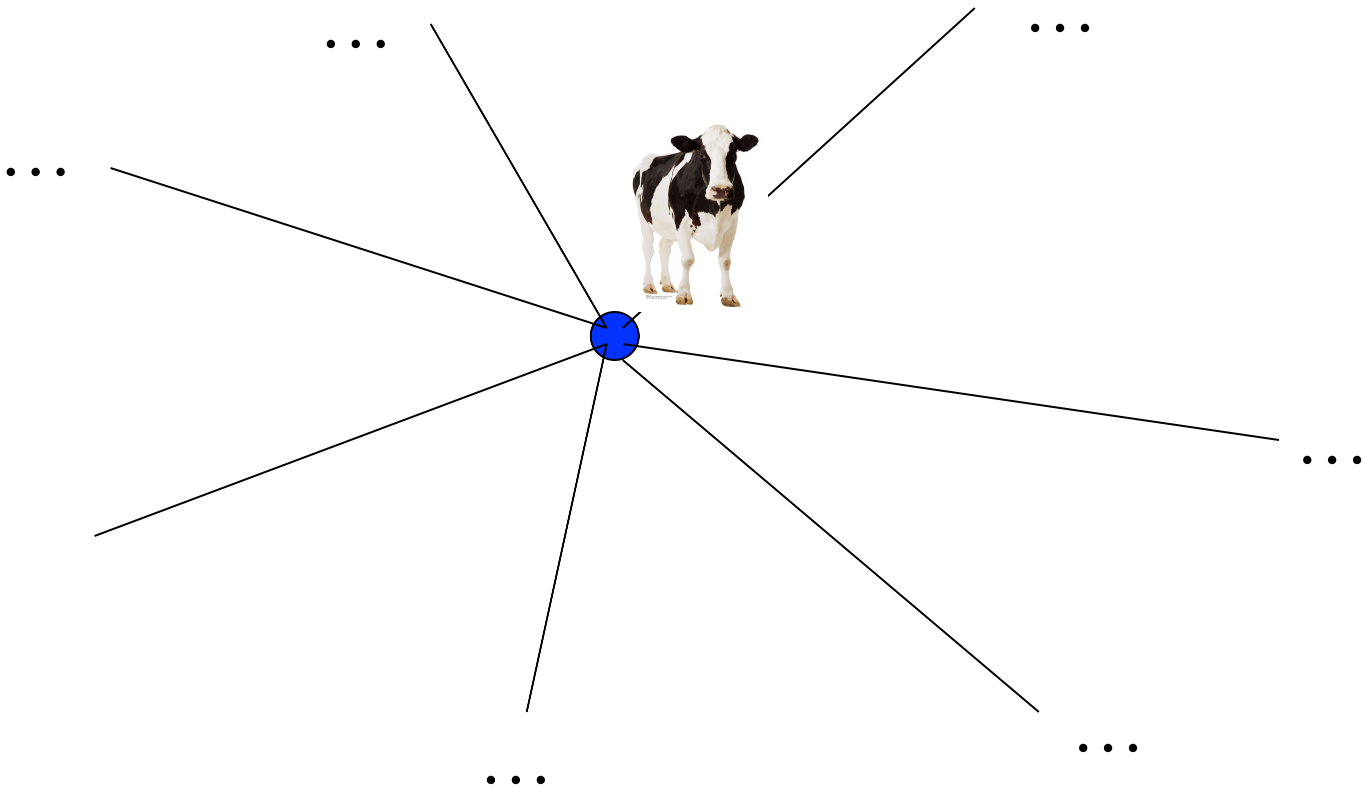
This seems vaguely familiar…



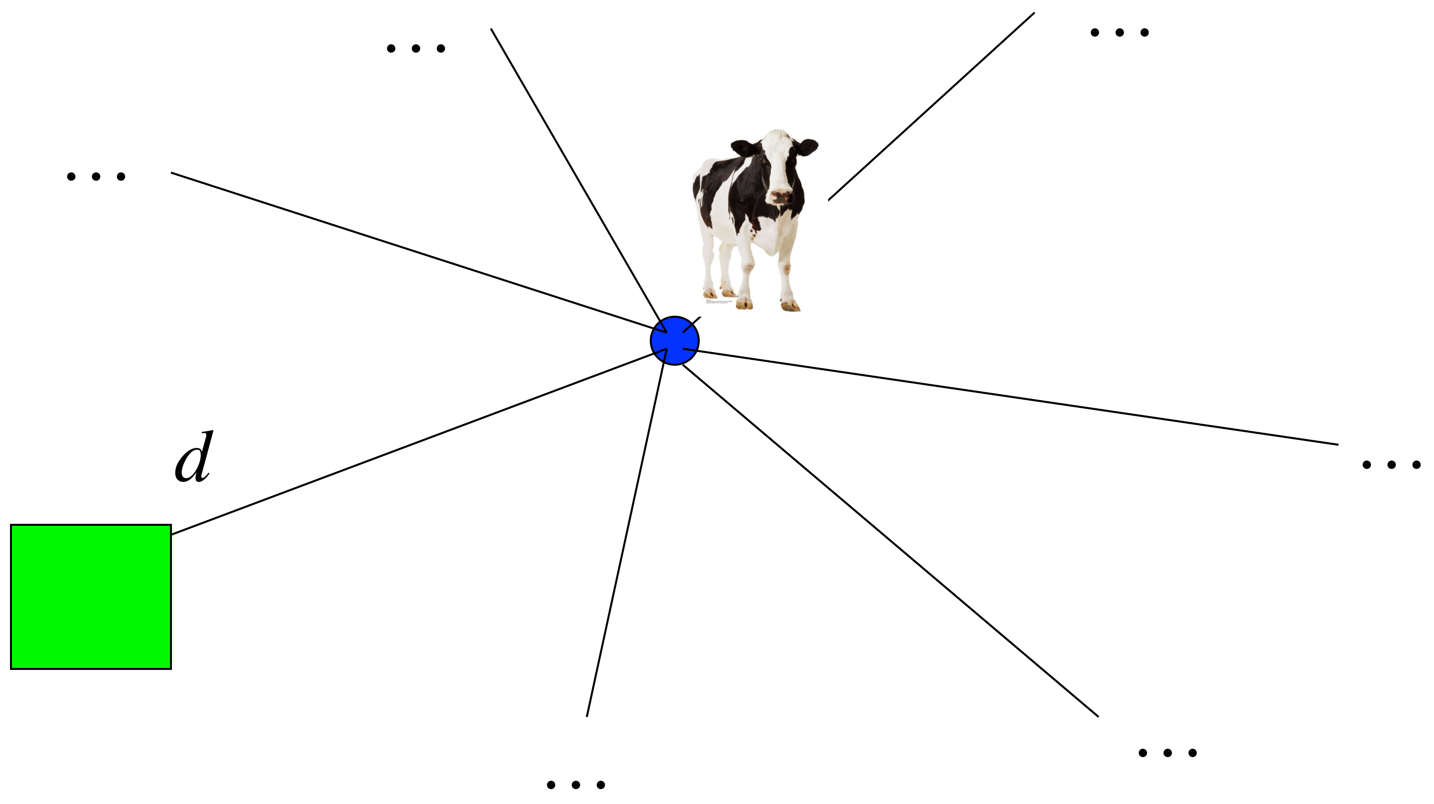
m lanes



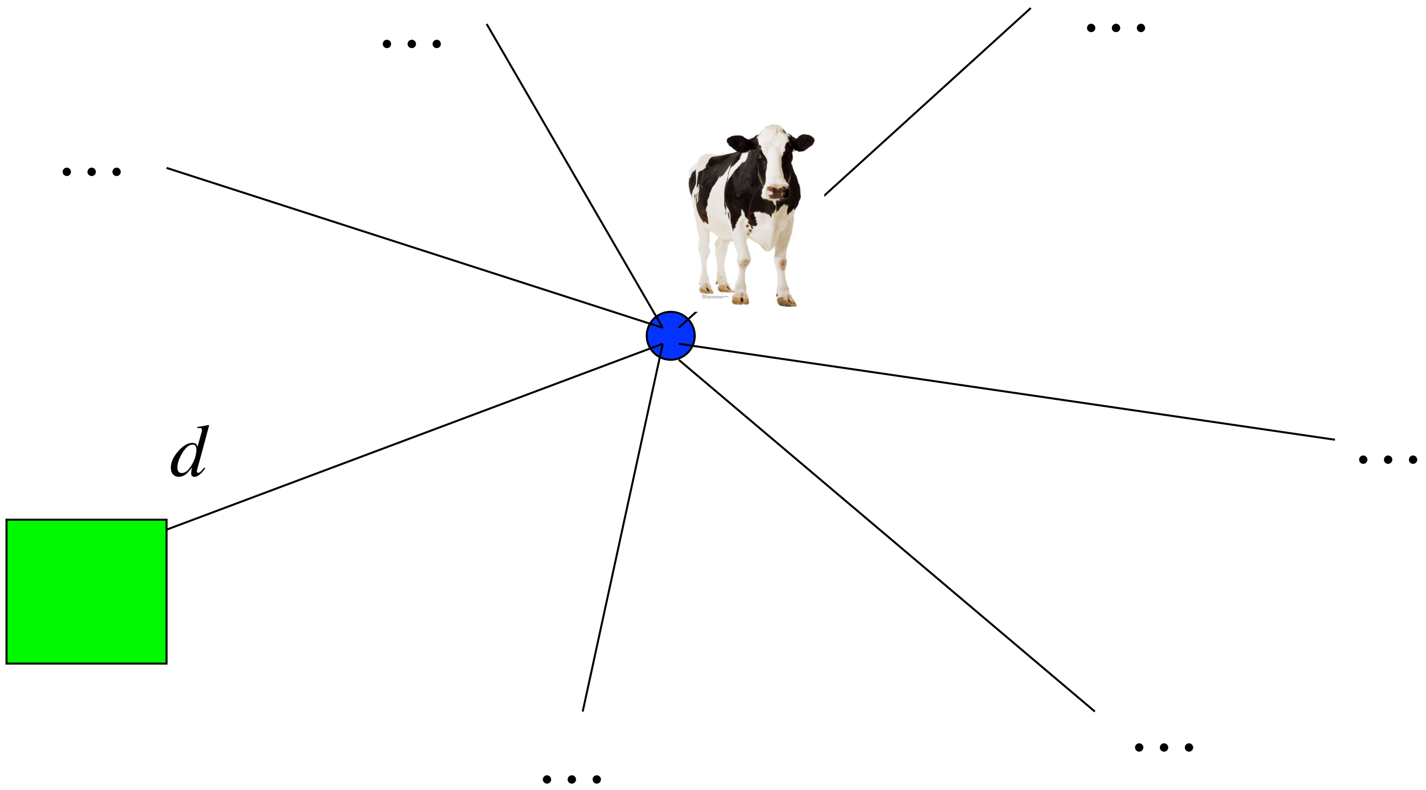
m lanes, a cow



m lanes, a cow and a pasture

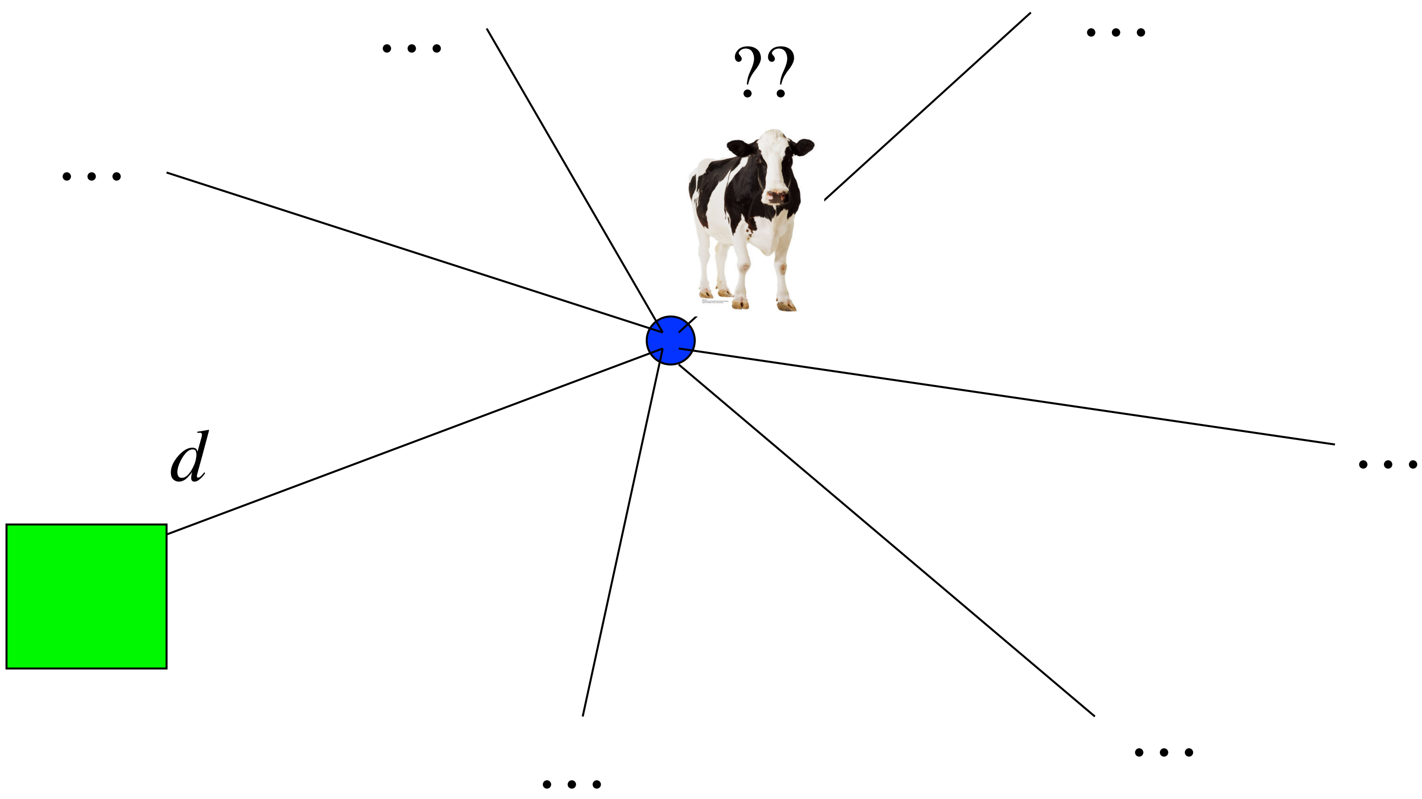


m lanes, a cow and a pasture



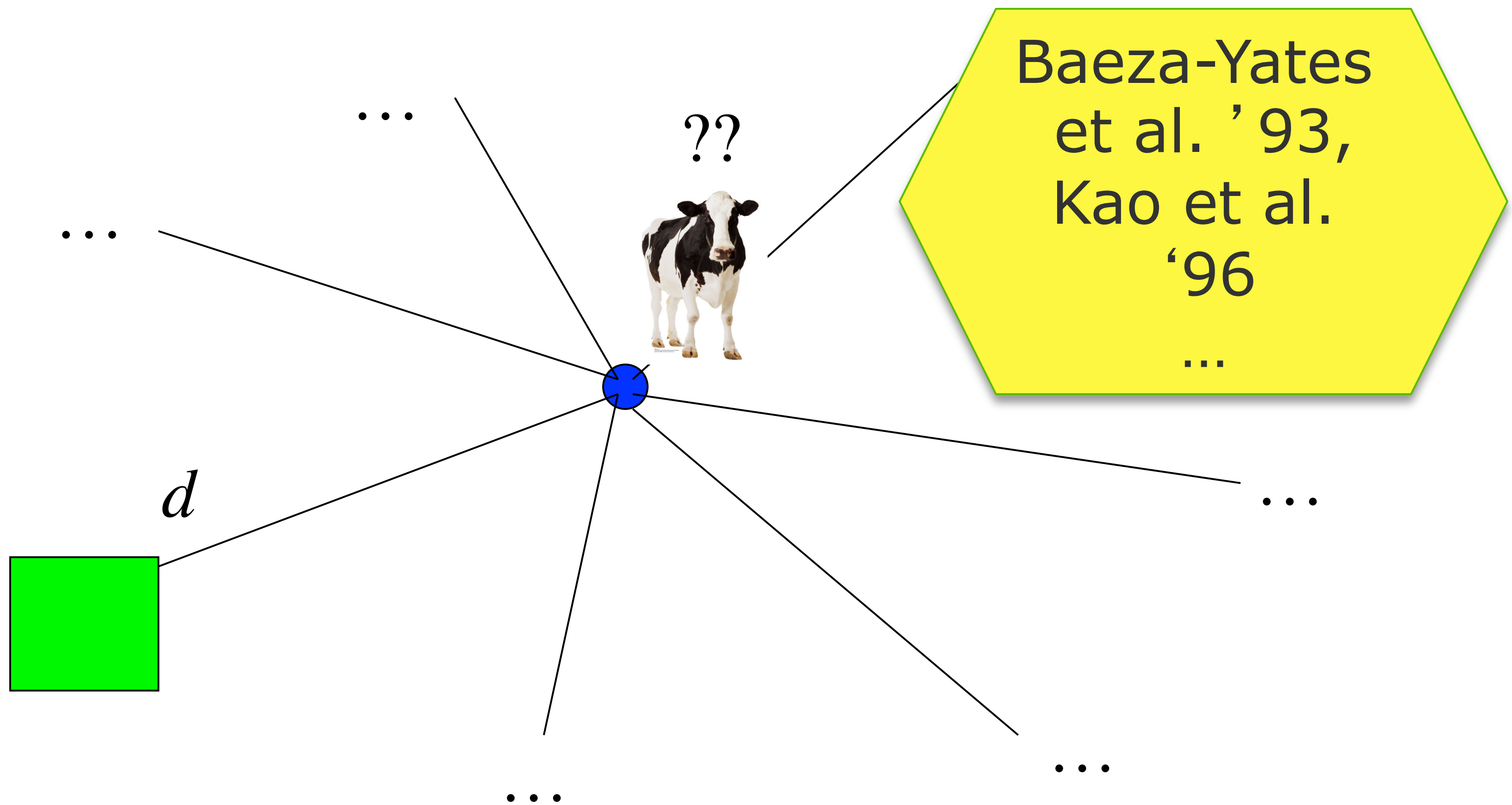
m-lane cow-paths problem

m lanes, a cow and a pasture

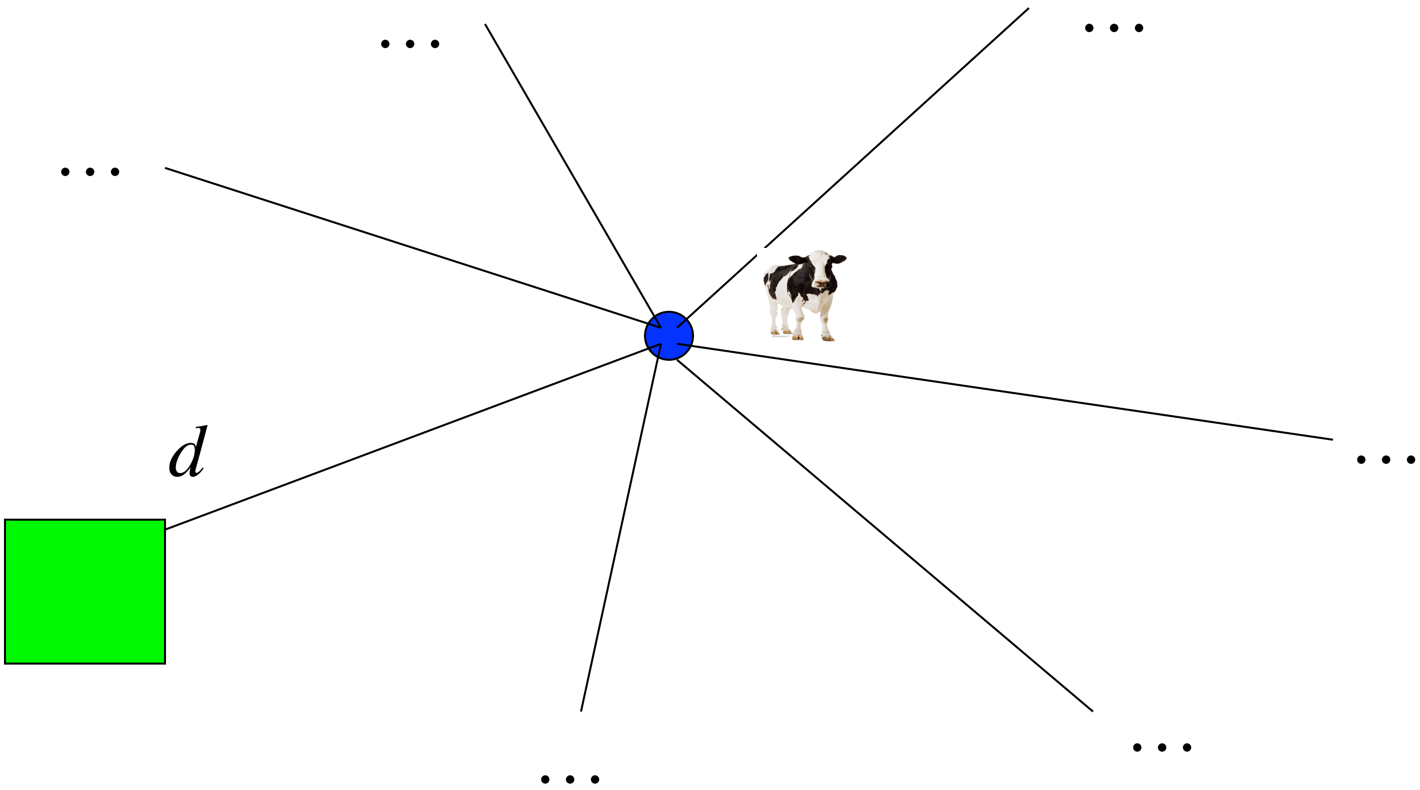


m-lane cow-paths problem

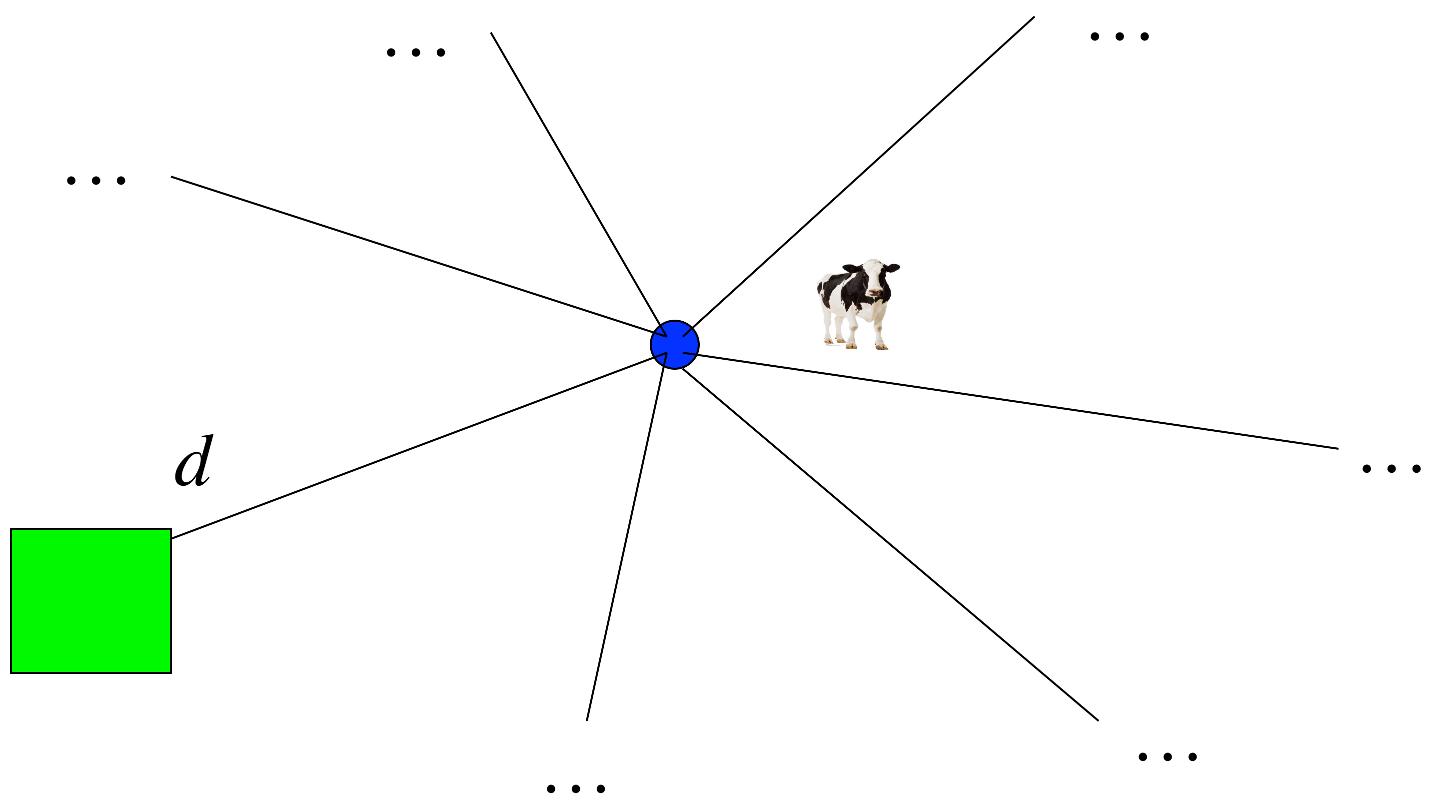
m lanes, a cow and a pasture



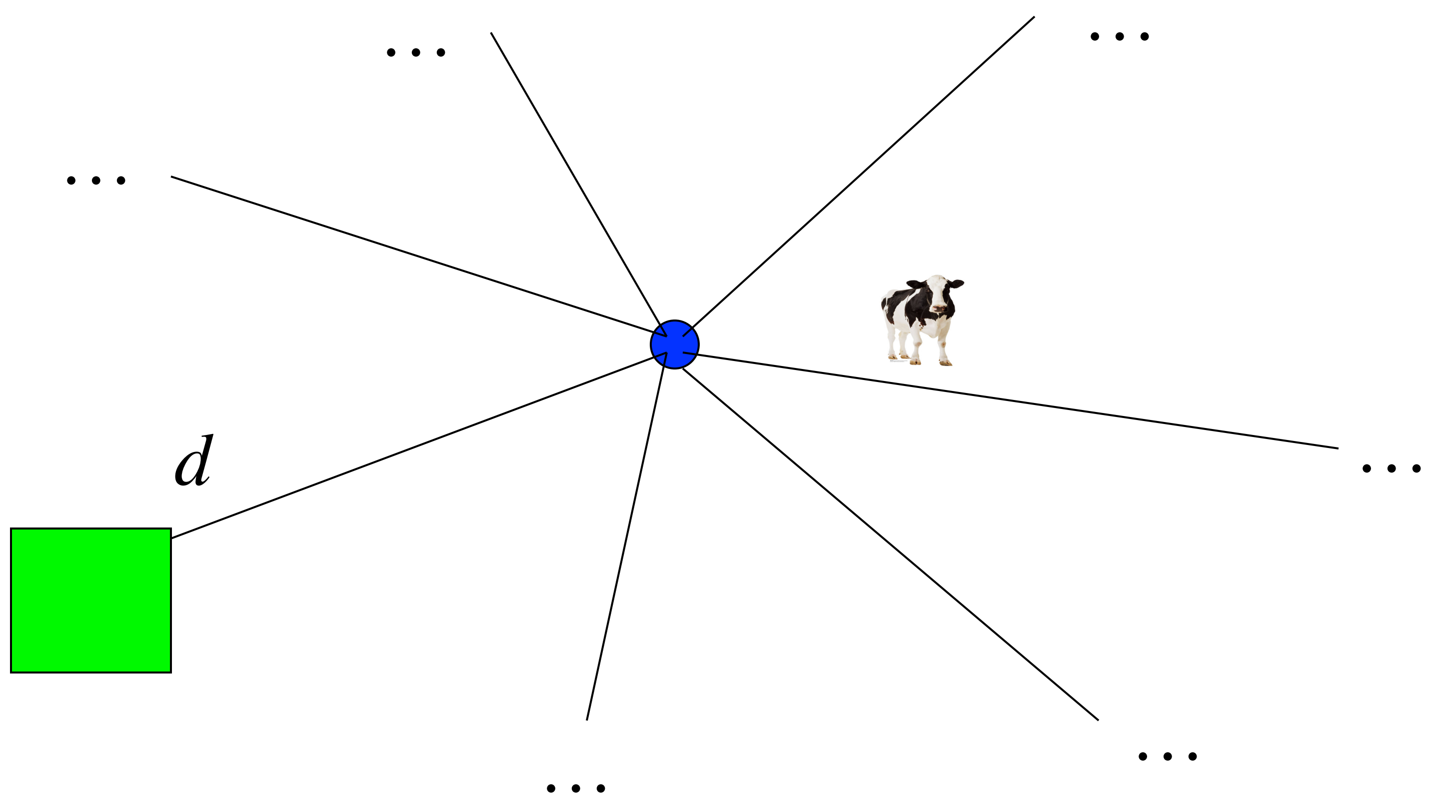
m-lane cow-paths problem



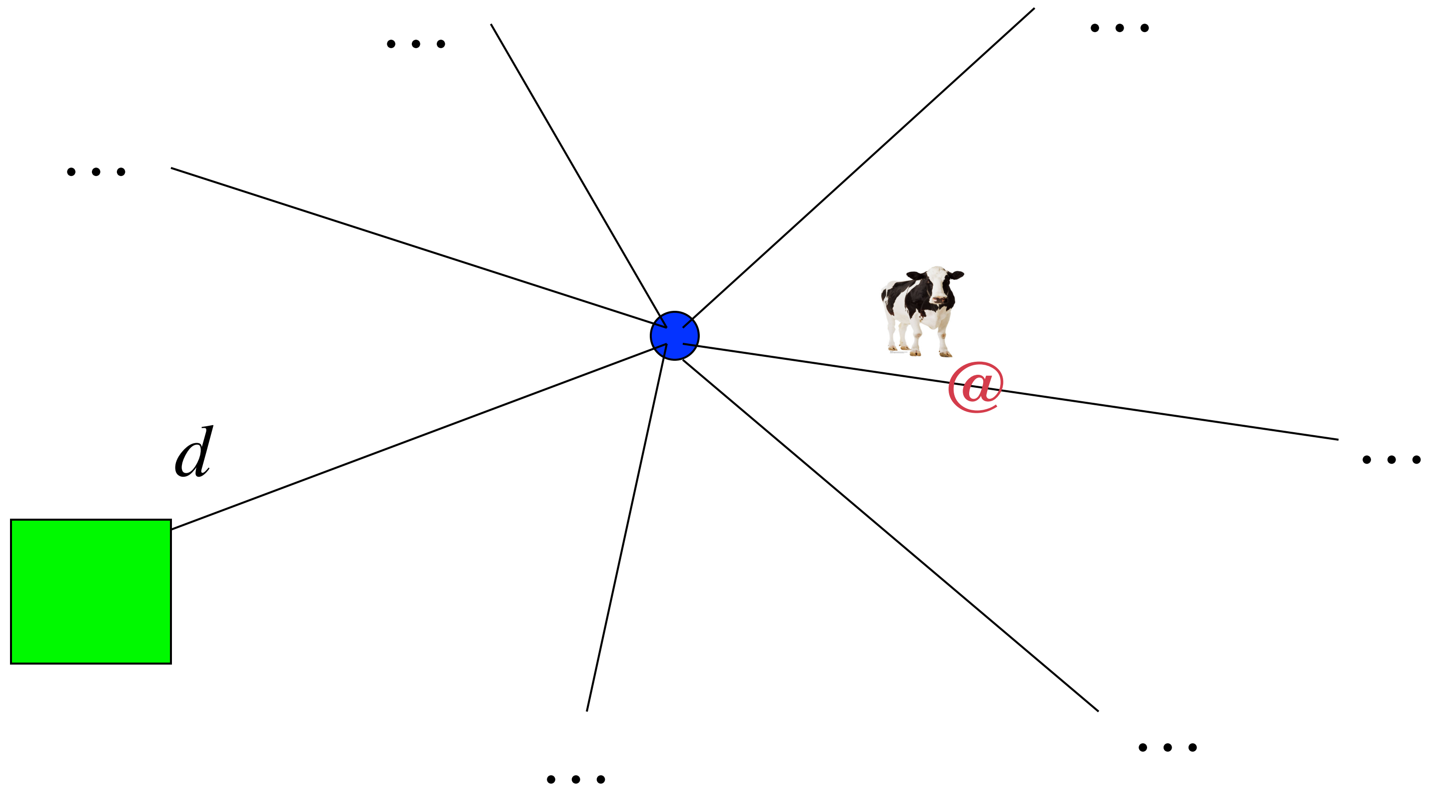
m-lane cow-paths problem



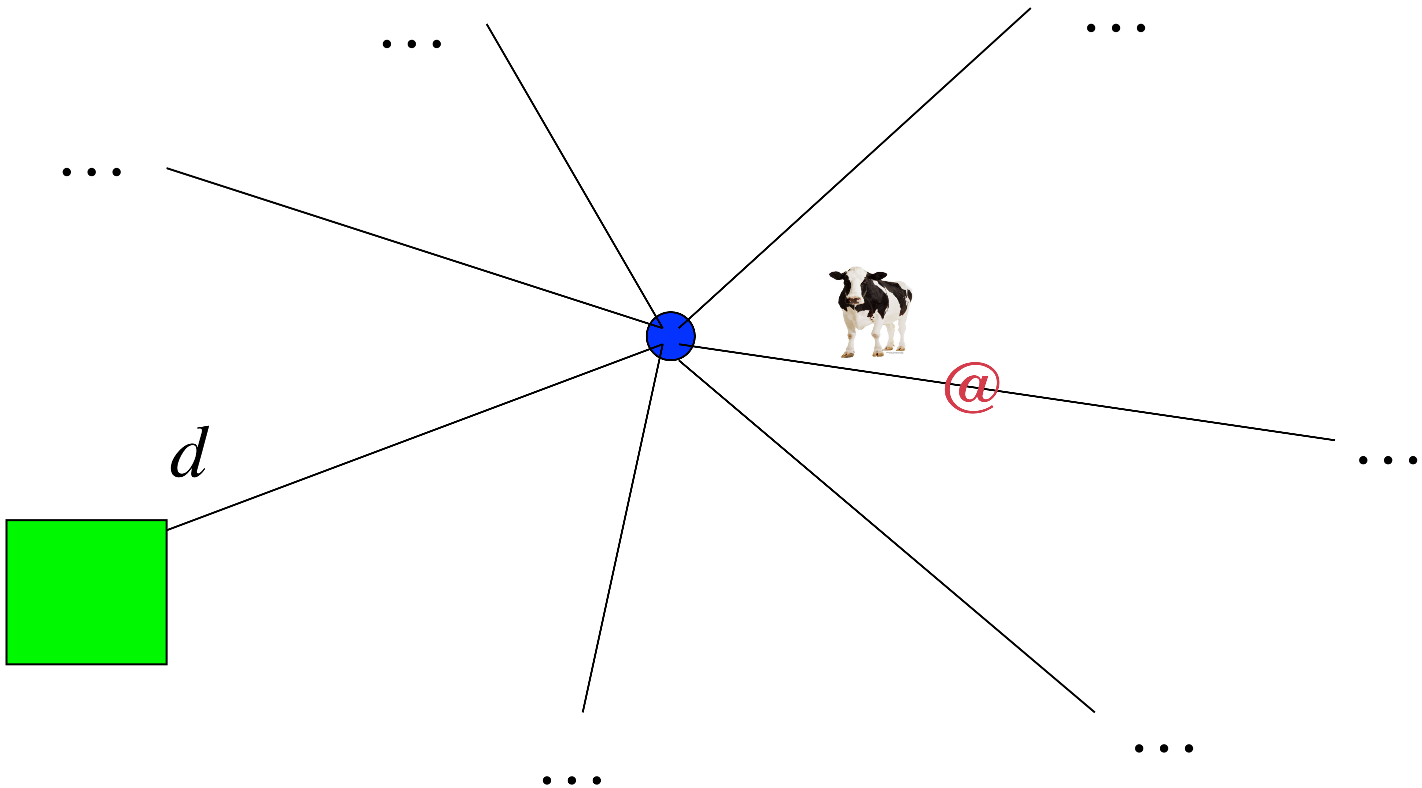
m-lane cow-paths problem



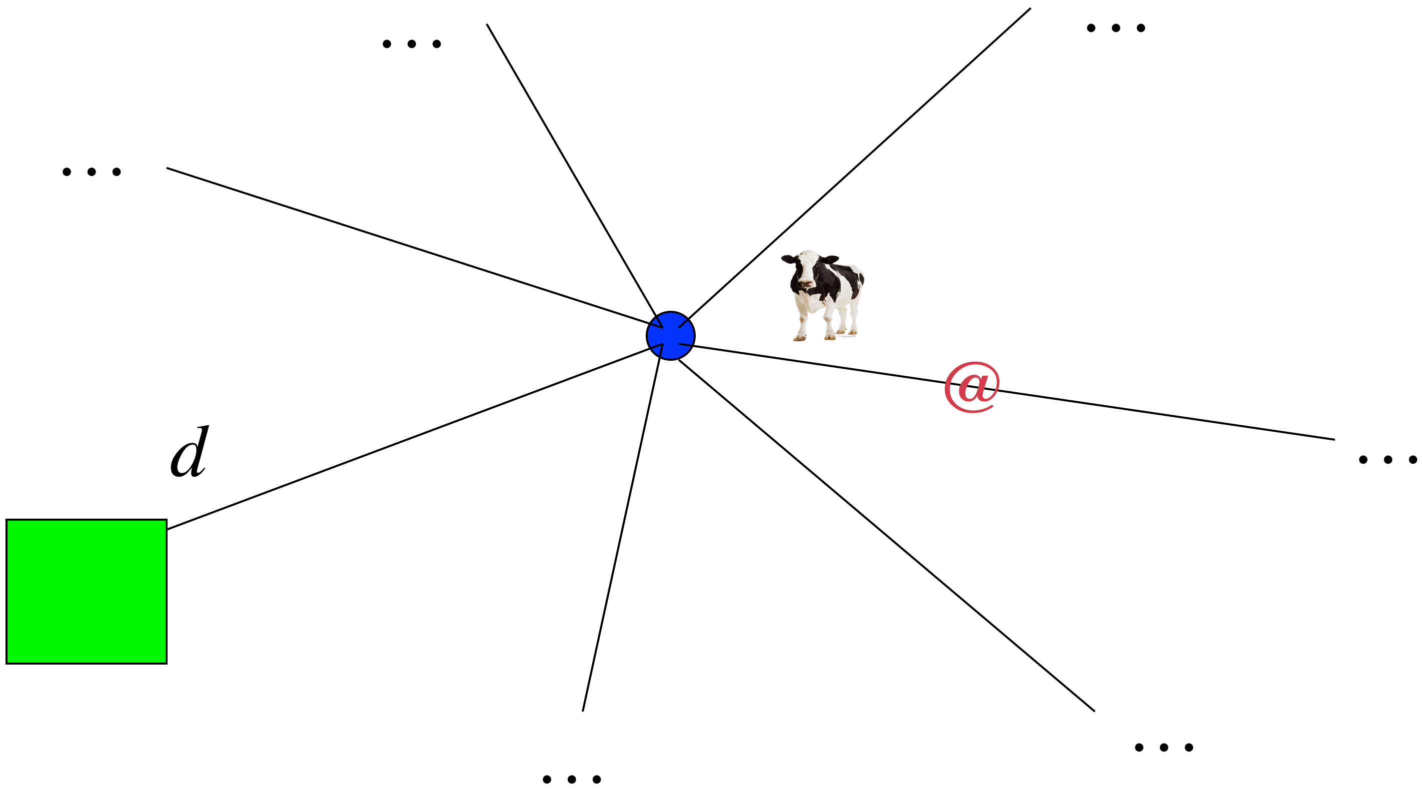
m-lane cow-paths problem



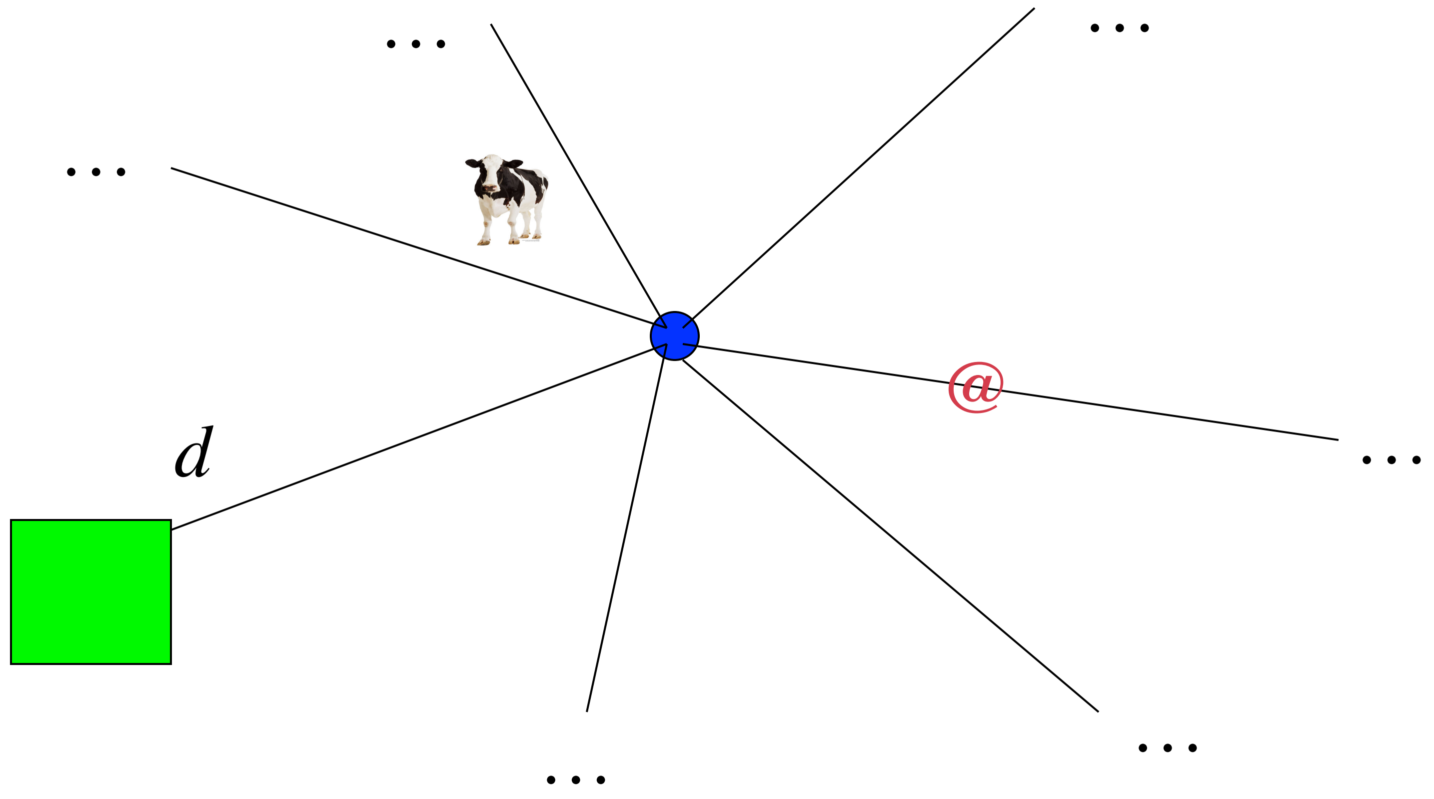
m-lane cow-paths problem



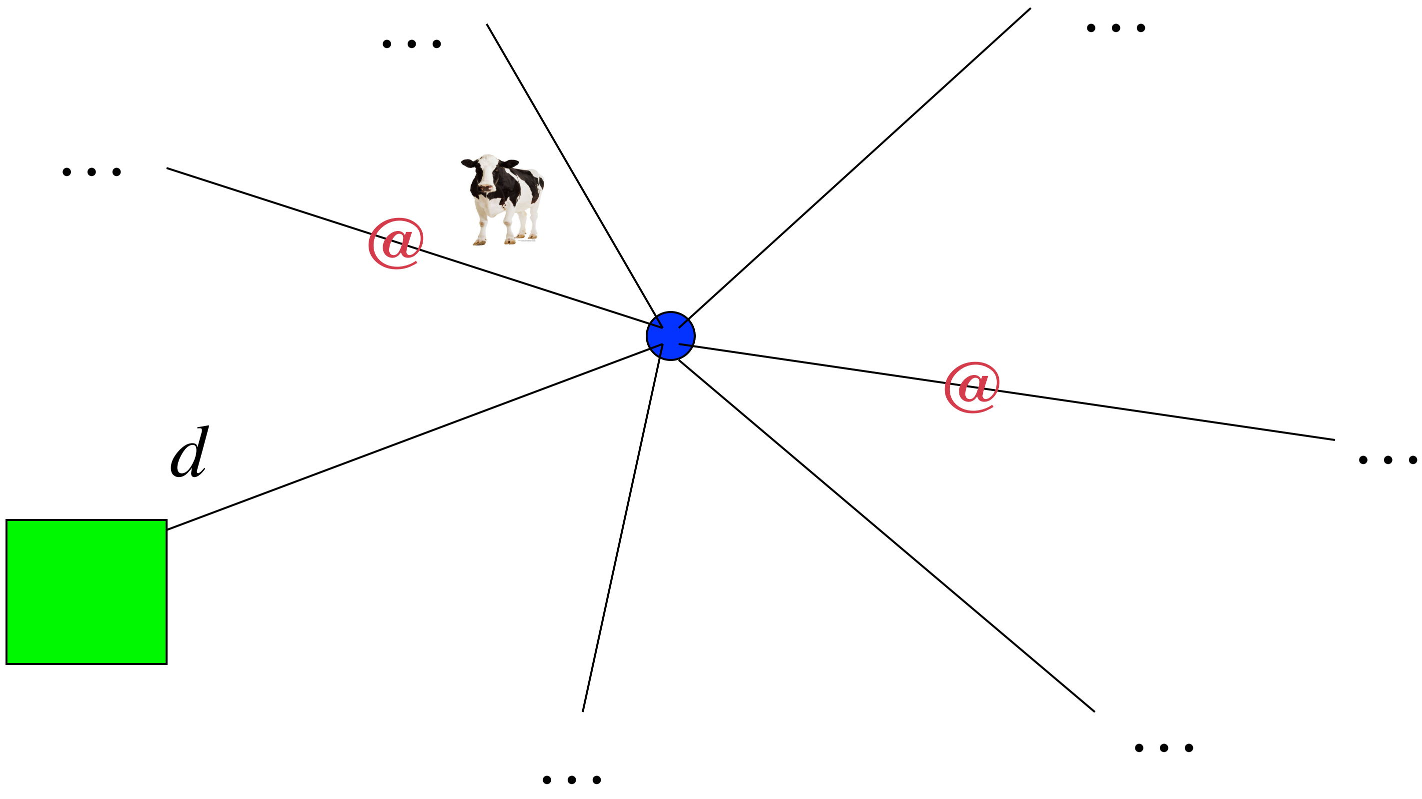
m-lane cow-paths problem



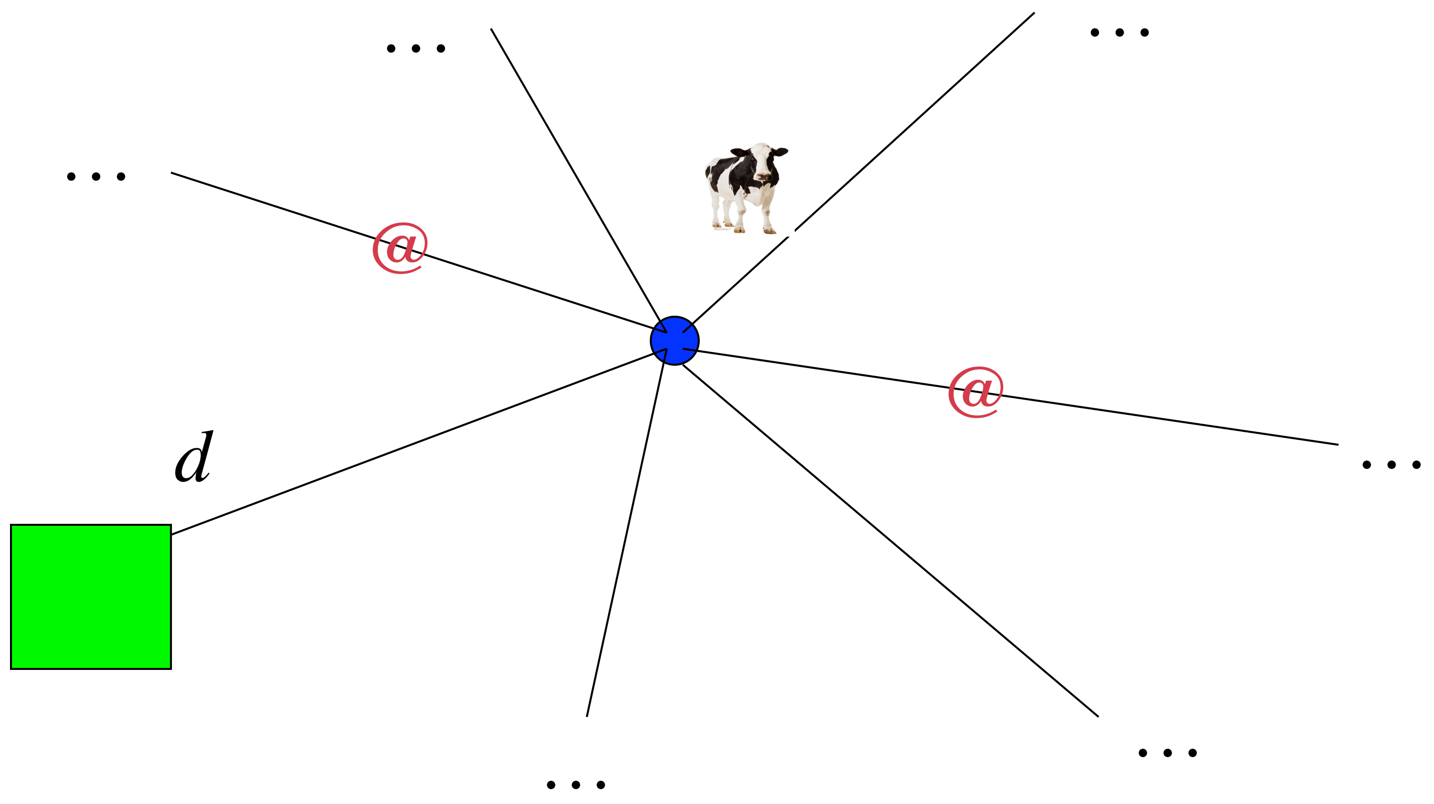
m-lane cow-paths problem



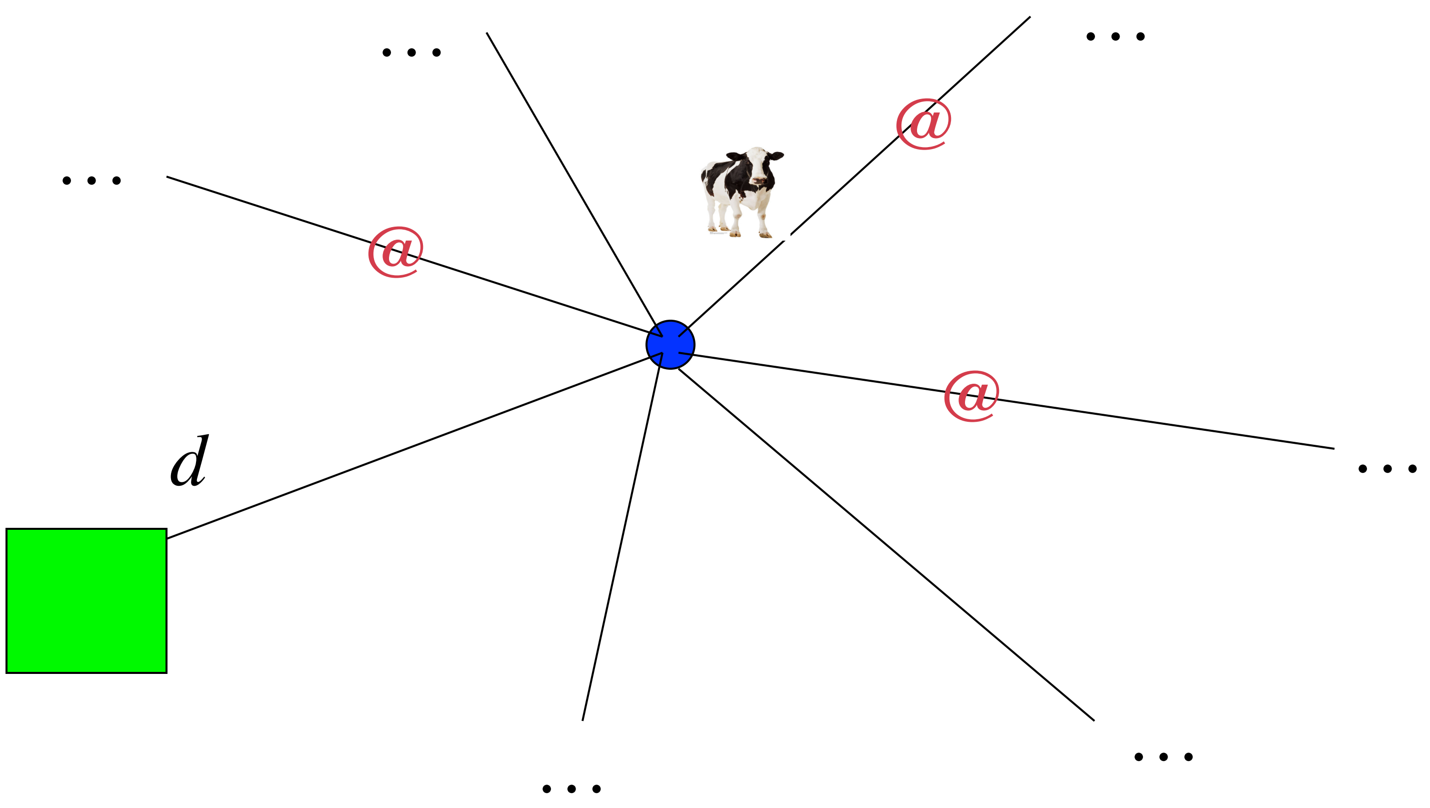
m-lane cow-paths problem



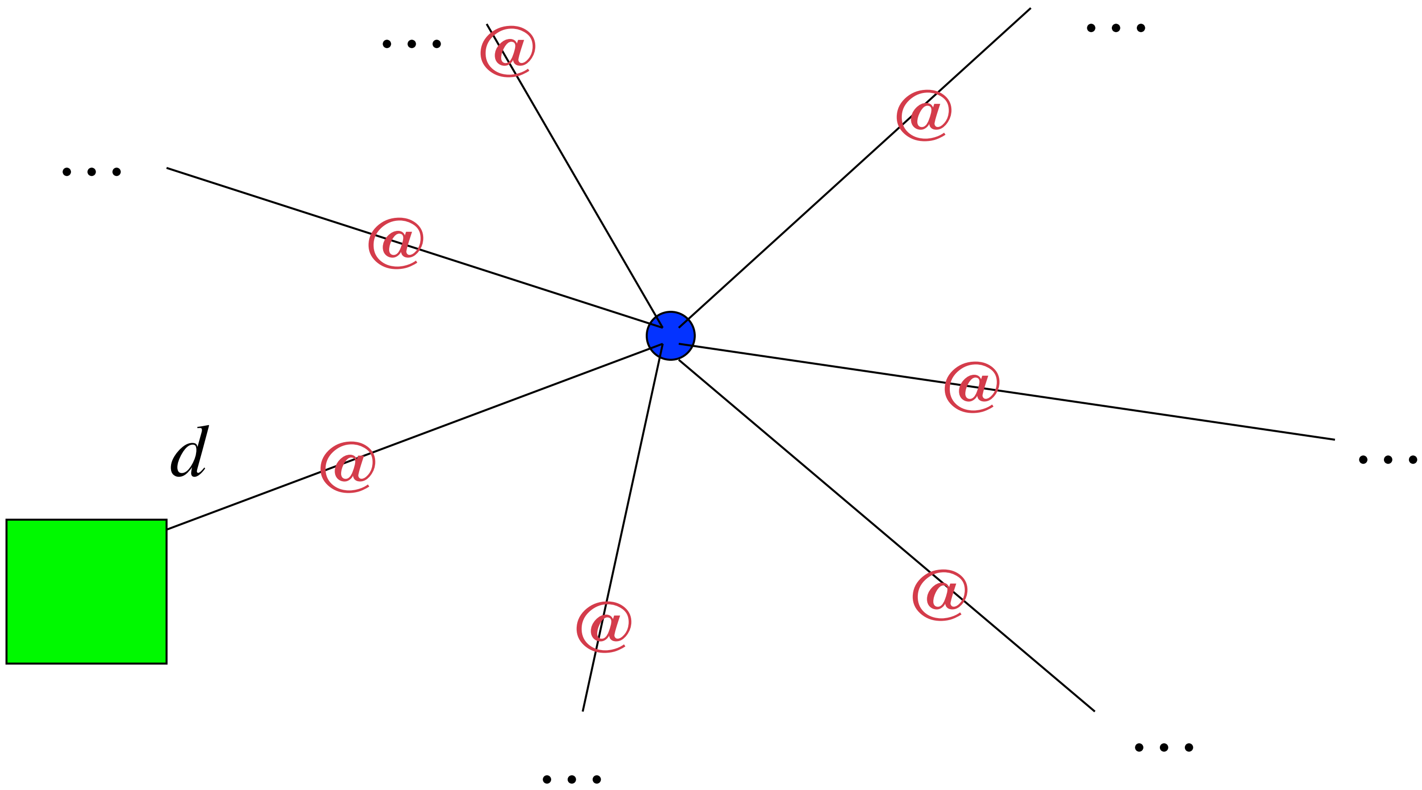
m-lane cow-paths problem



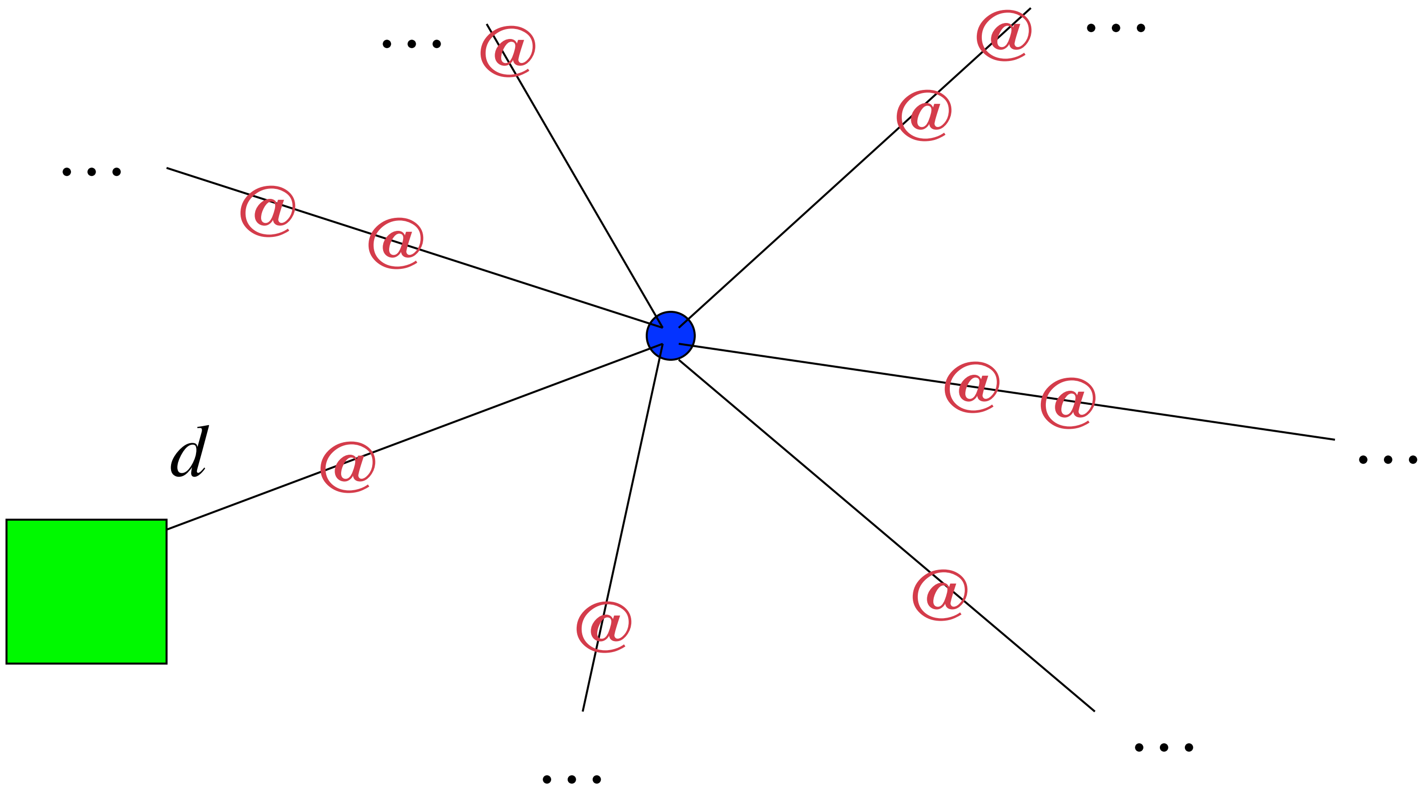
m-lane cow-paths problem



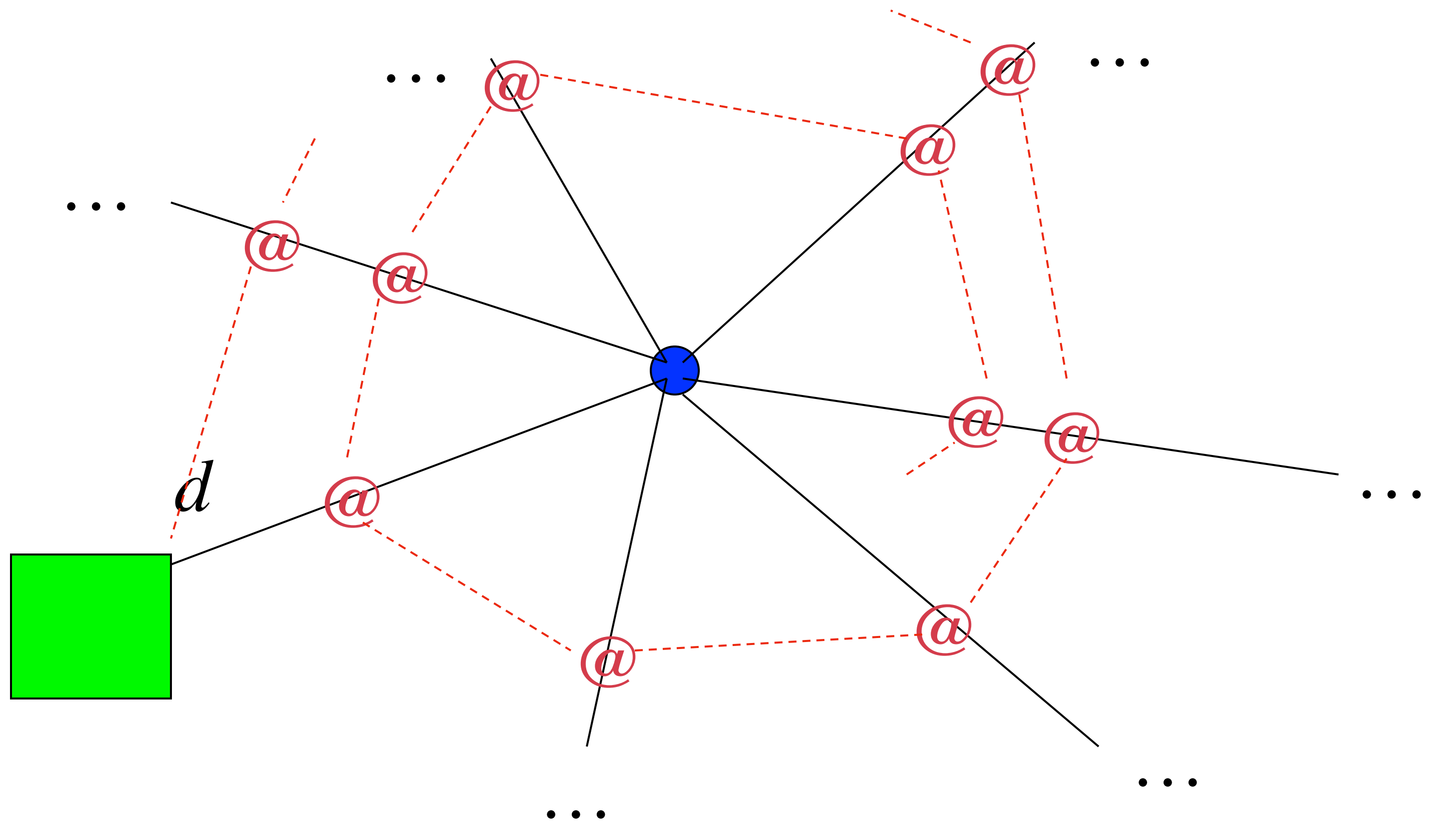
m-lane cow-paths problem



m-lane cow-paths problem

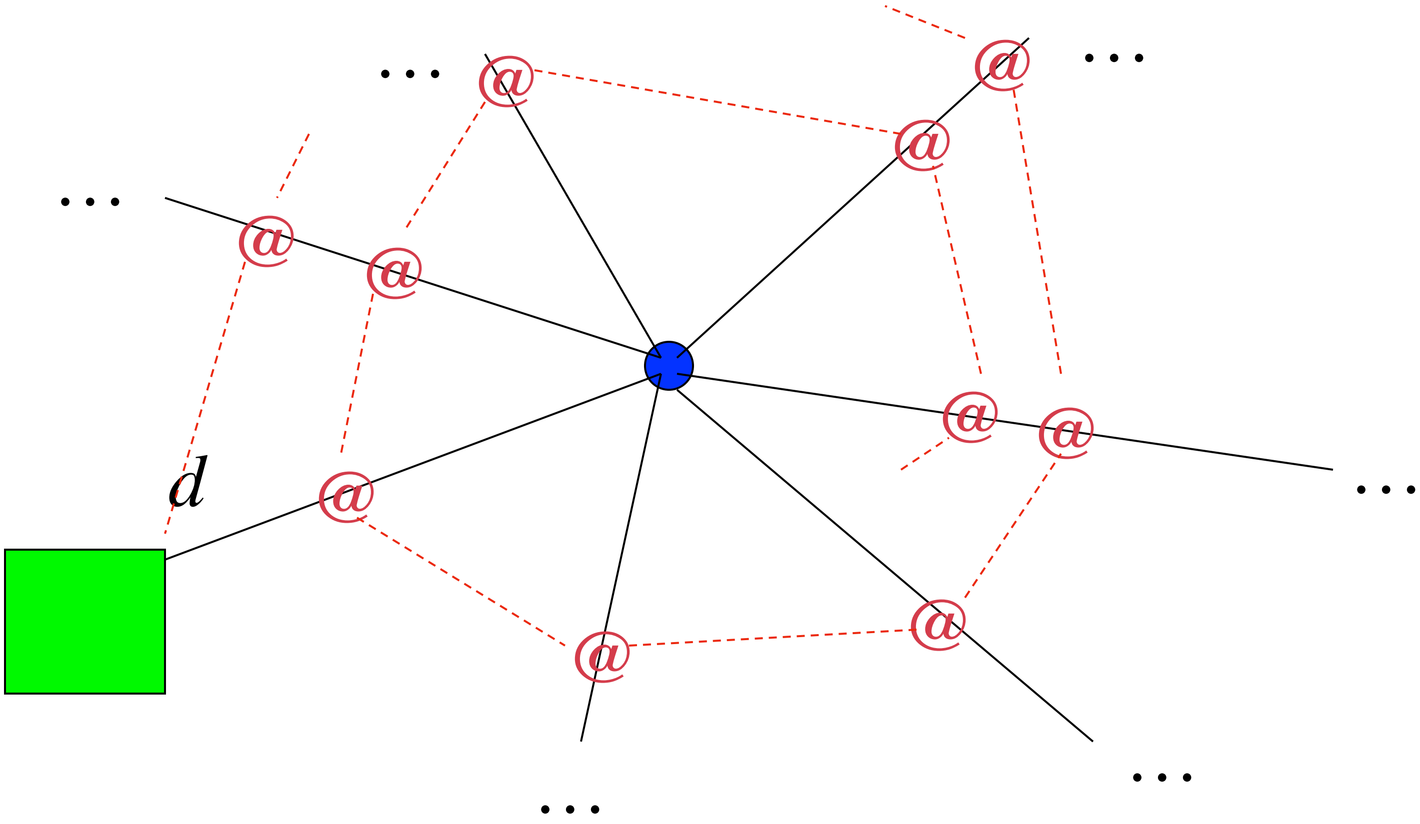


m-lane cow-paths problem



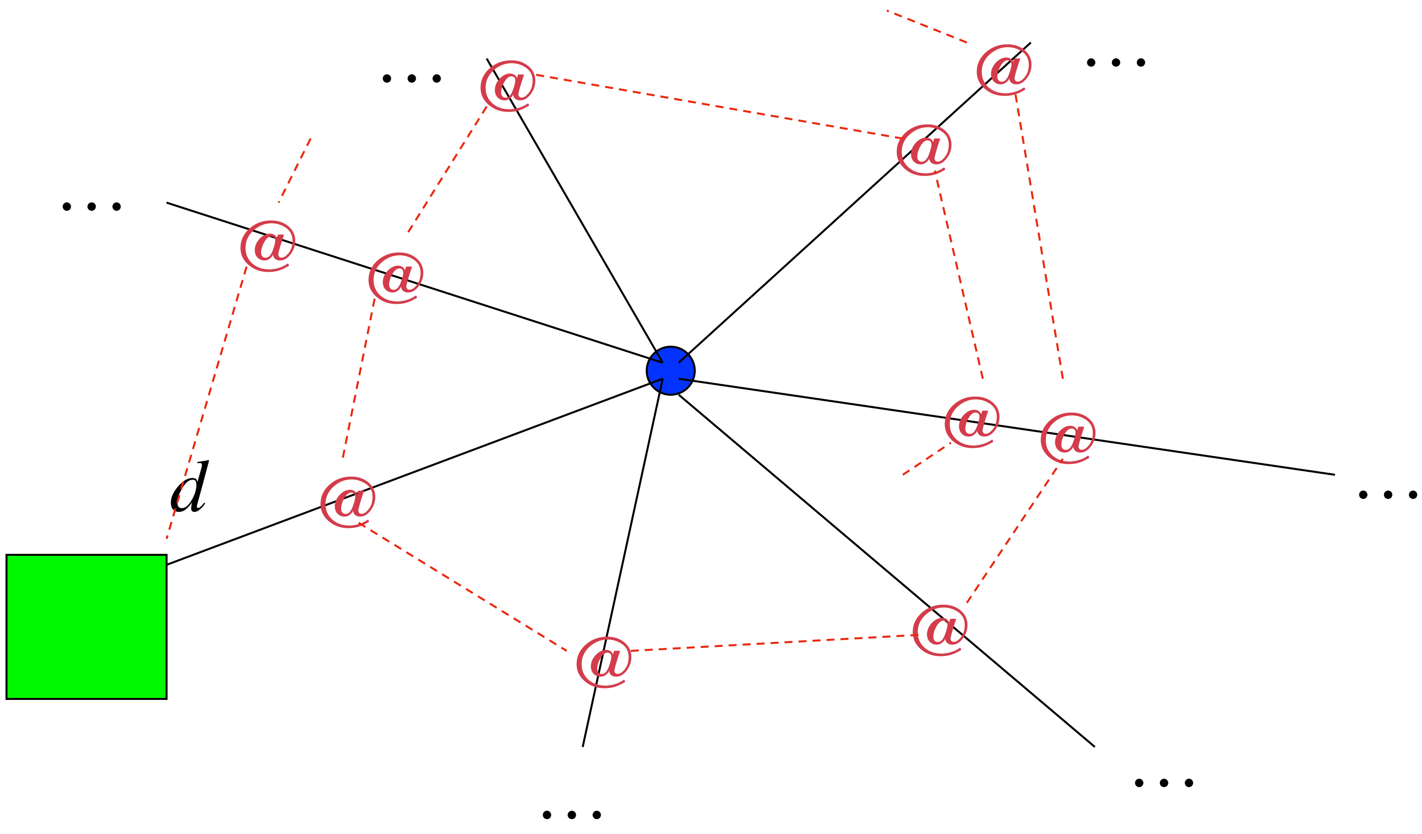
m-lane cow-paths problem

“spiraling” breadth-first (equitable) search



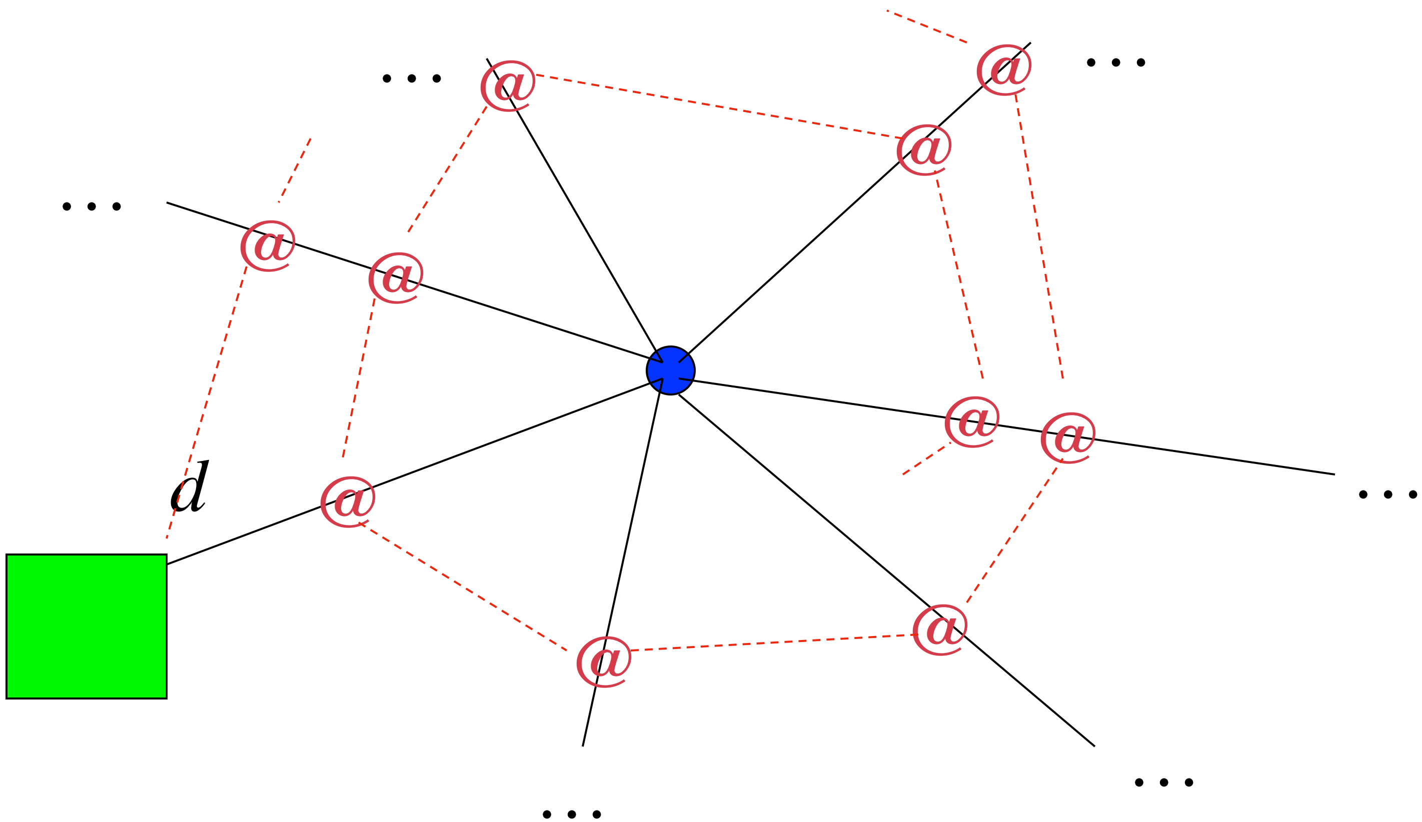
m-lane cow-paths problem

remains optimal under a variety of cost models



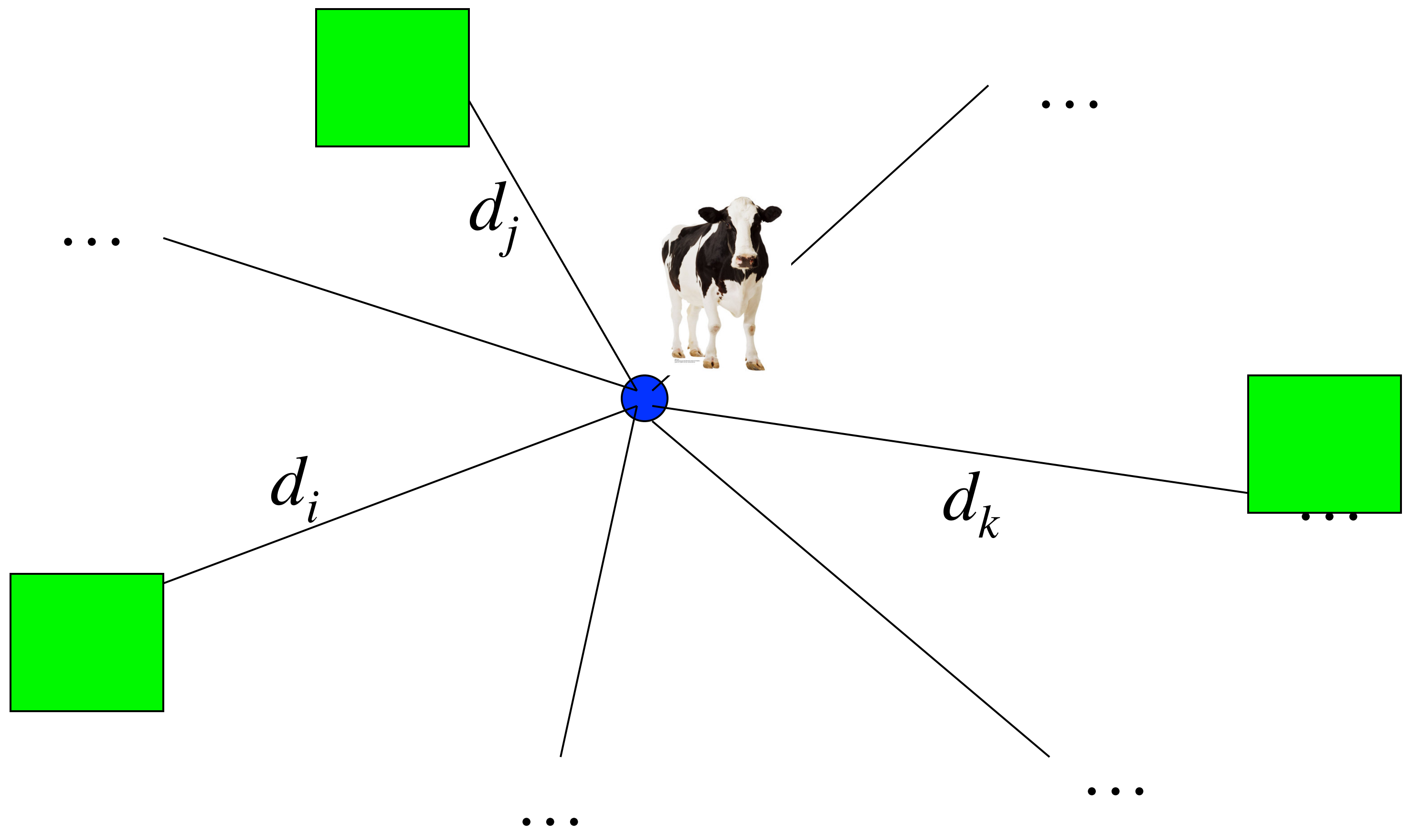
m-lane cow-paths problem

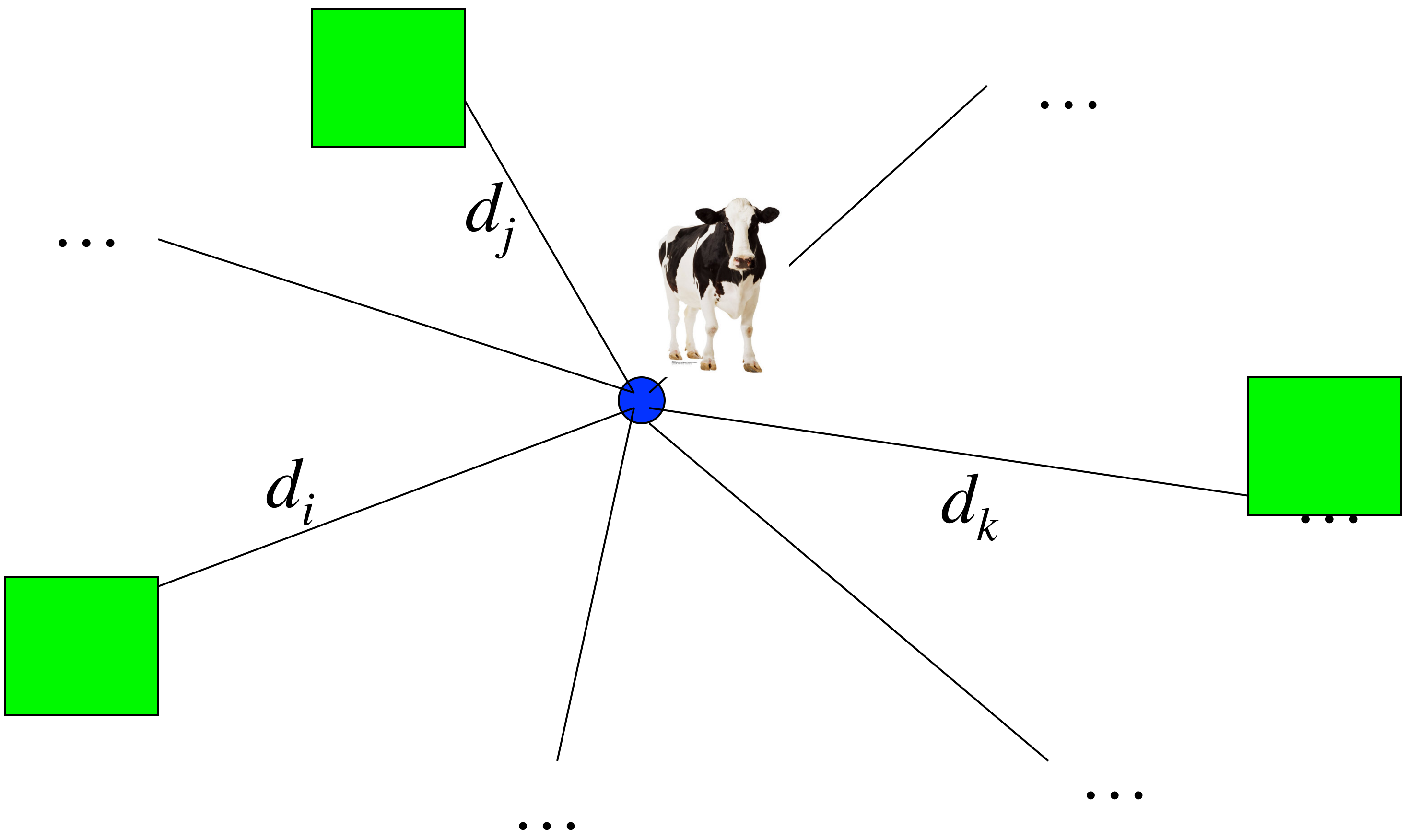
remains optimal under a variety of cost models



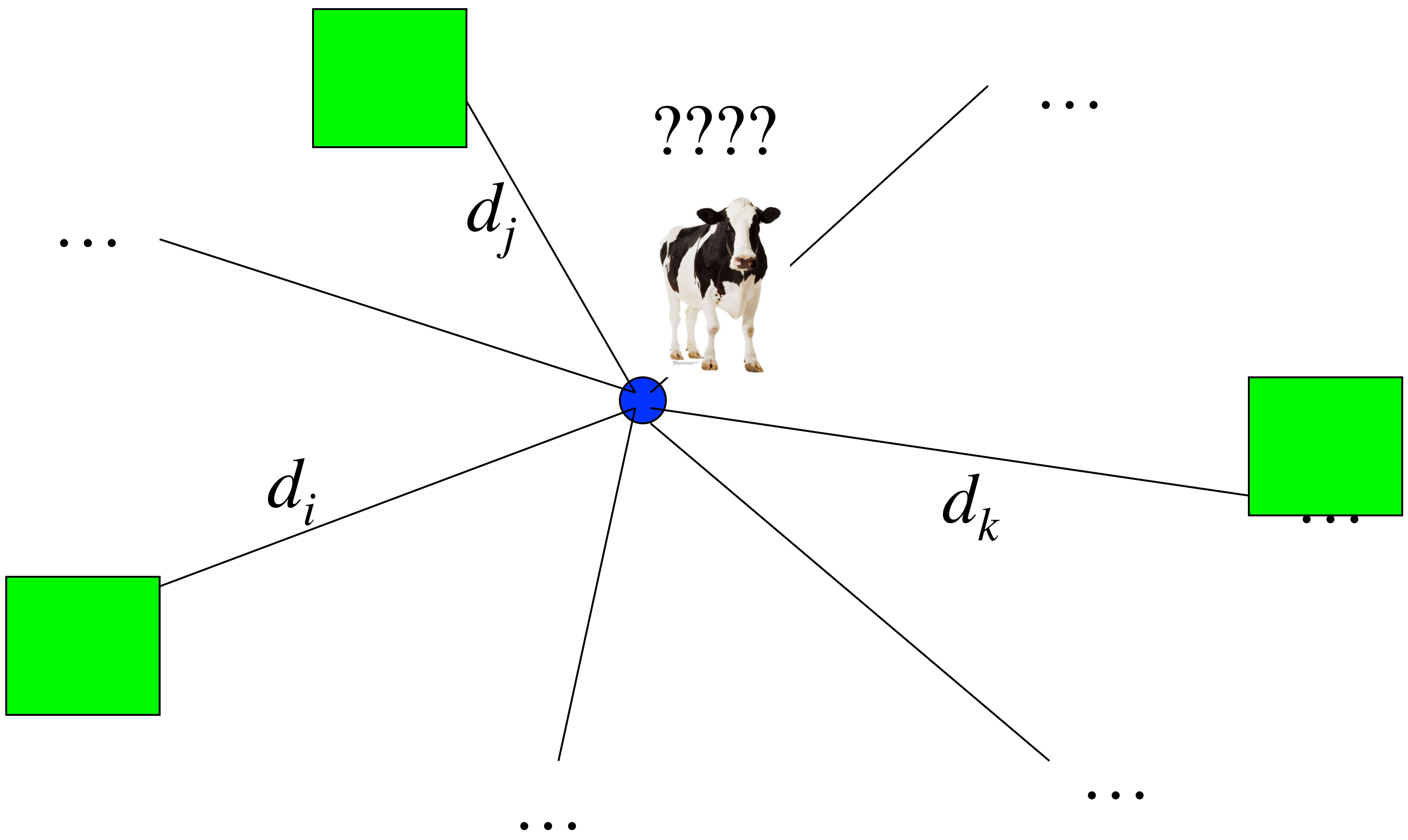
optimal multi-process "dovetailing"

What if there is more than one pasture?



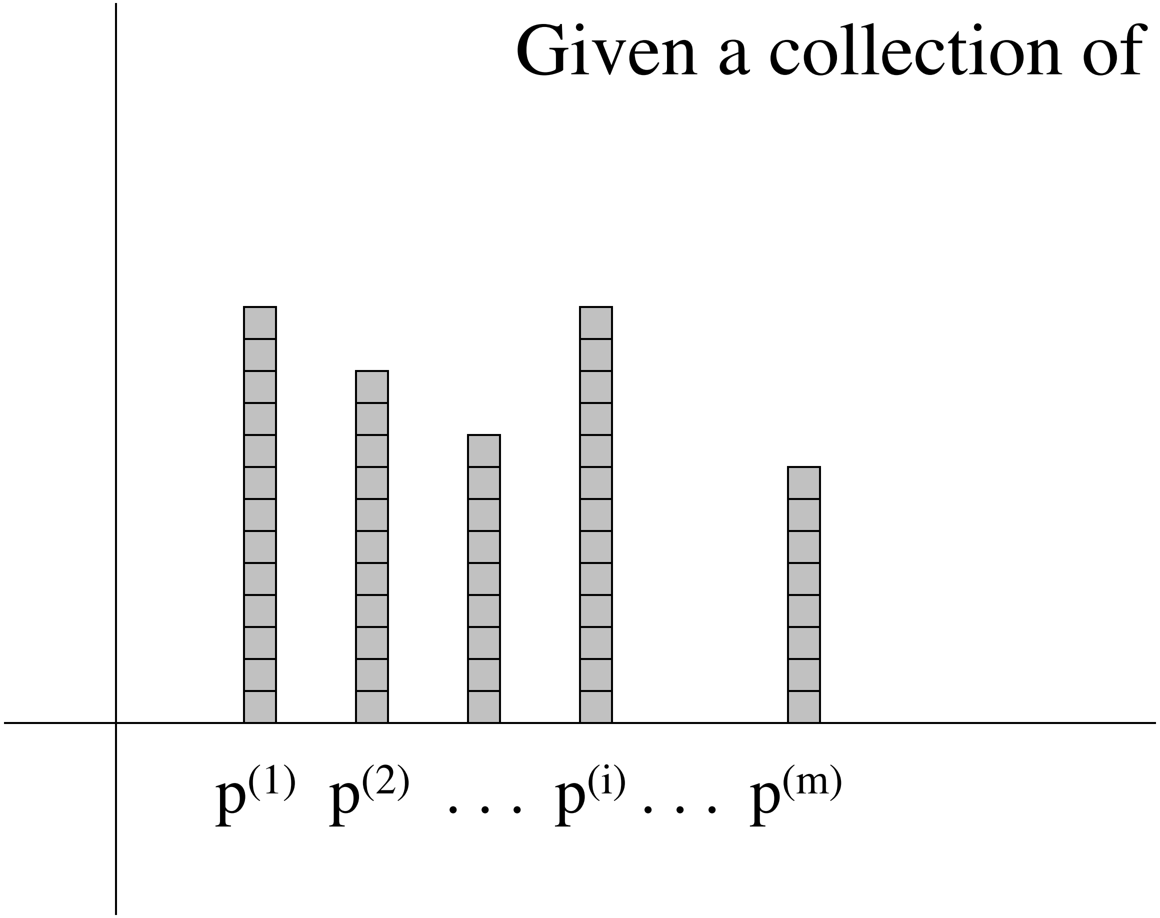


multi-pasture cow-paths problem

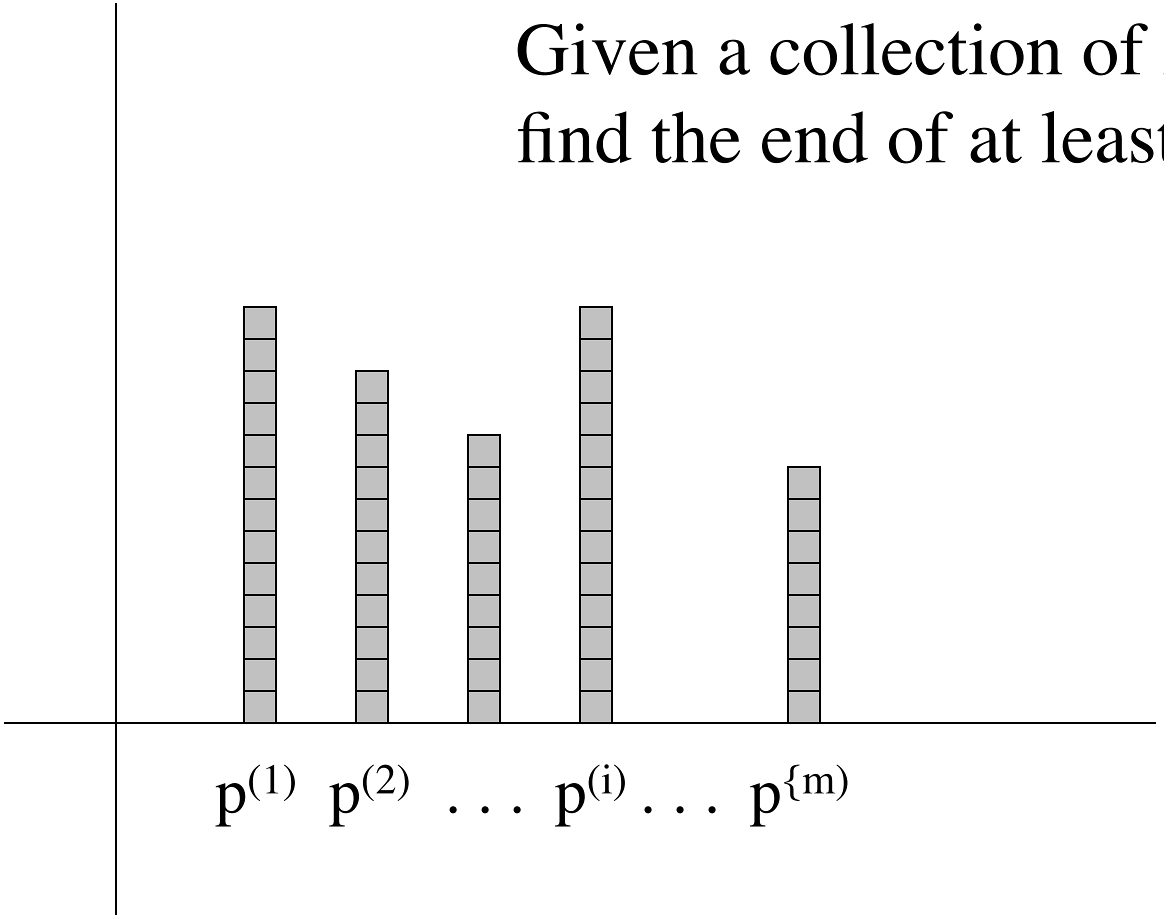


multi-pasture cow-paths problem

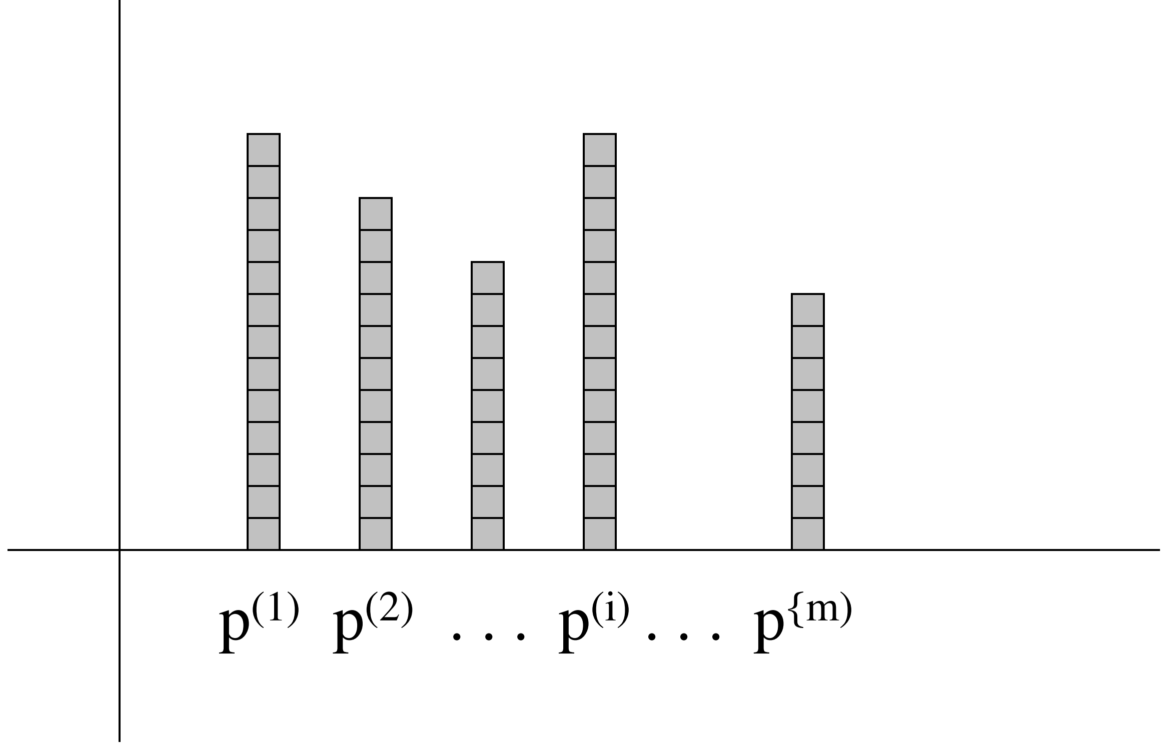
Given a collection of *m* lists:



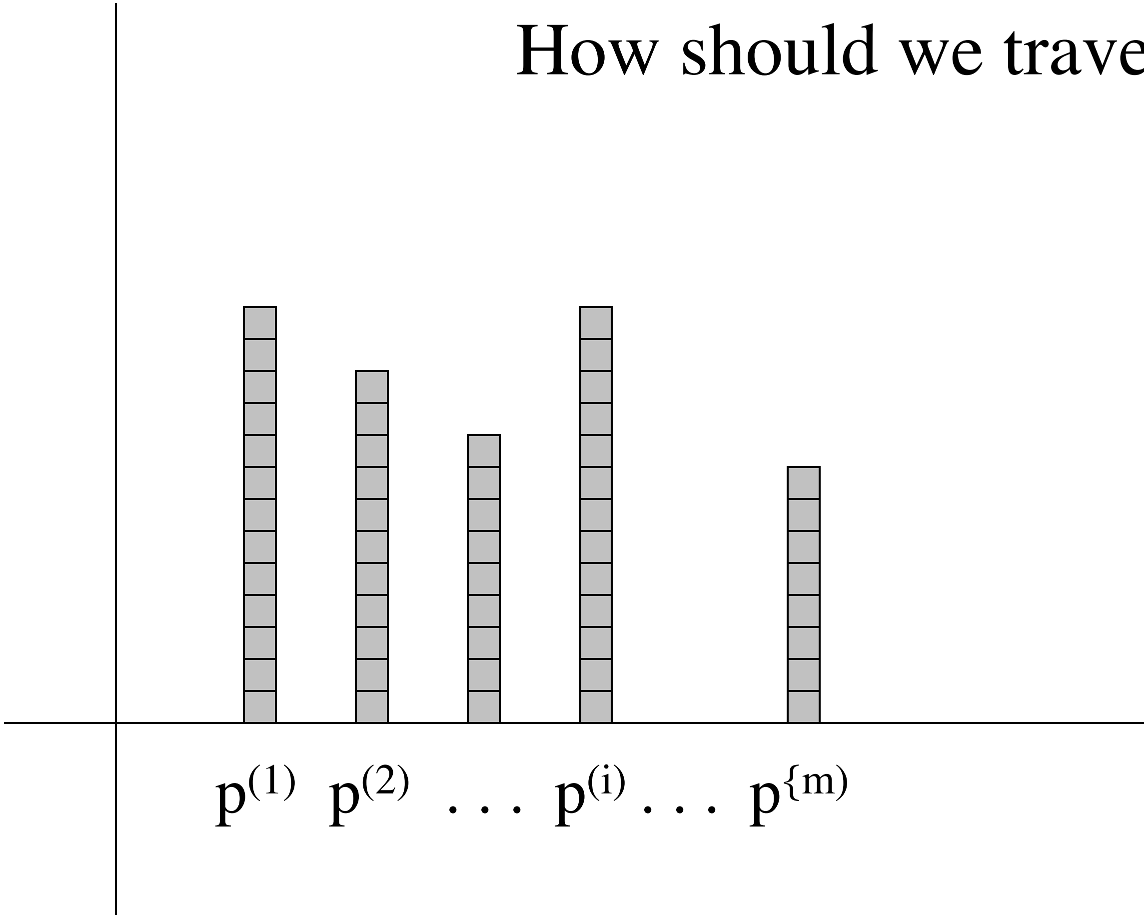
Given a collection of *m* lists: find the end of at least one



Given a collection of *m* lists:
find the end of at least one

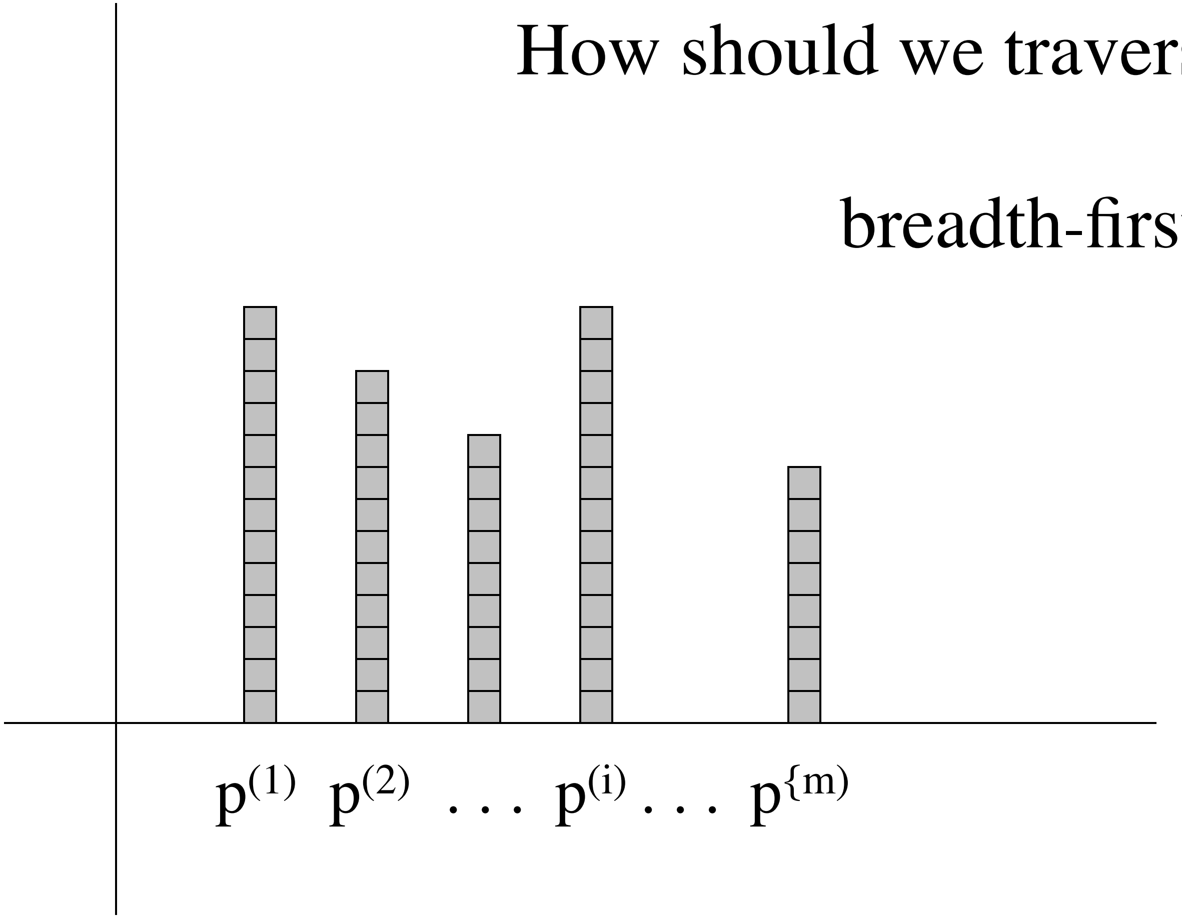


How should we traverse?

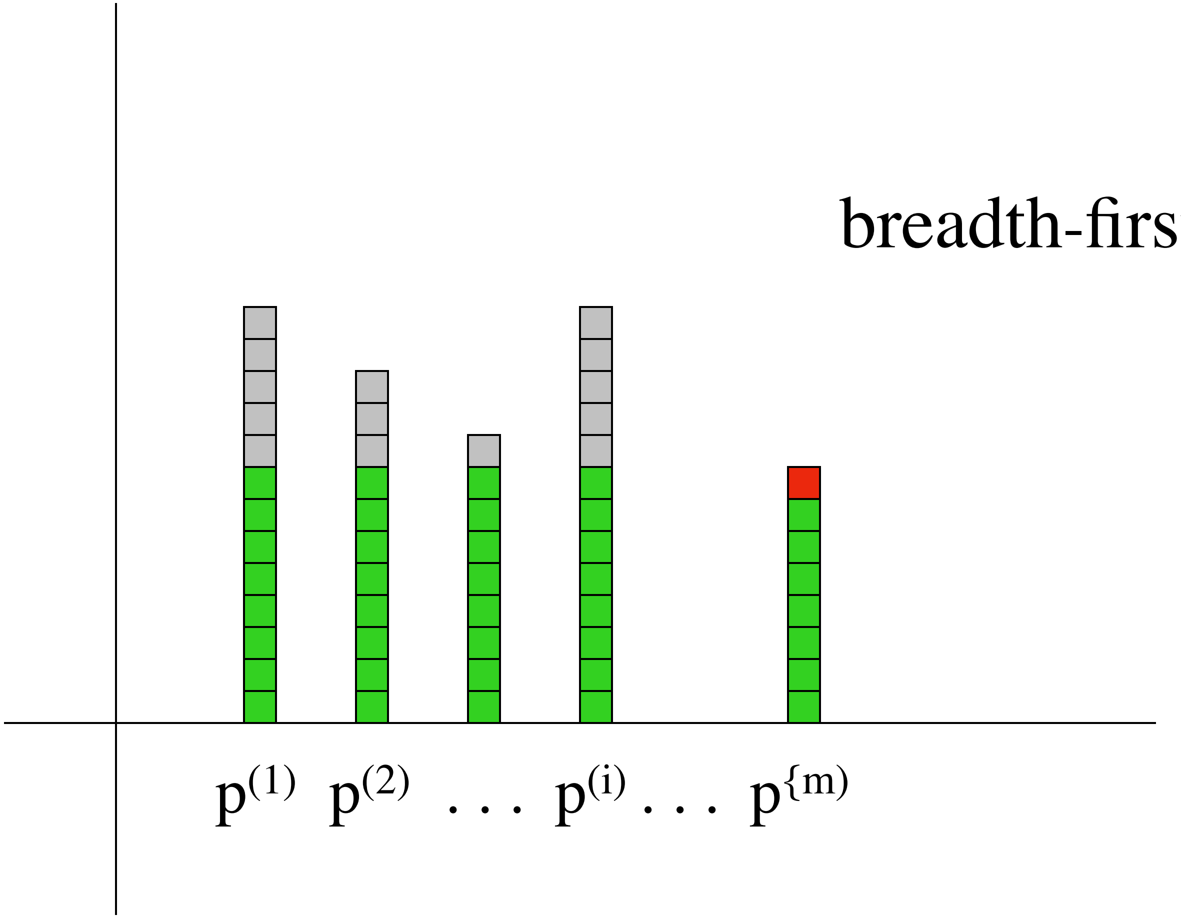


How should we traverse?

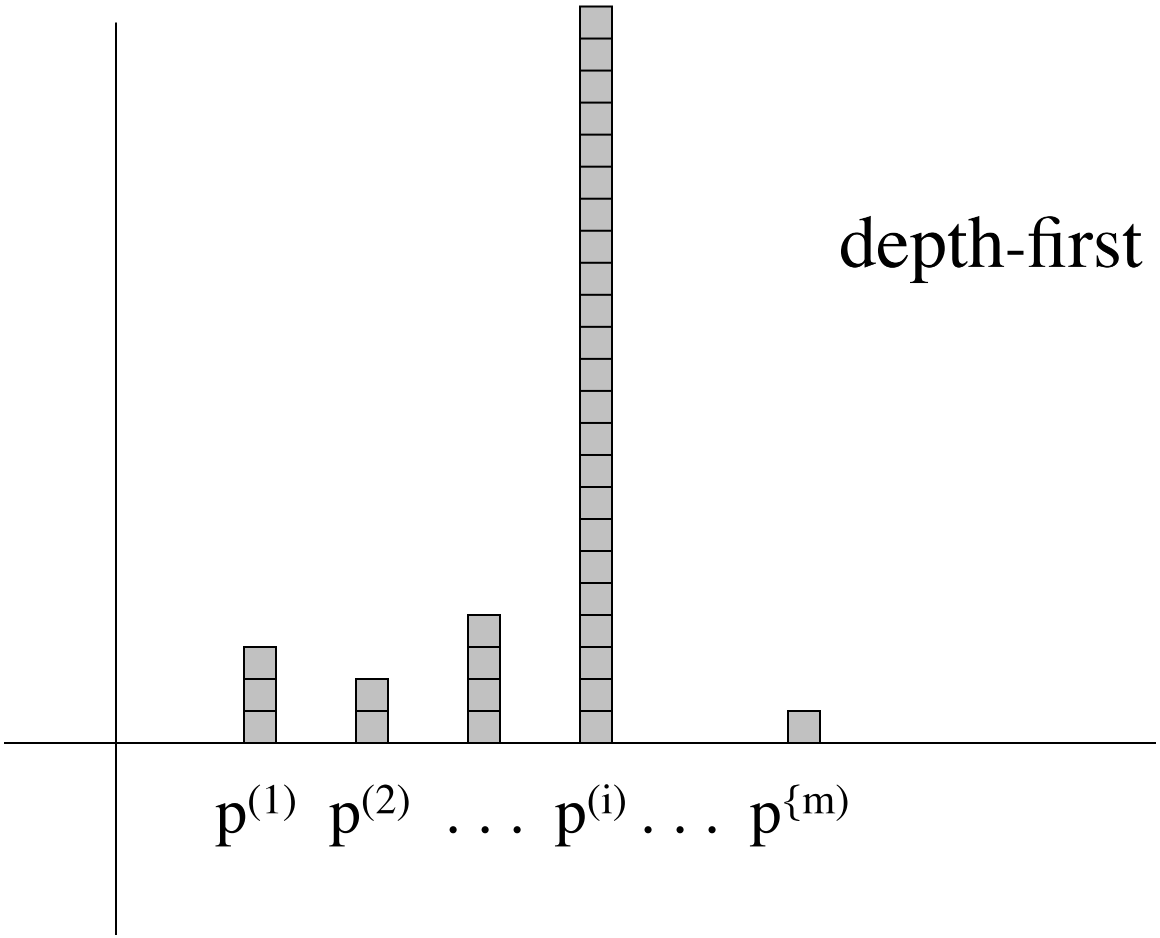
breadth-first search?



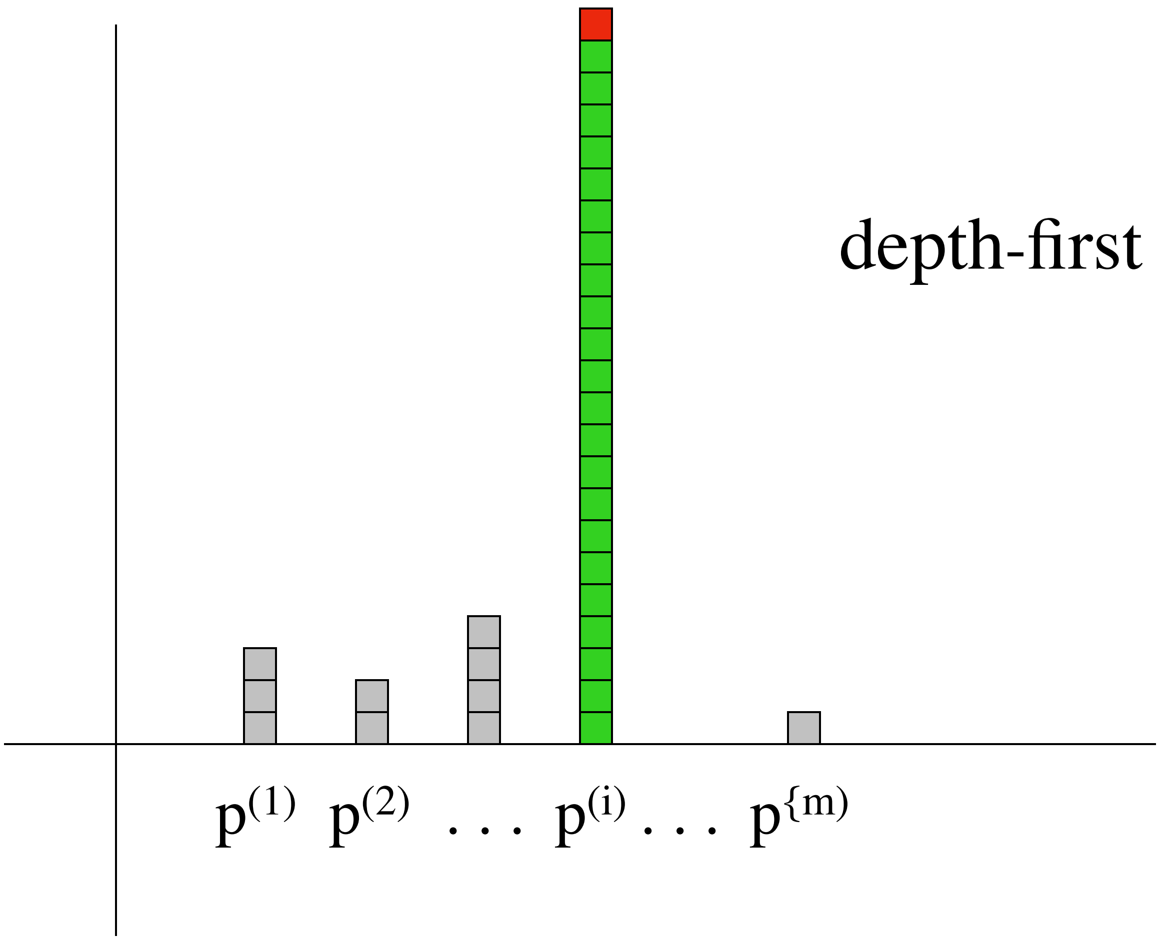
breadth-first search?



depth-first search?



depth-first search?



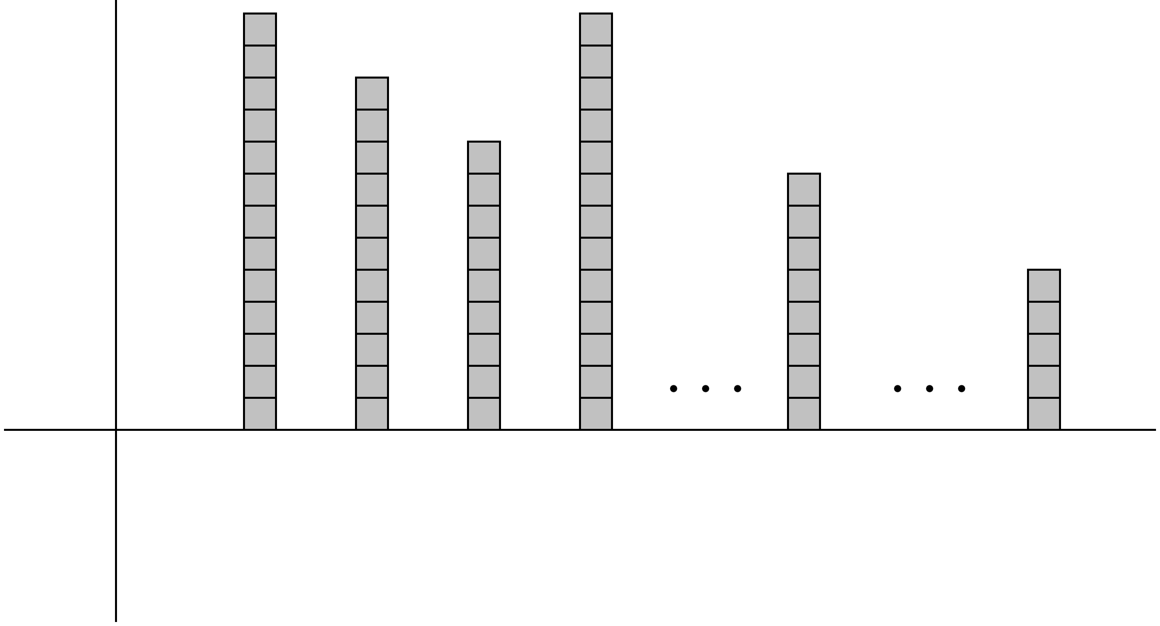
Both breadth-first and depth-first search can be *arbitrarily bad*

Both breadth-first and depth-first search can be *arbitrarily bad* -- *relative to the size of the shortest certificate*.

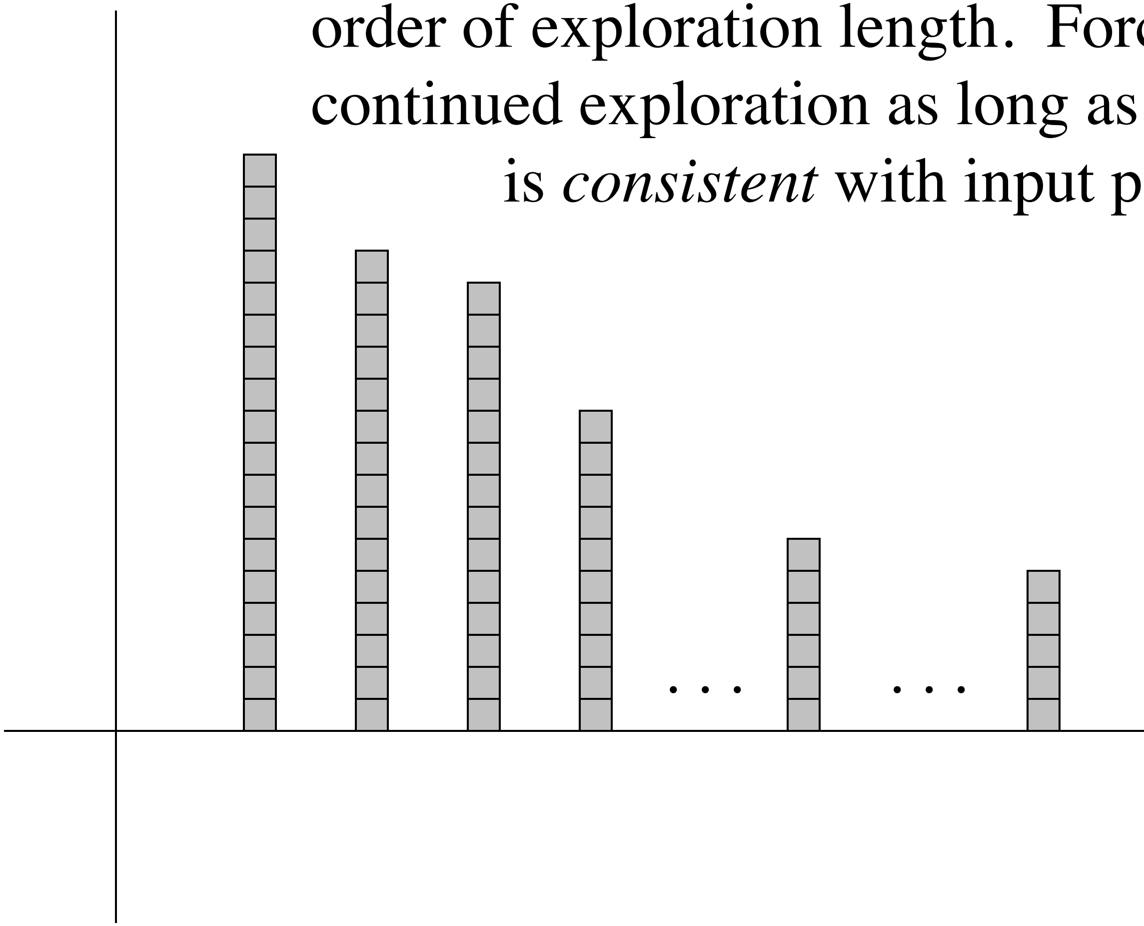
Both breadth-first and depth-first search can be *arbitrarily bad* *-- relative to the size of the shortest certificate*.

But can we hope to discover short certificates quickly?

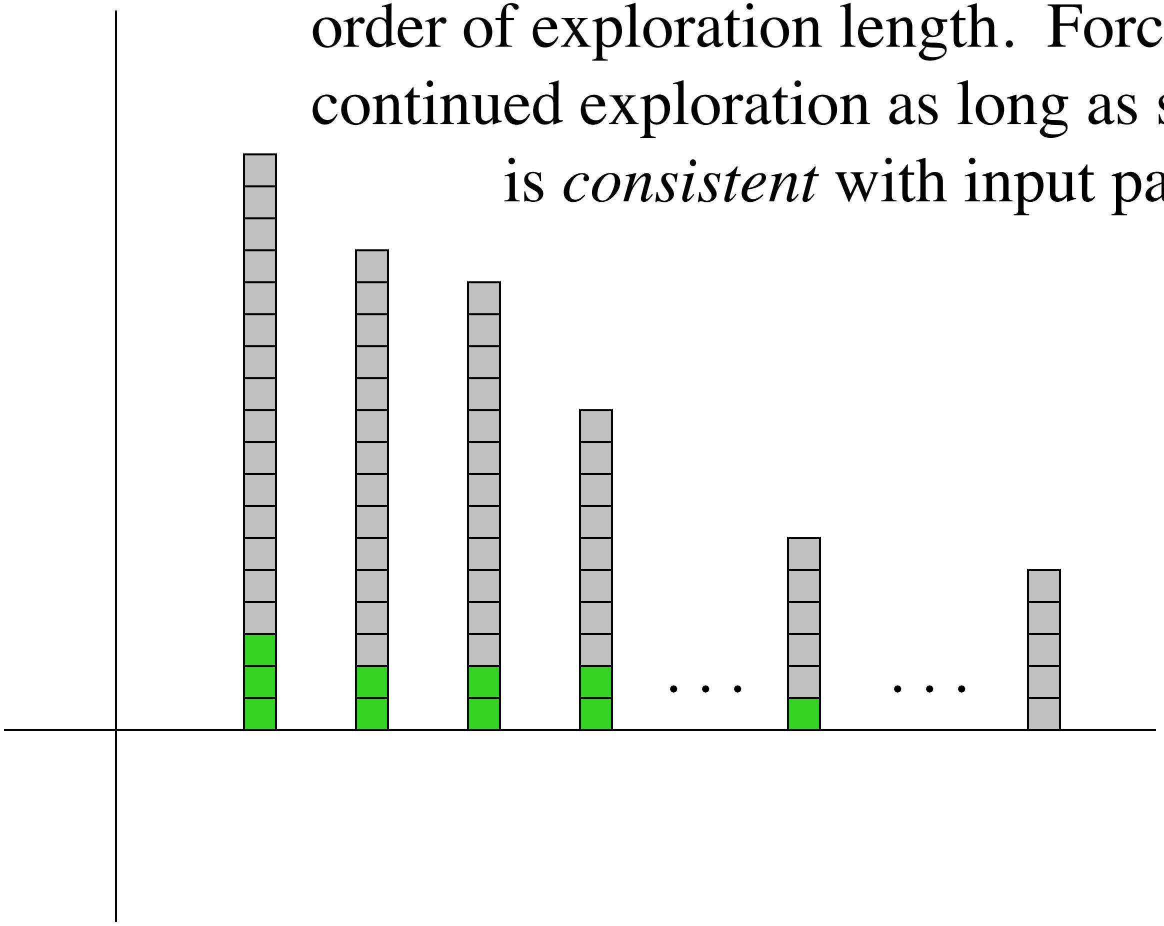
Suppose an algorithm is given the *pattern* $\pi=\{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ of list lengths, but not their *presentation*.



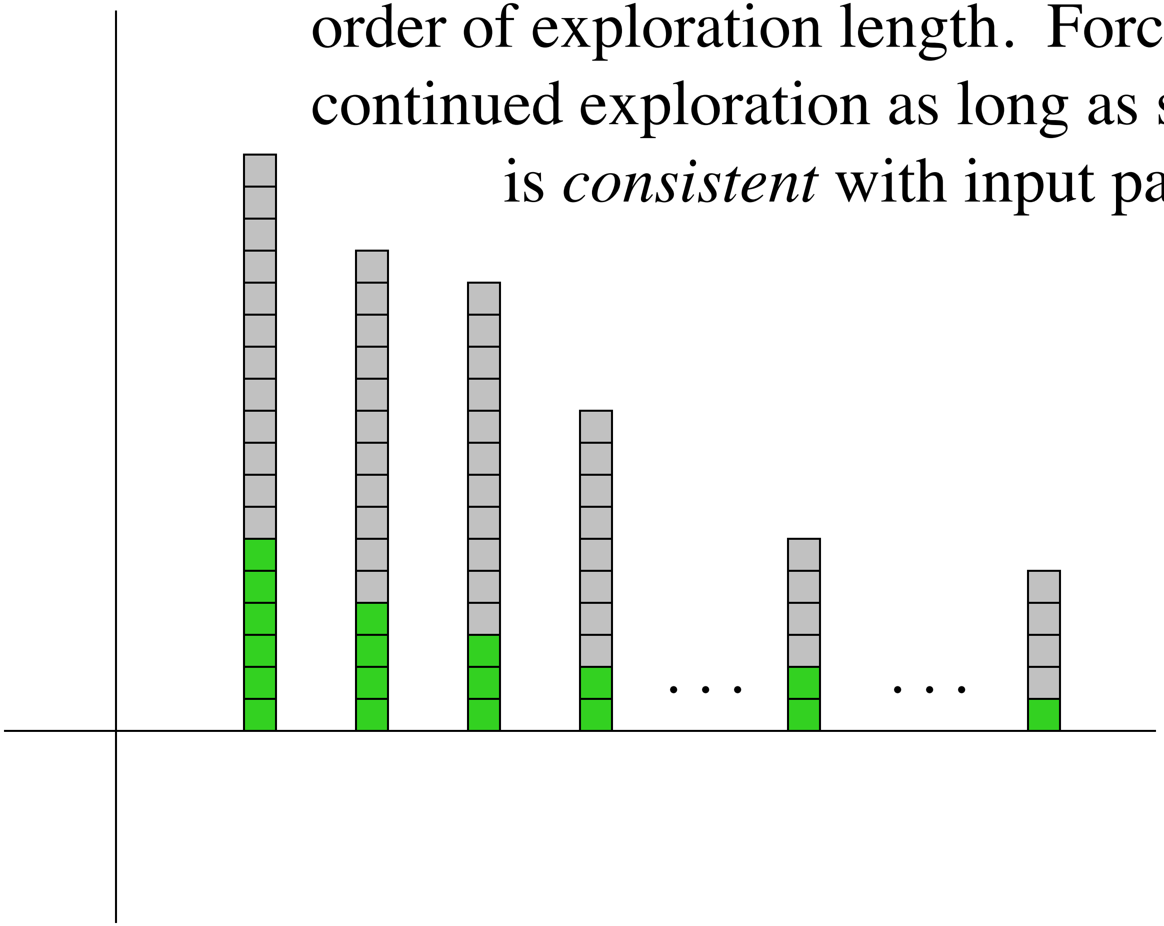
Adversary strategy: Maintain lists in order of exploration length. Force continued exploration as long as sequence is *consistent* with input pattern π.



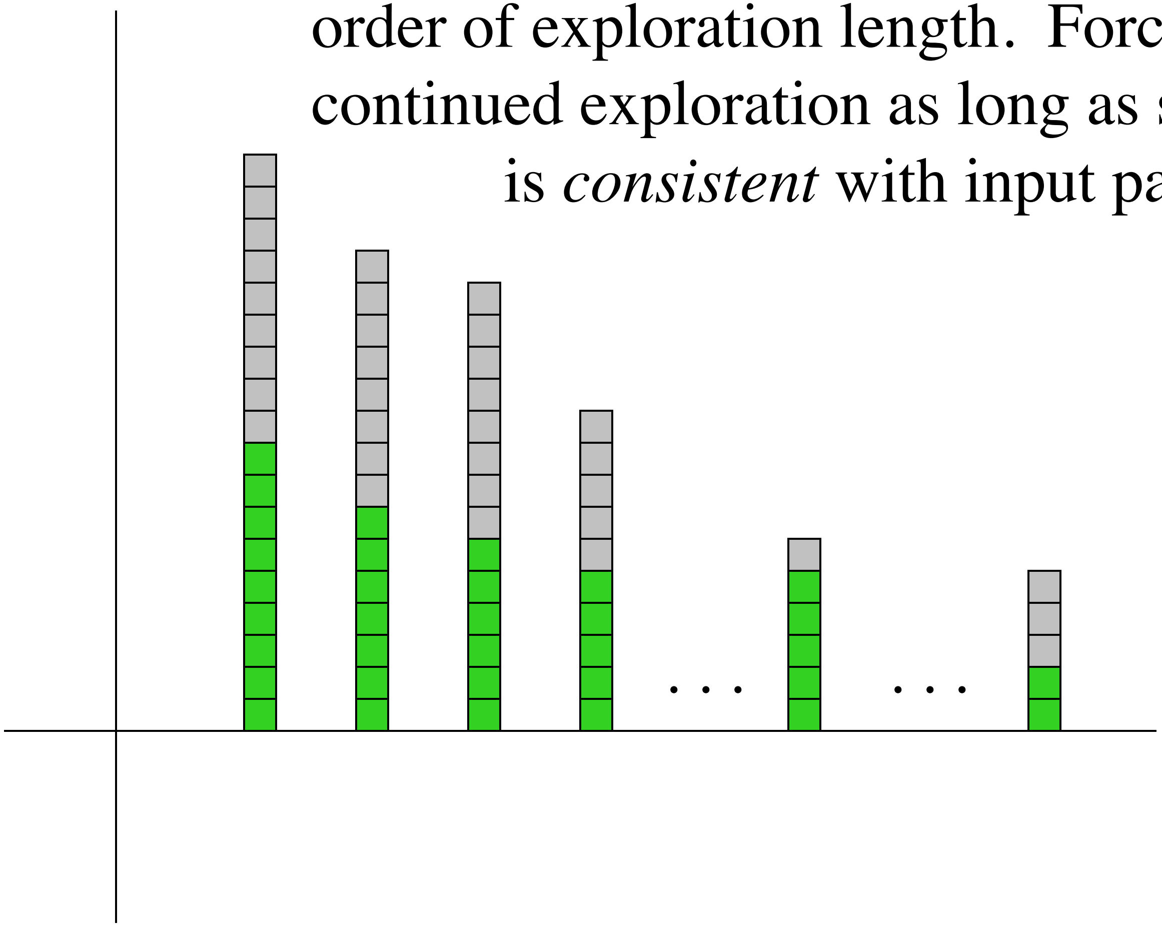
Adversary strategy: Maintain lists in order of exploration length. Force continued exploration as long as sequence is *consistent* with input pattern π.



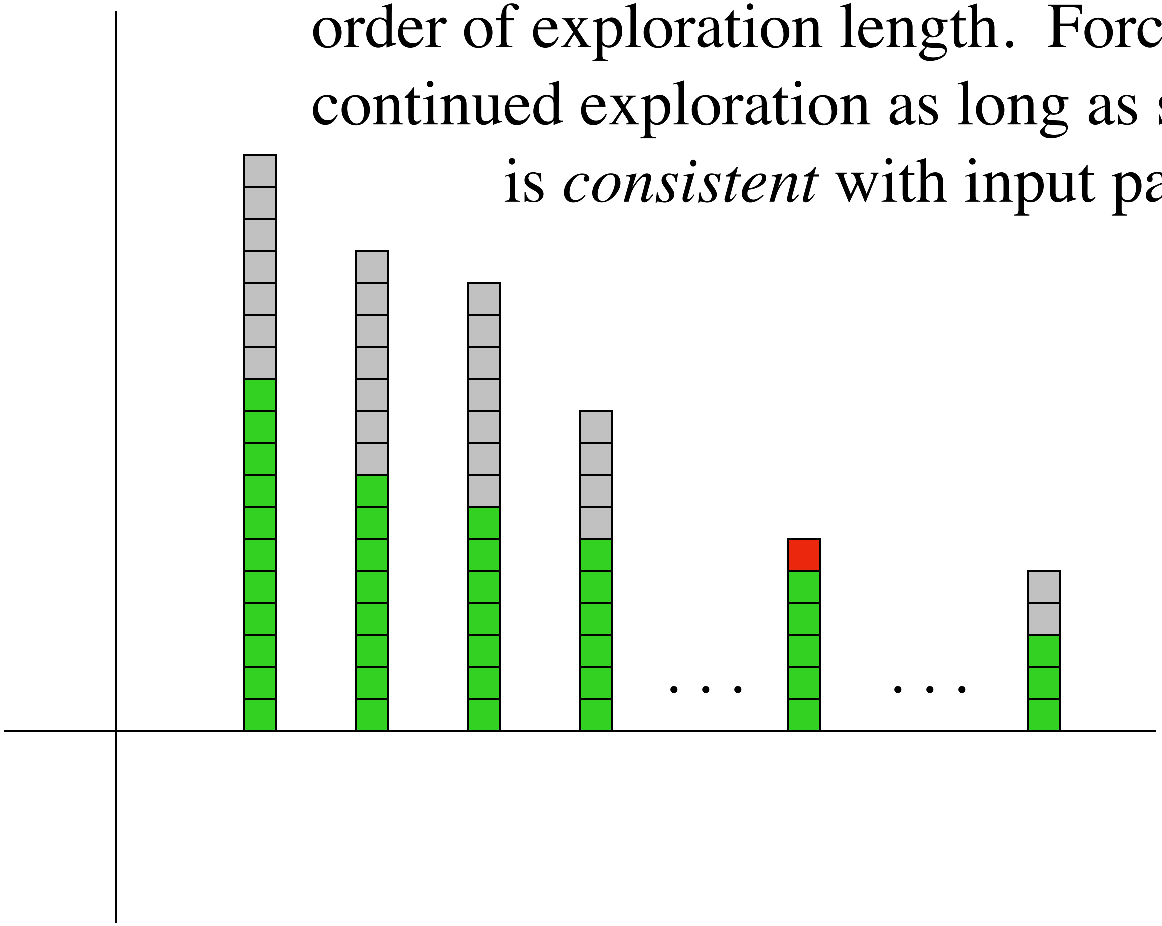
Adversary strategy: Maintain lists in order of exploration length. Force continued exploration as long as sequence is *consistent* with input pattern π.



Adversary strategy: Maintain lists in order of exploration length. Force continued exploration as long as sequence is *consistent* with input pattern π.

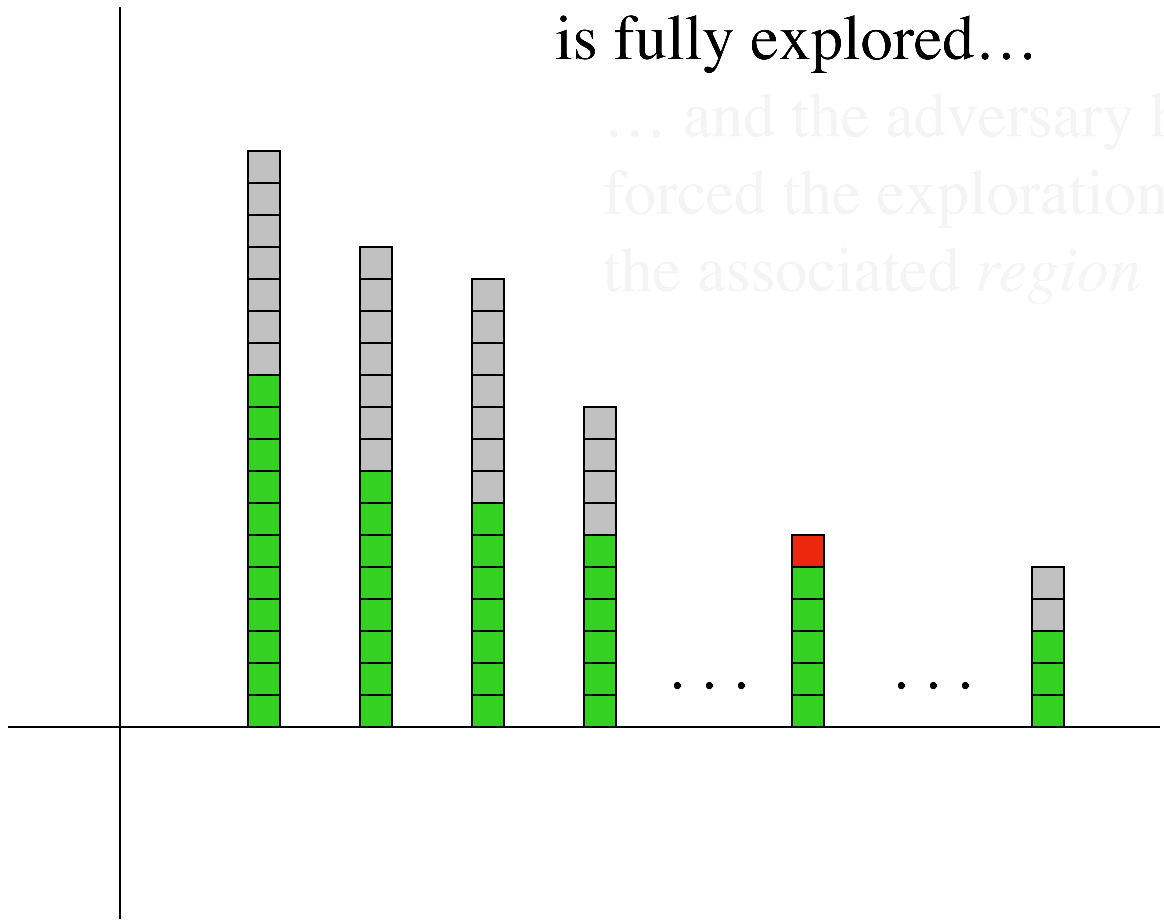


Adversary strategy: Maintain lists in order of exploration length. Force continued exploration as long as sequence is *consistent* with input pattern π.



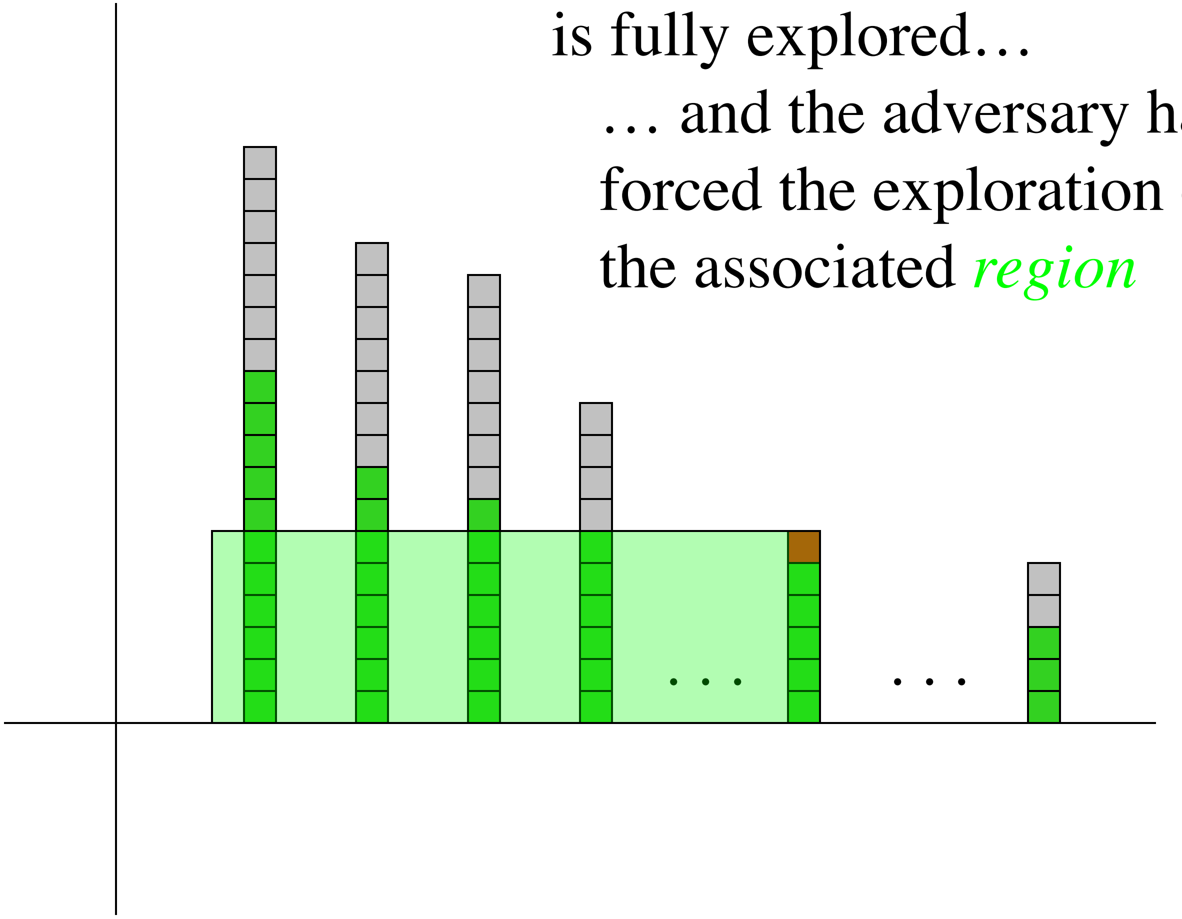
Game ends when some list is fully explored…

… and the adversary has forced the exploration of the associated *region*

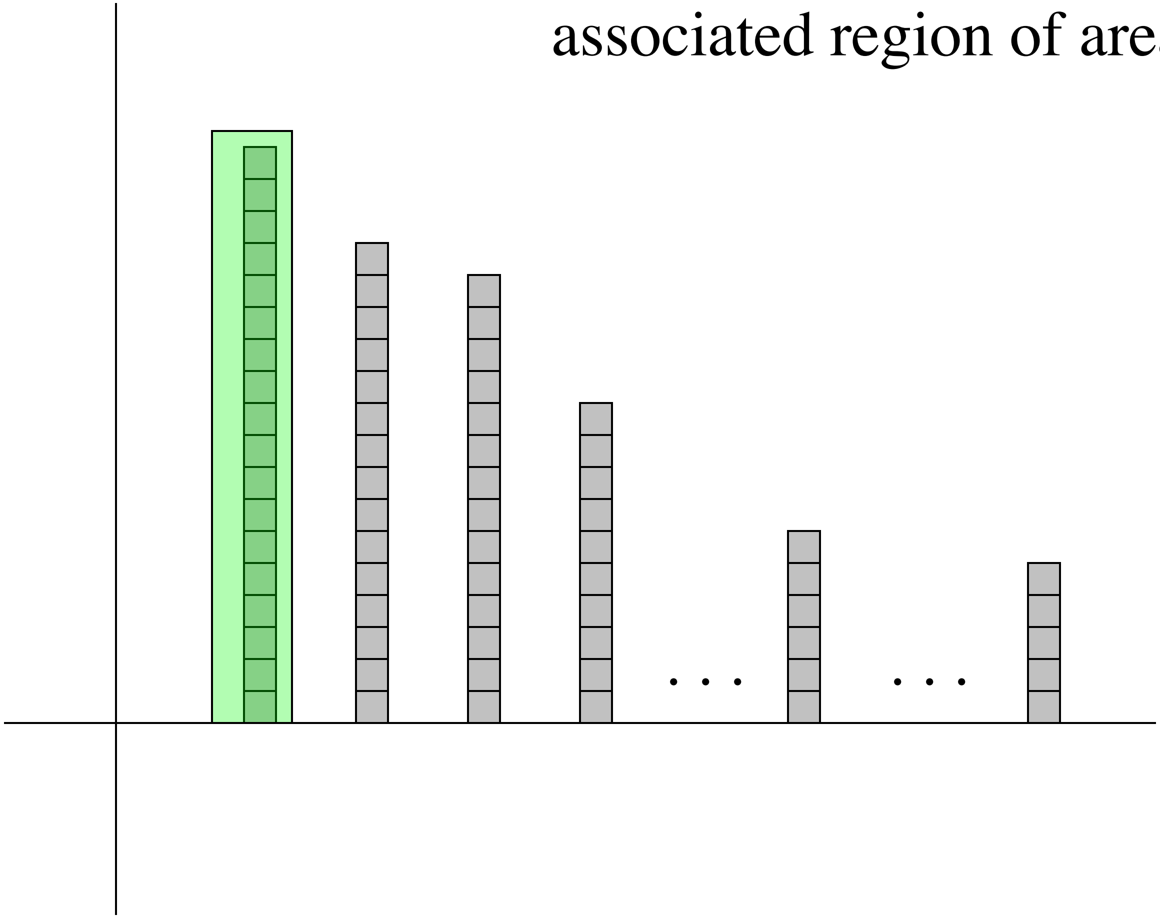


Game ends when some list is fully explored…

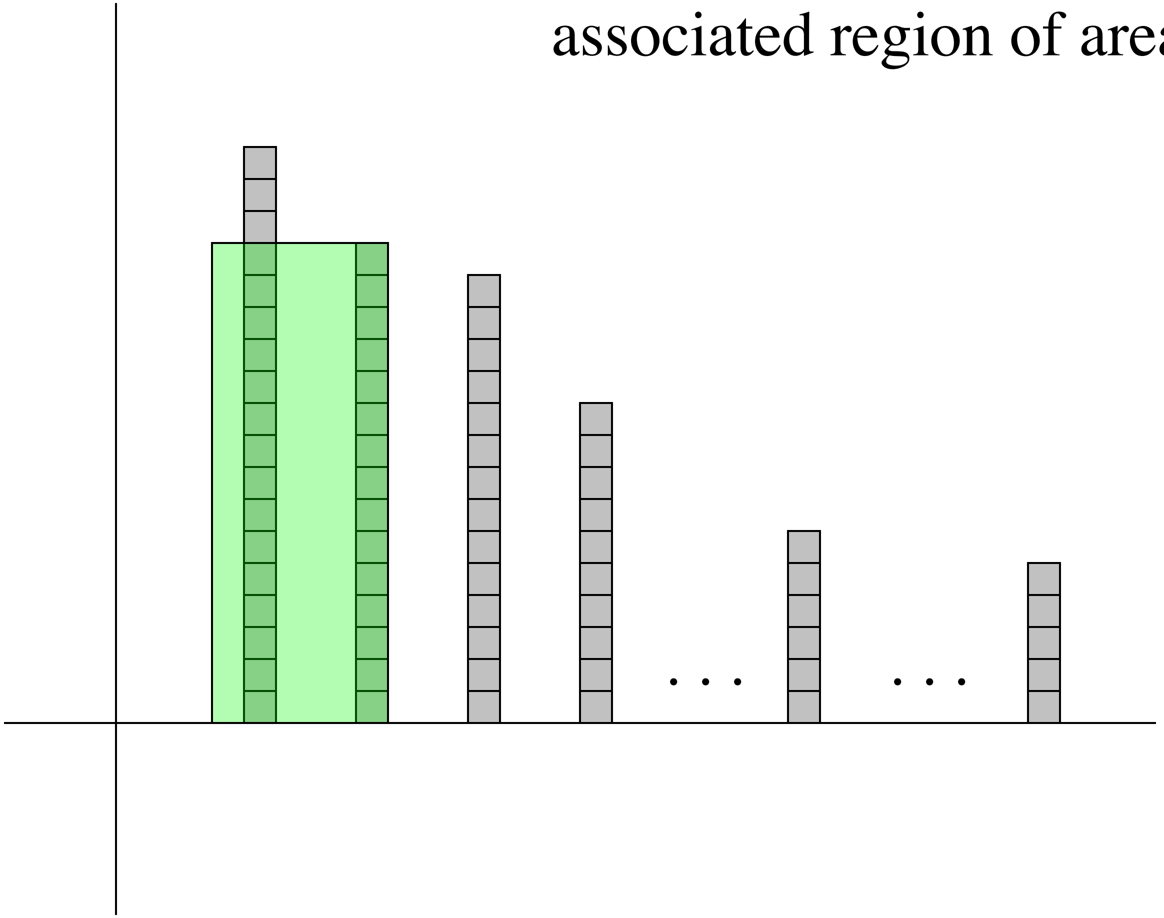
… and the adversary has forced the exploration of the associated *region*



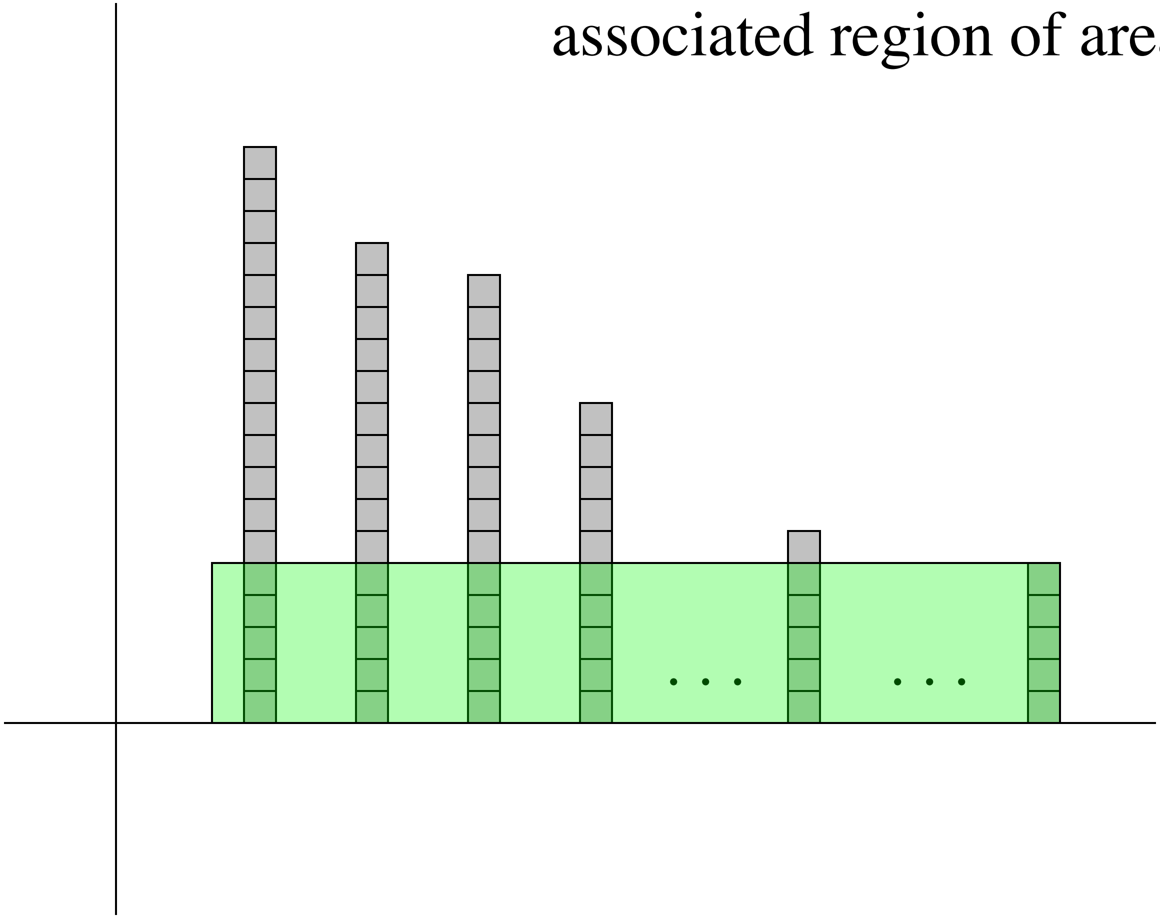
The i-th longest list has an associated region of area c_i



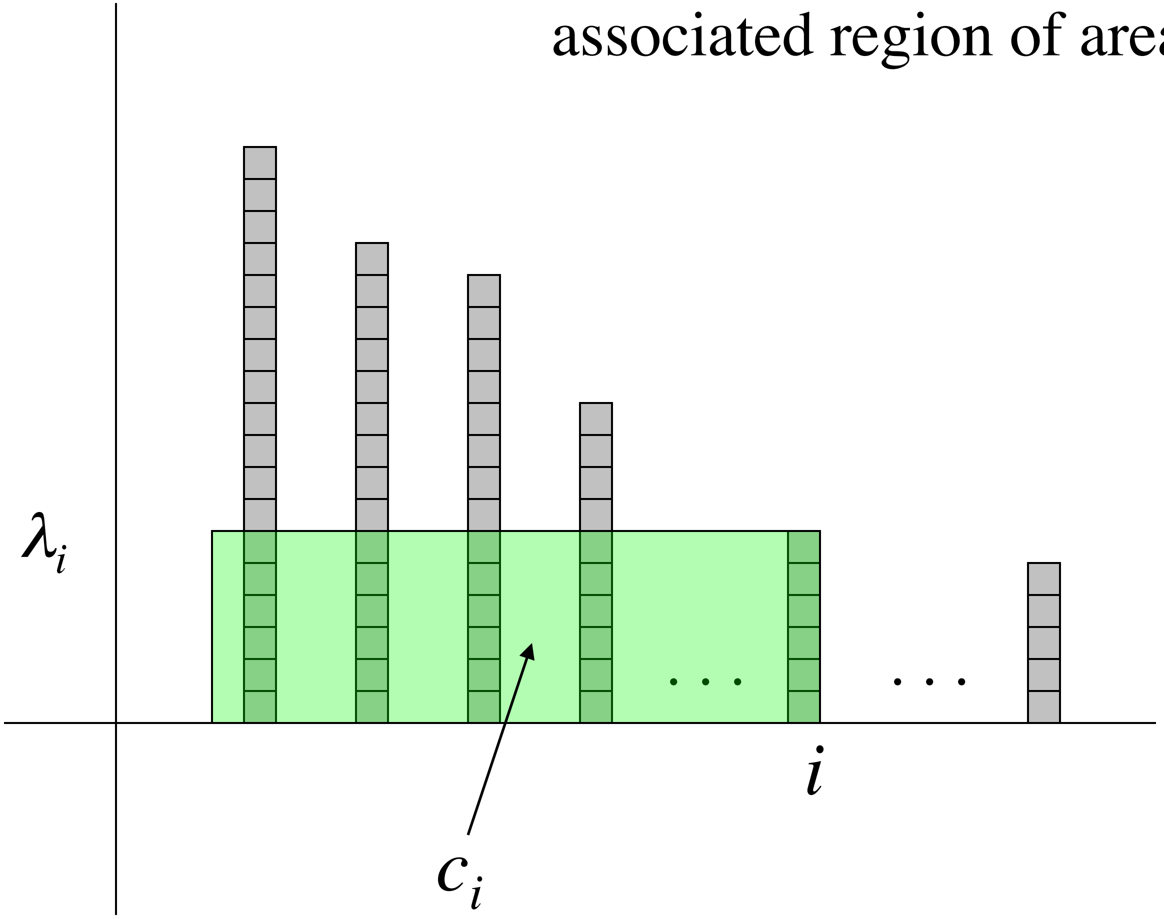
The i-th longest list has an associated region of area c_i



The i-th longest list has an associated region of area c_i



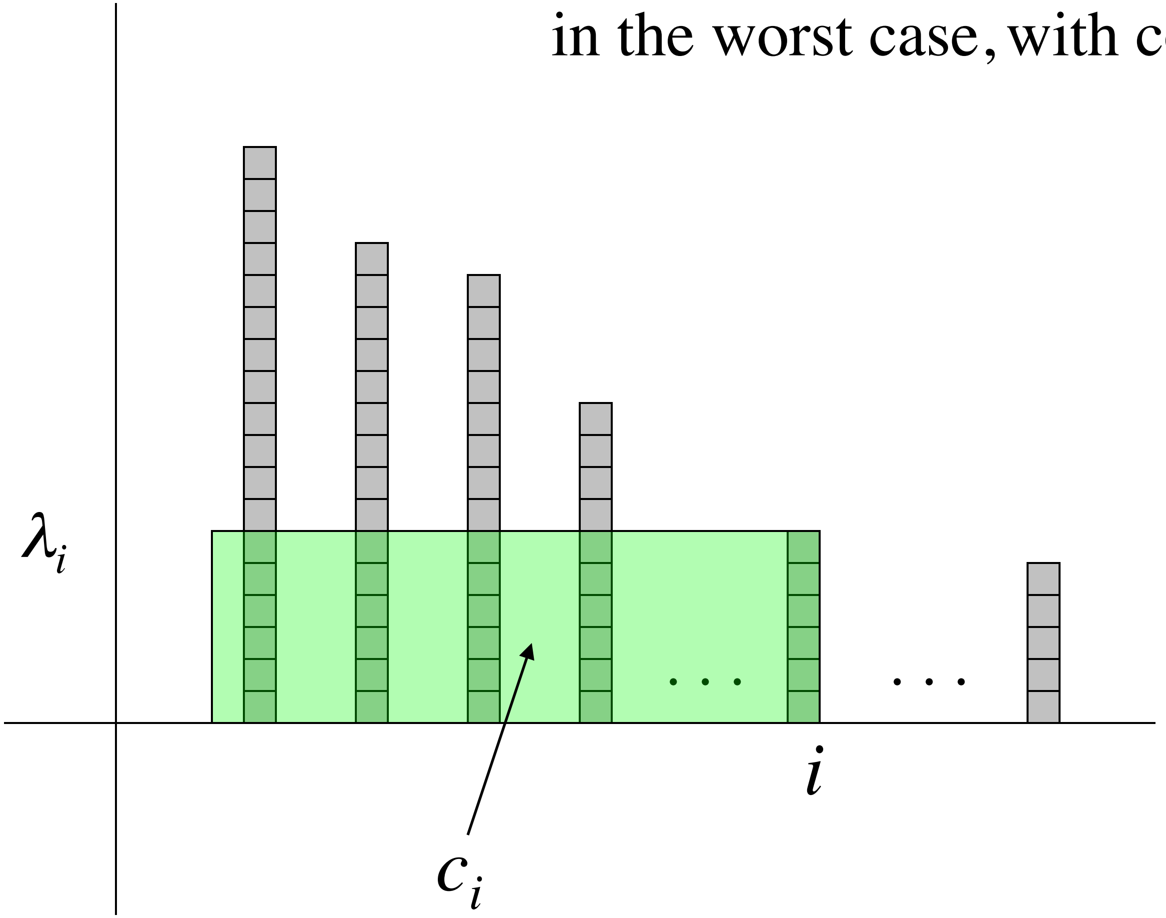
The i-th longest list has an associated region of area $c_i = i\lambda_i$



Theorem A1. Any algorithm that solves the list-exploration problem with inputs of pattern π can be forced to make $\min_i \{c_i\}$ steps, *even if the algorithm knows π.*

Theorem A1. Any algorithm that solves the list-exploration problem with inputs of pattern π can be forced to make $\min_i \{c_i\}$ steps, *even if the algorithm knows* π.

The strategy **BDFS(λ_i)** succeeds,
in the worst case, with cost $c_i = i\lambda_i$



Theorem A1. Any algorithm that solves the list-exploration problem with inputs of pattern π can be forced to make $\min_i \{c_i\}$ steps, *even if the algorithm knows* π.

So we refer to $c(\pi) = \min_i \{c_i\}$ as the ***intrinsic (worst-case) cost*** of the list-exploration problem with input pattern π.

Theorem A2. There is an algorithm that solves the list-exploration problem (with arbitrary inputs) in

$O(\ c(\pi) \ln \min \{m, c(\pi)\}\ )$ steps,

without knowing the input pattern π.

Theorem A2. There is an algorithm that solves the list-exploration problem (with arbitrary inputs) in
$O(\, c(\pi) \ln \min \{m, c(\pi)\}\,)$ steps,
without knowing the input pattern π.

Overview

- Introduction and motivation
  - Input-thrifty algorithms
- List search
- Hyperbolic dovetailing
- Applications to input-thrifty algorithms
- Extensions & generalizations

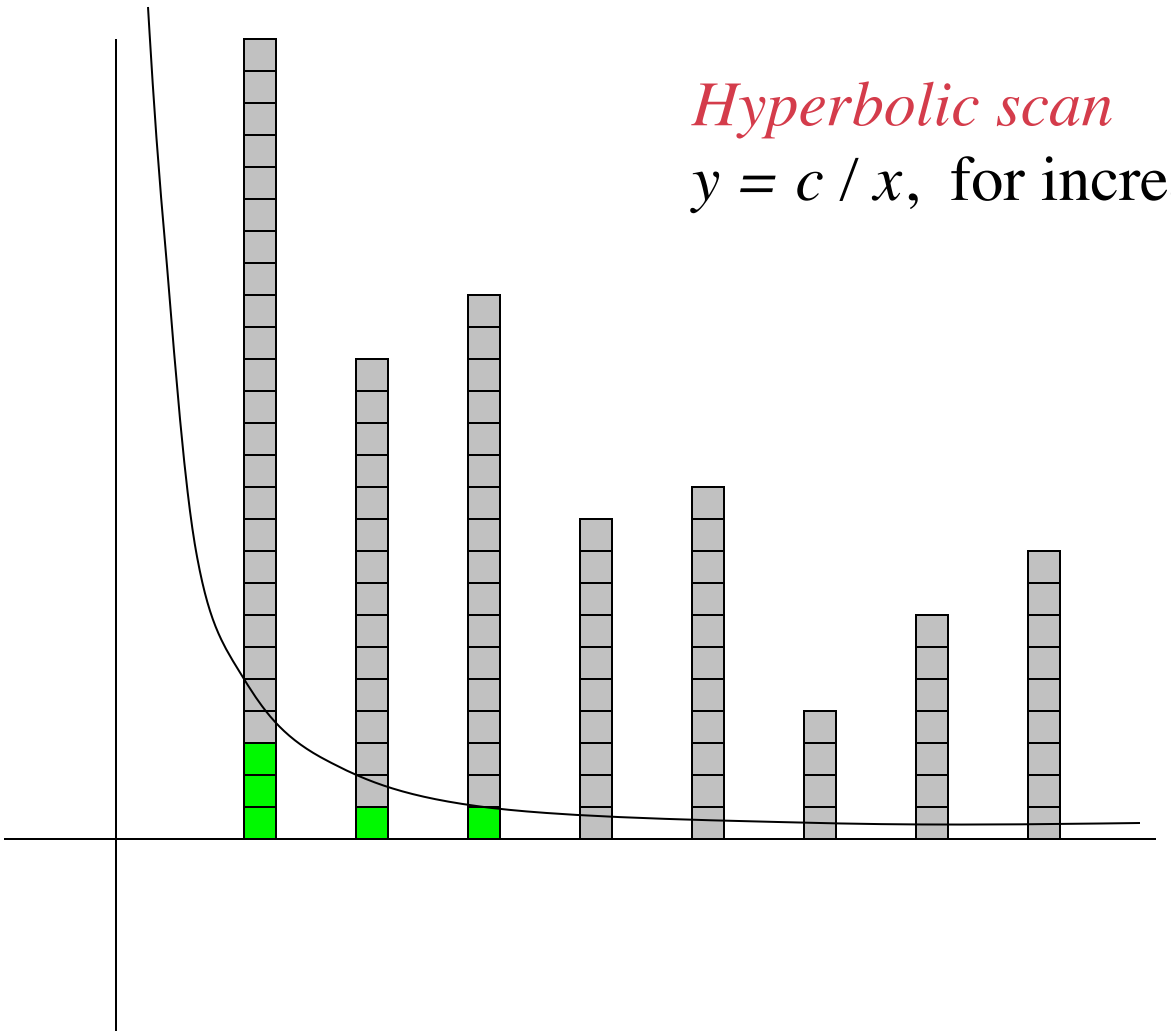
- Introduction and motivation
- List search
- Hyperbolic dovetailing
- Extensions & generalizations
- Applications to input-thrifty algorithms

Overview

```
c = 1;
repeat until some list end is reached
    for i = 1 to m
        continue exploration of list i
         up to position c / i
    increment c
```

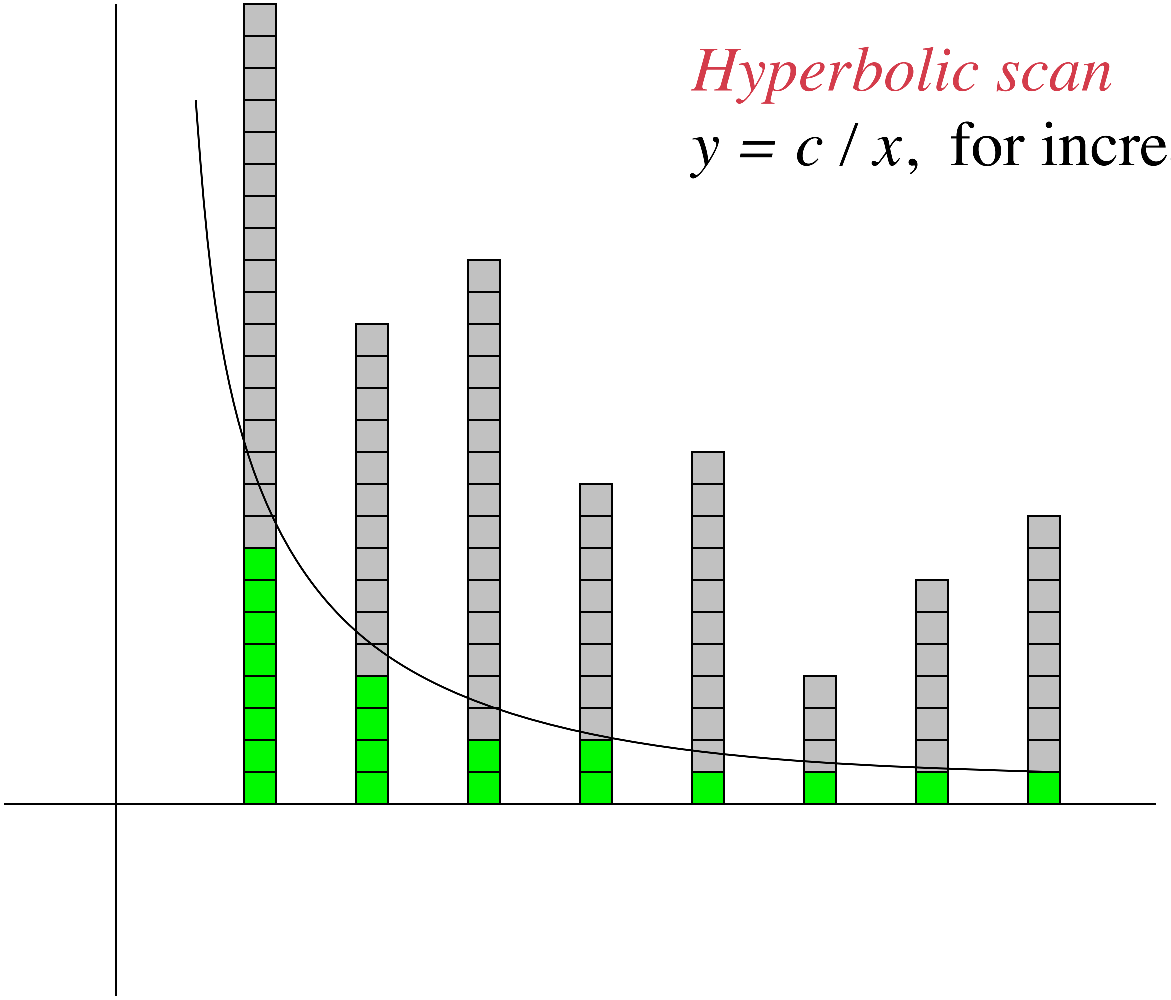
Hyperbolic scan

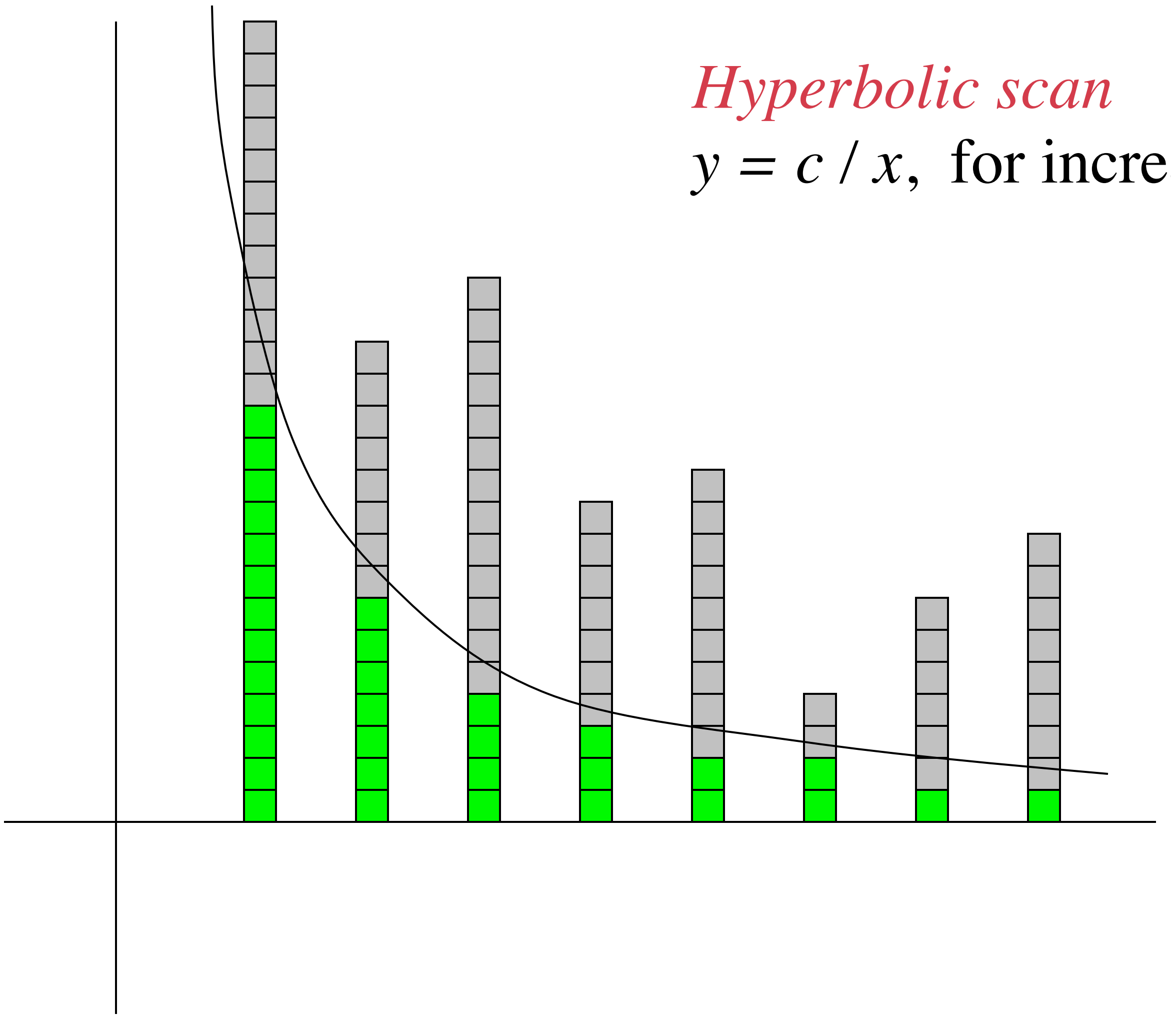
$y = c / x$, for increasing c



Hyperbolic scan

$y = c\,/\,x$, for increasing c



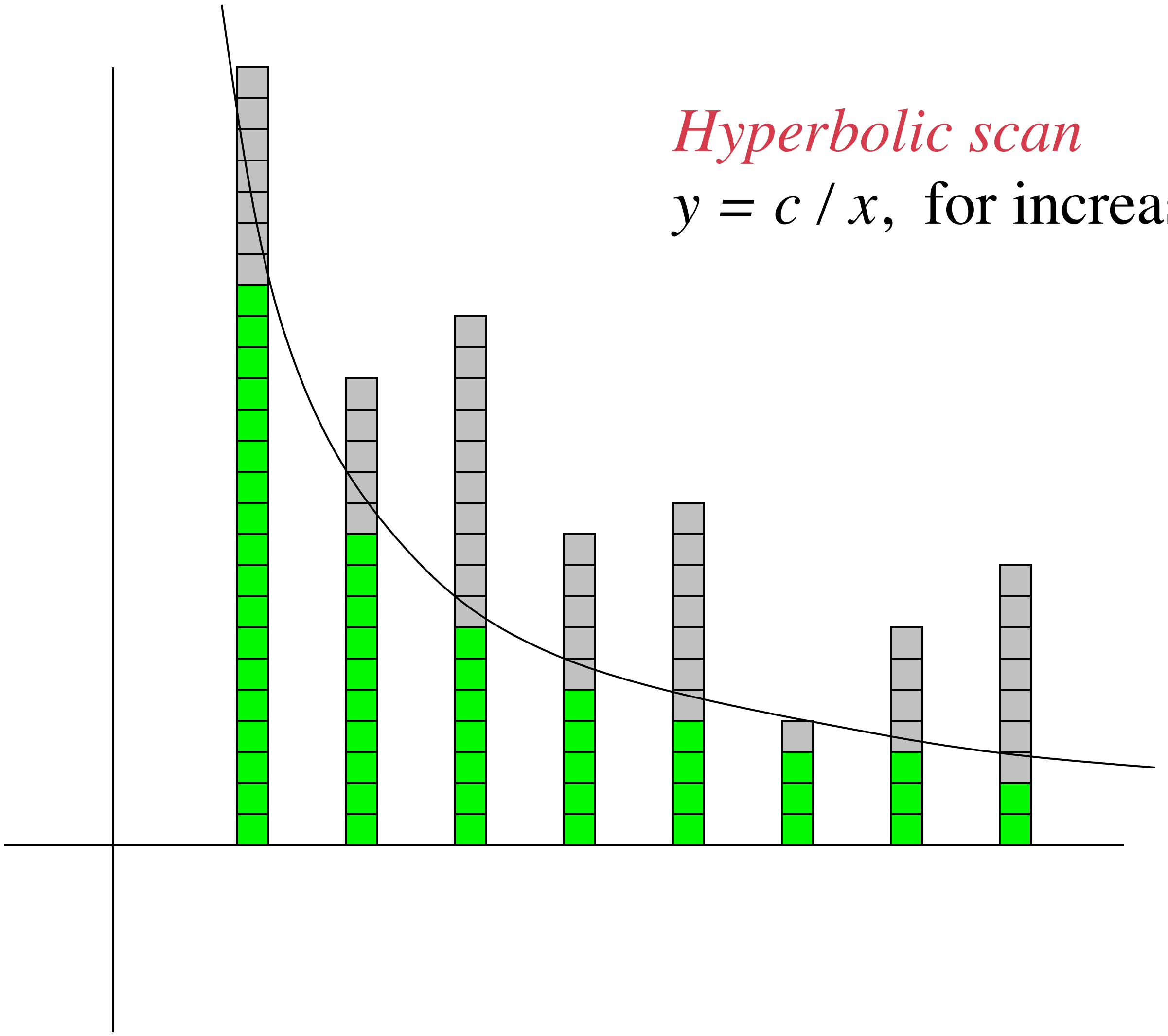


Hyperbolic scan

$y = c\ /\ x$, for increasing c

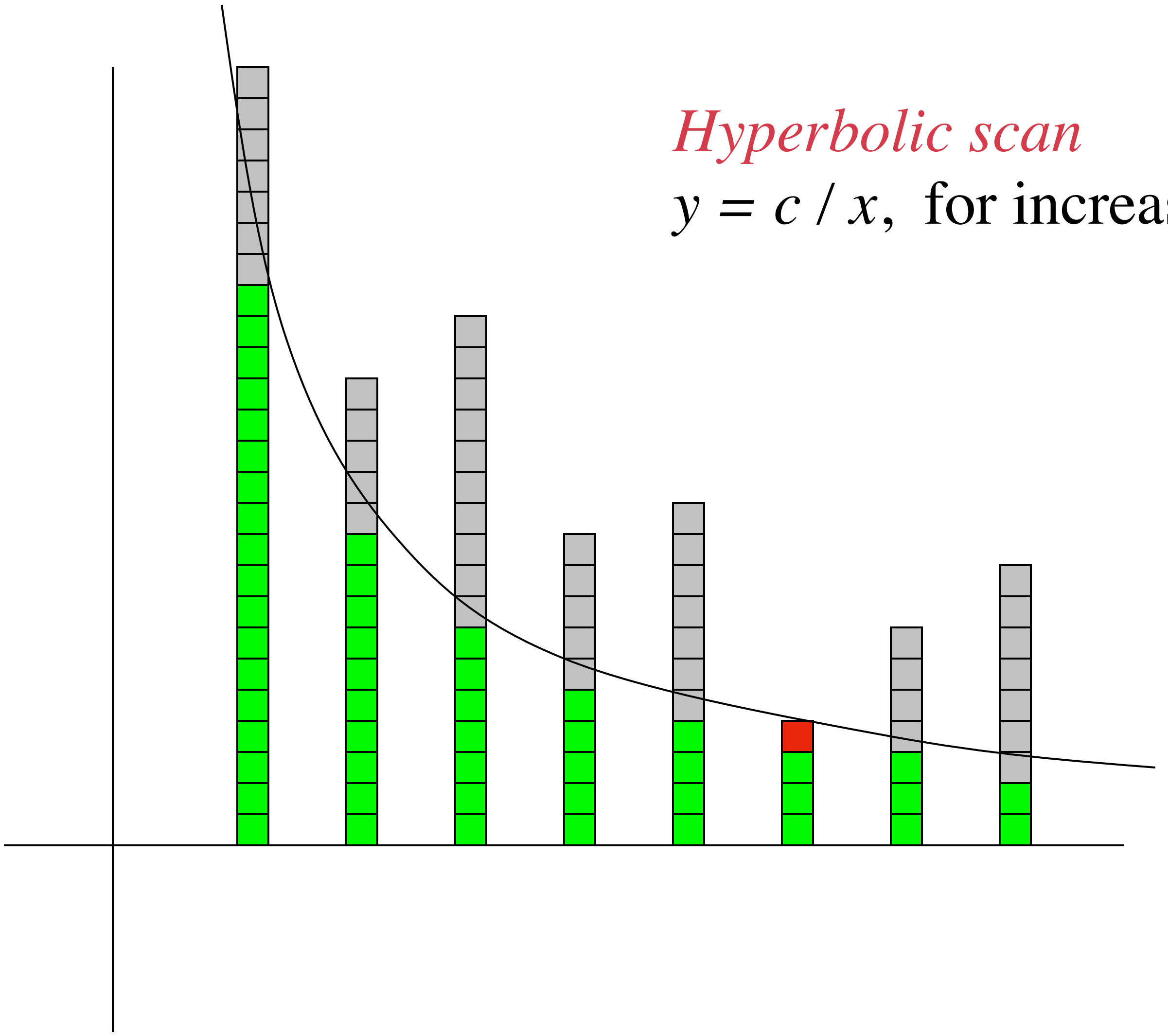
Hyperbolic scan

$y = c / x$, for increasing c

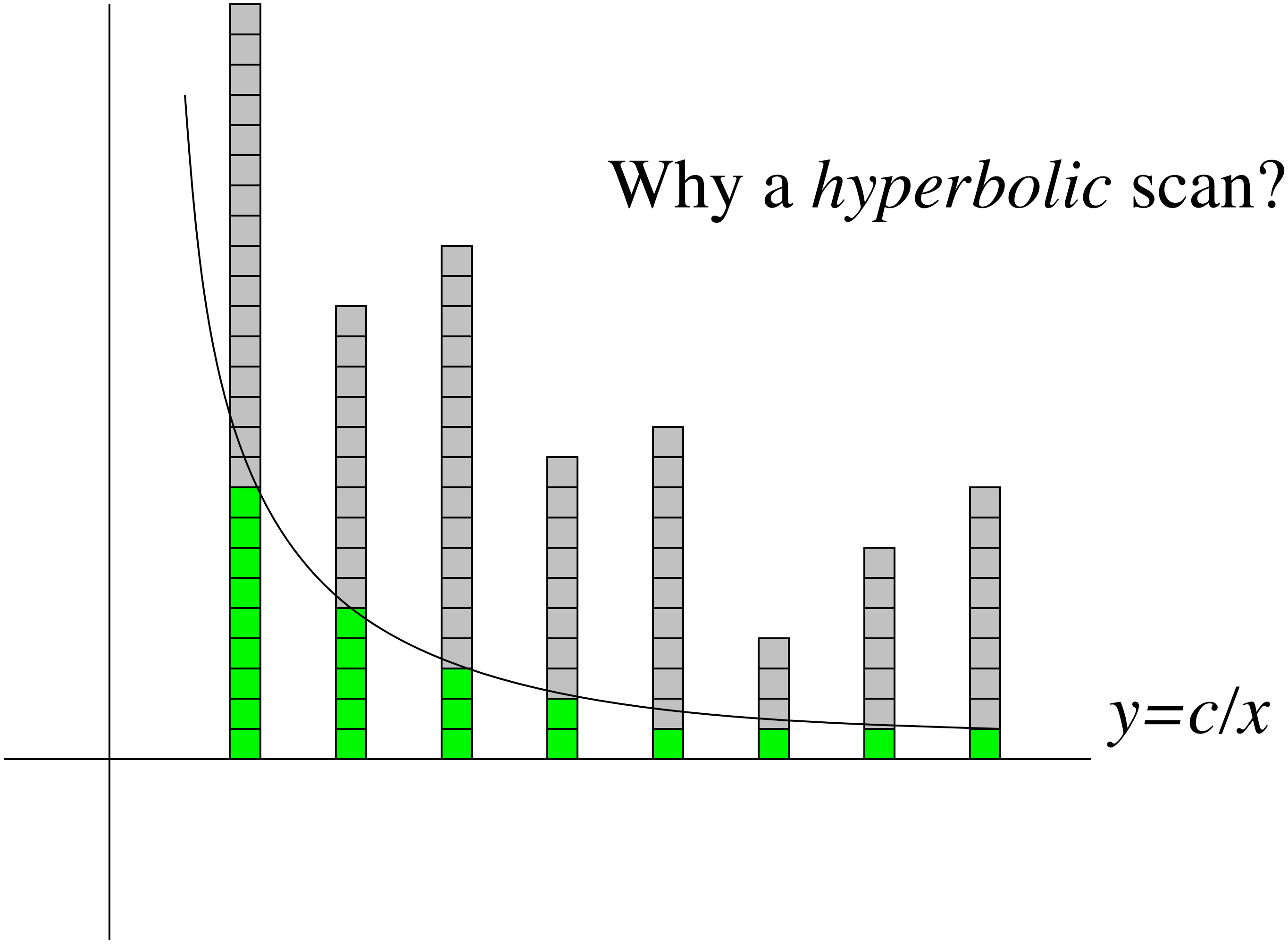


Hyperbolic scan

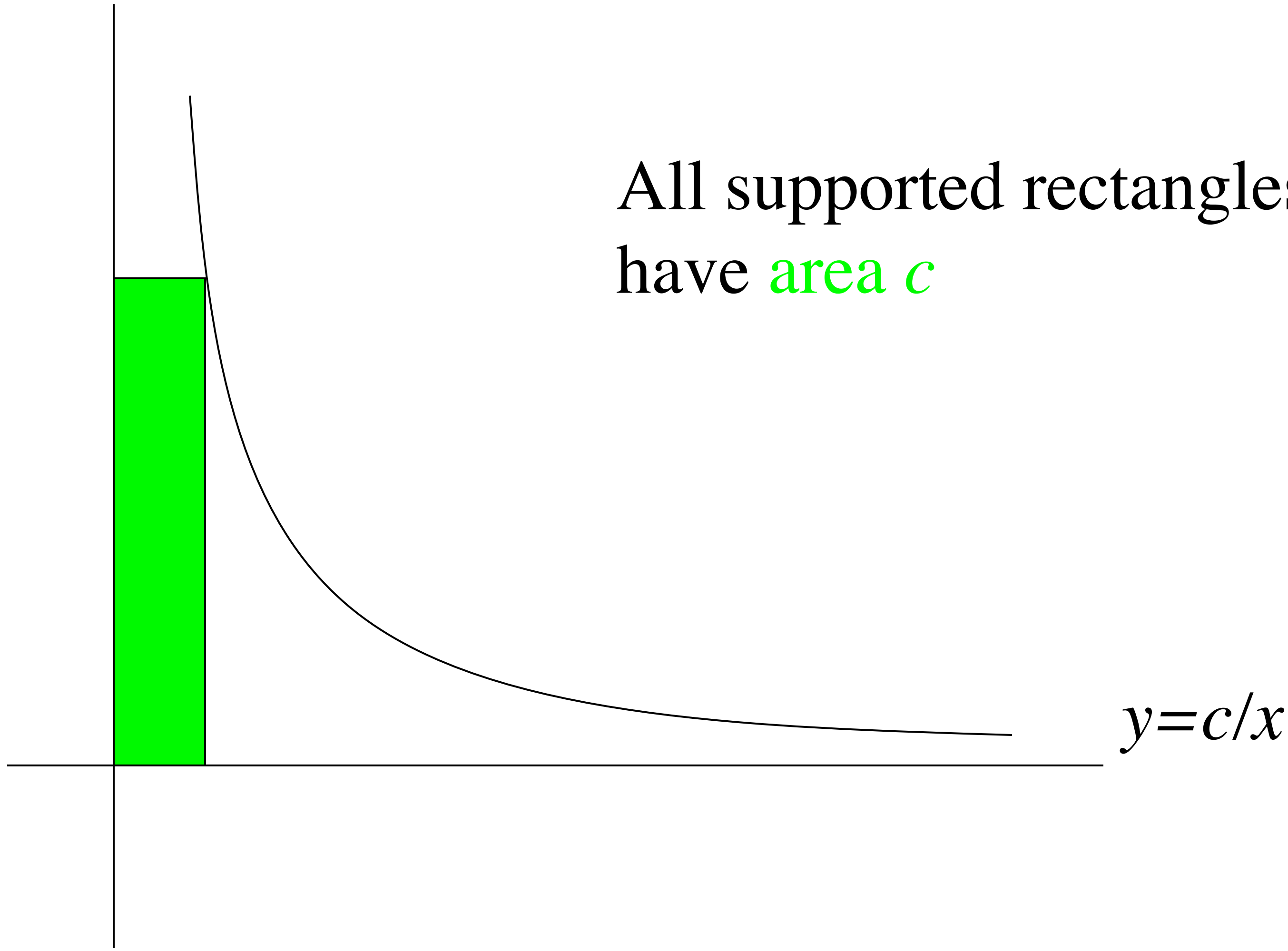
$y = c\ /\ x$, for increasing c



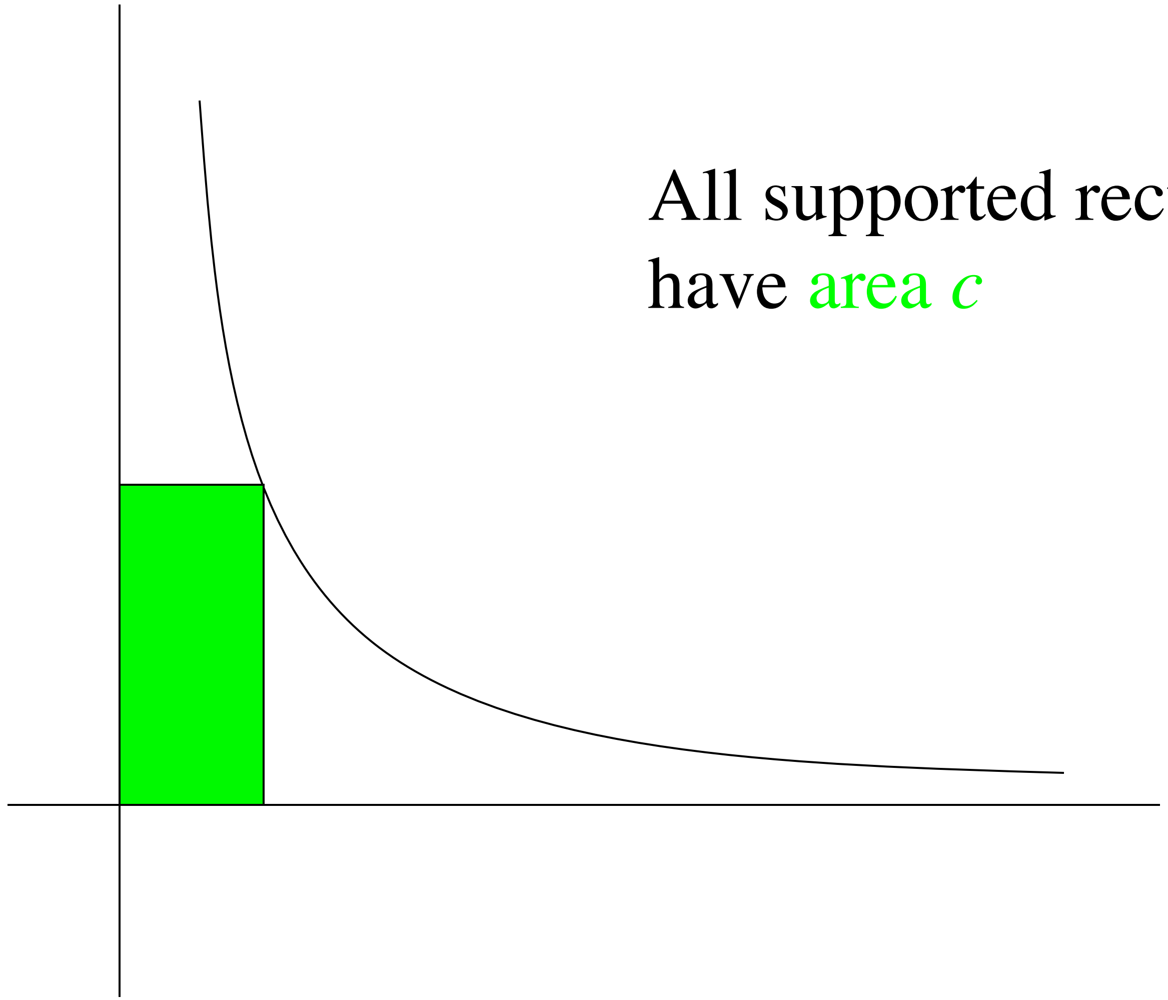
Why a *hyperbolic* scan?



All supported rectangles have area c

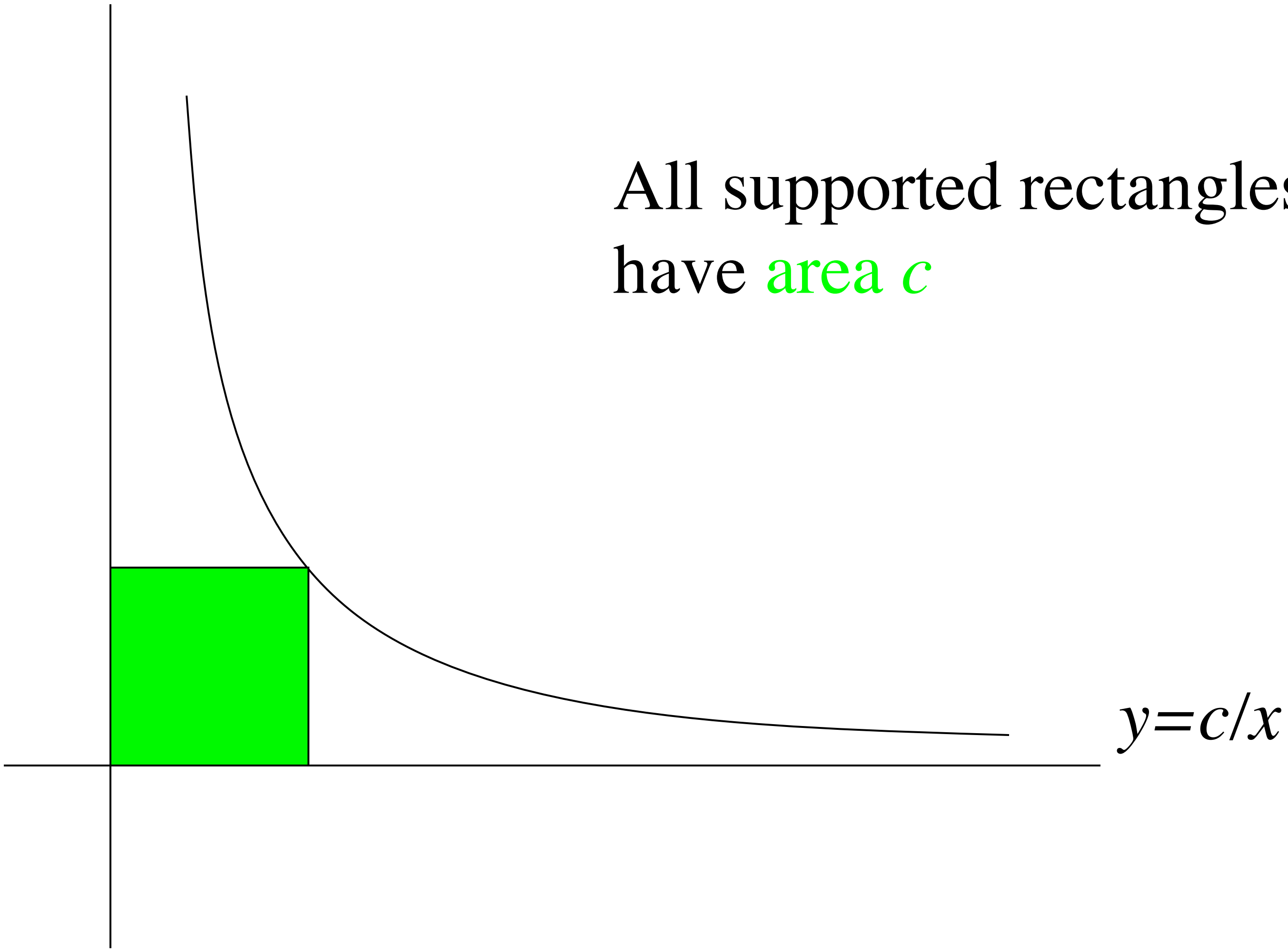


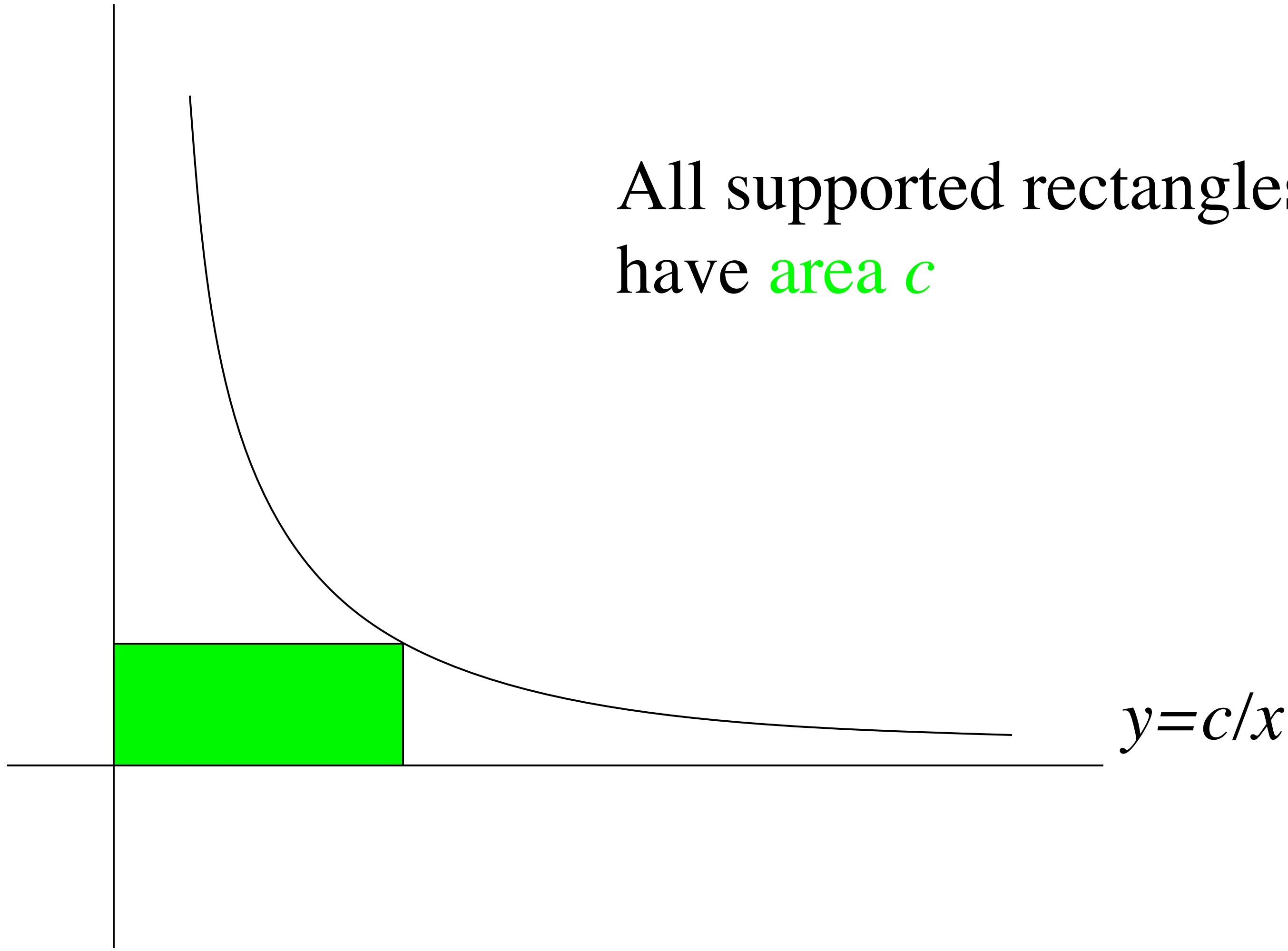
All supported rectangles have area c



$y=c/x$

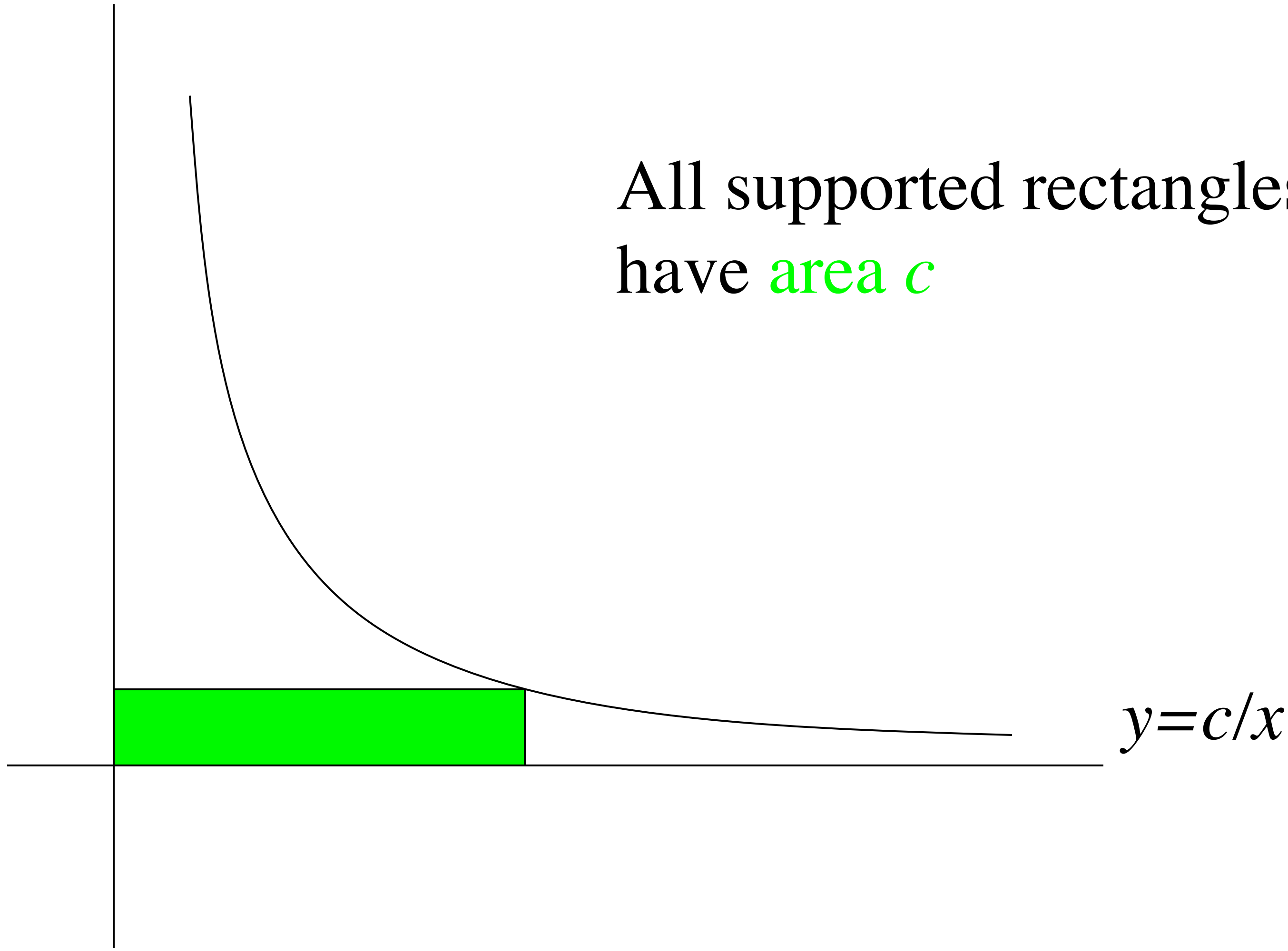
All supported rectangles have area c

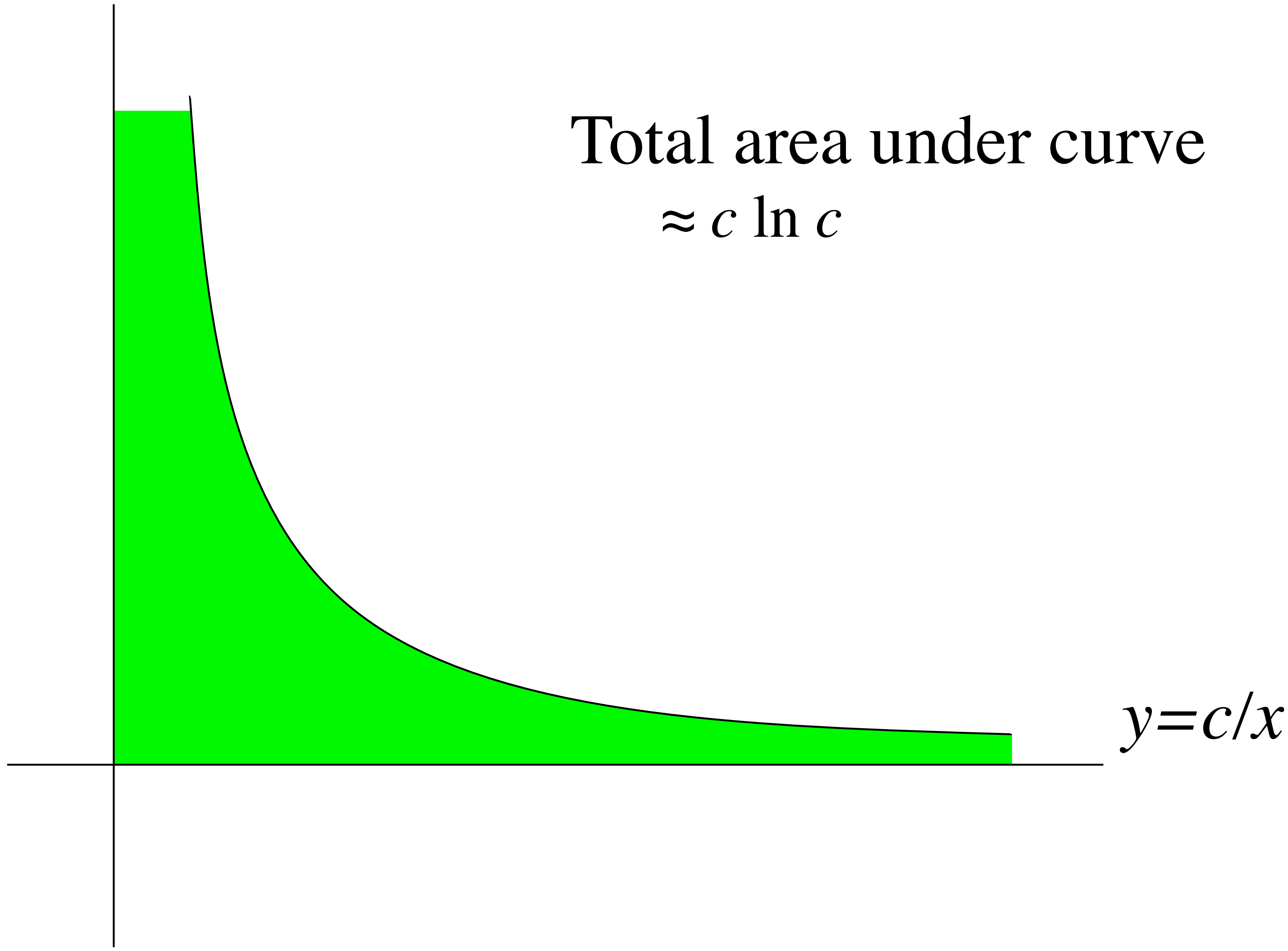




All supported rectangles have area c

All supported rectangles have area c



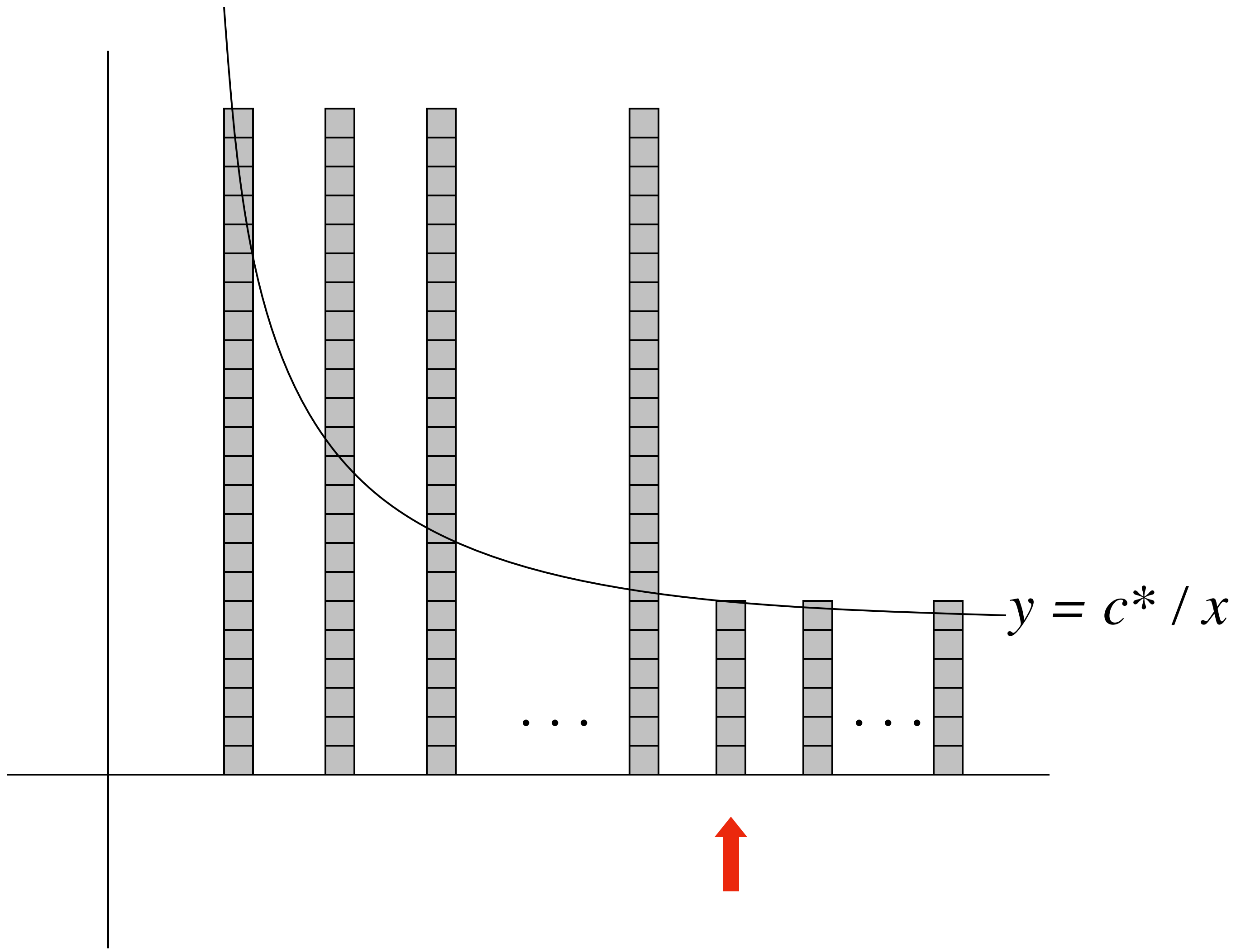

Total area under curve
≈ c ln c
y=c/x

Theorem A2. There is an algorithm that solves the list-exploration problem (with arbitrary inputs) in
$O(\, c(\pi) \ln \min\{m, c(\pi)\}\,)$ steps,
without knowing the input pattern π.

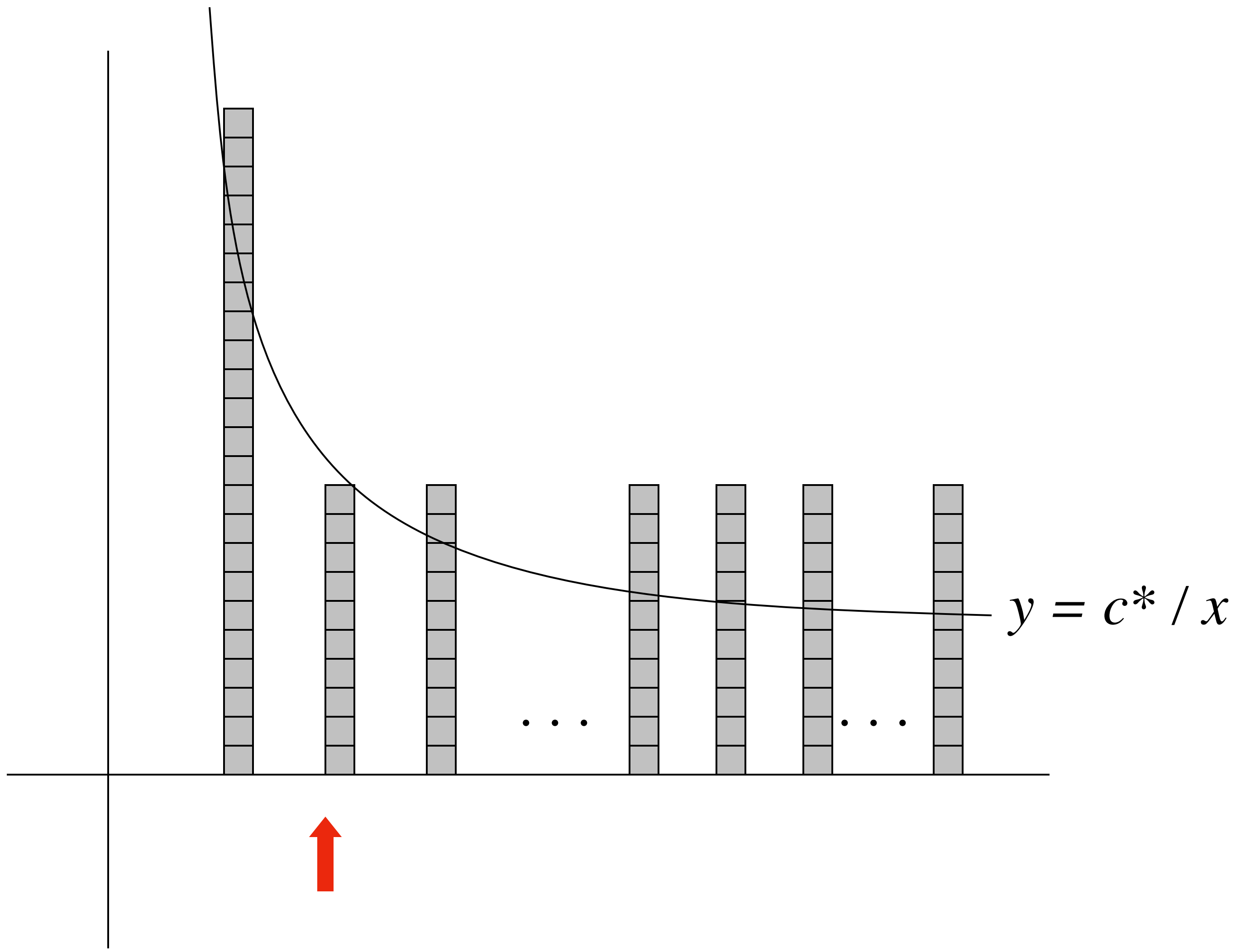
Theorem A2. There is an algorithm that solves the list-exploration problem (with arbitrary inputs) in
$O(\ c(\pi)\ \ln \min\{m, c(\pi)\}\ )$ steps,
without knowing the input pattern π.

Can we do better?

Theorem A3. Any algorithm that solves the list-exploration problem can be forced to make $\Omega\,(c^*\ \ln c^*\ )$ steps,
even if the algorithm knows that the input pattern π *satisfies* $c(\pi) = c^*$



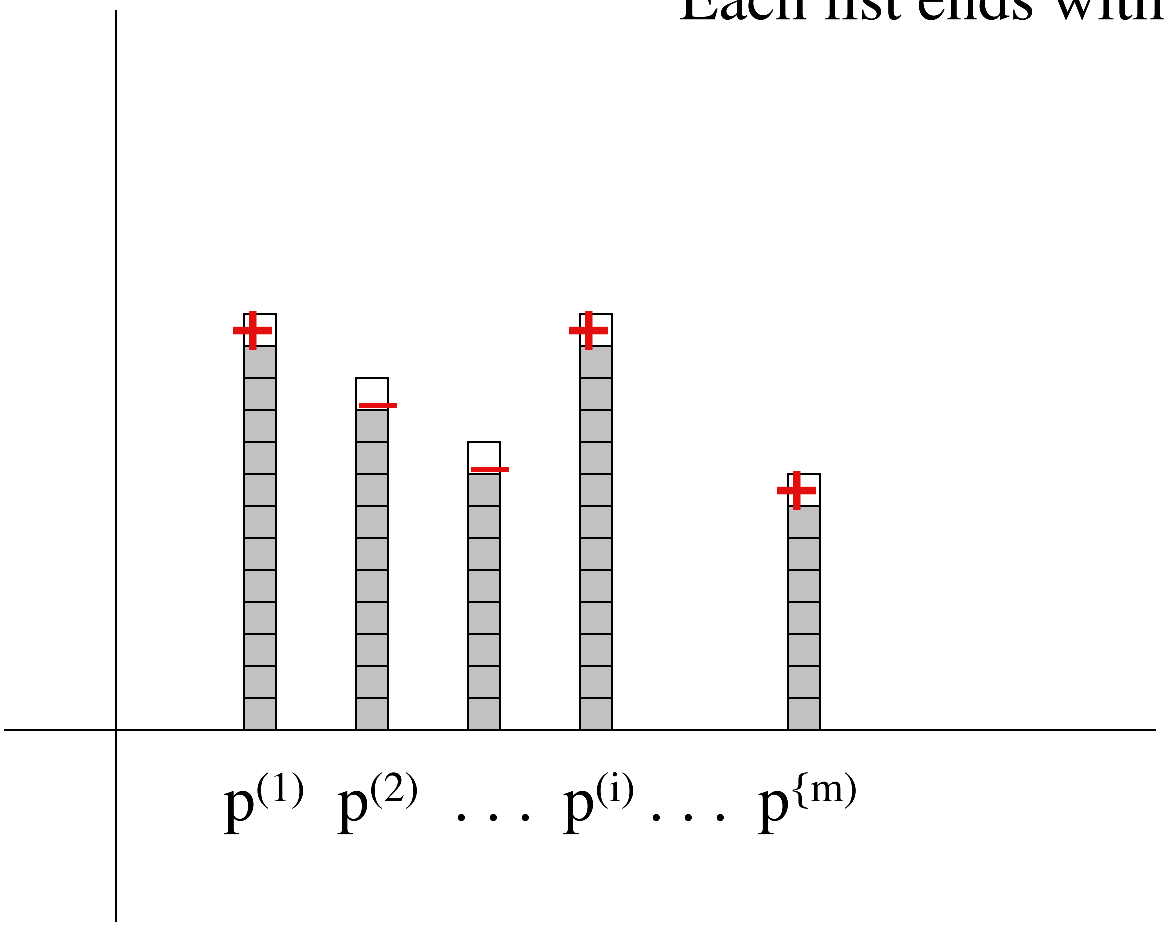
y = c* / x


$y = c^* / x$

Overview

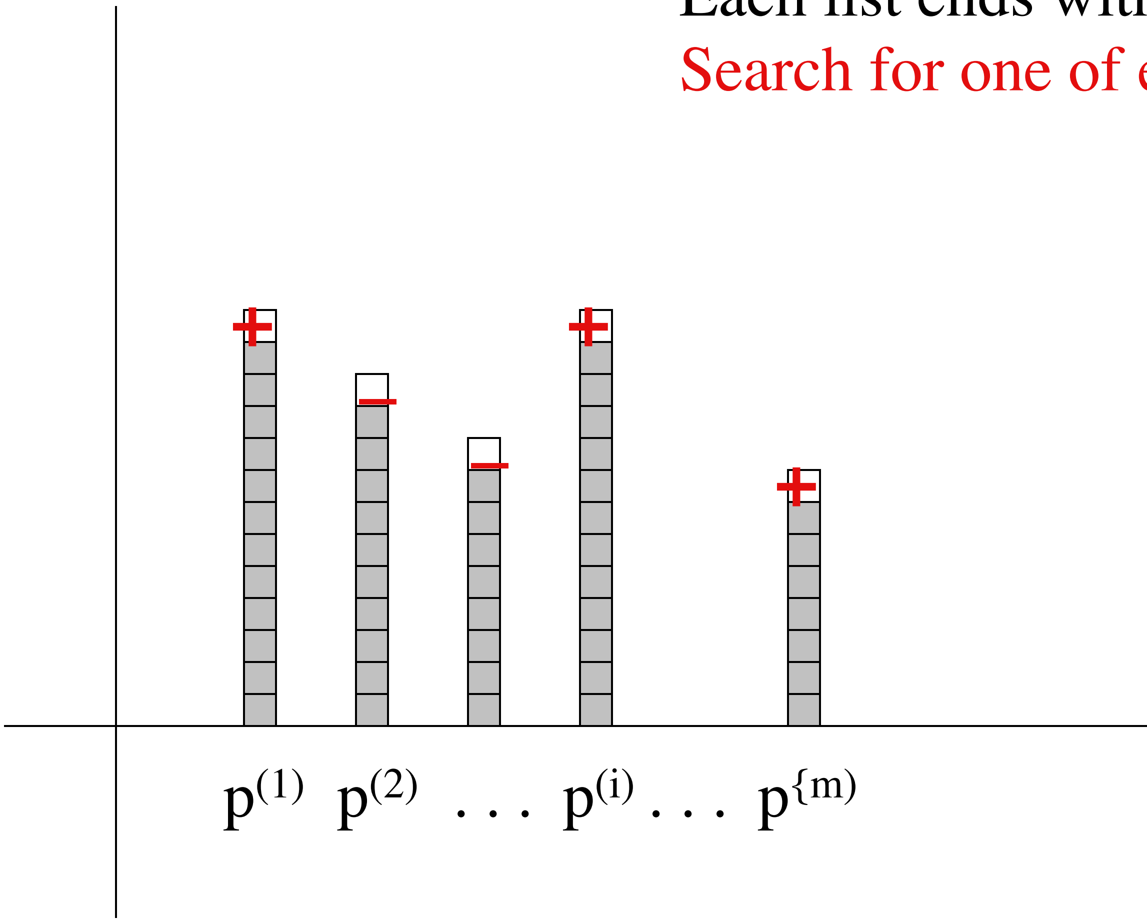
- Introduction and motivation
  - Input-thrifty algorithms
- List search
- Hyperbolic dovetailing
- **Applications to input-thrifty algorithms**
- Extensions & generalizations

Each list ends with a sign

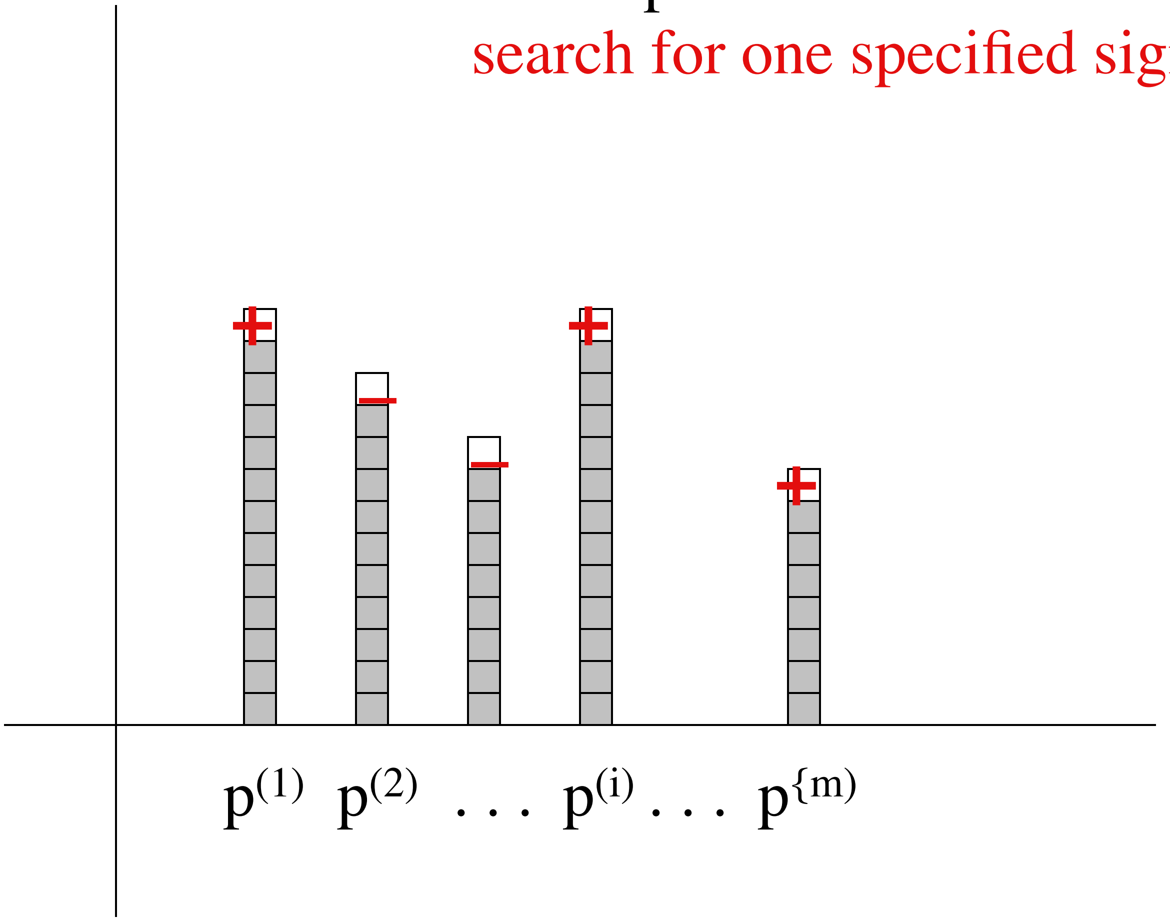


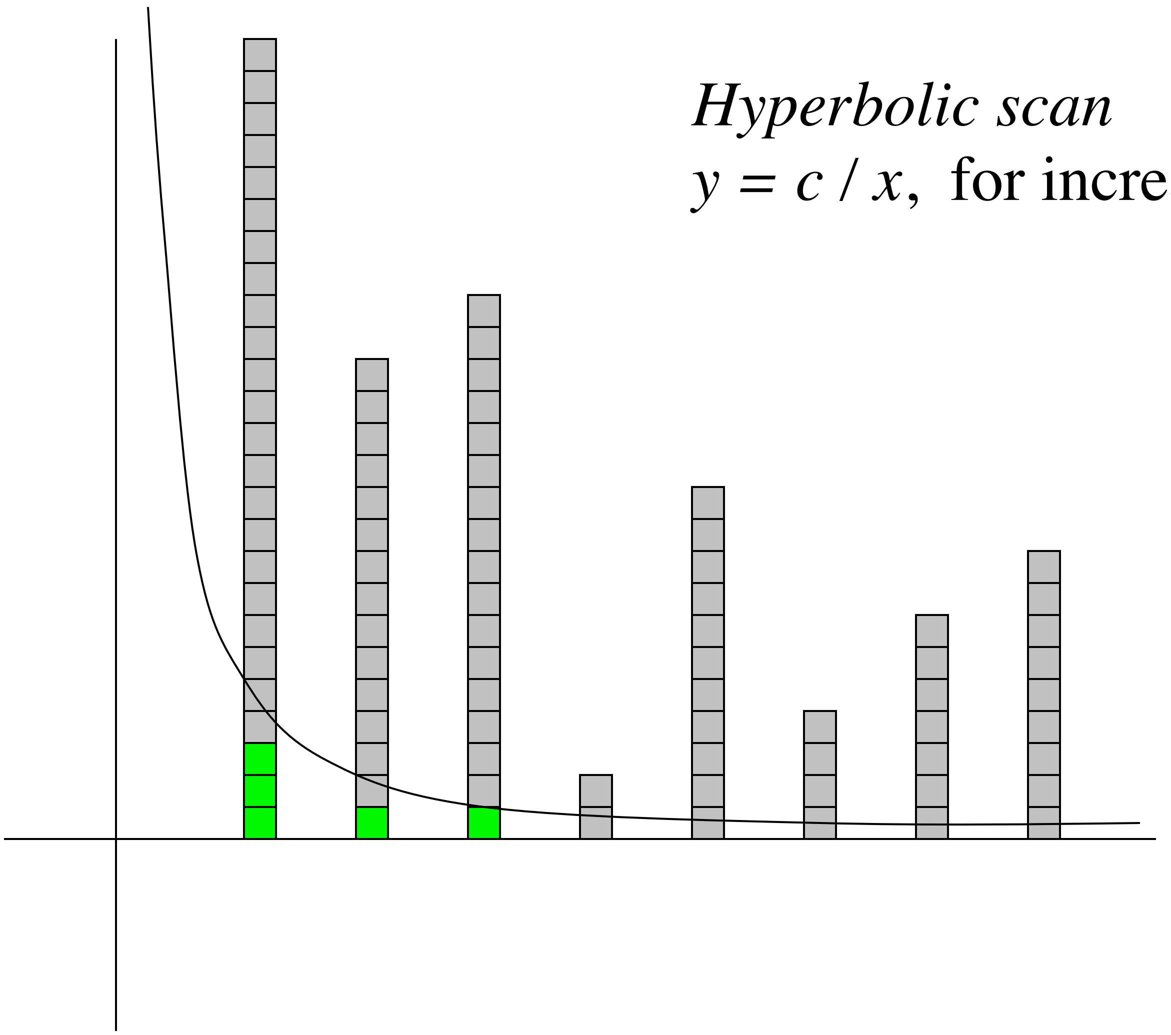
Each list ends with a sign

Search for one of each type



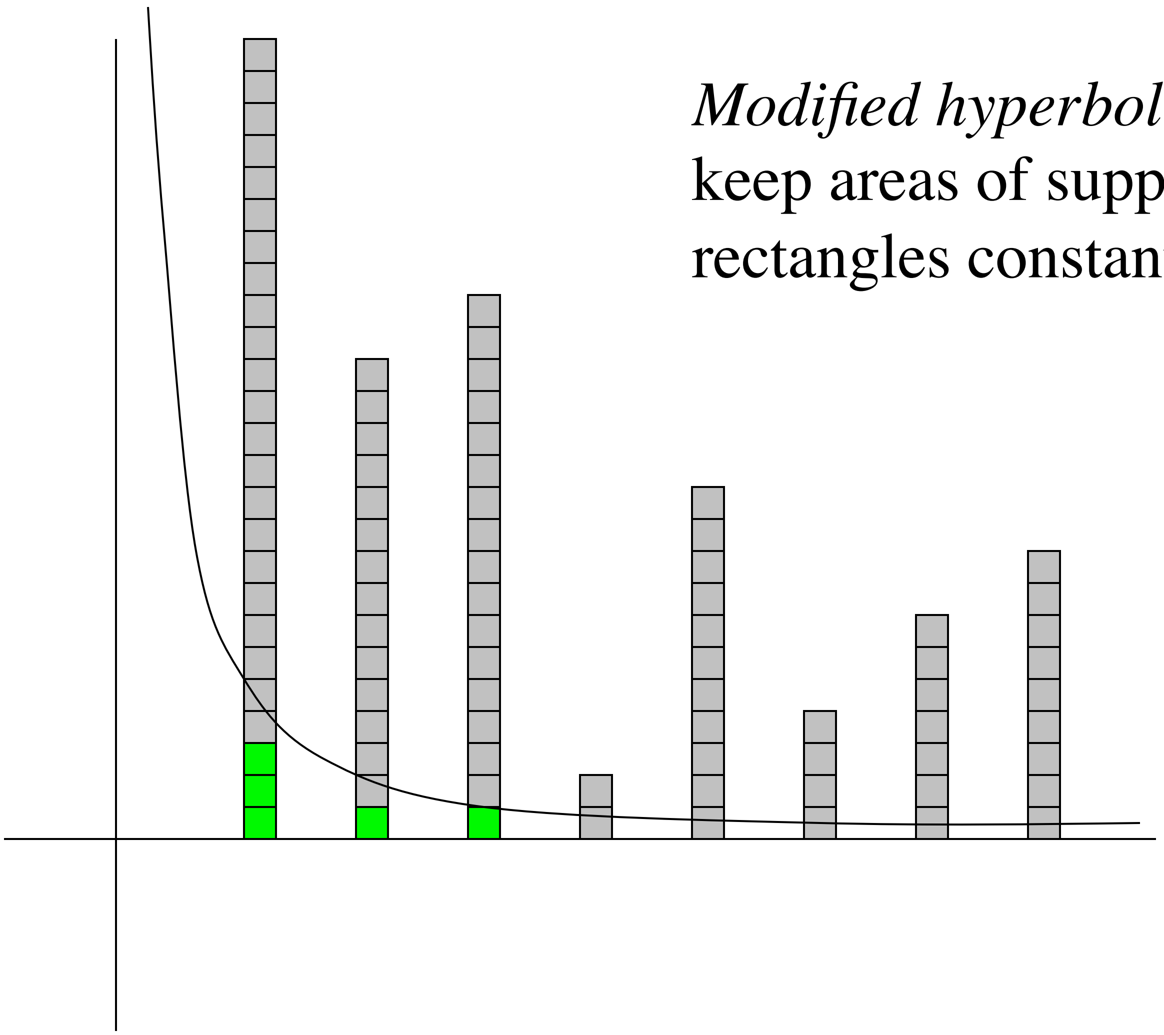
Given previous solution it suffices to search for one specified sign (+)





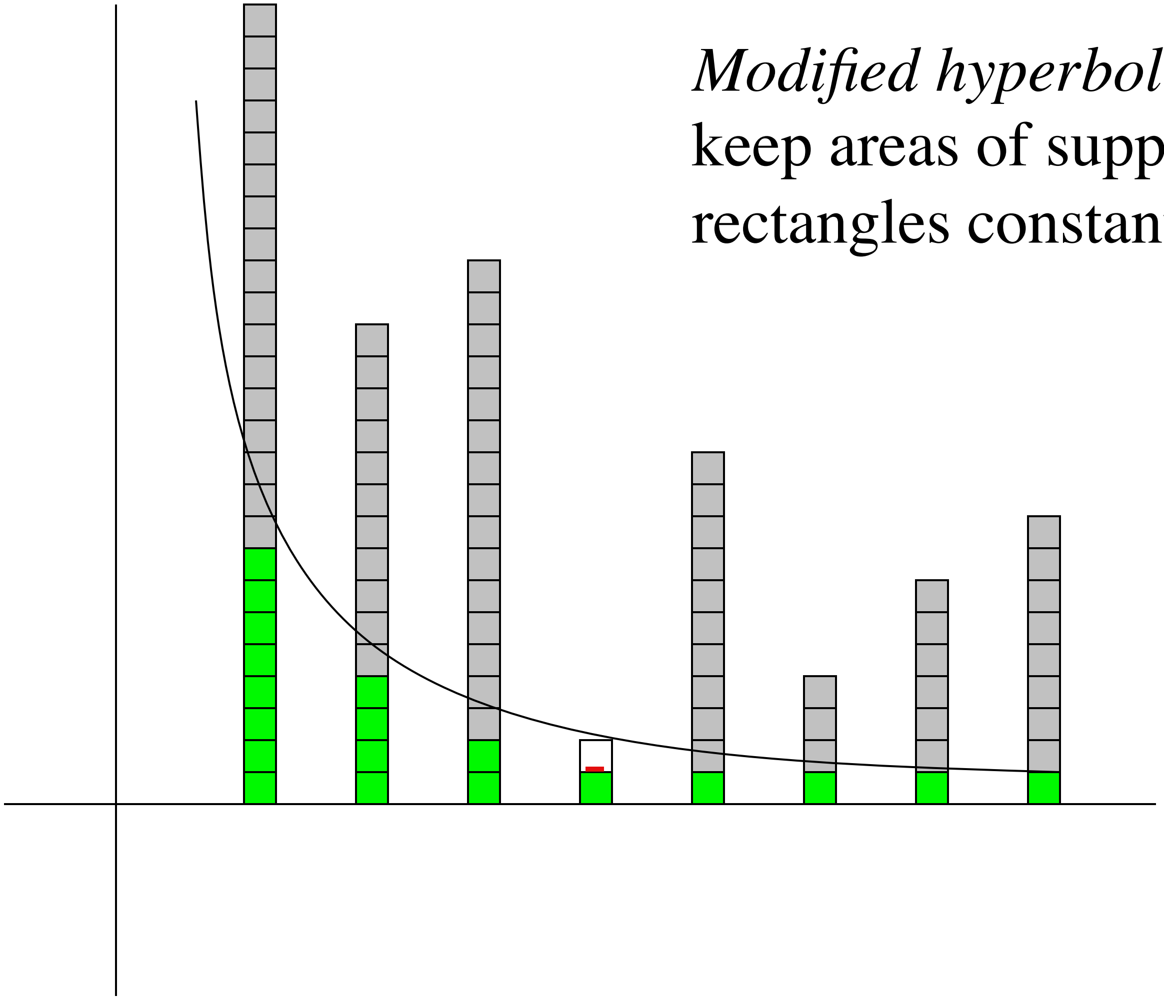
Hyperbolic scan

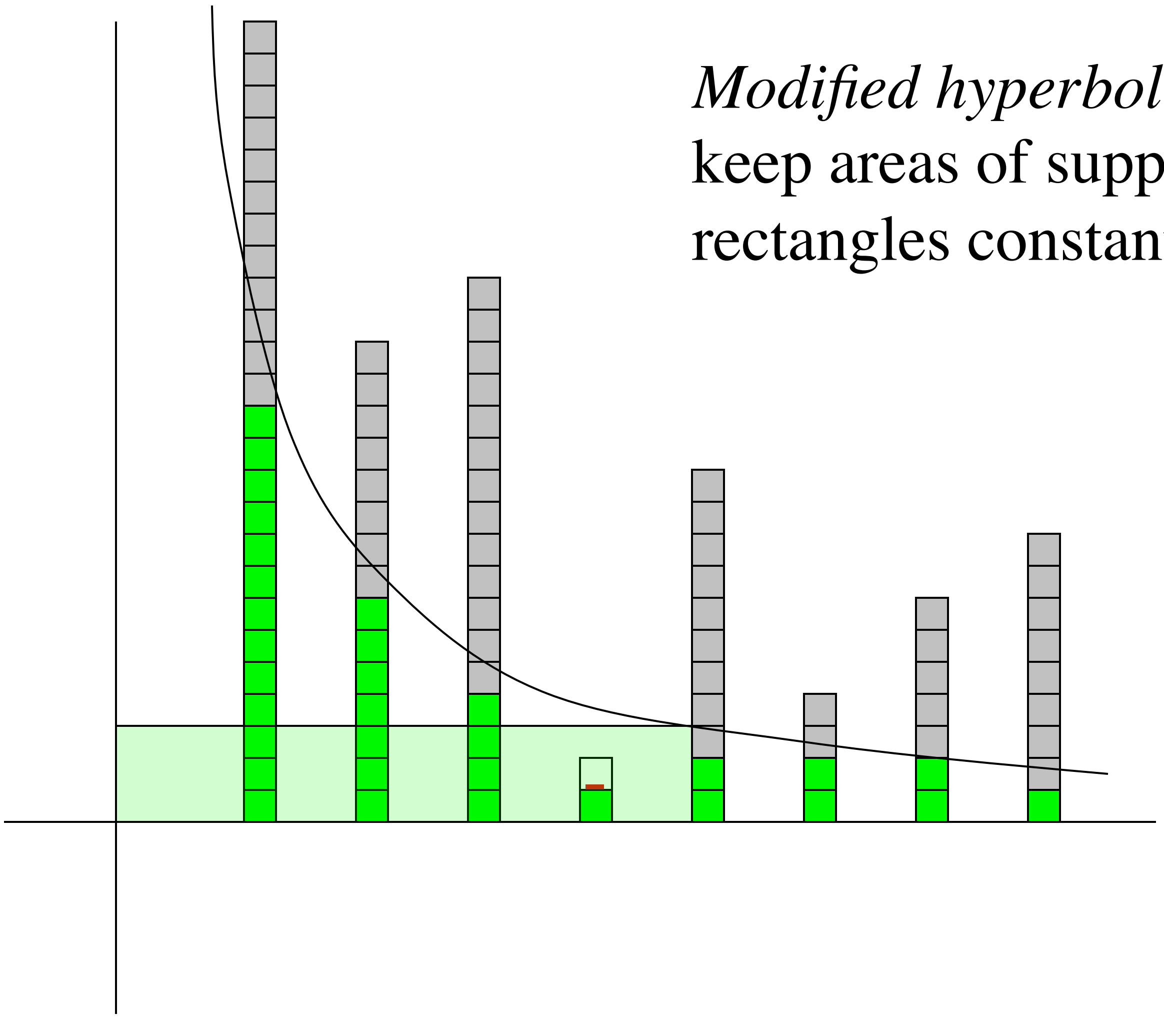
$y = c / x$, for increasing c



Modified hyperbolic scan:
keep areas of supported
rectangles constant

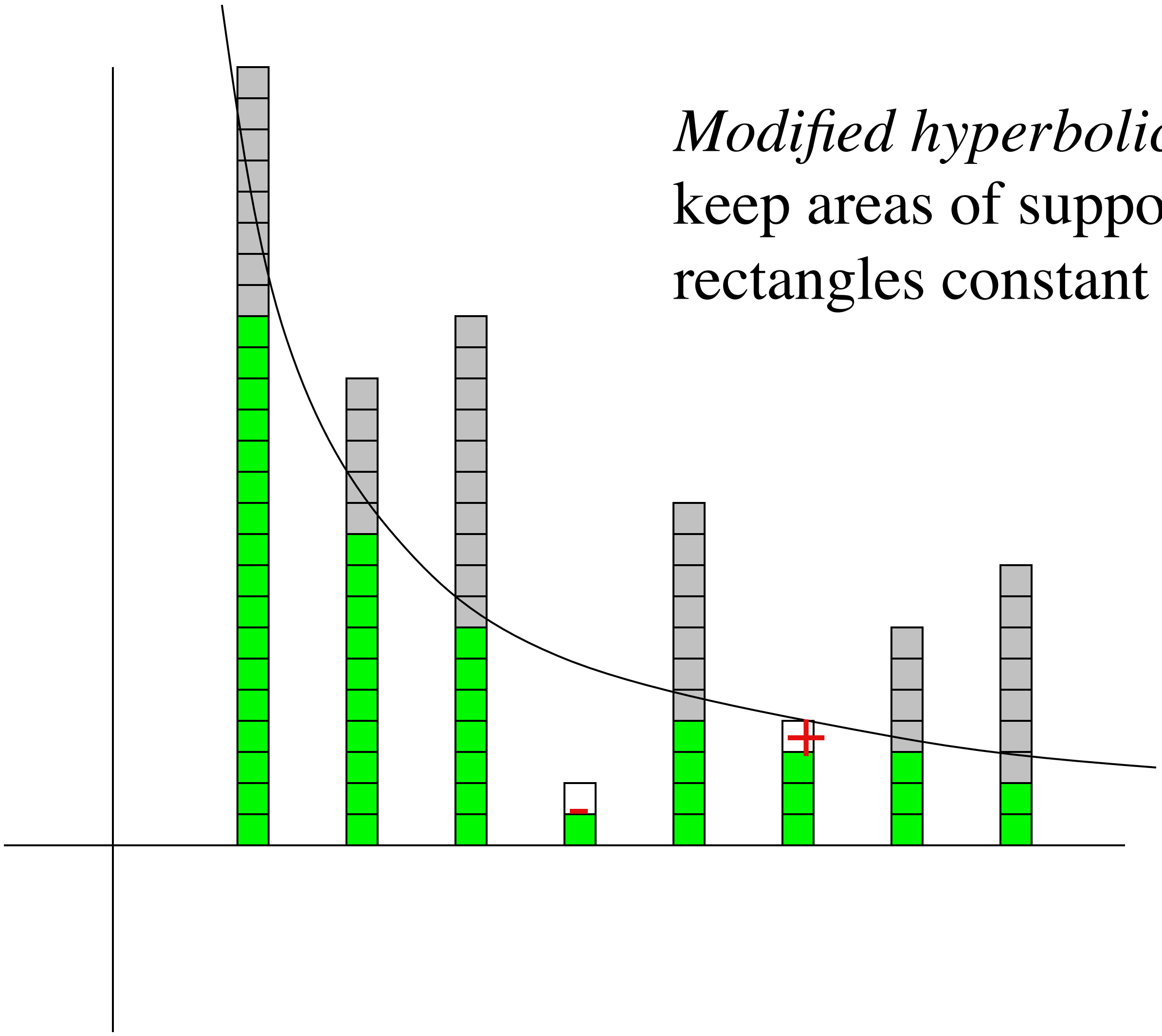
Modified hyperbolic scan:
keep areas of supported
rectangles constant





Modified hyperbolic scan:
keep areas of supported rectangles constant

Modified hyperbolic scan:
keep areas of supported rectangles constant



What about the average case?

What about the average case?

Similar results….

What about the average case?

Similar results….

- intrinsic cost is $\min_i m\lambda_i/(m-i)$
-

What about the average case?

Similar results….

- intrinsic cost is $\min_i m\lambda_i/(m-i)$
- hyperbolic scan remains log-competetive

Other generalizations…

- searching for a goal in a general symmetric tree

- searching for multiple goals

Thanks for your attention