Revisiting The Monge Property

Mordecai Golin
Hong Kong UST

Joint Work with
Amotz Bar-Noy, Yi Feng,
Rudolf Fleischer, Yan Zhang

(Revision of 18/6/08)

Well known that, under “special” circumstances, Dynamic Programming can be sped up.

Well known that, under "special" circumstances, Dynamic Programming can be sped up.

(a) $H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$

Well known that, under "special" circumstances, Dynamic Programming can be sped up.

(a) $$H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$$

(b) $$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$$

Well known that, under "special" circumstances, Dynamic Programming can be sped up.

(a) $$H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$$

$0 \le i \le n$ $\Theta(n^2)$ time

(b) $$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$$

$0 \le i \le n$ $\Theta(Dn^2)$ time

$0 \le d \le D$

Well known that, under "special" circumstances, Dynamic Programming can be sped up.

+ Monge Property

(a) $H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$

$0 \le i \le n$ $\qquad$ $\Theta(n^2)$ time

(b) $H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$

$0 \le i \le n$ $\qquad$ $\Theta(Dn^2)$ time

$0 \le d \le D$

Well known that, under "special" circumstances, Dynamic Programming can be sped up.

$+$ Monge Property

(a) $H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$

$0 \le i \le n$ $\qquad$ $\cancel{\Theta(n^2)\text{ time}}$ $\qquad$ $\Theta(n)$ time

(b) $H(i,d) = \min_{0 \le j < i} \Big(H(j,d-1) + w^{(d)}(j,i) \Big)$

$0 \le i \le n$ $\qquad$ $\cancel{\Theta(Dn^2)\text{ time}}$ $\qquad$ $\Theta(Dn)$ time

$0 \le d \le D$

$$H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$$

$$H(i,d) = \min_{0 \le j < i} \Big(H(j,d-1) + w^{(d)}(j,i) \Big)$$

$$0 \le i \le n$$
$$0 \le d \le D$$

$$H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$$

$$H(i,d) = \min_{0 \le j < i} \Big(H(j,d-1) + w^{(d)}(j,i) \Big)$$

$$0 \le i \le n \qquad n^2 \longrightarrow n$$

$$0 \le d \le D \qquad Dn^2 \longrightarrow Dn$$

$$H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$$

$$H(i,d) = \min_{0 \le j < i} \Big(H(j,d-1) + w^{(d)}(j,i) \Big)$$

$0 \le i \le n \qquad n^2 \longrightarrow n$

$0 \le d \le D \qquad Dn^2 \longrightarrow Dn$

Calculating $H(n,D)$ requires only $O(n)$ space.
Constructing explicit path in DP table yielding this solution, requires storing entire DP table $\Rightarrow \Theta(Dn)$ space.

$$H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$$

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$$

$0 \le i \le n \qquad n^2 \longrightarrow n$

$0 \le d \le D \qquad Dn^2 \longrightarrow Dn$

Calculating $H(n, D)$ requires only $O(n)$ space.
Constructing explicit path in DP table yielding this solution,
requires storing entire DP table $\Rightarrow \Theta(Dn)$ space.

First new result is reduction to $O(n)$ space.

$$H(i) = \min_{0 \le j < i} \Big(H(j) + w(j,i) \Big)$$

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$$

$0 \le i \le n \qquad n^2 \longrightarrow n$

$0 \le d \le D \qquad Dn^2 \longrightarrow Dn$

Calculating $H(n, D)$ requires only $O(n)$ space.
Constructing explicit path in DP table yielding this solution, requires storing entire DP table $\Rightarrow \Theta(Dn)$ space.

First new result is reduction to $O(n)$ space.

Speedup works by batching calculations.
Data (the $w(j,i)$) must be known in advance so that proper batching order can be used. In particular, speedup fails if data is given online, i.e., $i = 1, 2, 3, \ldots$.

$$H(i) = \min_{0\le j<i}\Big(H(j) + w(j,i)\Big)$$

$$H(i,d) = \min_{0\le j<i}\Big(H(j,d-1) + w^{(d)}(j,i)\Big)$$

$0 \le i \le n \qquad n^2 \longrightarrow n$

$0 \le d \le D \qquad Dn^2 \longrightarrow Dn$

Calculating $H(n,D)$ requires only $O(n)$ space.
Constructing explicit path in DP table yielding this solution, requires storing entire DP table $\Rightarrow \Theta(Dn)$ space.

First new result is reduction to $O(n)$ space.

Speedup works by batching calculations.
Data (the $w(j,i)$) must be known in advance so that proper batching order can be used. In particular, speedup fails if data is given online, i.e., $i = 1, 2, 3, \ldots$.

Second new result is how to maintain the speedup for online data; $O(1)$ or $O(D)$ per update.

Outline

- Review of the Monge Speedup
- Saving Space While Saving Time
- Maintaining the Speedup in an Online Setting

The Monge Speedup

- M is an $m \times n$ matrix

The Monge Speedup

- M is an $m \times n$ matrix
- $\text{RM}_M(i)$ is column index of (rightmost) min item on row i of M.
- M is Monotone if $\forall i \leq i'$, $\text{RM}_M(i) \leq \text{RM}_M(i')$.

The Monge Speedup

- M is an $m \times n$ matrix
- $\text{RM}_M(i)$ is column index of (rightmost) min item on row i of M.
- M is Monotone if $\forall i \leq i'$, $\text{RM}_M(i) \leq \text{RM}_M(i')$.

7	2	4	3	9	9
5	1	5	1	6	5
7	1	2	0	3	1
9	4	5	1	3	2
8	4	5	3	4	3
9	6	7	5	6	5

$\text{RM}_M(1) = 2$

$\text{RM}_M(2) = 4$

$\text{RM}_M(3) = 4$

$\text{RM}_M(4) = 4$

$\text{RM}_M(5) = 6$

$\text{RM}_M(6) = 6$

The Monge Speedup

- M is an $m \times n$ matrix
- $\text{RM}_M(i)$ is column index of (rightmost) min item on row i of M.
- M is Monotone if $\forall i \le i'$, $\text{RM}_M(i) \le \text{RM}_M(i')$.
- 2×2 monotone matrices have form

$$\begin{bmatrix} 2 & 4 \\ 4 & 5 \end{bmatrix} \qquad \begin{bmatrix} 2 & 3 \\ 5 & 3 \end{bmatrix} \qquad \begin{bmatrix} 7 & 1 \\ 2 & 2 \end{bmatrix} \qquad \xcancel{\begin{bmatrix} 7 & 1 \\ 2 & 3 \end{bmatrix}}$$

- An $m \times n$ matrix M is Totally Monotone (TM) if every 2×2 submatrix is Monotone.

(submatrix: not necessarily contiguous in the original matrix)

SMAWK and LARSCH Algorithms

- Motivation:
  Find all m row minima of an implicitly given $m \times n$ matrix M

SMAWK and LARSCH Algorithms

- Motivation:
  Find all m row minima of an implicitly given $m \times n$ matrix M
- Naive Algorithm: $O(mn)$

SMAWK and LARSCH Algorithms

- Motivation:
  Find all m row minima of an implicitly given $m \times n$ matrix M
- Naive Algorithm: $O(mn)$

- SMAWK Algorithm
  [Aggarwal, Klawe, Moran, Shor, Wilber (1986)]
  - If M is Totally Monotone,
    all m row minima can be found in $O(m+n)$ time.
  - Usually $m = \Theta(n)$
    $\Theta(n)$ speedup: $O(n^2)$ down to $O(n)$.

SMAWK and LARSCH Algorithms

- Motivation:
  Find all m row minima of an implicitly given $m \times n$ matrix M
- Naive Algorithm: $O(mn)$

- SMAWK Algorithm
  [Aggarwal, Klawe, Moran, Shor, Wilber (1986)]
  - If M is Totally Monotone,
    all m row minima can be found in $O(m+n)$ time.
  - Usually $m = \Theta(n)$
    $\Theta(n)$ speedup: $O(n^2)$ down to $O(n)$.

- SMAWK was culmination of decade(s) of work on similar problems; speedups using convexity and concavity.
  Has been used to speed up many DP problems, e.g., computational geometry, bioinformatics, k-center on a line, etc.

SMAWK and LARSCH Algorithms

- Motivation:
  Find all m row minima of an implicitly given $m \times n$ matrix M
- Naive Algorithm: $O(mn)$

- SMAWK Algorithm
  [Aggarwal, Klawe, Moran, Shor, Wilber (1986)]
  - If M is Totally Monotone,
    all m row minima can be found in $O(m + n)$ time.
  - Usually $m = \Theta(n)$
    $\Theta(n)$ speedup: $O(n^2)$ down to $O(n)$.

- LARSCH Algorithm [Larmore, Schieber (1991)]
  More complicated solution to same problem.
  Allows dependencies of $M_{i,j}$ on earlier row minima in matrix.

The Monge Property

- Motivation: TM is often established via Monge property

The Monge Property

- Motivation: TM is often established via Monge property
- $m \times n$ matrix M is Monge if $\forall i \leq i'$ and $\forall j \leq j'$

$$M_{i,j} + M_{i',j'} \leq M_{i',j} + M_{i,j'}$$

The Monge Property

- Motivation: TM is often established via Monge property

- $m \times n$ matrix M is Monge if $\forall i \le i'$ and $\forall j \le j'$

$$M_{i,j} + M_{i',j'} \le M_{i',j} + M_{i,j'}$$

- M is Monge $\Rightarrow$ M is Totally Monotone

The Monge Property

- Motivation: TM is often established via Monge property

- $m \times n$ matrix M is Monge if $\forall i \leq i'$ and $\forall j \leq j'$

$$M_{i,j} + M_{i',j'} \leq M_{i',j} + M_{i,j'}$$

- M is Monge $\Rightarrow$ M is Totally Monotone

- Also, if $\forall i, j$, $\quad M_{i,j} + M_{i+1,j+1} \leq M_{i+1,j} + M_{i,j+1}$,
  $\Rightarrow$ M is Monge.

The Monge Property

- Motivation: TM is often established via Monge property
- $m \times n$ matrix M is Monge if $\forall i \le i'$ and $\forall j \le j'$

$$M_{i,j} + M_{i',j'} \le M_{i',j} + M_{i,j'}$$

- M is Monge $\Rightarrow$ M is Totally Monotone
- Also, if $\forall i, j$, $\quad M_{i,j} + M_{i+1,j+1} \le M_{i+1,j} + M_{i,j+1}$,
  $\Rightarrow$ M is Monge.
- $\Rightarrow$ Only need to prove Monge property for adjacent rows and columns.

Using The Monge Property

Suppose we are given DP (i.v. $H(i,0)$ known, $i \le n$, $d \le D$):

$$H(i,d) = \min_{0 \le j < i} \Big(H(j,d-1) + w^{(d)}(j,i) \Big)$$

Using The Monge Property

Suppose we are given DP (i.v. $H(i,0)$ known, $i \le n$, $d \le D$):

$$H(i,d) = \min_{0 \le j < i}\Big(H(j,d-1) + w^{(d)}(j,i)\Big)$$

For $j < i$, set $M_{j,i} = H(j,d-1) + w^{(d)}(j,i)$; else $M_{j,i} = \infty$

Using The Monge Property

Suppose we are given DP (i.v. $H(i,0)$ known, $i \le n$, $d \le D$):

$$H(i,d) = \min_{0 \le j < i} \Big(H(j,d-1) + w^{(d)}(j,i) \Big)$$

For $j < i$, set $M_{j,i} = H(j,d-1) + w^{(d)}(j,i)$; else $M_{j,i} = \infty$

To calculate $H(*,d)$, simply find row-minima in M

Using The Monge Property

Suppose we are given DP (i.v. $H(i,0)$ known, $i \le n$, $d \le D$):

$$H(i,d) = \min_{0 \le j < i} \Big(H(j,d-1) + w^{(d)}(j,i) \Big)$$

For $j < i$, set $M_{j,i} = H(j,d-1) + w^{(d)}(j,i)$; else $M_{j,i} = \infty$

To calculate $H(*,d)$, simply find row-minima in M

$\boxed{\textbf{Fact: If } w^{(d)}(j,i) \textbf{ are Monge} \Rightarrow M \textbf{ is Monge}}$

Using The Monge Property

Suppose we are given DP (i.v. $H(i,0)$ known, $i \le n$, $d \le D$):

$$H(i,d) = \min_{0 \le j < i} \Big(H(j,d-1) + w^{(d)}(j,i) \Big)$$

For $j < i$, set $M_{j,i} = H(j,d-1) + w^{(d)}(j,i)$; else $M_{j,i} = \infty$

To calculate $H(*,d)$, simply find row-minima in M

Fact: If $w^{(d)}(j,i)$ are Monge $\Rightarrow$ M is Monge

Then, for given d, SMAWK finds all $H(*,d)$ in $O(n)$ time;
iterating, finds all $H(i,d)$ in $O(nD)$ time.

Examples of $H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$

$i \le n,\ d \le D$

Examples of $H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$

$i \le n,\ d \le D$

- Length Limited Huffman Codes $0 \le p_1 \le p_2 \le \cdots \le p_n$

  $w^{(d)}(j,i) = S_{2j-i}$ where $S_k = \sum_{i=1}^{k} p_i$.

  $H(n-1, D)$ is cost of min-cost D-limited code

Examples of $i \le n,\ d \le D$

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$$

- Length Limited Huffman Codes $0 \le p_1 \le p_2 \le \cdots \le p_n$

  $w^{(d)}(j,i) = S_{2j-i}$ where $S_k = \sum_{i=1}^{k} p_i$.

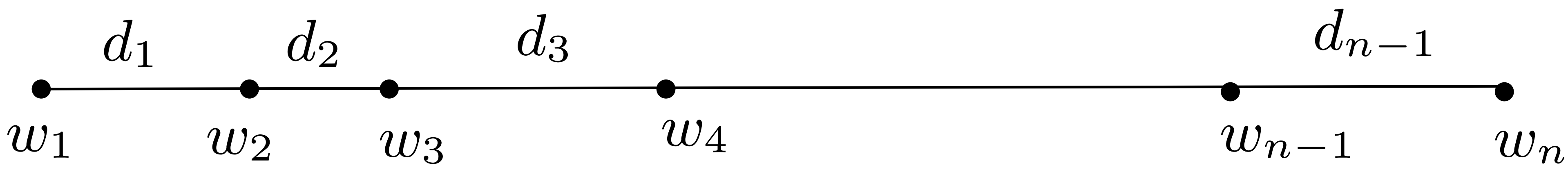
  $H(n-1, D)$ is cost of min-cost D-limited code

- Wireless mobile paging $p_1 \ge p_2 \ge \cdots \ge p_n \ge 0$

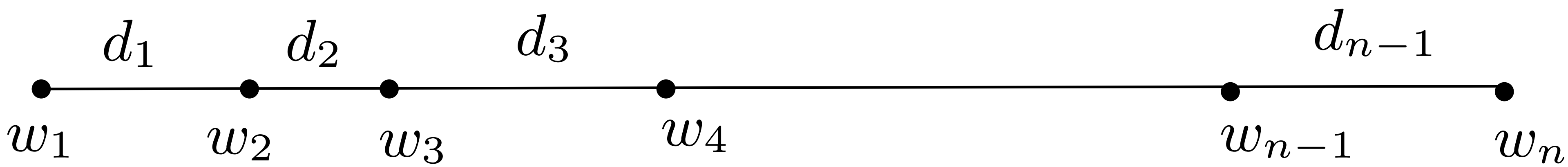
  $$w^{(d)}(j,i) = i \left(\sum_{\ell=j+1}^{i} p_\ell \right)$$

  $H(n, D)$ is min expected bandwidth required to page all items using $\le D$ paging rounds

- D-Medians on a Directed Line *Woeginger '00*



• D-Medians on a Directed Line *Woeginger '00*



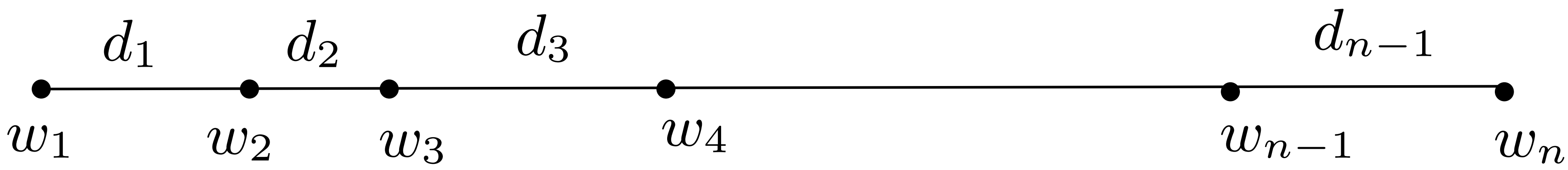
Identify D nodes as service centers.

Nodes can only be serviced by node to their left (or themselves) so node 1 must be a service center.

Cost of servicing request w_i, is w_i times distance from node i to nearest service center.

Problem is to find location of D service centers
that minimize total service cost.

- D-Medians on a Directed Line *Woeginger '00*



Let $H(i,d)$ be cost of
servicing nodes $[1,i]$ using exactly d servers.

$$H(i,d) = \begin{cases} 0 & n = d \\ w^{(d)}_{0,i} & d = 0,\ i \geq 1 \\ \min\limits_{d-1 \leq j < i} \big(H(j,d-1) + w^{(d)}(j,i)\big), & 1 \leq d < n \end{cases}$$

$$w^{(d)}_{j,i} = \textstyle\sum_{l=j+1}^{i} w_l(v_l - v_{j+1}), \qquad v_k = \textstyle\sum_{j=1}^{k-1} d_j$$

Examples of $\quad H(i,d) = \min_{0\le j<i}\Big(H(j,d-1) + w^{(d)}(j,i)\Big)$

$i \le n,\ d \le D$

- Length Limited Huffman Codes

  $w^{(d)}(j,i) = S_{2j-i}$ where $S_k = \sum_{i=1}^{k} p_i$.

- Wireless mobile paging $\quad w^{(d)}(j,i) = i\left(\sum_{\ell=j+1}^{i} p_\ell\right)$

- D-Medians on a Directed Line $\quad w^{(d)}(j,i) = \sum_{l=j+1}^{i} w_l(v_l - v_{j+1})$

Examples of $i \le n,\ d \le D$

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$$

- Length Limited Huffman Codes

  $w^{(d)}(j,i) = S_{2j-i}$ where $S_k = \sum_{i=1}^{k} p_i$.

- Wireless mobile paging $\quad w^{(d)}(j,i) = i \left(\sum_{\ell=j+1}^{i} p_\ell \right)$

- D-Medians on a Directed Line $\quad w^{(d)}(j,i) = \sum_{l=j+1}^{i} w_l (v_l - v_{j+1})$

All these $w^{(d)}(j,i) = w_{j,i}$ satisfy Monge property

$$w_{j,i} + w_{j+1,i+1} \le w_{j,i+1} + w_{j+1,i}$$

$\Rightarrow H(n,D)$ can be calculated in $O(nD)$ time

Outline

- Review of the Monge Speedup
- Saving Space While Saving Time
- Maintaining the Speedup in an Online Setting

Given a DP in the form

$$H(i,d) = \min_{0 \le j < i} \left(H(j, d-1) + w^{(d)}(j,i) \right) \qquad \begin{array}{c} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

in which, for fixed d, the $w^{(d)}$ are Monge, e.g., D-limited Huffman Encoding, D-Median on a line or Wireless Paging , the $H(\cdot,\cdot)$ table can be filled in using only $O(nD)$ time.

Given a DP in the form

$$H(i,d) = \min_{0\le j<i}\Big(H(j,d-1) + w^{(d)}(j,i)\Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

in which, for fixed d, the $w^{(d)}$ are Monge, e.g., D-limited Huffman Encoding, D-Median on a line or Wireless Paging , the $H(\cdot,\cdot)$ table can be filled in using only $O(nD)$ time.

Furthermore, calculation of $H(\cdot,d)$ only requires knowledge of $H(\cdot,d-1)$. So, if $H(n,D)$ is final goal, we can fill in table iteratively, for $d = 1,2,\ldots,D$, using only $O(n)$ space.

On the other hand, finding actual "solution path" of DP, corresponding to min-cost tree, median locations or paging schedule, requires backtracking through DP table. This implies storing entire table, using $\Theta(nD)$ space.

Context:

$$H(i,d) = \min_{0\le j<i}\Big(H(j,d-1) + w^{(d)}(j,i)\Big) \qquad \begin{array}{c} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

D-Length-Limited Huffman Coding

(*) $\quad w^{(d)}(j,i) = S_{2j-i}$ where $S_k = \sum_{i=1}^{k} p_i$.

Context:

$$H(i,d) = \min_{0\le j<i}\Big(H(j,d-1) + w^{(d)}(j,i)\Big) \qquad \begin{matrix} 0 \le i \le n \\ 0 \le d \le D \end{matrix}$$

D-Length-Limited Huffman Coding

(*) $w^{(d)}(j,i) = S_{2j-i}$ where $S_k = \sum_{i=1}^{k} p_i$.

Larmore & Hirschberg ('90) $O(nD)$ time, $O(n)$ space.

Very clever special-purpose algorithm; culmination of a long series of papers by various authors on this problem.

Context:

$$H(i,d) = \min_{0 \le j < i} \left(H(j, d-1) + w^{(d)}(j,i) \right) \qquad \begin{matrix} 0 \le i \le n \\ 0 \le d \le D \end{matrix}$$

D-Length-Limited Huffman Coding

(*) $w^{(d)}(j,i) = S_{2j-i}$ where $S_k = \sum_{i=1}^{k} p_i$.

Larmore & Hirschberg ('90) $O(nD)$ time, $O(n)$ space.

Very clever special-purpose algorithm; culmination of a long series of papers by various authors on this problem.

Larmore & Przytycka ('91) Derived (*) DP formulation

Easy $O(nD)$ time (Monge) algorithm but not interesting since it requires $\Theta(nD)$ space as well.

Context:

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

D-Length-Limited Huffman Coding

(*) $\quad w^{(d)}(j,i) = S_{2j-i}$ where $S_k = \sum_{i=1}^{k} p_i$.

Larmore & Hirschberg ('90) $\qquad$ $O(nD)$ time, $O(n)$ space.

Very clever special-purpose algorithm; culmination of a long series of papers by various authors on this problem.

Larmore & Przytycka ('91) $\qquad$ Derived (*) DP formulation

Easy $O(nD)$ time (Monge) algorithm but not interesting since it requires $\Theta(nD)$ space as well.

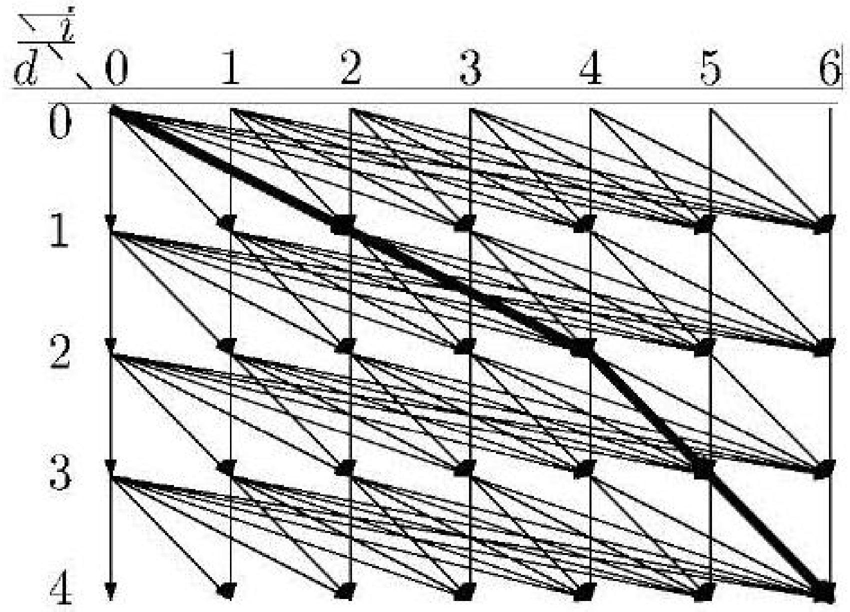
Would like to reduce space for (*) down to $\Theta(n)$

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{c} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

$$H(i,d) = \min_{0 \le j < i} \left(H(j, d-1) + w^{(d)}(j,i) \right) \qquad \begin{matrix} 0 \le i \le n \\ 0 \le d \le D \end{matrix}$$

Alternative Interpretation:

Consider a layered graph in which edges only go down one level and to the right.

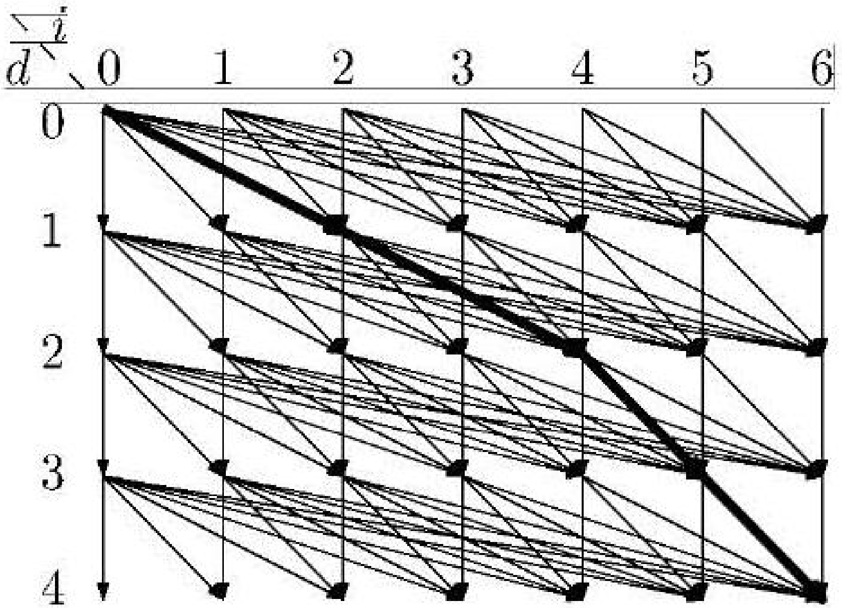


$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{matrix} 0 \le i \le n \\ 0 \le d \le D \end{matrix}$$

Alternative Interpretation:

Consider a layered graph in which edges only go down one level and to the right.

$$w\Big(\ (d-1, j) \rightarrow (d, i)\ \Big) = w^{(d)}(j,i)$$

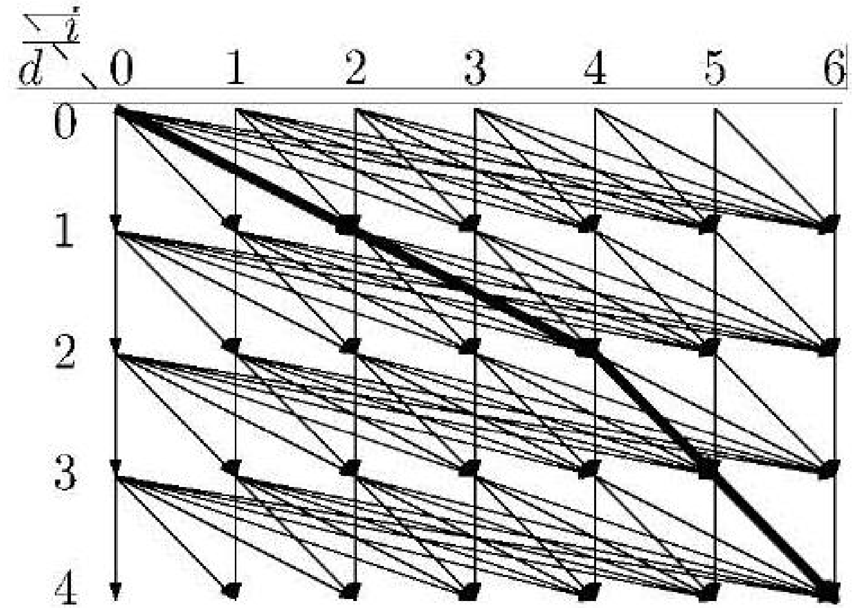


$$H(i,d) = \min_{0 \le j < i} \left(H(j, d-1) + w^{(d)}(j,i) \right) \qquad \begin{array}{c} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

Alternative Interpretation:

Consider a layered graph in which edges only go down one level and to the right.

$$w\Big(\ (d-1, j) \to (d,i) \ \Big) = w^{(d)}(j,i)$$



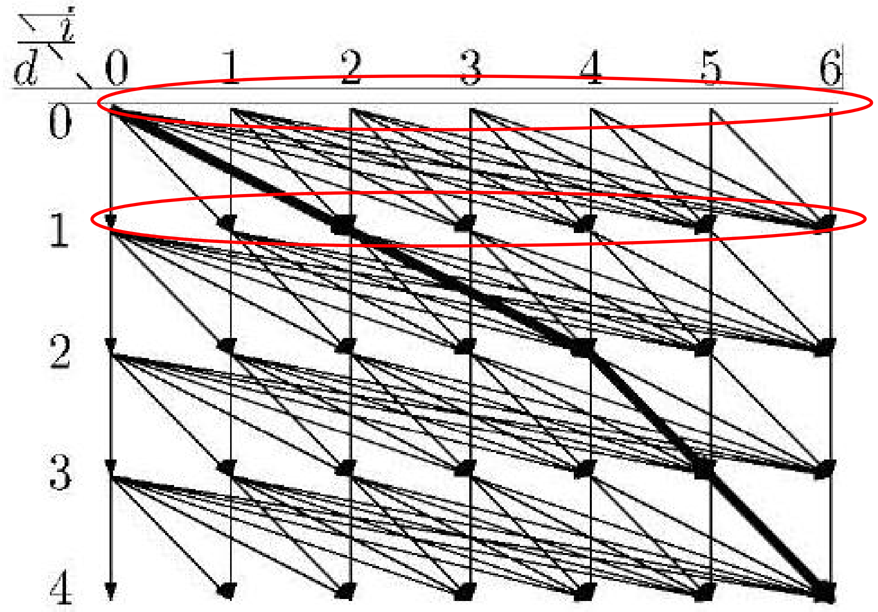
$H(i,d) =$ cost of min-cost path from $(0,0)$ to (d,i).

Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{matrix} 0 \le i \le n \\ 0 \le d \le D \end{matrix}$$

Alternative Interpretation:

Consider a layered graph in which edges only go down one level and to the right.



$$w\Big(\ (d-1, j) \to (d, i) \ \Big) = w^{(d)}(j,i)$$

$H(i,d) =$ cost of min-cost path from $(0,0)$ to (d,i).

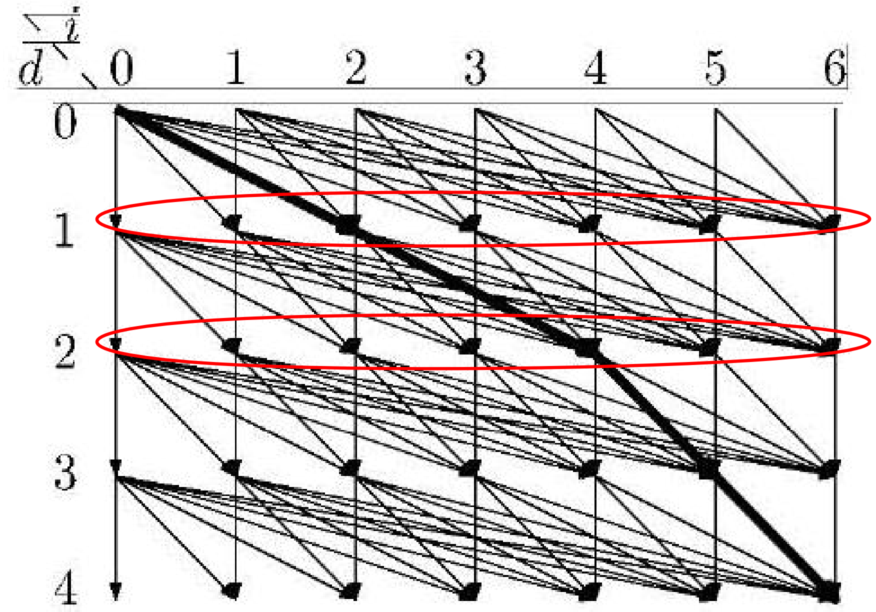
Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{matrix} 0 \le i \le n \\ 0 \le d \le D \end{matrix}$$

Alternative Interpretation:

Consider a layered graph in which edges only go down one level and to the right.

$$w\Big(\ (d-1, j) \to (d, i)\ \Big) = w^{(d)}(j,i)$$



$H(i,d) =$ cost of min-cost path from $(0,0)$ to (d,i).

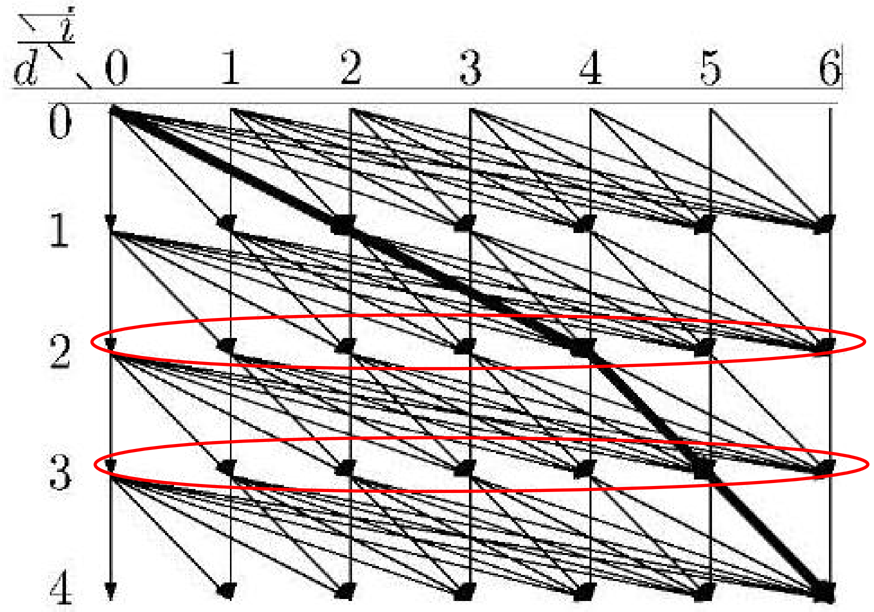
Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{matrix} 0 \le i \le n \\ 0 \le d \le D \end{matrix}$$

Alternative Interpretation:

Consider a layered graph in which edges only go down one level and to the right.

$$w\Big(\ (d-1, j) \to (d, i) \ \Big) = w^{(d)}(j,i)$$



$H(i,d) =$ cost of min-cost path from $(0,0)$ to (d,i).

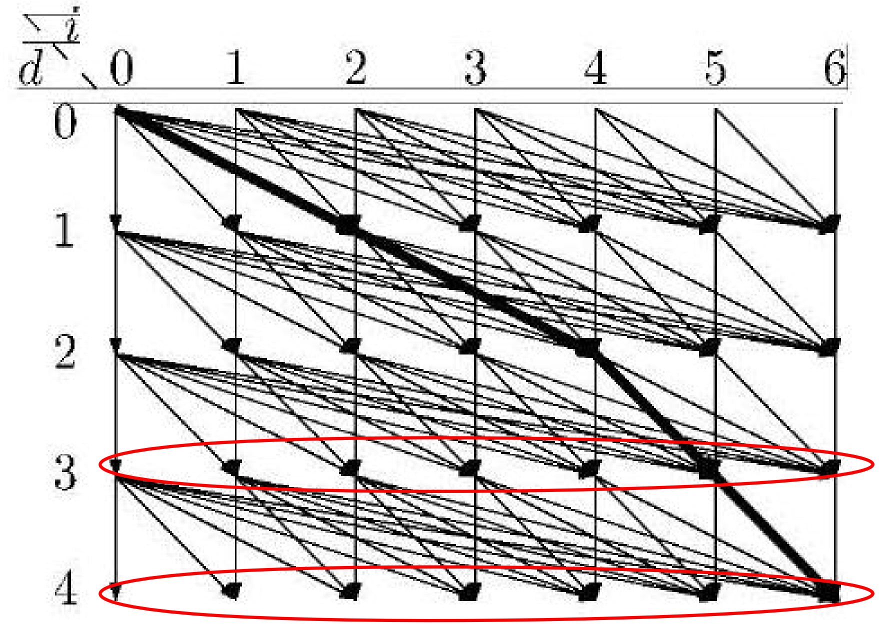
Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{c} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

Alternative Interpretation:

Consider a layered graph in which edges only go down one level and to the right.

$$w\Big(\ (d-1, j) \to (d, i)\ \Big) = w^{(d)}(j,i)$$



$H(i,d) =$ cost of min-cost path from $(0,0)$ to (d,i).

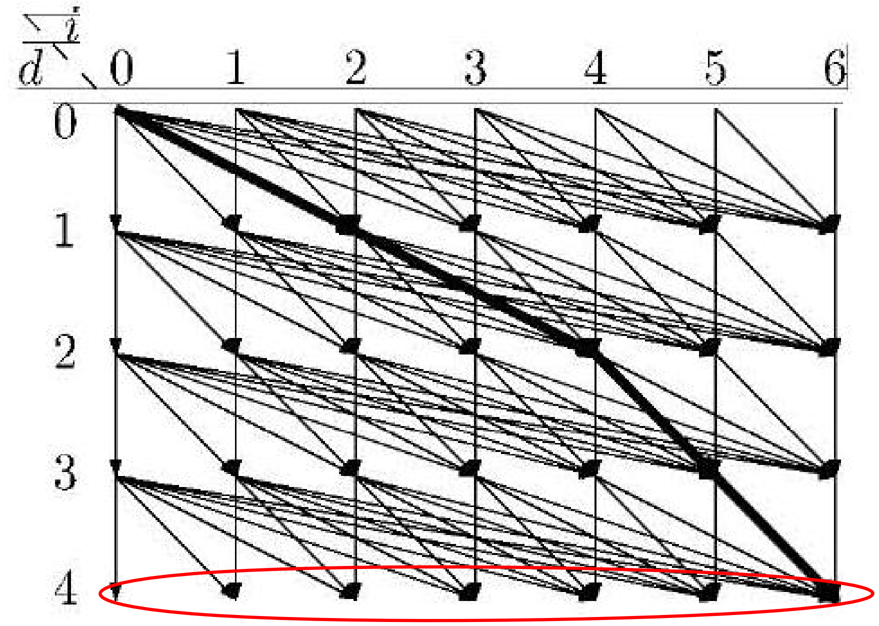
Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{c} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

Alternative Interpretation:

Consider a layered graph in which edges only go down one level and to the right.

$$w\Big(\ (d-1, j) \to (d,i)\ \Big) = w^{(d)}(j,i)$$



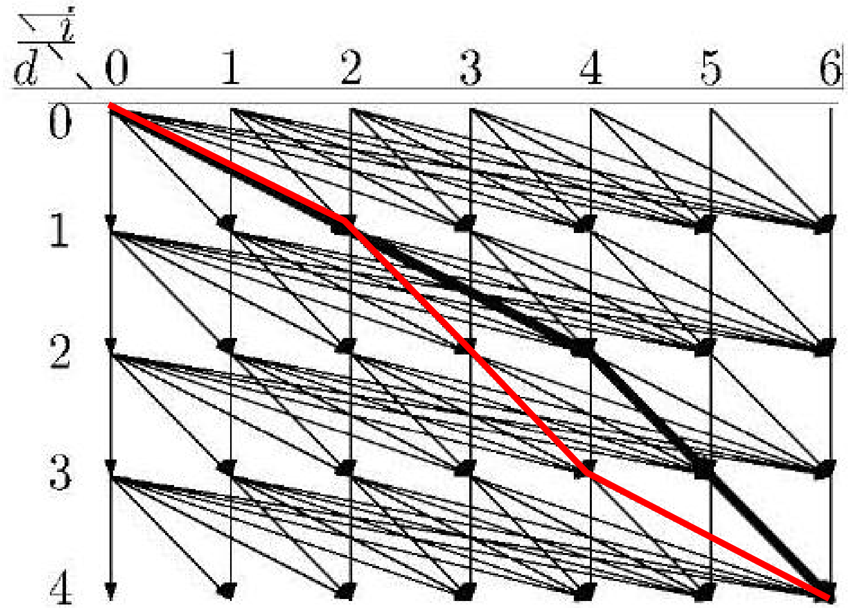
$H(i,d) =$ cost of min-cost path from $(0,0)$ to (d,i).

Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

$$H(i,d) = \min_{0\le j<i}\left(H(j,d-1)+w^{(d)}(j,i)\right) \qquad \begin{array}{l} 0\le i\le n \\ 0\le d\le D \end{array}$$

Alternative Interpretation:

Consider a layered graph in which edges only go down one level and to the right.



$$w\Big(\ (d-1,j)\rightarrow(d,i)\ \Big) = w^{(d)}(j,i)$$

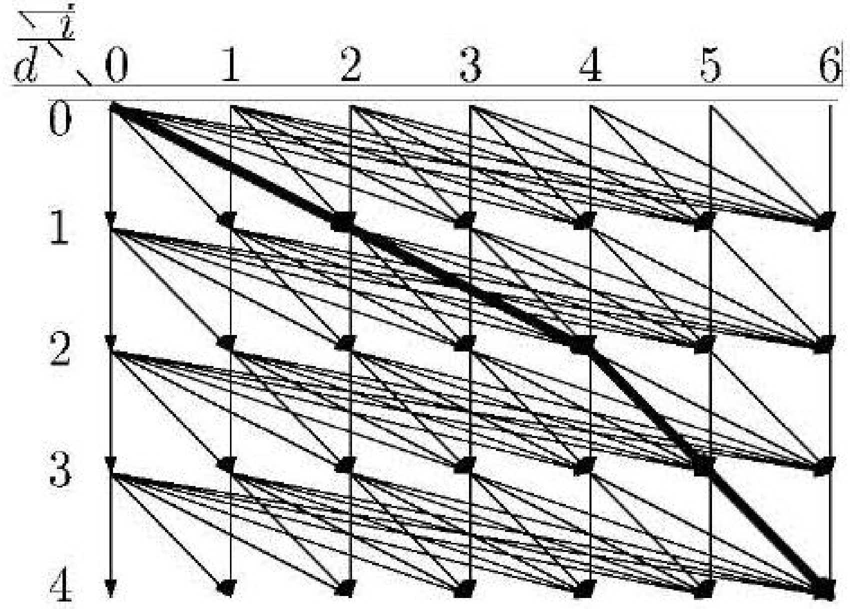
$H(i,d)$ = cost of min-cost path from $(0,0)$ to (d,i).

Given row $H(\cdot,d-1)$, SMAWK calculates row $H(\cdot,d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot,D)$ in $O(nD)$ time and $O(D)$ space.

On the other hand, finding optimal path to $H(D,n)$ requires keeping entire $\Theta(nD)$ space table to backtrack through

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

We will now see how to find path using $O(D+n)$ space.



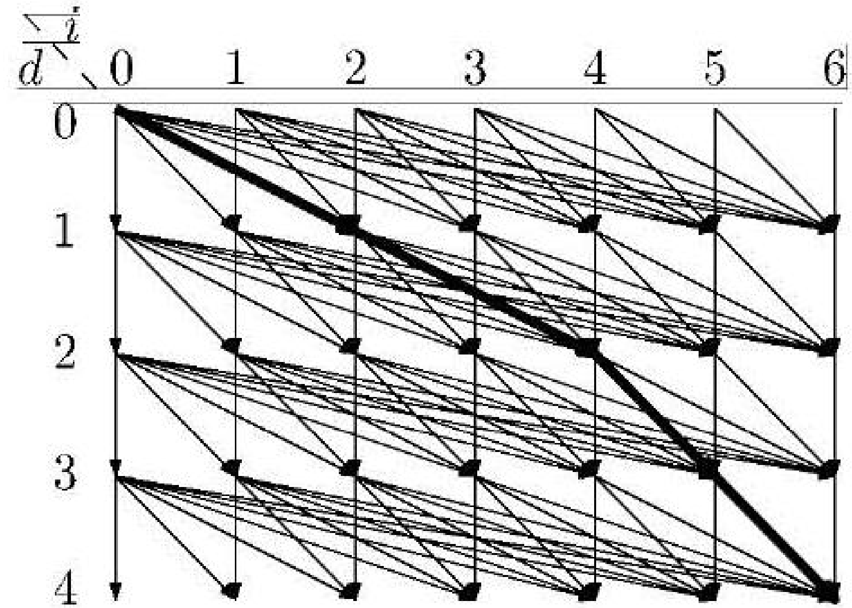
Modification of idea due to

Hirschberg ('75)

Munro & Ramirez ('82)

$$H(i,d) = \min_{0 \le j < i} \left(H(j, d-1) + w^{(d)}(j,i) \right) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

We will now see how to find path using $O(D+n)$ space.

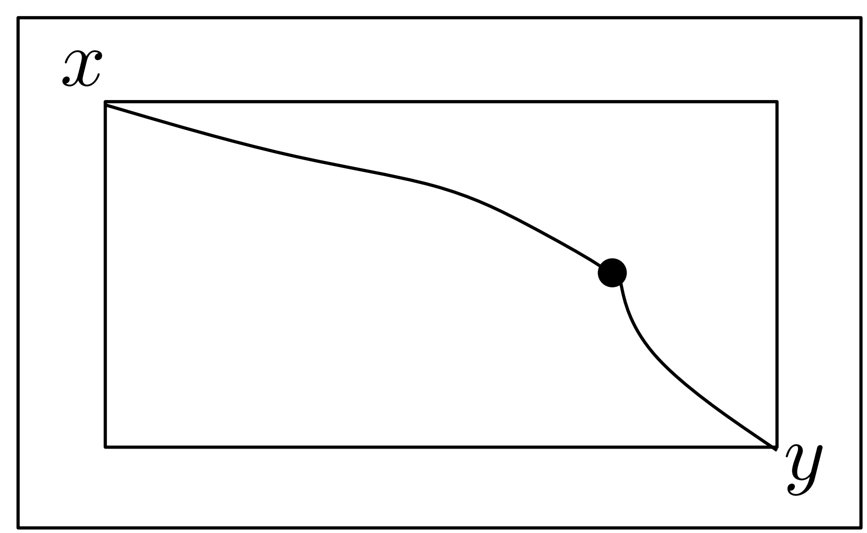


Modification of idea due to

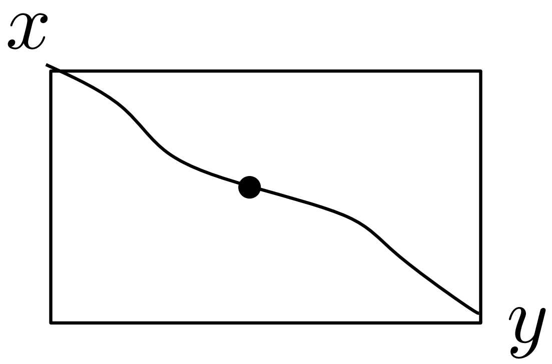
Hirschberg ('75)

Munro & Ramirez ('82)

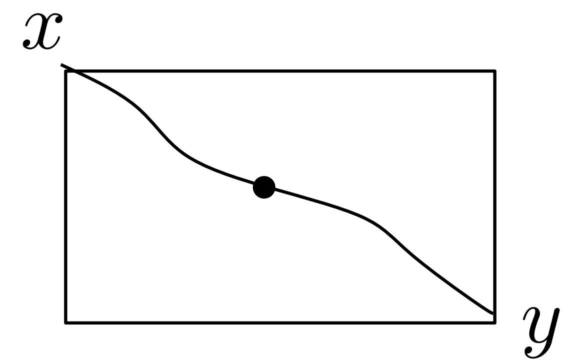
Let y be below and to the right of x. Assume existence of an oracle $Mid(x,y)$ that returns a midpoint (hop distance) on some min-cost x-y path.



$Mid(x, y)$ returns a midpoint (hop distance) on some min-cost x-y path.



$Mid(x, y)$ returns a midpoint (hop distance) on some min-cost x-y path.



We now have a simple recursive procedure for building min-cost path

Buildpath(x,y)

If $y_d = x_{d+1}$

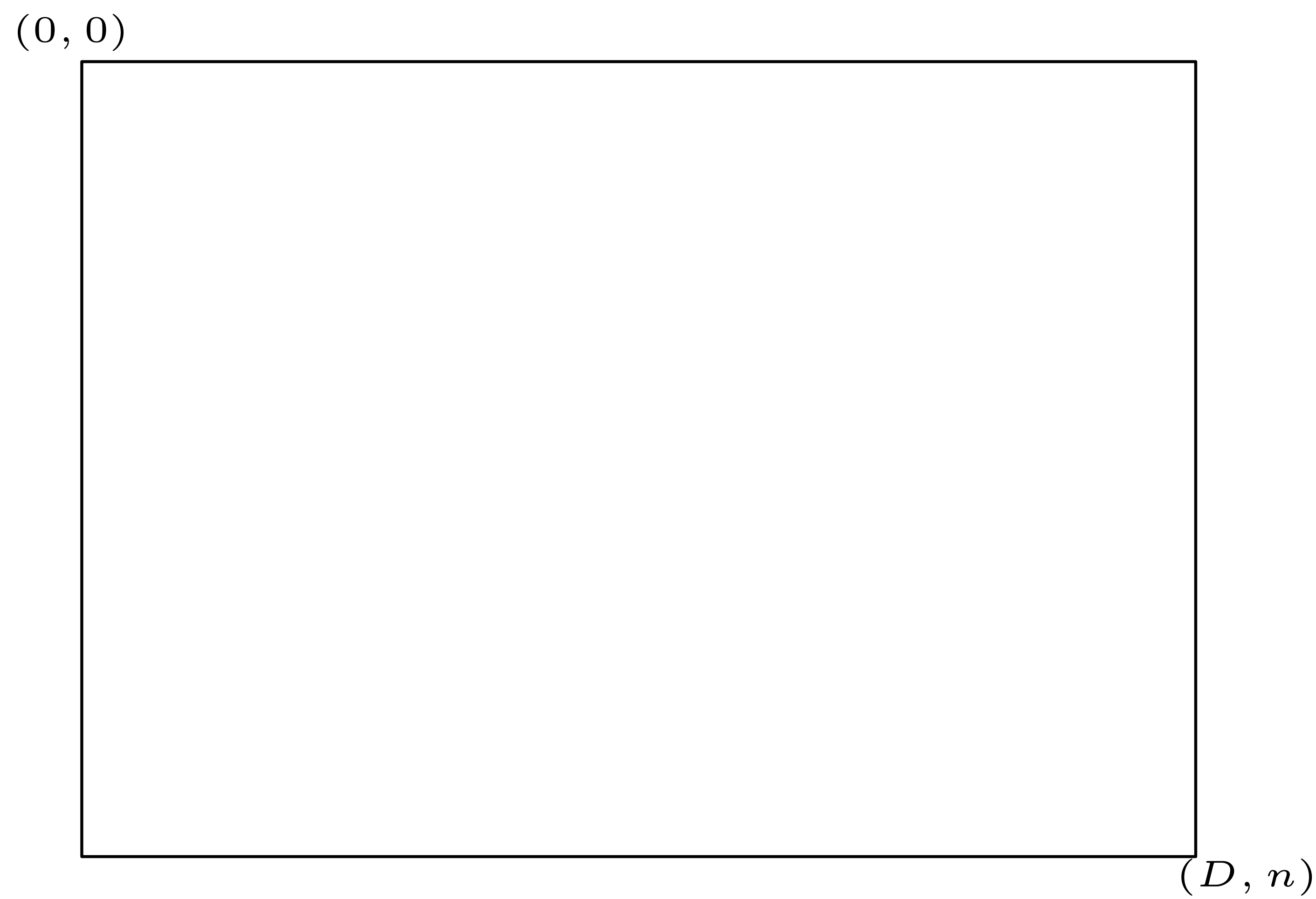
return $(x \rightarrow y)$

else

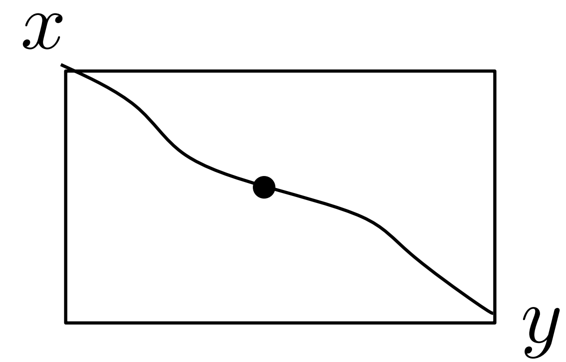
$z = Mid(x, y)$

Buildpath(x,z)

Buildpath(z,y)



$Mid(x, y)$ returns a midpoint (hop distance) on some min-cost x-y path.



We now have a simple recursive procedure for building min-cost path

Buildpath(x,y)

If $y_d = x_{d+1}$

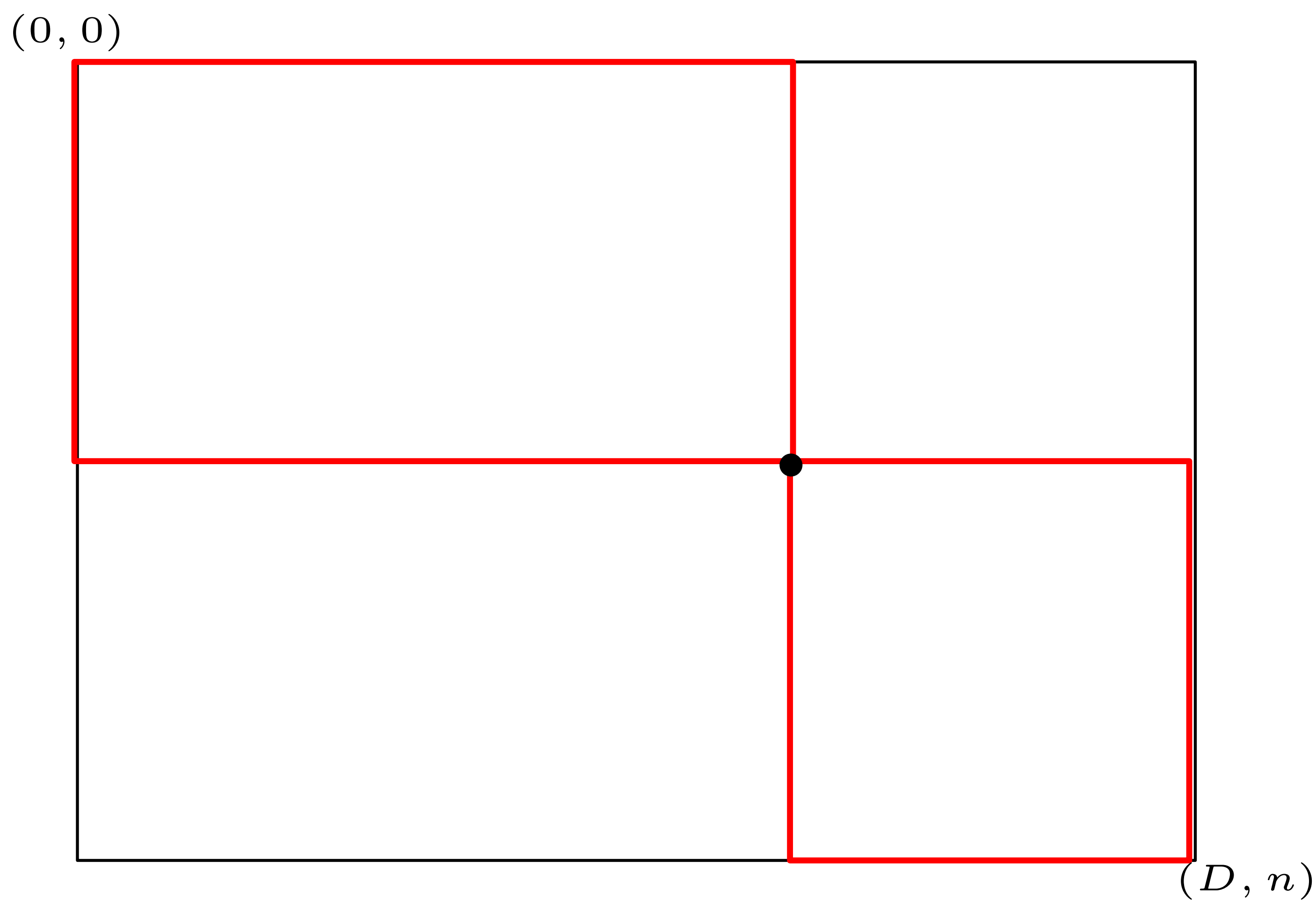
return $(x \rightarrow y)$

else

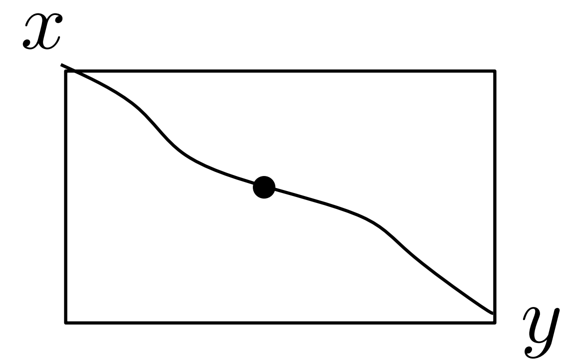
$z = Mid(x, y)$

Buildpath(x,z)

Buildpath(z,y)



$Mid(x, y)$ returns a midpoint (hop distance) on some min-cost x-y path.



We now have a simple recursive procedure for building min-cost path

Buildpath(x,y)

If $y_d = x_{d+1}$

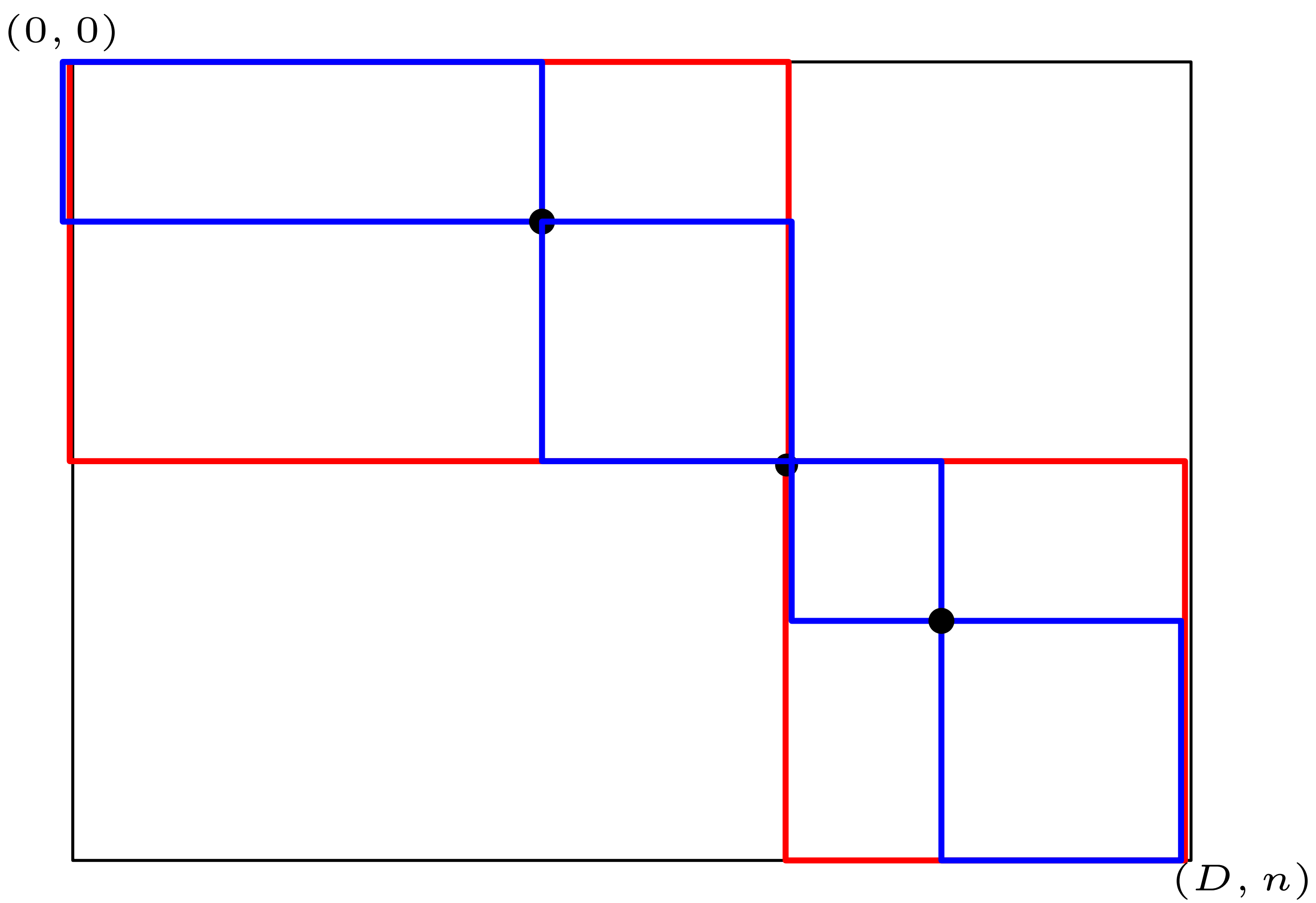
return $(x \rightarrow y)$

else

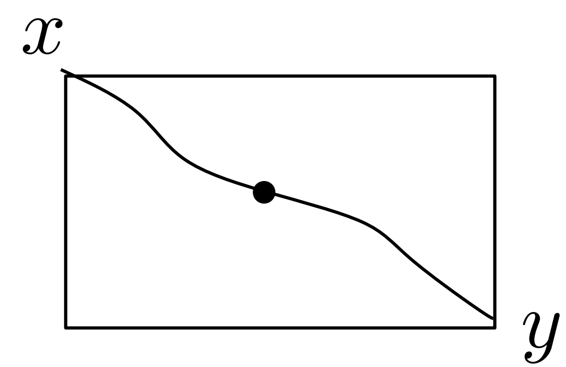
$z = Mid(x, y)$

Buildpath(x,z)

Buildpath(z,y)



$Mid(x, y)$ returns a midpoint (hop distance) on some min-cost x-y path.



We now have a simple recursive procedure for building min-cost path

Buildpath(x,y)

If $y_d = x_{d+1}$

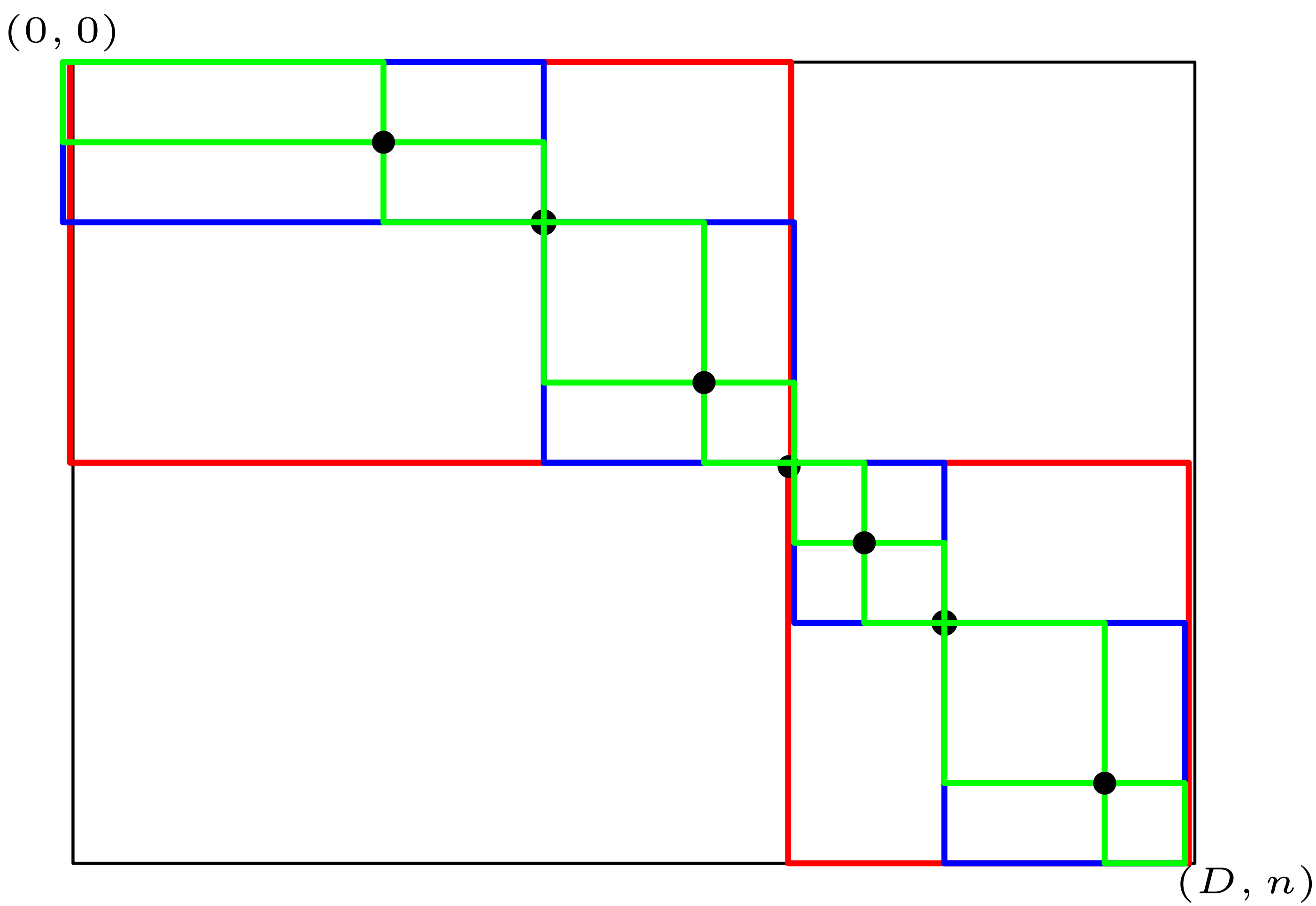
return $(x \rightarrow y)$

else

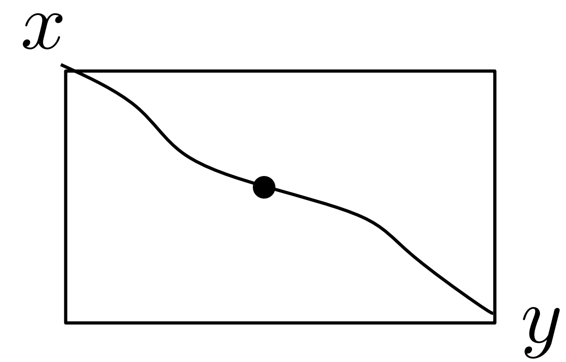
$z = Mid(x, y)$

Buildpath(x,z)

Buildpath(z,y)



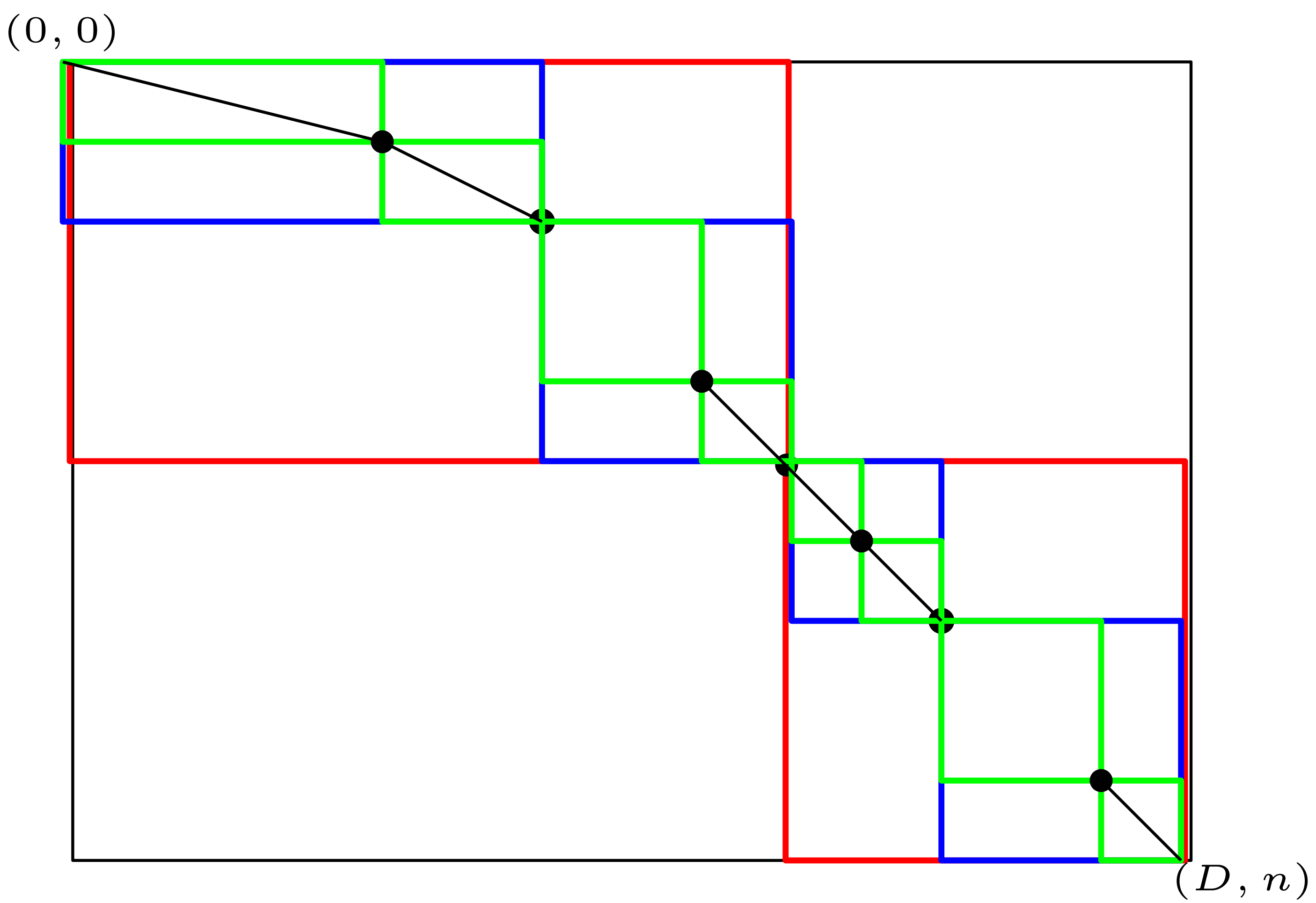
$Mid(x, y)$ returns a midpoint (hop distance) on some min-cost x-y path.



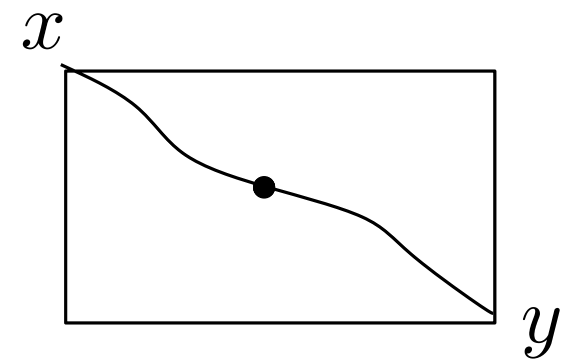
We now have a simple recursive procedure for building min-cost path

Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \to y)$
else
  $z = Mid(x, y)$
  Buildpath(x,z)
  Buildpath(z,y)



$Mid(x, y)$ returns a midpoint (hop distance) on some min-cost x-y path.



We now have a simple recursive procedure for building min-cost path

Buildpath(x,y)

If $y_d = x_{d+1}$

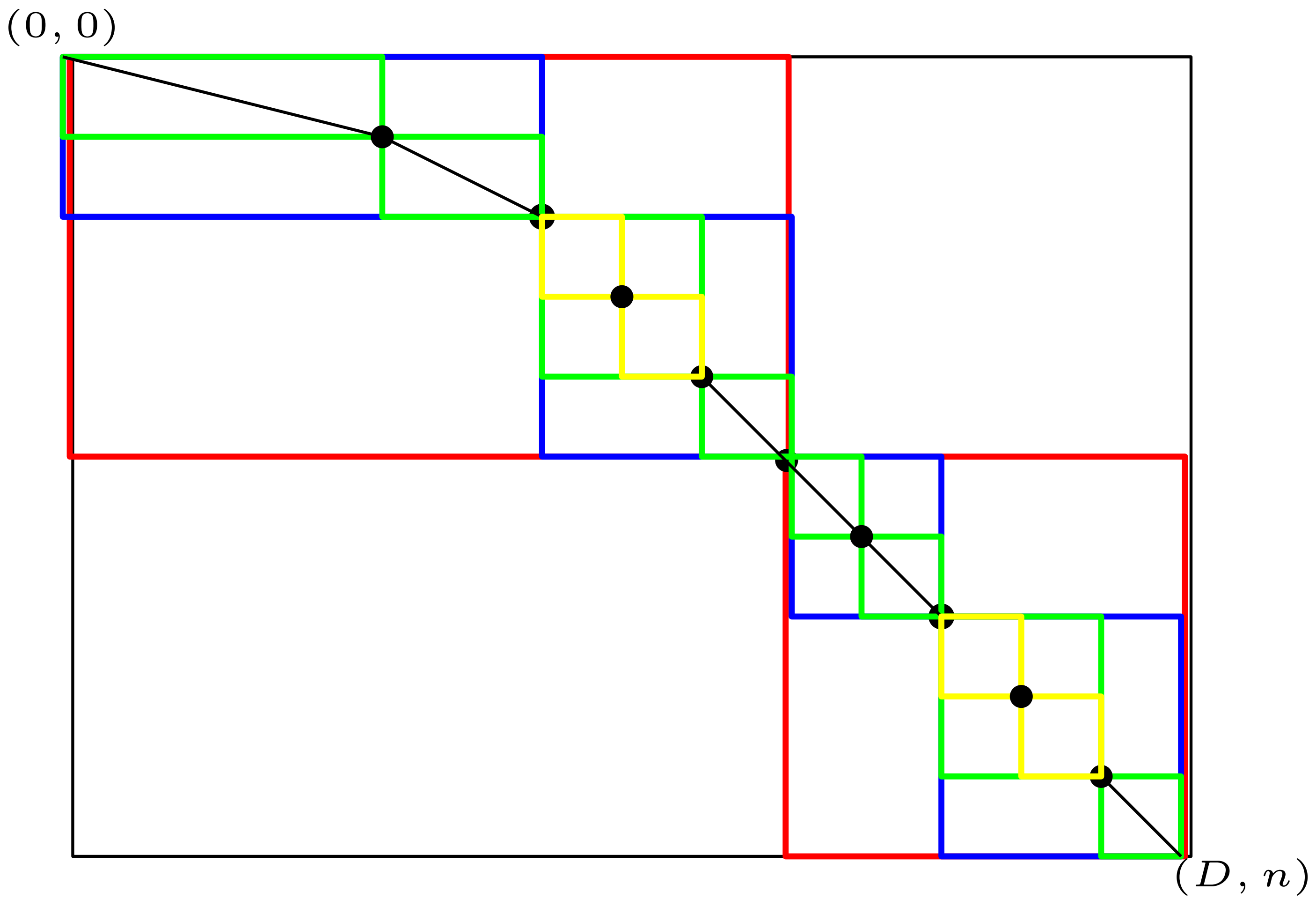
return $(x \to y)$

else

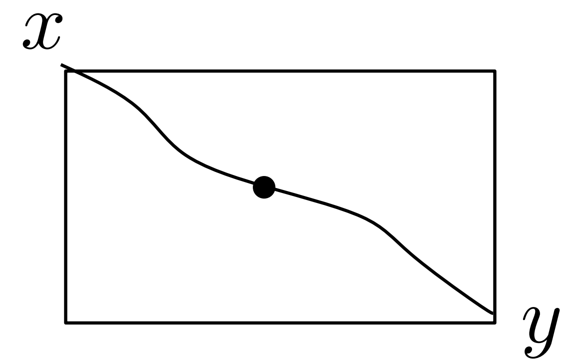
$z = Mid(x, y)$

Buildpath(x,z)

Buildpath(z,y)



$Mid(x, y)$ returns a midpoint (hop distance) on some min-cost x-y path.



We now have a simple recursive procedure for building min-cost path

Buildpath(x,y)

If $y_d = x_{d+1}$

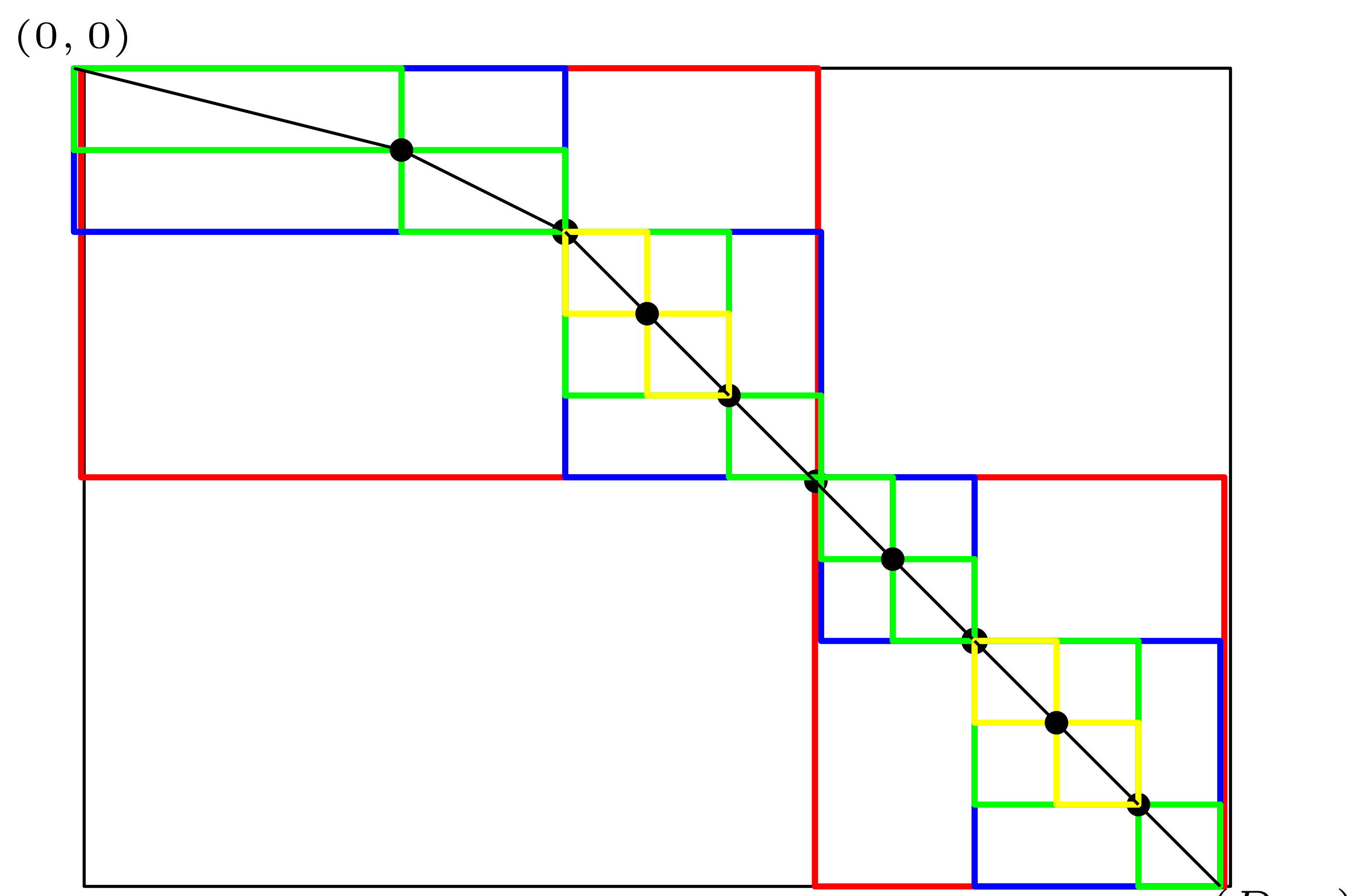
return $(x \to y)$

else

$z = Mid(x, y)$

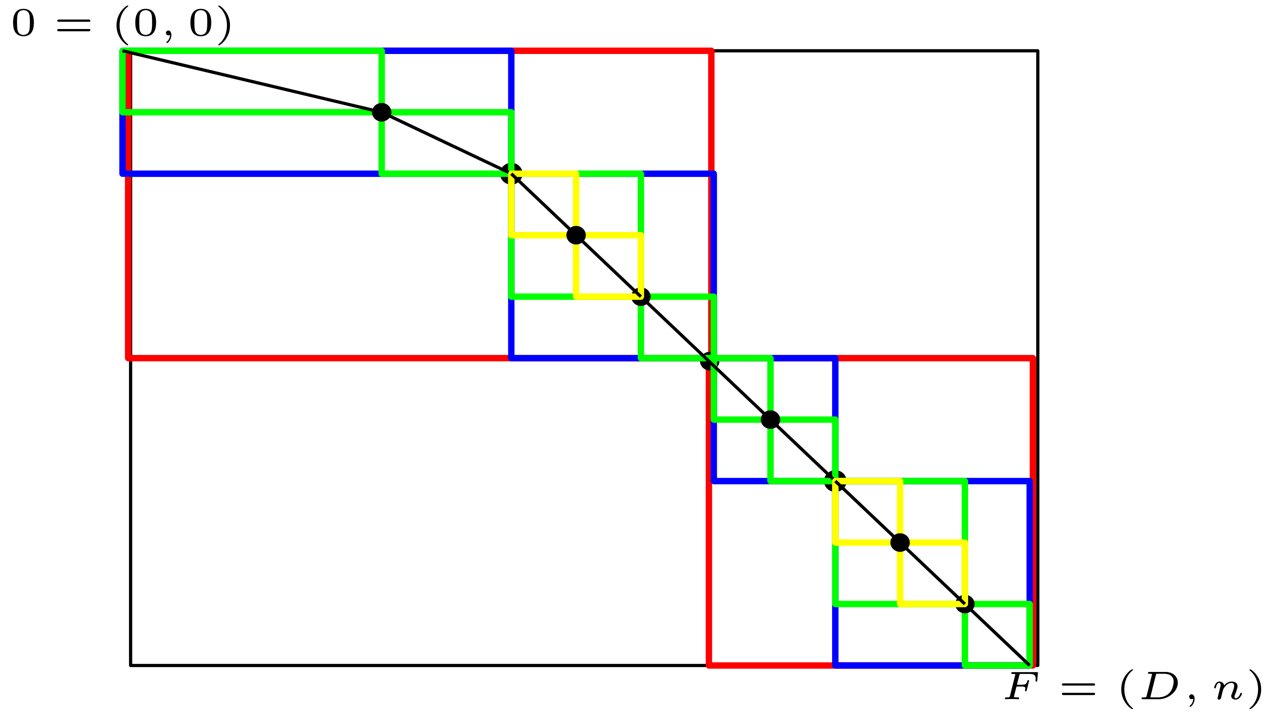
Buildpath(x,z)

Buildpath(z,y)



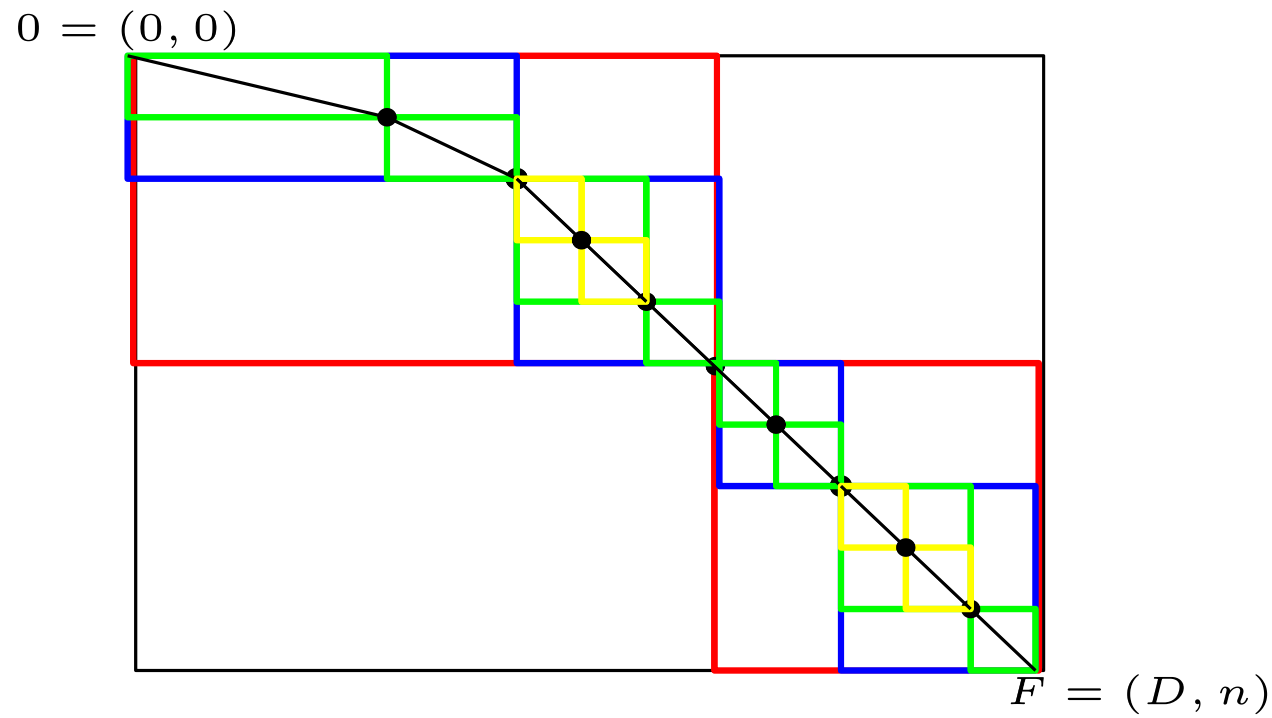
Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \longrightarrow y)$
else
  $z = Mid(x, y)$
  Buildpath(x,z)
  Buildpath(z,y)



Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \longrightarrow y)$
else
  $z = Mid(x, y)$
  Buildpath(x,z)
  Buildpath(z,y)

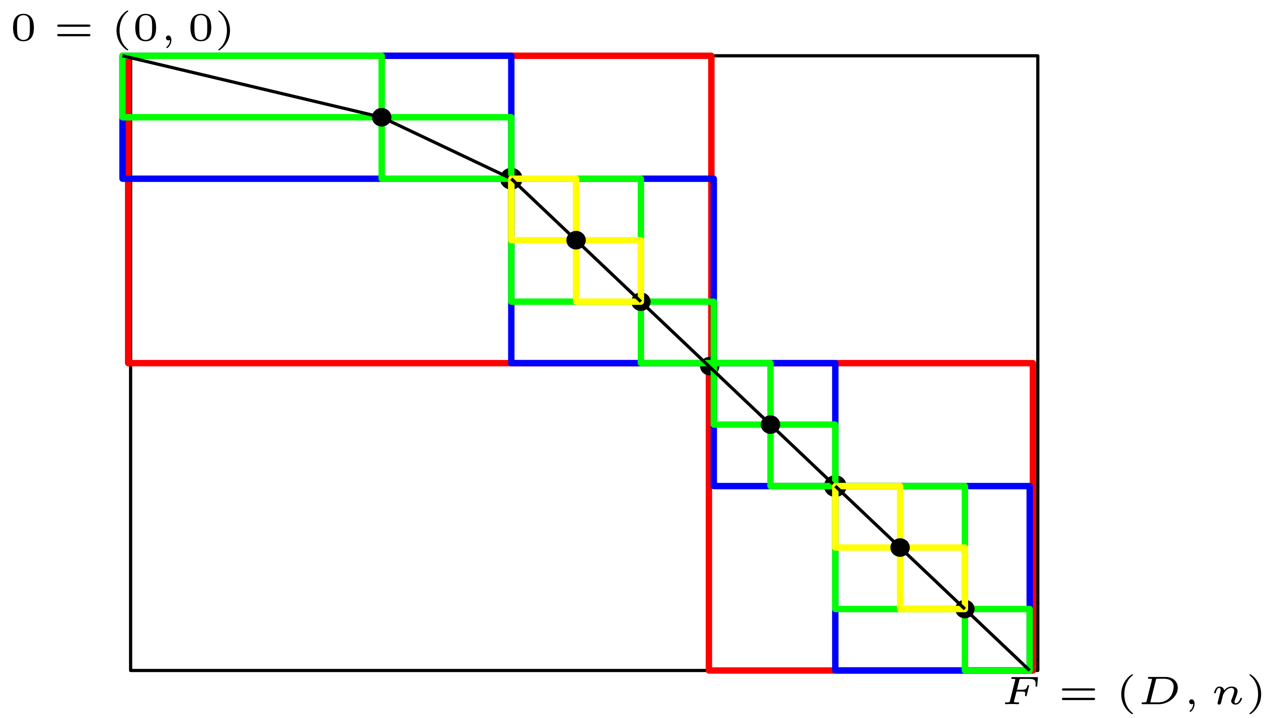


Lemma: If $Mid(x, y)$ uses $O(D + n)$ space

$\Longrightarrow$ Buildpath(0,F) uses $O(D + n)$ space

Buildpath(x,y)

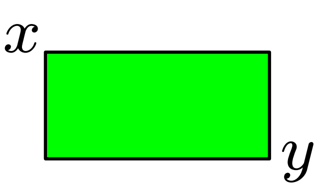
If $y_d = x_{d+1}$
  return $(x \rightarrow y)$
else
  $z = Mid(x,y)$
  Buildpath(x,z)
  Buildpath(z,y)



Lemma: If $Mid(x,y)$ uses $O(D+n)$ space

$\Longrightarrow$ Buildpath(0,F) uses $O(D+n)$ space

Lemma: Let $Area(x,y)$ be area of x,y box

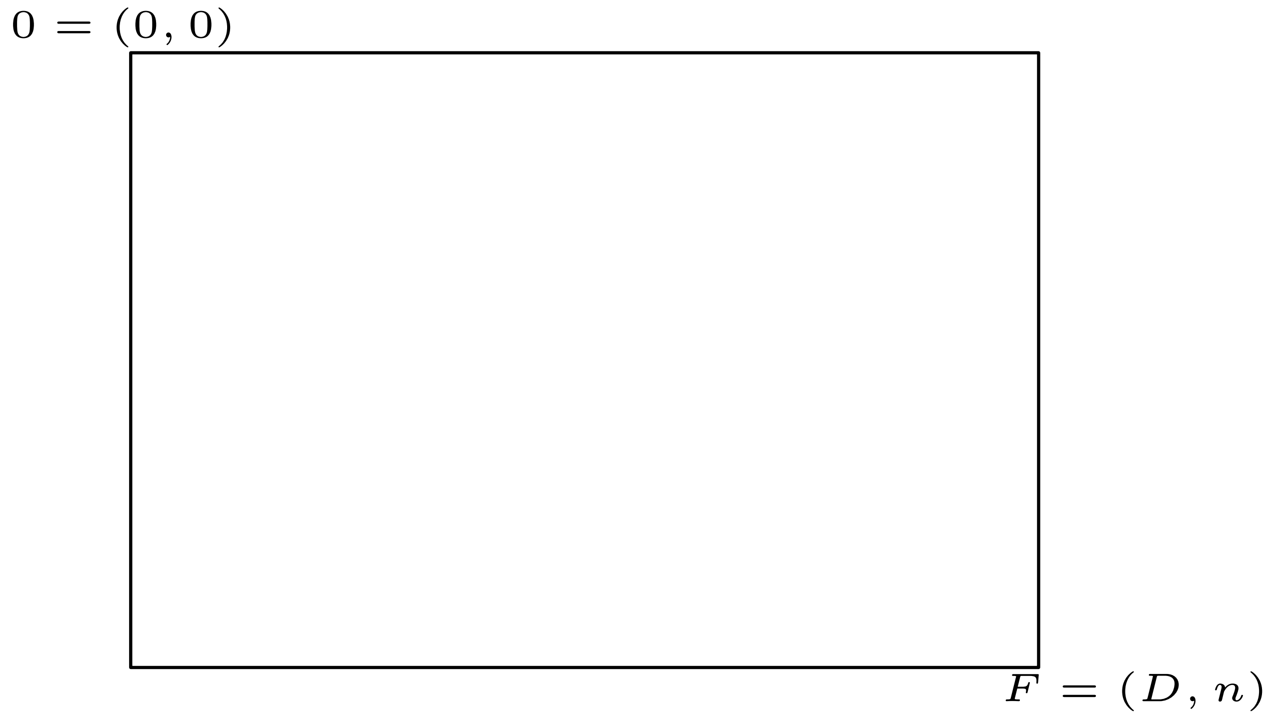


If $Mid(x,y)$ uses $O(Area(x,y))$ time

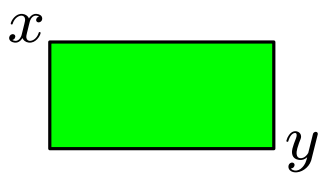
$\Longrightarrow$ Buildpath(0,F) uses $O(Dn)$ time

Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \rightarrow y)$
else
  $z = Mid(x, y)$
  Buildpath(x,z)
  Buildpath(z,y)



Lemma: Let $Area(x, y)$ be area of x, y box

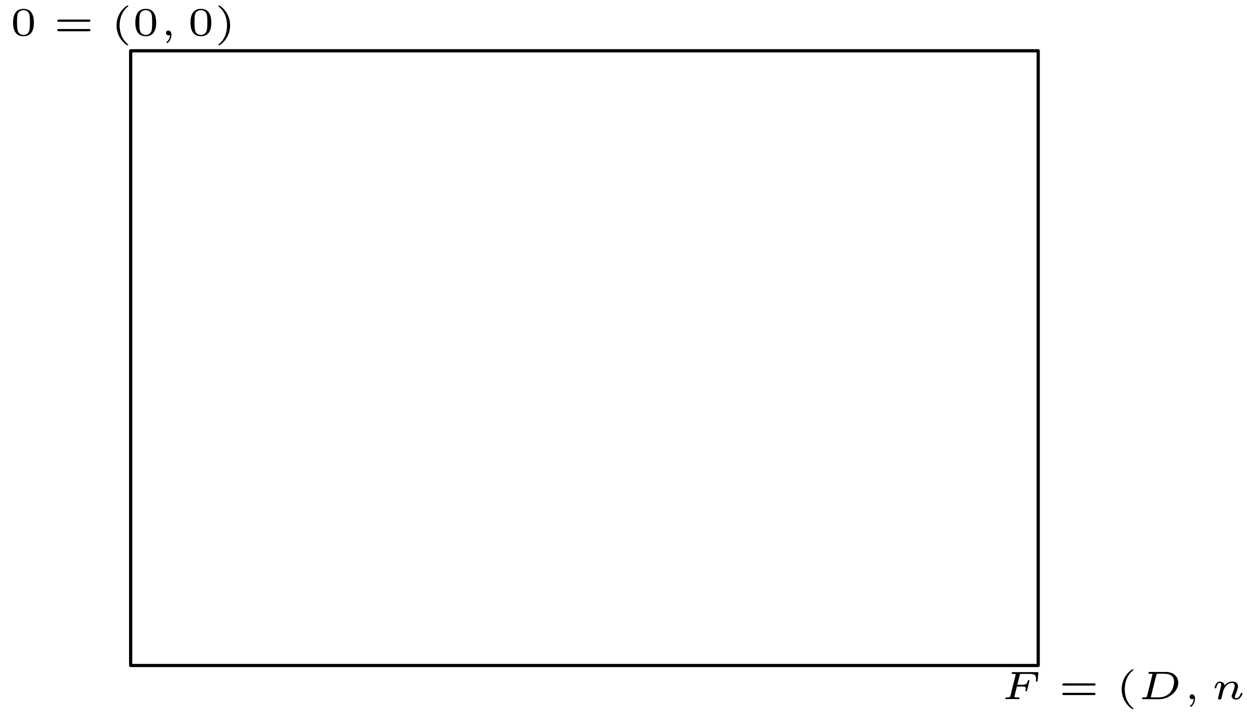


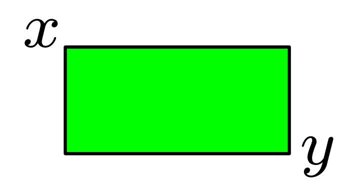
If $Mid(x, y)$ uses $O(Area(x, y))$ time

$\Longrightarrow$ Buildpath(0,F) uses $O(Dn)$ time

Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \longrightarrow y)$
else
  $z = Mid(x, y)$
  Buildpath(x,z)
  Buildpath(z,y)





Lemma: Let $Area(x, y)$ be area of x, y box

If $Mid(x, y)$ uses $O(Area(x, y))$ time

$\Longrightarrow$ Buildpath(0,F) uses $O(Dn)$ time

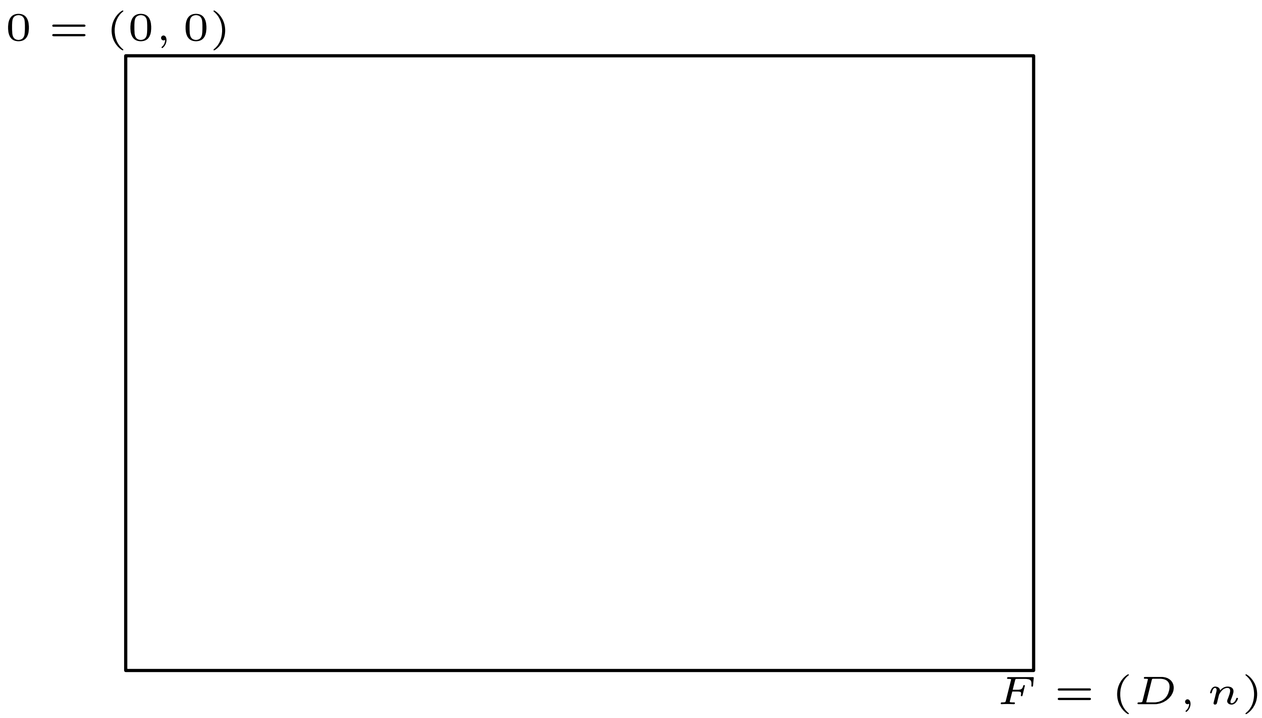
Proof: Rectangles at recursion level i are height $\leq D/2^i$

$\Longrightarrow$ Total work at level i is $\leq nD/2^i$

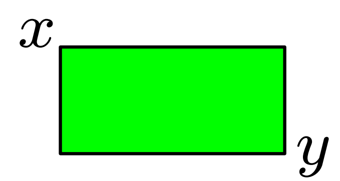
$\Longrightarrow$ Total work $\leq$

Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \rightarrow y)$
else
  $z = Mid(x, y)$
  Buildpath(x,z)
  Buildpath(z,y)



Lemma: Let $Area(x, y)$ be area of x, y box



If $Mid(x, y)$ uses $O(Area(x, y))$ time

$\Longrightarrow$ Buildpath(0,F) uses $O(Dn)$ time

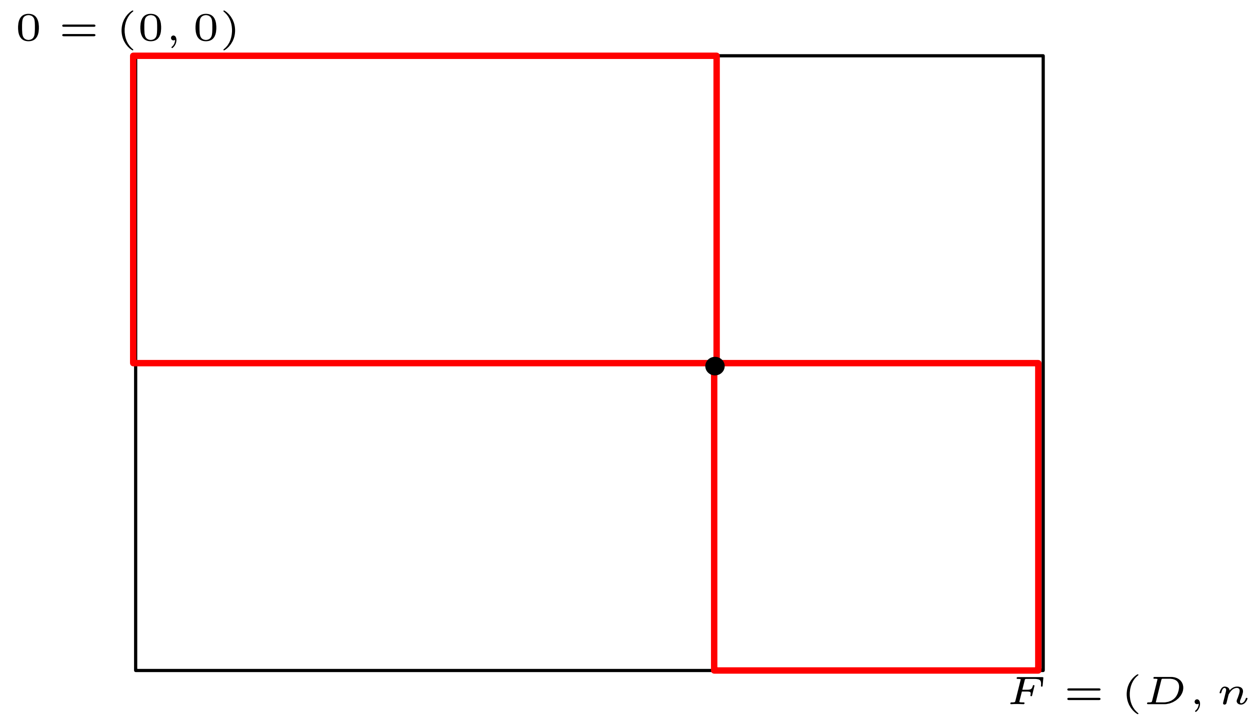
Proof: Rectangles at recursion level i are height $\leq D/2^i$

$\Longrightarrow$ Total work at level i is $\leq nD/2^i$

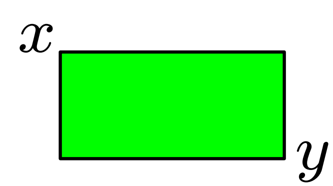
$\Longrightarrow$ Total work $\leq n \Big(\dfrac{D}{2^0}$

Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \to y)$
else
  $z = Mid(x, y)$
  Buildpath(x,z)
  Buildpath(z,y)



Lemma: Let $Area(x, y)$ be area of x, y box



If $Mid(x, y)$ uses $O(Area(x, y))$ time

$\Longrightarrow$ Buildpath(0,F) uses $O(Dn)$ time

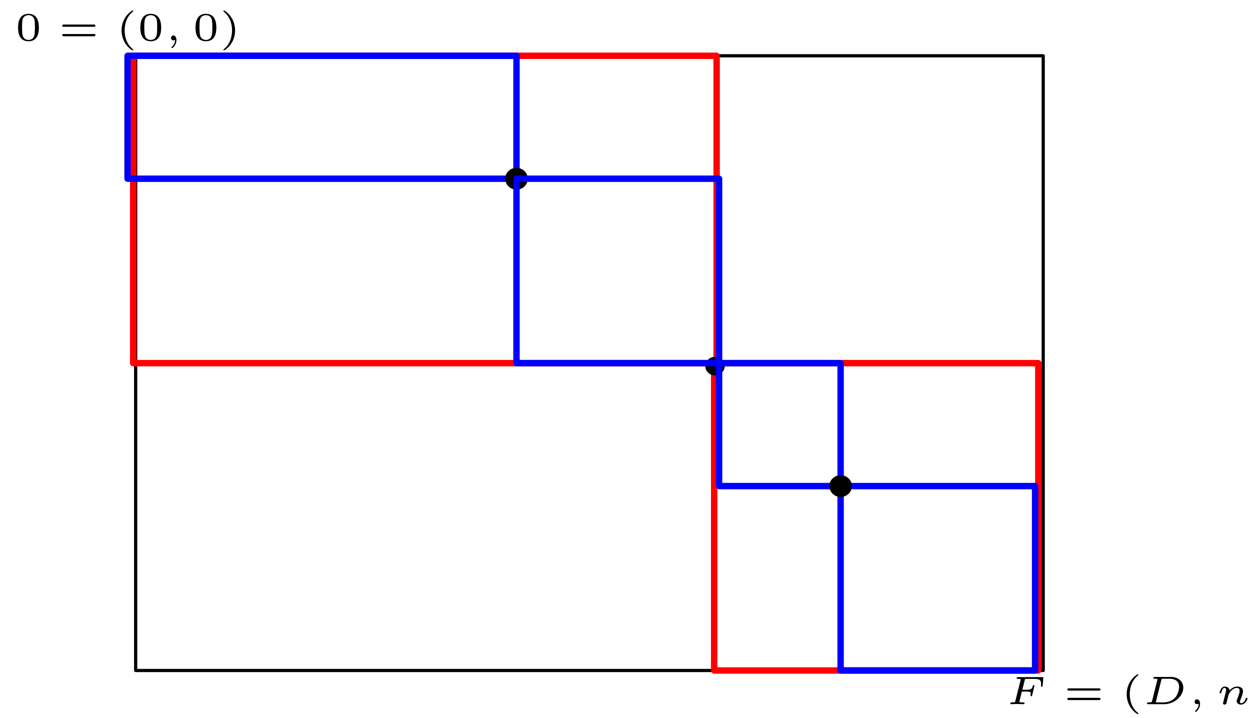
Proof: Rectangles at recursion level i are height $\leq D/2^i$

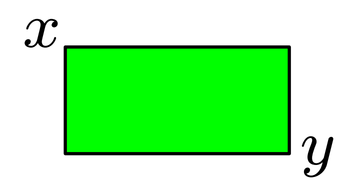
$\Longrightarrow$ Total work at level i is $\leq nD/2^i$

$\Longrightarrow$ Total work $\quad \leq n \left(\frac{D}{2^0} + \frac{D}{2^1} \right.$

Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \to y)$
else
  $z = Mid(x,y)$
  Buildpath(x,z)
  Buildpath(z,y)





Lemma: Let $Area(x,y)$ be area of x,y box

If $Mid(x,y)$ uses $O(Area(x,y))$ time

$\Longrightarrow$ Buildpath(0,F) uses $O(Dn)$ time

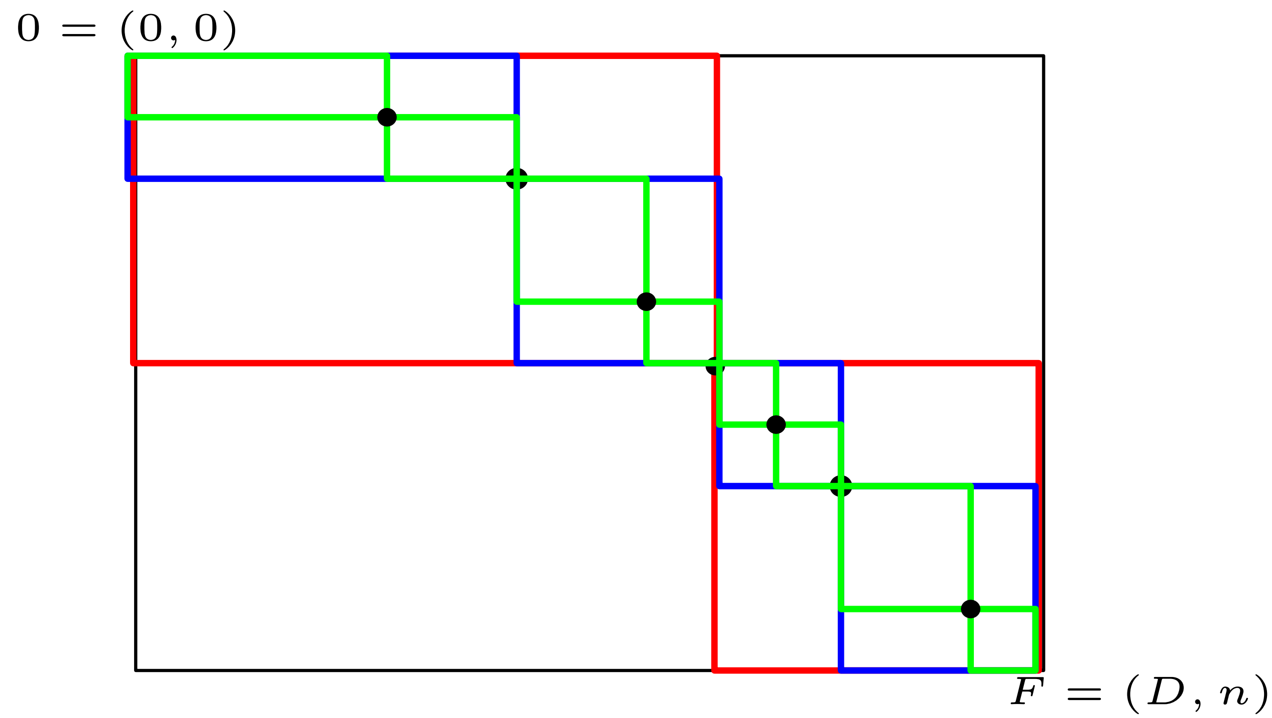
Proof: Rectangles at recursion level i are height $\le D/2^i$

$\Longrightarrow$ Total work at level i is $\le nD/2^i$

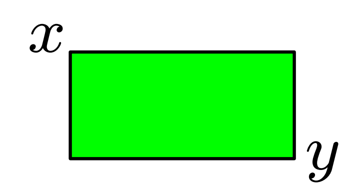
$\Longrightarrow$ Total work $\le n\left(\frac{D}{2^0} + \frac{D}{2^1} + \frac{D}{2^2}\right.$

Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \to y)$
else
  $z = Mid(x,y)$
  Buildpath(x,z)
  Buildpath(z,y)



Lemma: Let $Area(x,y)$ be area of x,y box



If $Mid(x,y)$ uses $O(Area(x,y))$ time

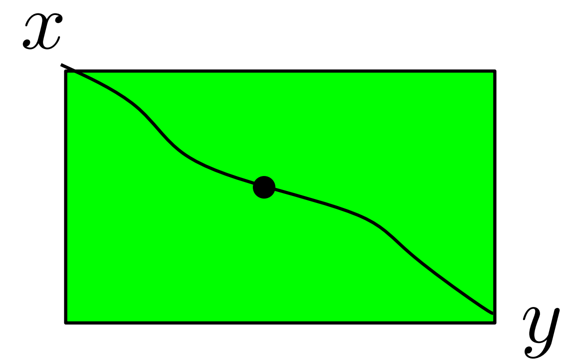
$\Longrightarrow$ Buildpath(0,F) uses $O(Dn)$ time

Proof: Rectangles at recursion level i are height $\leq D/2^i$

$\Longrightarrow$ Total work at level i is $\leq nD/2^i$

$\Longrightarrow$ Total work $\leq n\left(\frac{D}{2^0} + \frac{D}{2^1} + \frac{D}{2^2} + \frac{D}{2^3} + \cdots\right) \leq 2nD$

Just saw that if $Mid(x, y)$ can be implemented using $O(D + n)$ space and $Area(x, y)$ time, then path can be built using $O(D + n)$ space and $O(Dn)$ time.



There are two different methods in literature for implementing $Mid(x, y)$. They can both be used here, but we will use (b).

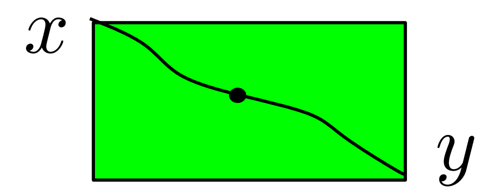
(a) Hirschberg ('75)

For longest common subsequence problem.
Runs two modified Dijkstra's that meet in "middle"
Every vertex had constant outdegree (≤ 3)
Used extensively in bioinformatics.

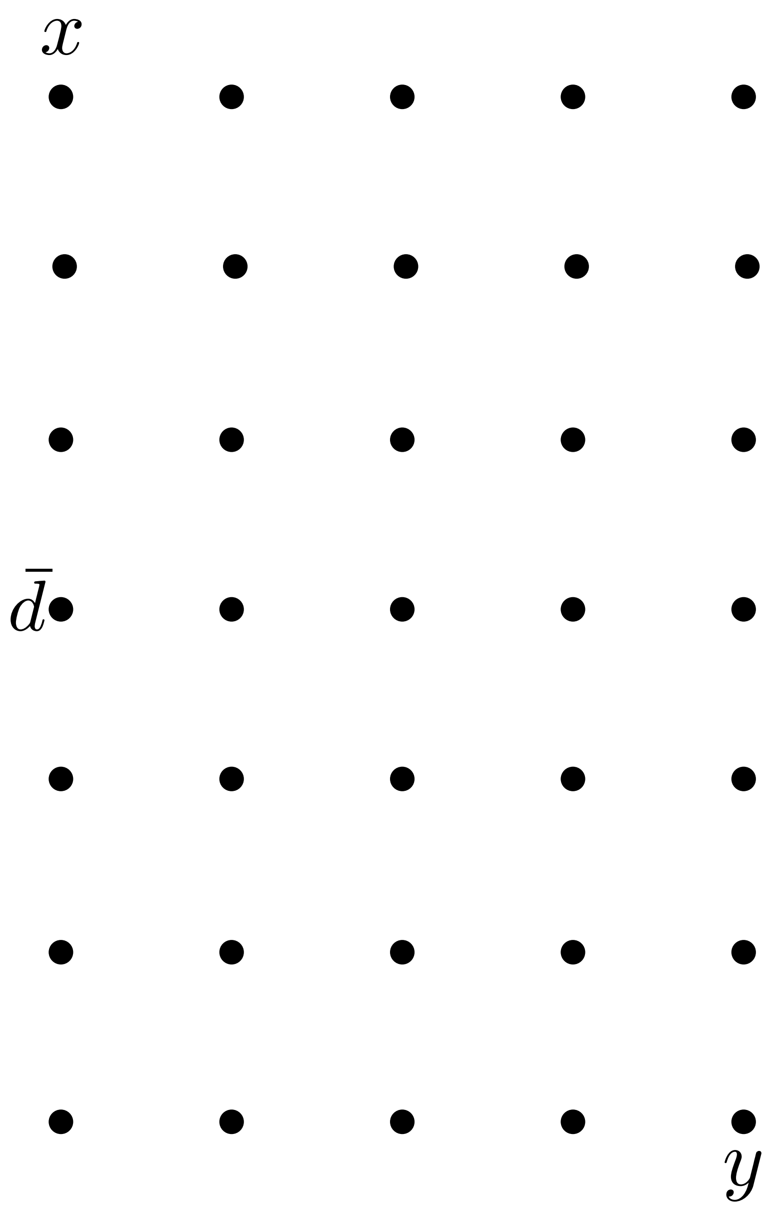
(b) Munro & Ramirez ('82)

For graphs like our's
Runs one modified Dijkstra
Uses $\Theta(Dn^2)$ time (we can improve to $\Theta(Dn)$ with Monge)

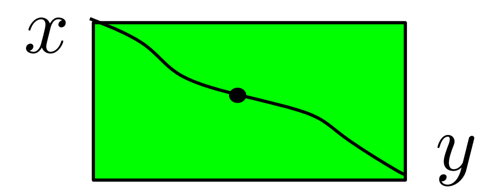
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



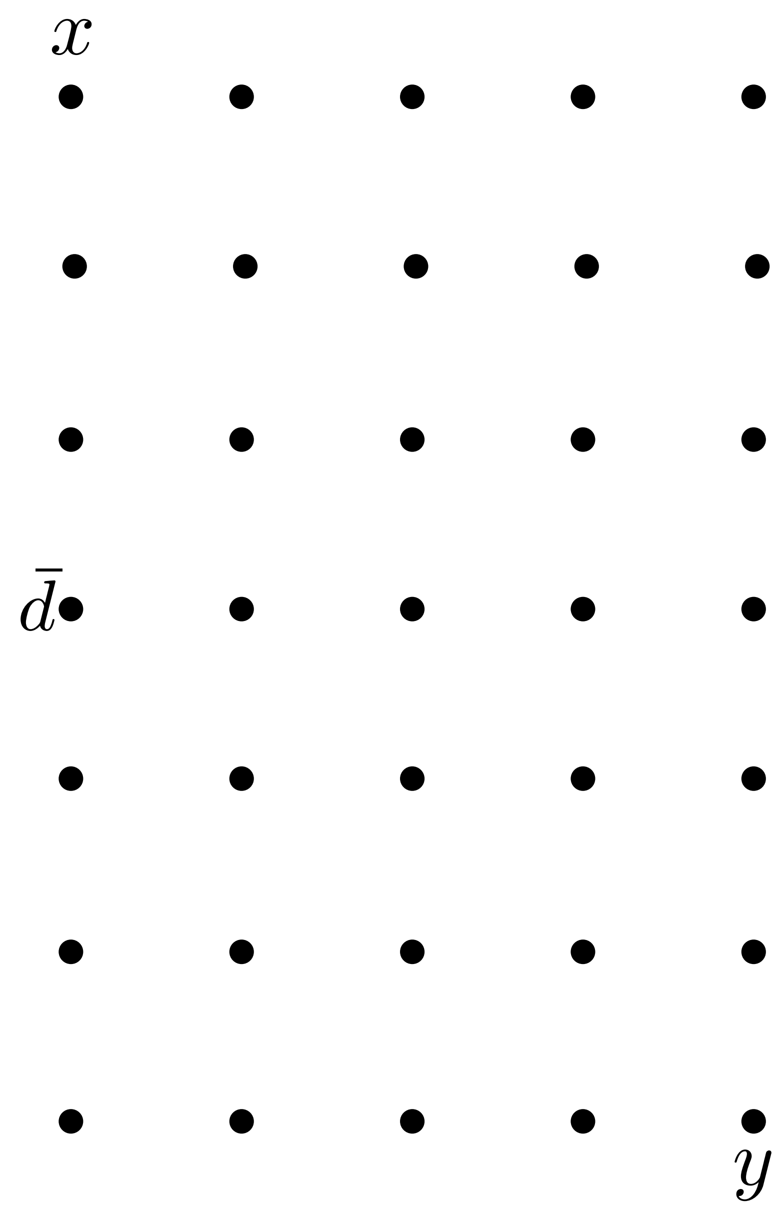
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



Implementing $Mid(x,y)$ in $O(D+n)$ space and $Area(x,y)$ time

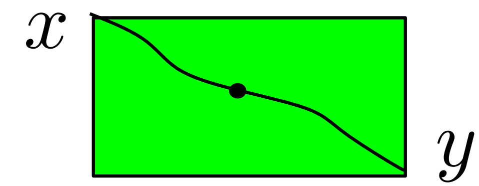


For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

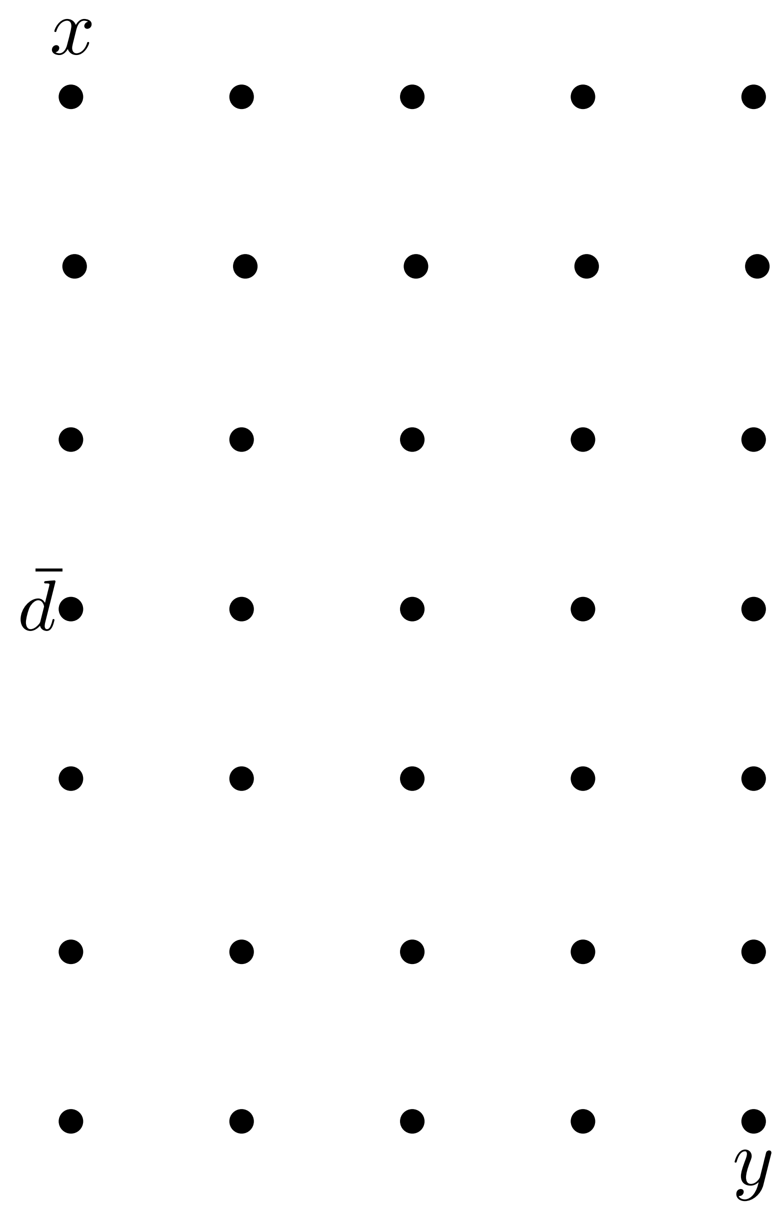
Implementing $Mid(x,y)$ in $O(D+n)$ space and $Area(x,y)$ time



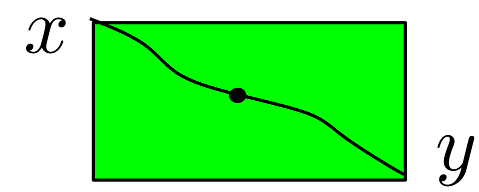
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

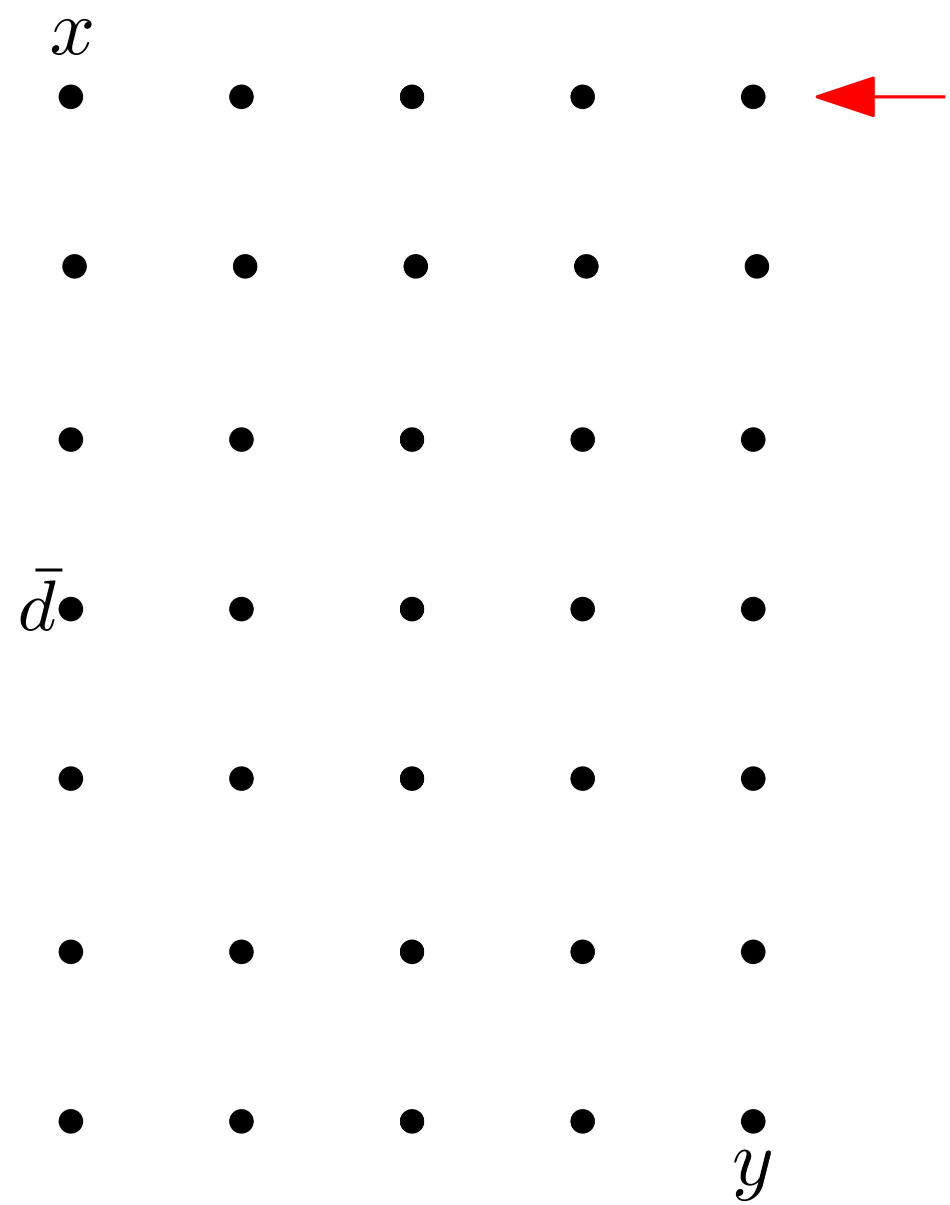
All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d-1$.



Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



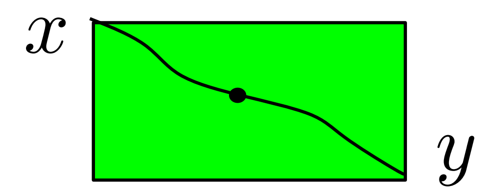
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



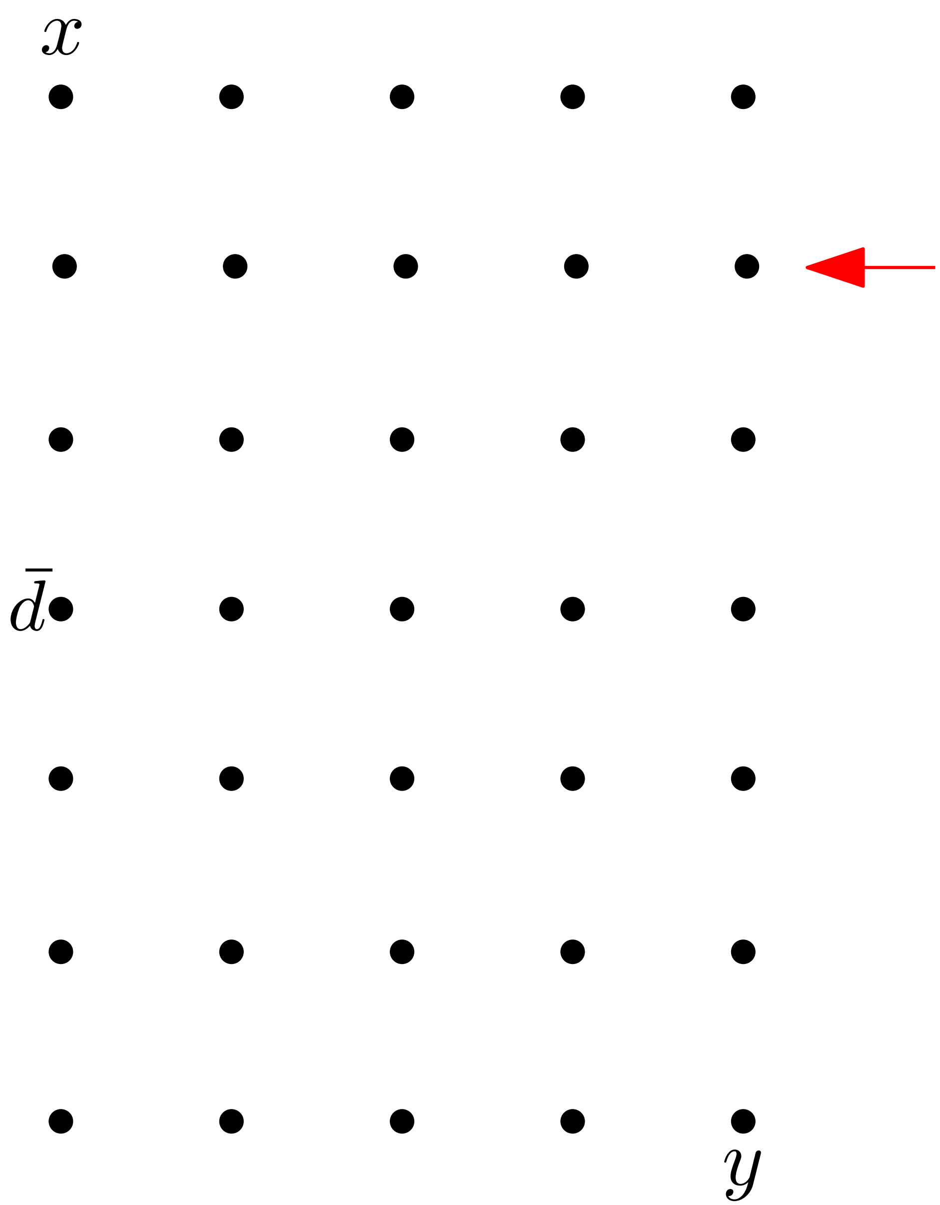
If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.

Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



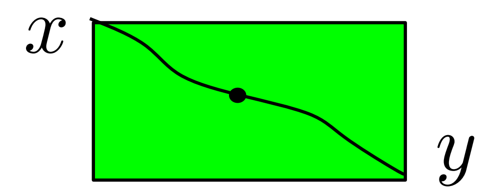
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.

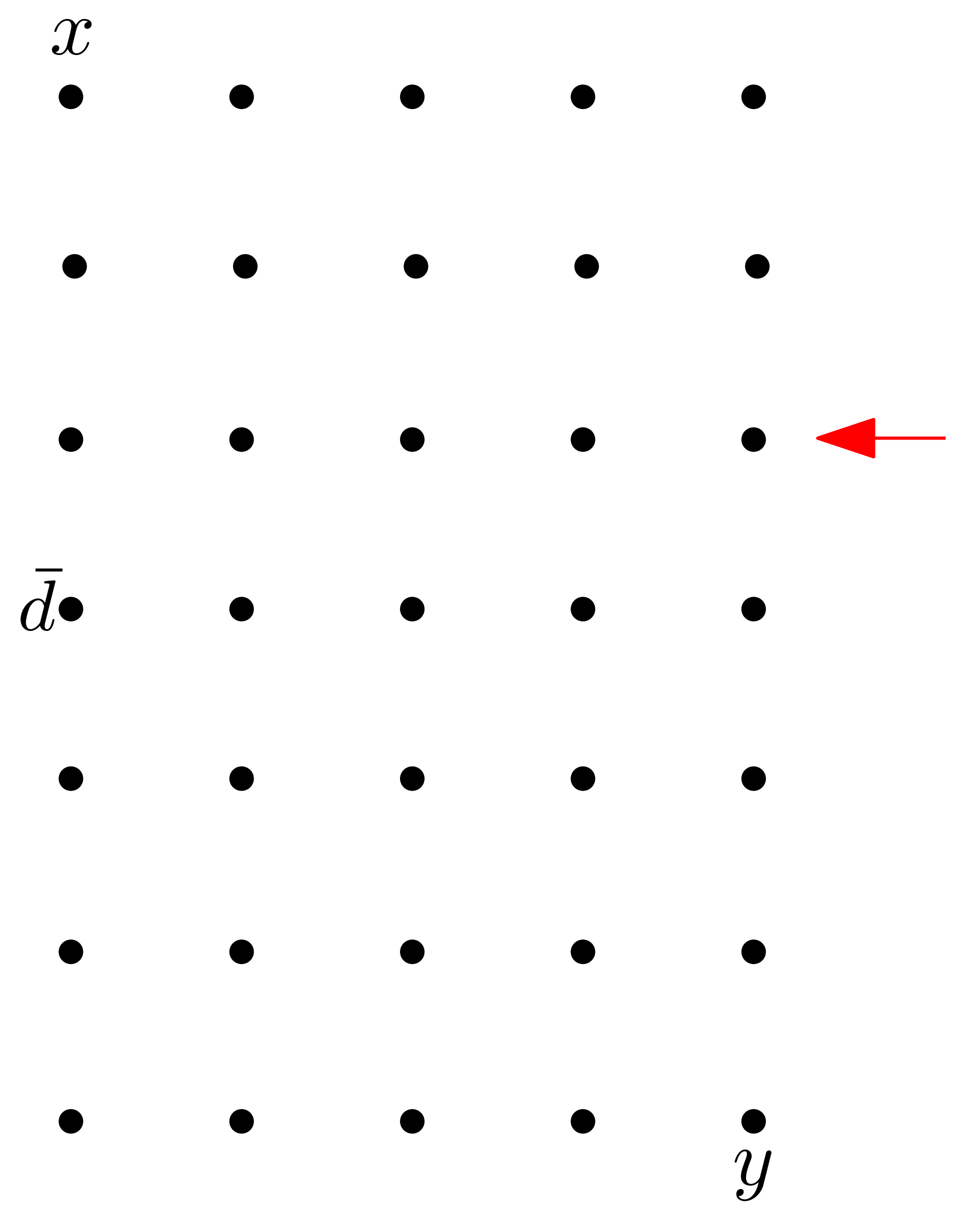
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



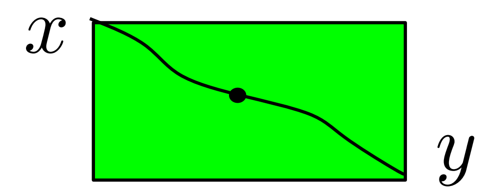
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

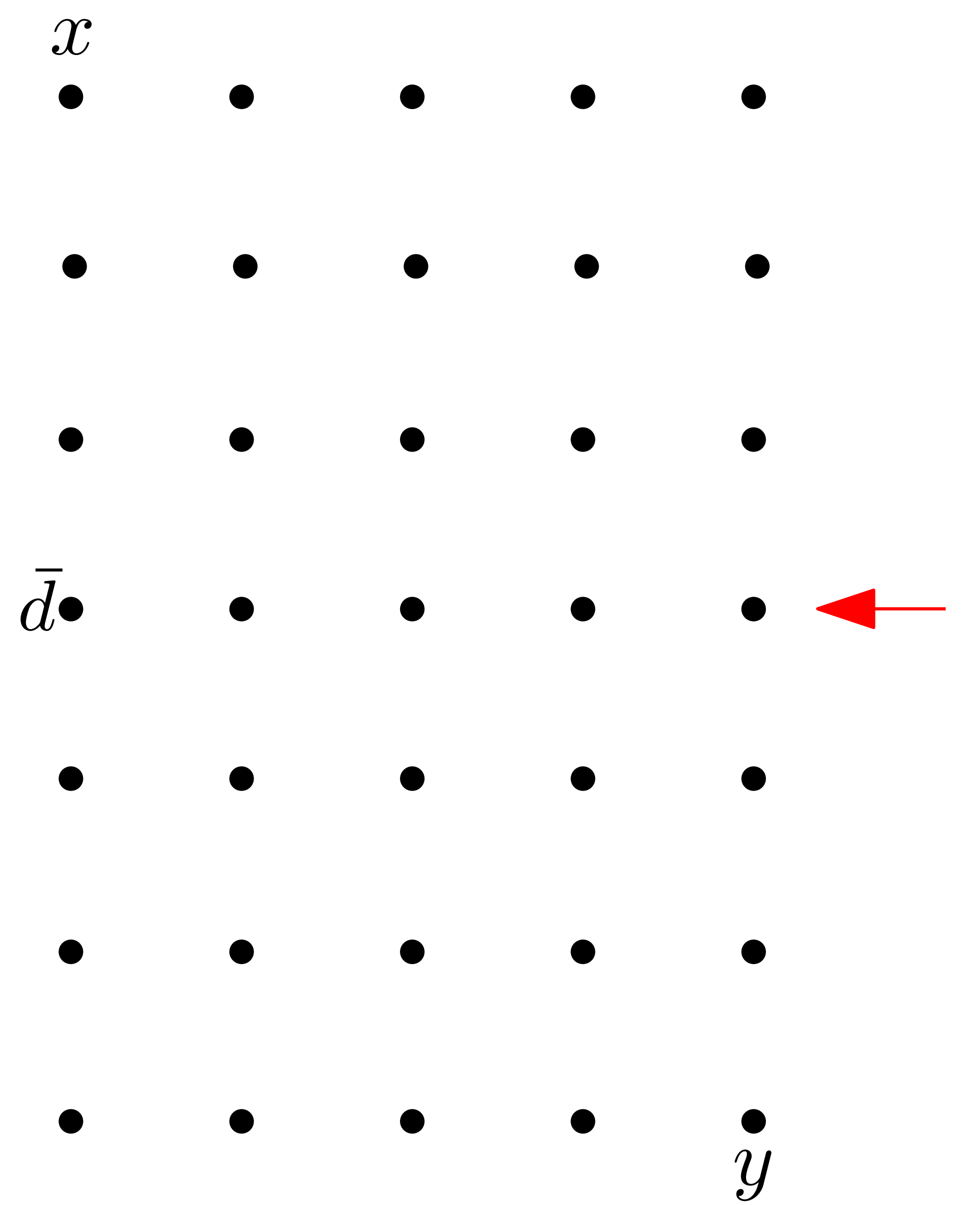
All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.



Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



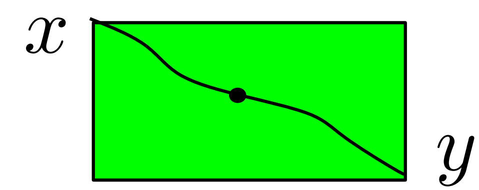
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.

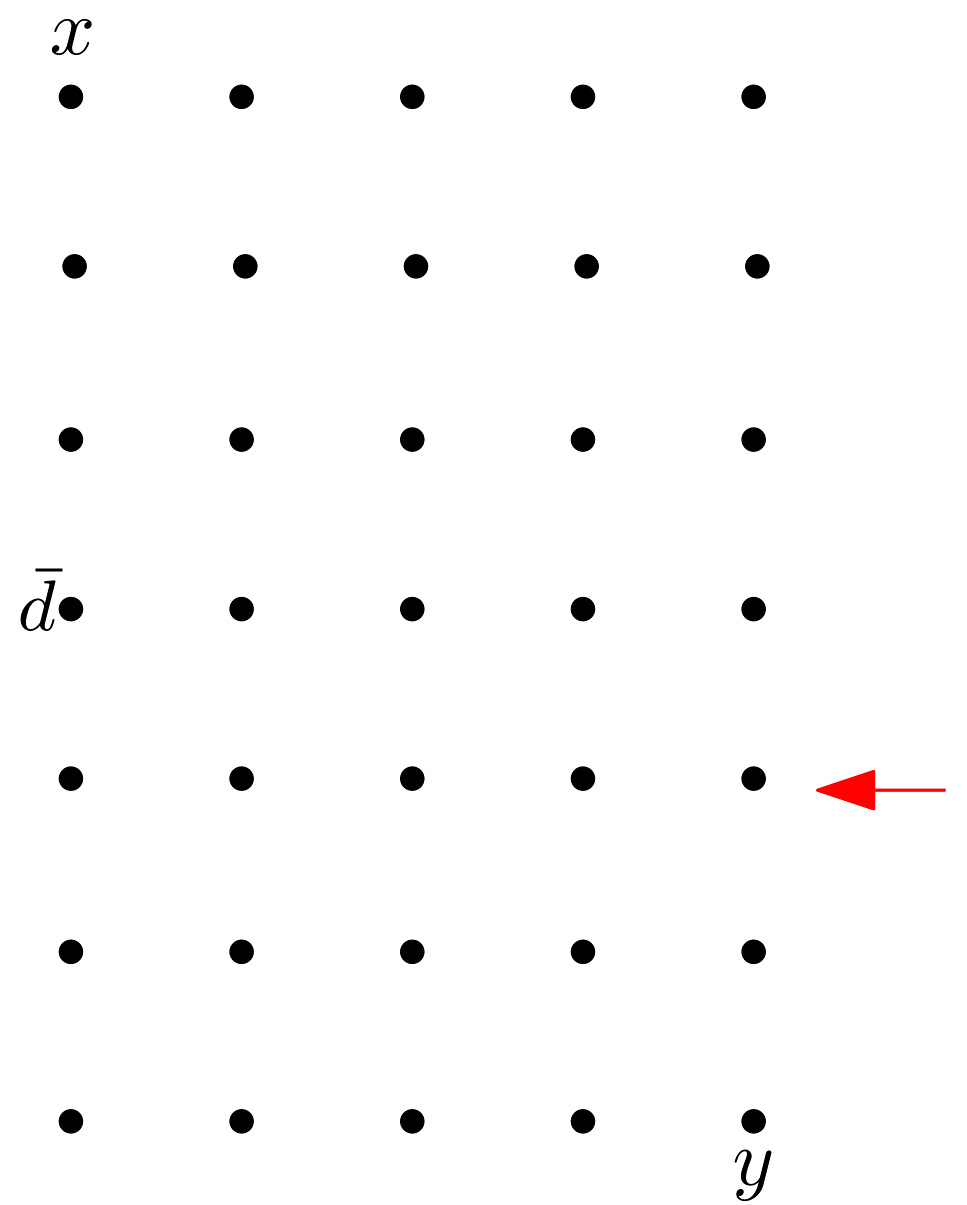
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



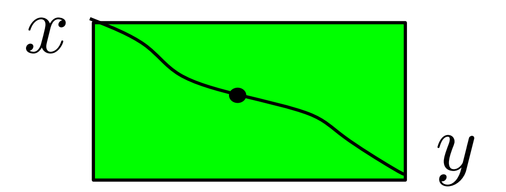
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.



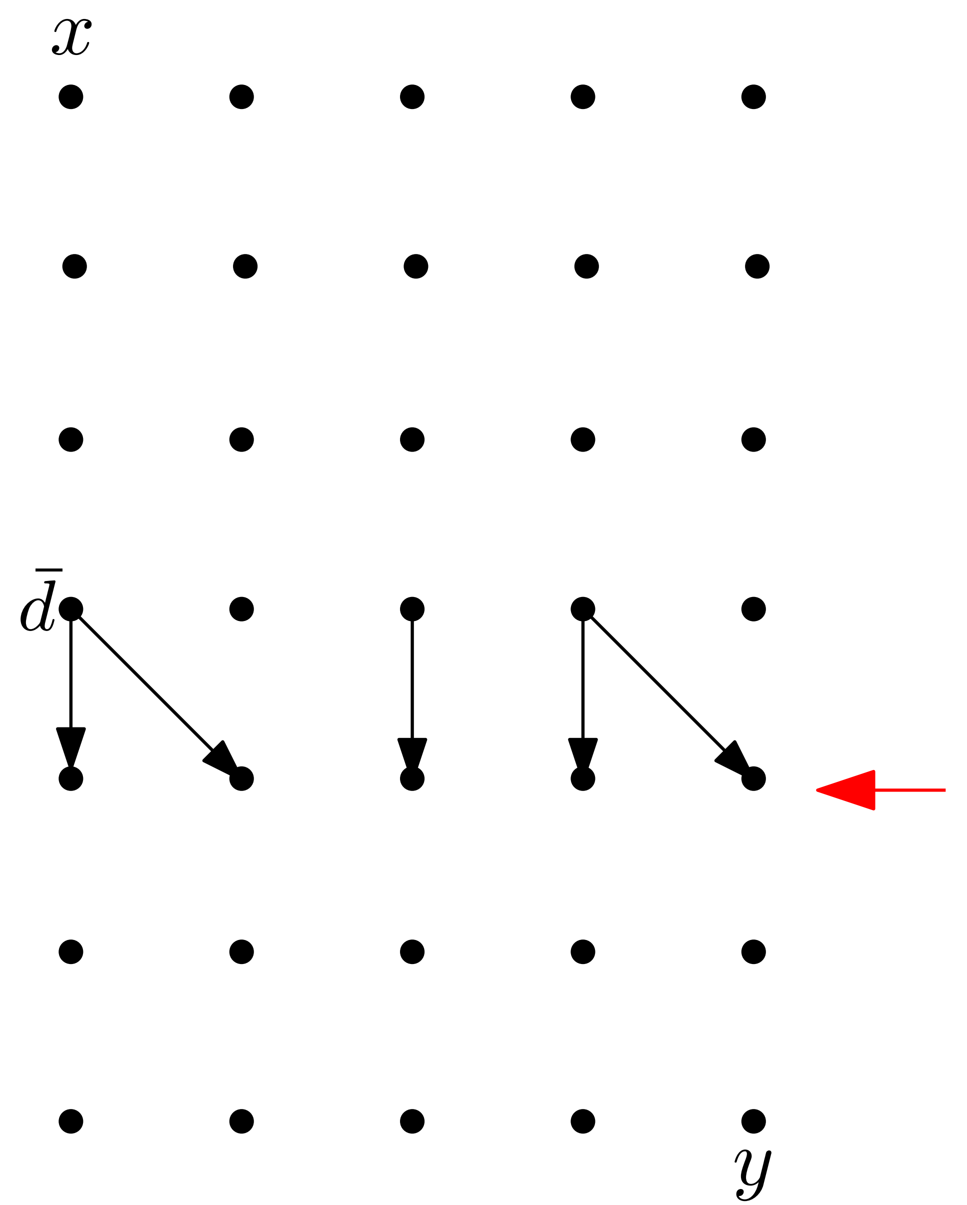
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



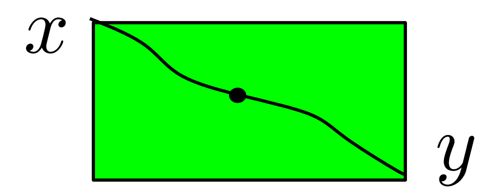
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

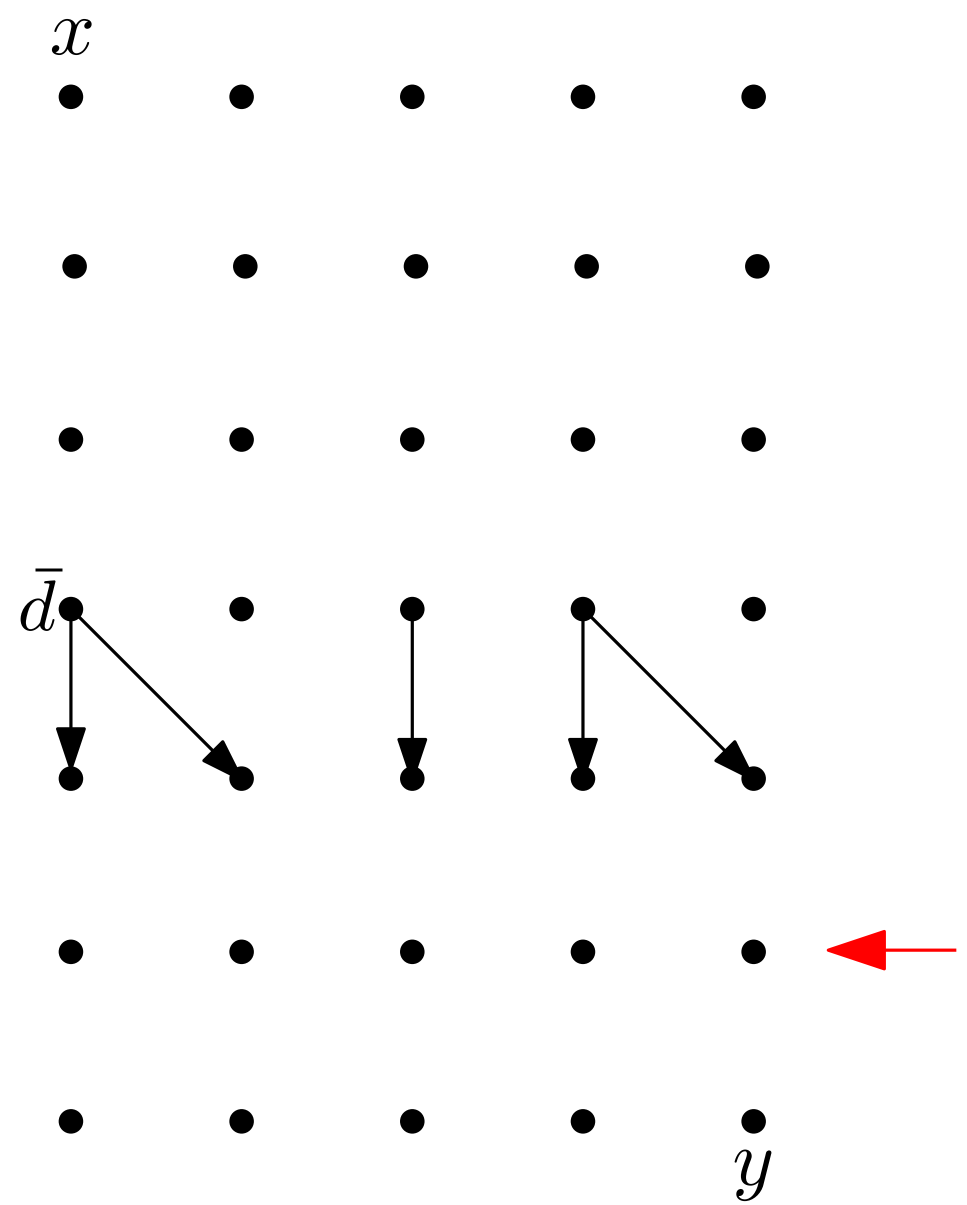
All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.



Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



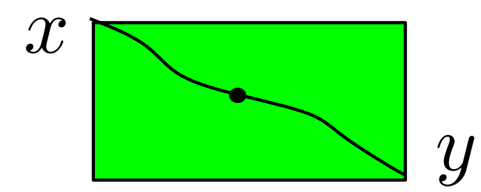
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



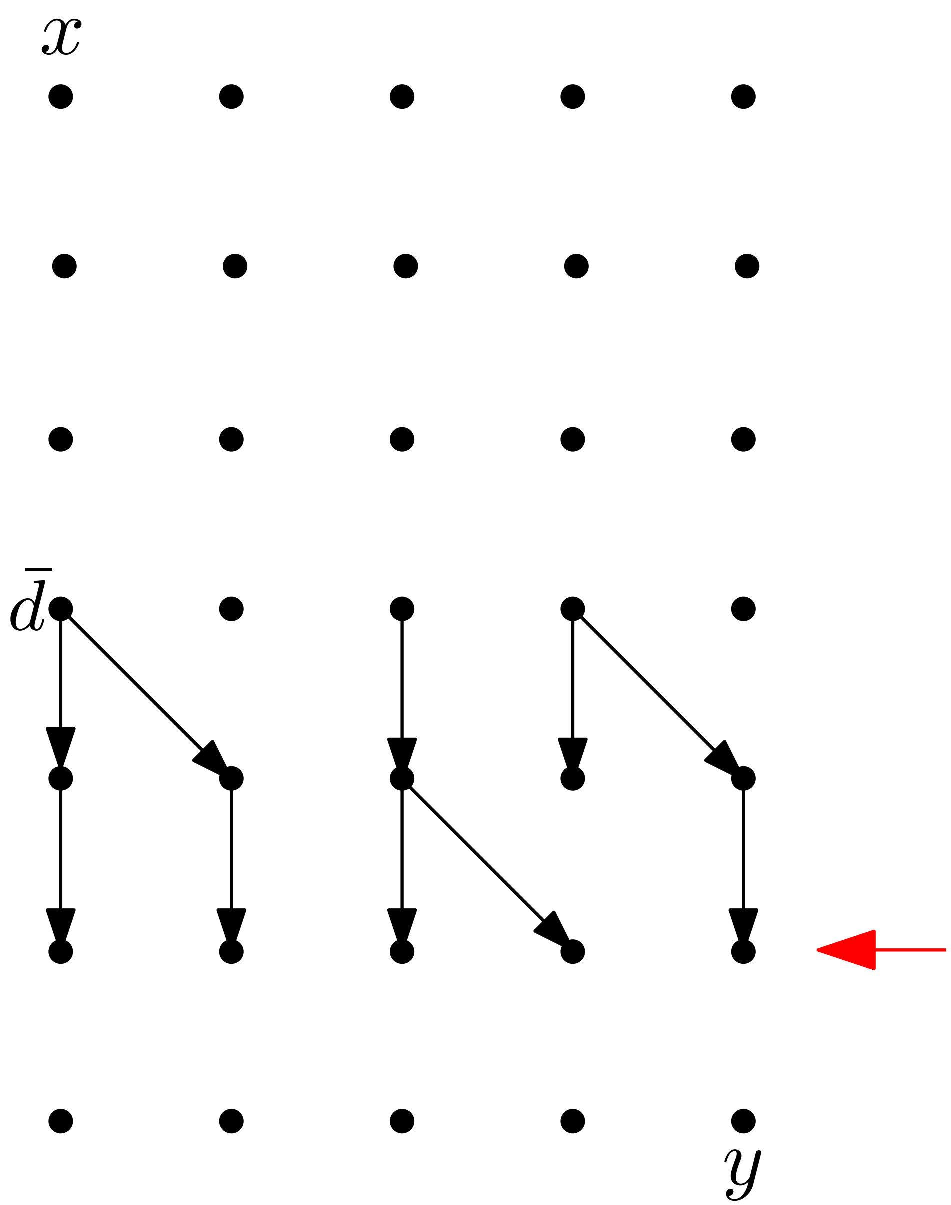
If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.

Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



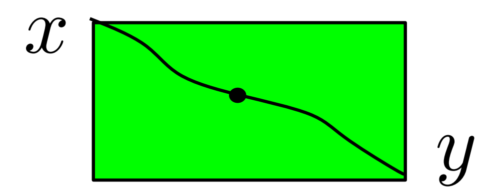
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.

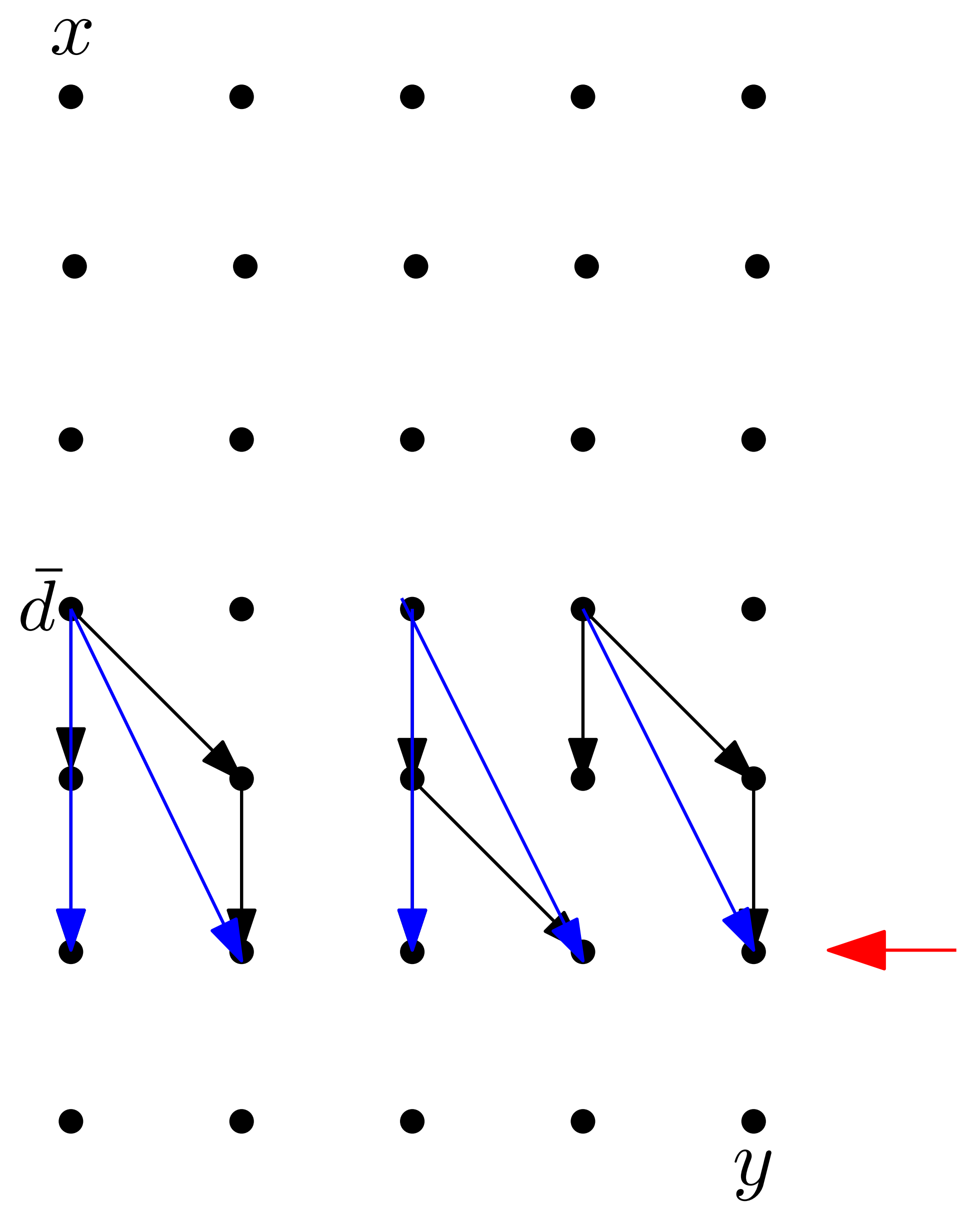
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



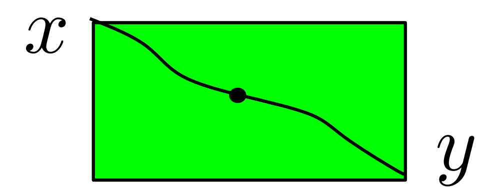
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.



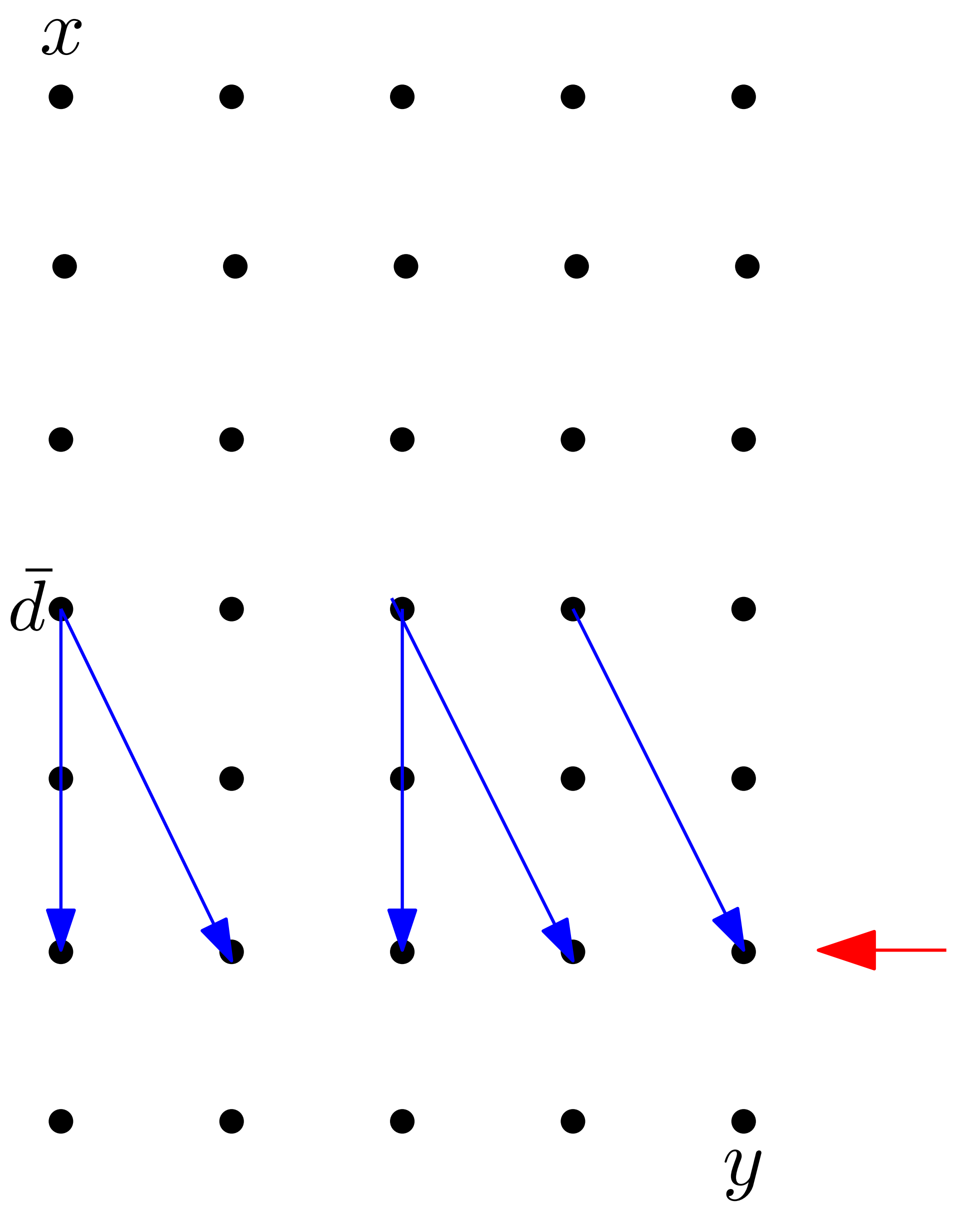
Implementing $Mid(x,y)$ in $O(D+n)$ space and $Area(x,y)$ time



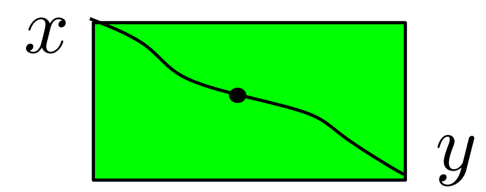
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d-1$.



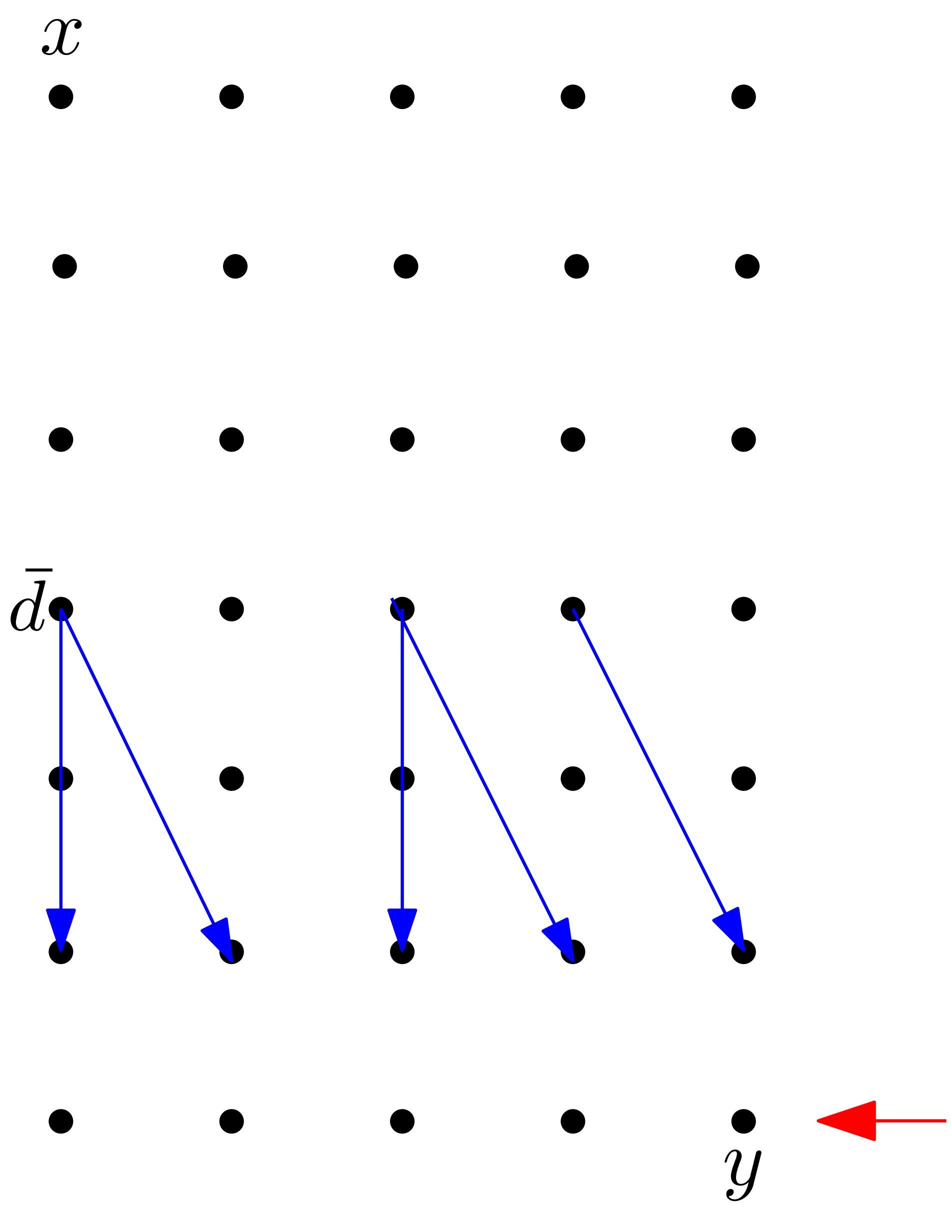
Implementing $Mid(x,y)$ in $O(D+n)$ space and $Area(x,y)$ time



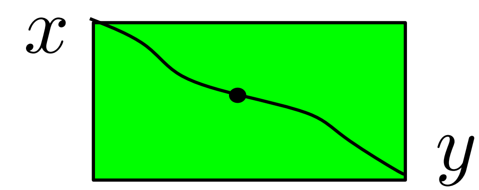
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d-1$.



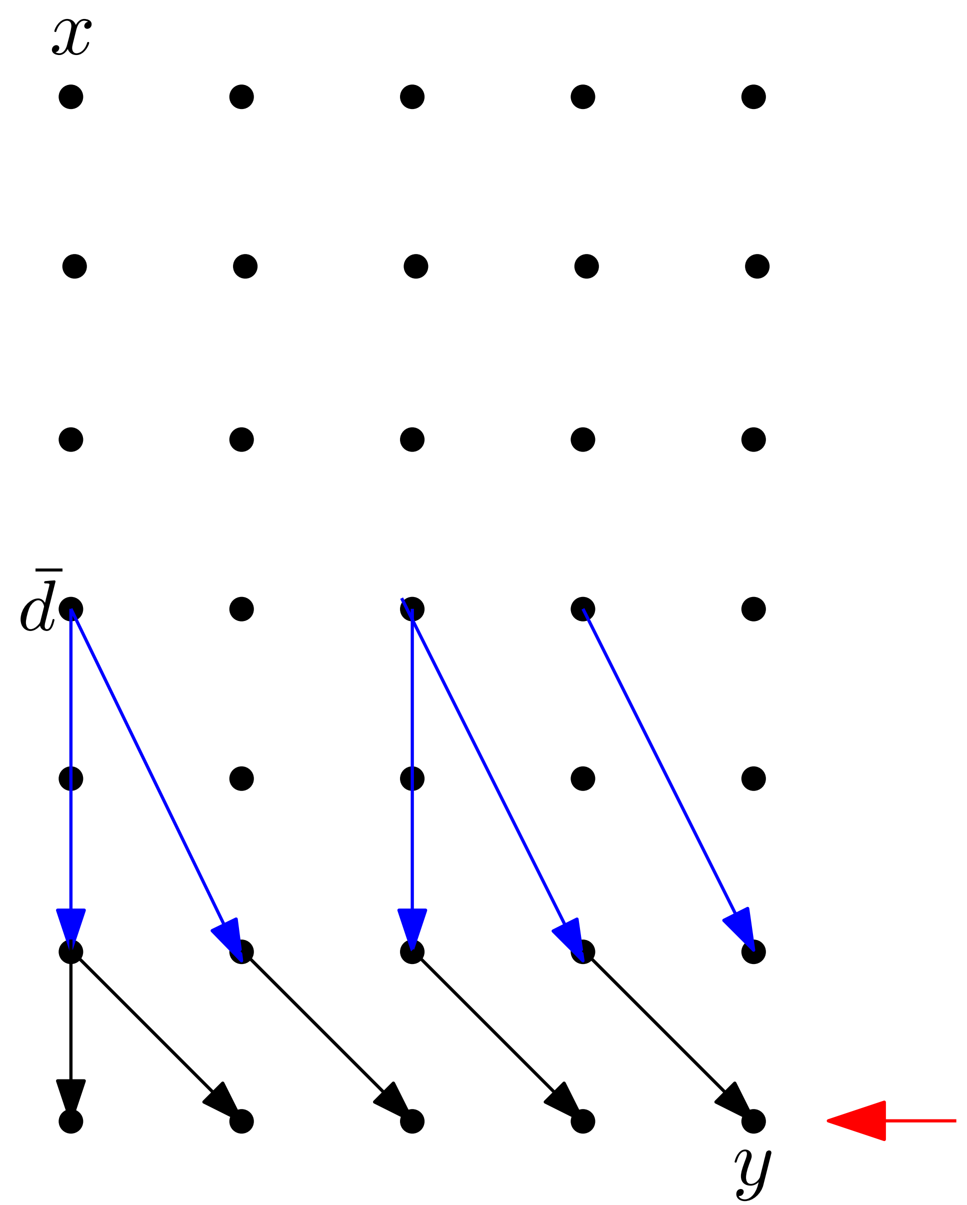
Implementing $Mid(x,y)$ in $O(D+n)$ space and $Area(x,y)$ time



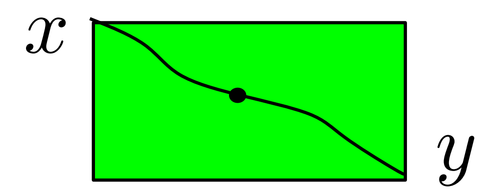
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d-1$.



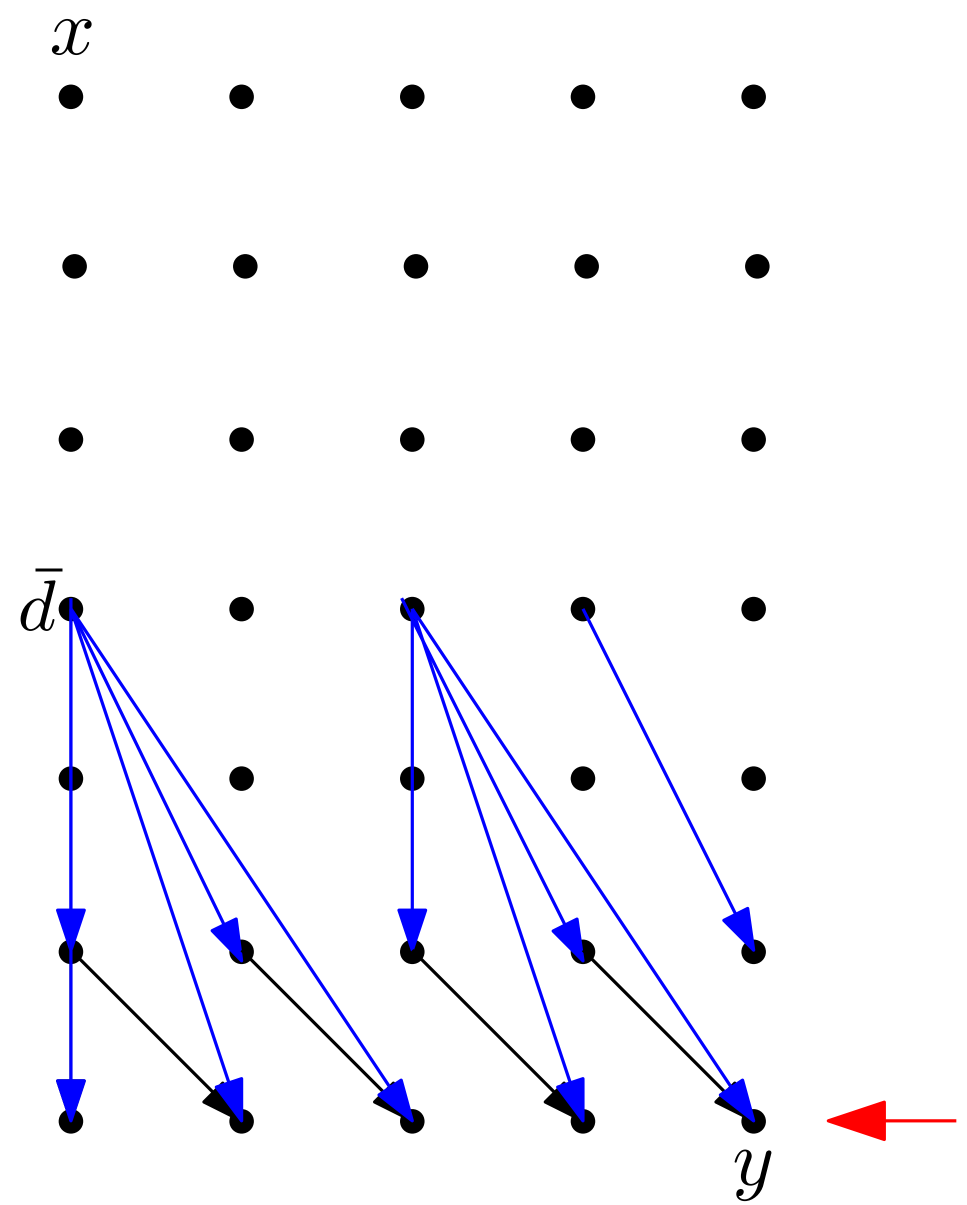
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



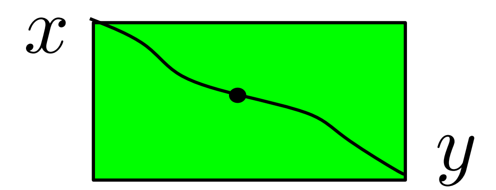
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

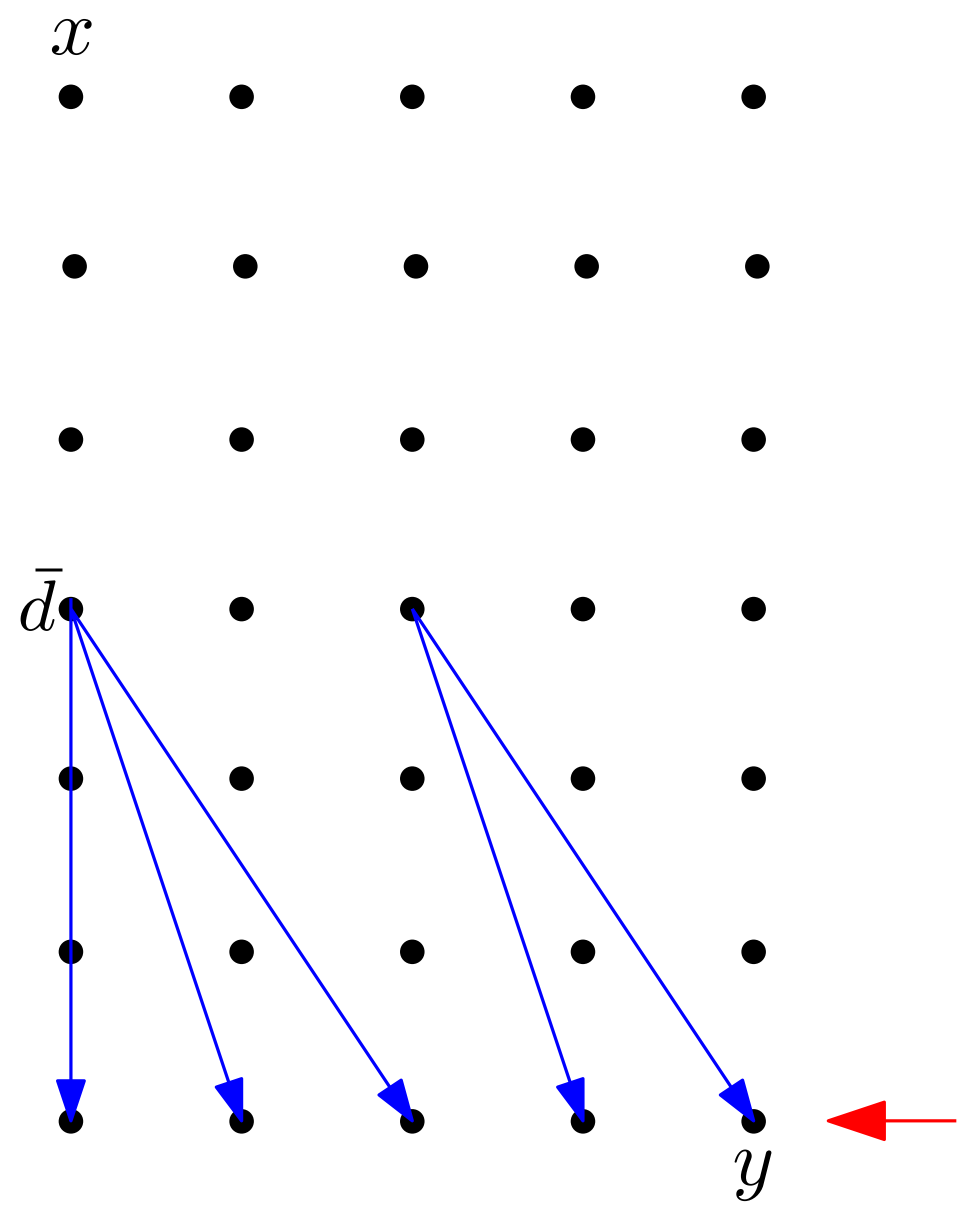
All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.



Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



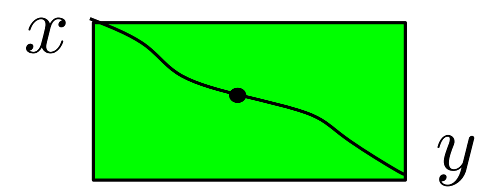
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.

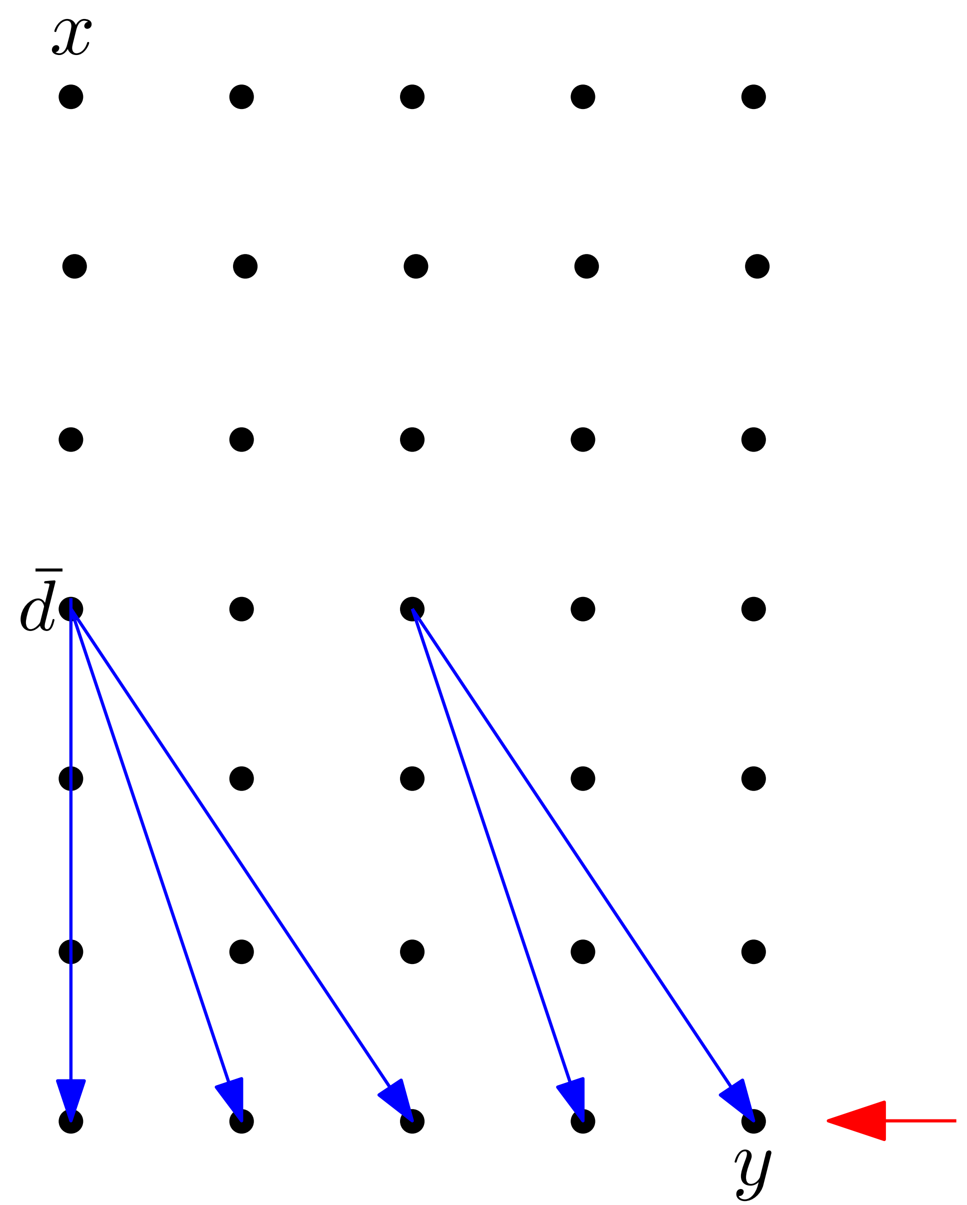
Implementing $Mid(x, y)$ in $O(D + n)$ space and $\boxed{Area(x, y) \text{ time}}$



For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.



Outline

- Review of the Monge Speedup

- Saving Space While Saving Time

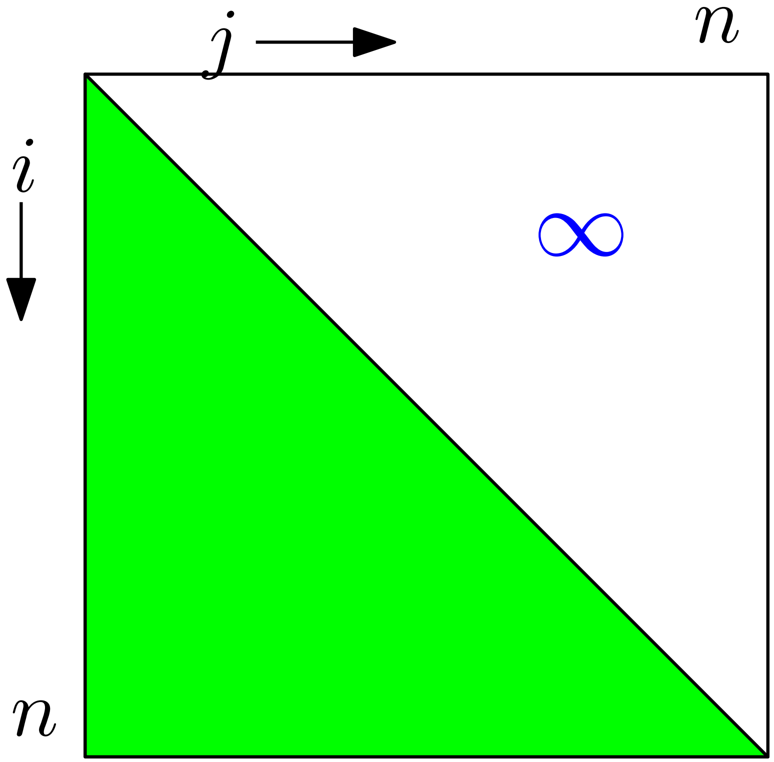
- Maintaining the Speedup in an Online Setting

$$H(i,d) = \min_{0 \leq j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{l} 0 \leq i \leq n \\ 0 \leq d \leq D \end{array}$$

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

For any fixed d, the problem is to find the row minima of a lower triangular matrix $M = \{a_{j,i}\}$ where

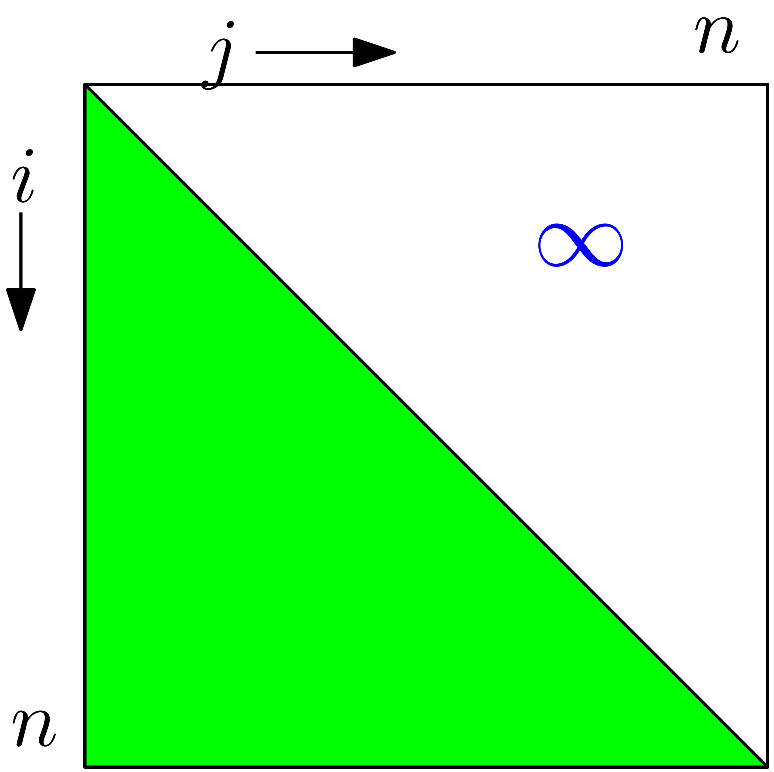
For $j < i$, $\quad M_{j,i} = H(j, d-1) + w^{(d)}(j,i)$; else $M_{j,i} = \infty$



$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

For any fixed d, the problem is to find the row minima of a lower triangular matrix $M = \{a_{j,i}\}$ where

For $j < i$, $\quad M_{j,i} = H(j, d-1) + w^{(d)}(j,i)$; else $M_{j,i} = \infty$

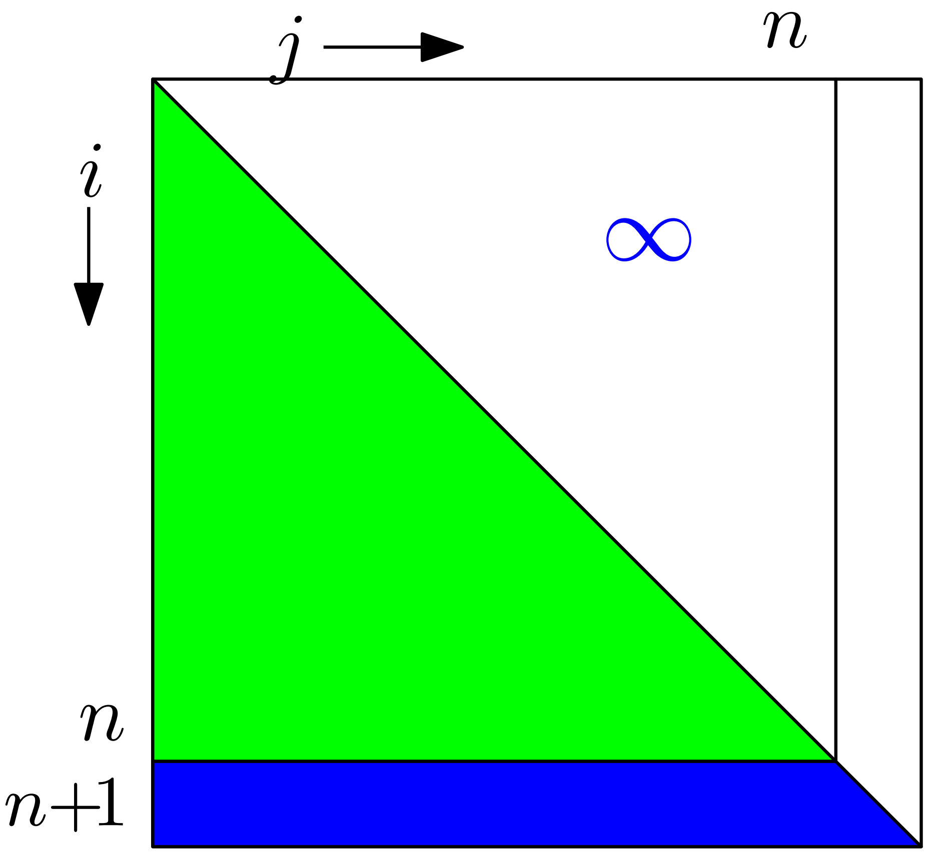


If $n \to (n+1)$ must find minimum of new row.

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

For any fixed d, the problem is to find the row minima of a lower triangular matrix $M = \{a_{j,i}\}$ where

For $j < i$, $\quad M_{j,i} = H(j, d-1) + w^{(d)}(j,i)$; else $M_{j,i} = \infty$

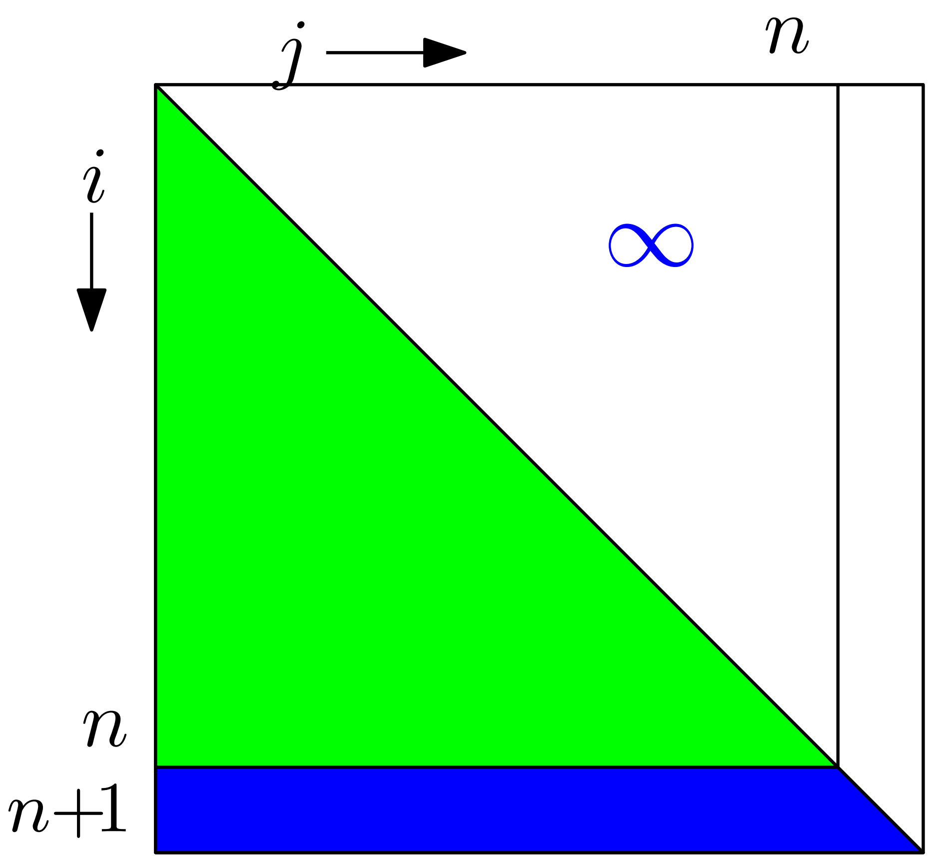


If $n \to (n+1)$ must find minimum of new row.

$$H(i,d) = \min_{0\le j<i}\Big(H(j,d-1) + w^{(d)}(j,i)\Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

For any fixed d, the problem is to find the row minima of a lower triangular matrix $M = \{a_{j,i}\}$ where

For $j < i$, $\quad M_{j,i} = H(j,d-1) + w^{(d)}(j,i)$; else $M_{j,i} = \infty$



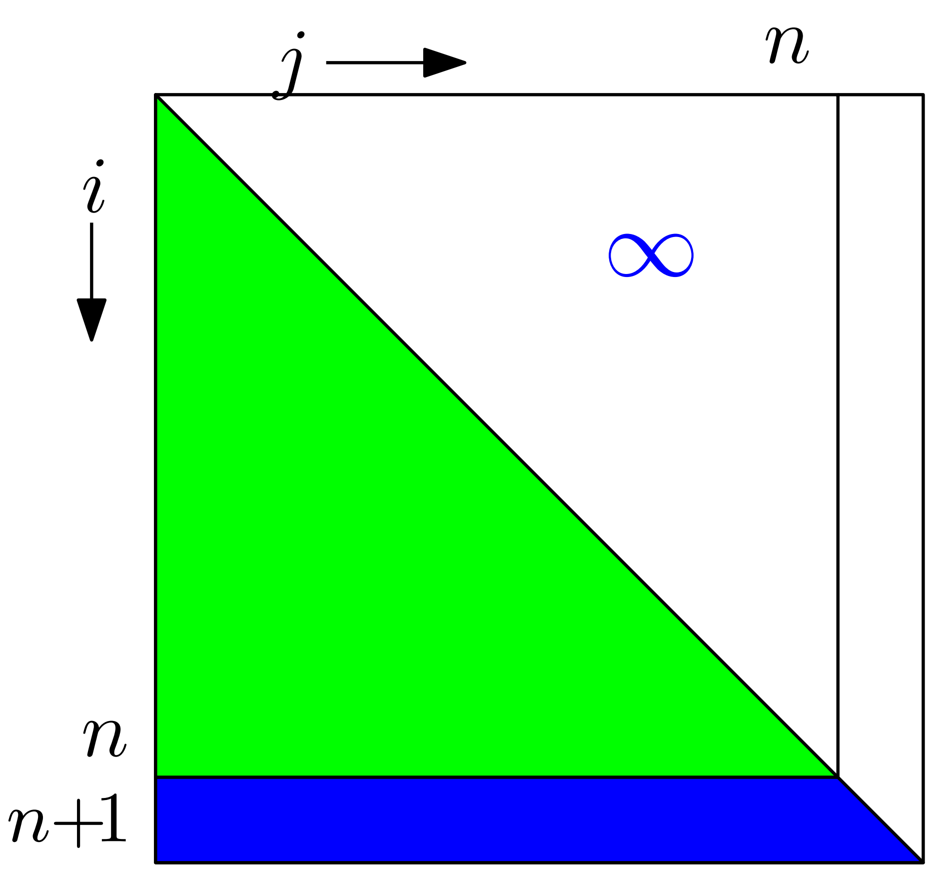
If $n \to (n+1)$ must find minimum of new row.

Context: Adding new point to right of line in D-median problem requires updating median locations. This requires finding "min" of new row on bottom of Monge matrices.

$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

For any fixed d, the problem is to find the row minima of a lower triangular matrix $M = \{a_{j,i}\}$ where

For $j < i$, $\quad M_{j,i} = H(j, d-1) + w^{(d)}(j,i)$; else $M_{j,i} = \infty$



If $n \to (n+1)$ must find minimum of new row.

SMAWK/LARSCH require batching queries. They do not provide *online processing* (in $O(1)$ time per step).

Suppose we are given an implicitly defined lower triangular matrix $A = \{a(n, j)\}$ in which we want to find *row minima*.

$$h(n) = \min_{1 \le j < n} a(n, j)$$

Suppose we are given an implicitly defined lower triangular matrix $A = \{a(n,j)\}$ in which we want to find *row minima*.

$$h(n) = \min_{1 \le j < n} a(n,j)$$

We say that the $a(n,j)$ satisfy the *online Monge property*, if

$$\forall 1 \le j < n, \qquad a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n,$$

where c_n, β_n and δ_j are constants satisfying

$$\beta_n \ge 0, \quad \text{and} \quad \delta_1 \ge \delta_2 \ge \delta_3 \cdots.$$

Suppose we are given an implicitly defined lower triangular matrix $A = \{a(n,j)\}$ in which we want to find *row minima*.

$$h(n) = \min_{1 \le j < n} a(n,j)$$

We say that the $a(n,j)$ satisfy the *online Monge property*, if

$$\forall 1 \le j < n, \qquad a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n,$$

where c_n, β_n and δ_j are constants satisfying

$$\beta_n \ge 0, \quad \text{and} \quad \delta_1 \ge \delta_2 \ge \delta_3 \cdots.$$

Theorem: If $\forall n, i$, the value of $a(n,i)$ can be computed in $O(1)$ time, provided that the values of $h(j)$ for $1 \le j < n$ are known,

Suppose we are given an implicitly defined lower triangular matrix $A = \{a(n,j)\}$ in which we want to find *row minima*.

$$h(n) = \min_{1 \le j < n} a(n,j)$$

We say that the $a(n,j)$ satisfy the *online Monge property*, if

$$\forall 1 \le j < n, \qquad a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n,$$

where c_n, β_n and δ_j are constants satisfying

$$\beta_n \ge 0, \quad \text{and} \quad \delta_1 \ge \delta_2 \ge \delta_3 \cdots.$$

Theorem: If $\forall n, i$, the value of $a(n,i)$ can be computed in $O(1)$ time, provided that the values of $h(j)$ for $1 \le j < n$ are known,

$\Rightarrow$ The $h(i)$ can be computed consecutively $h(1), h(2), \ldots$ using $O(1)$ amortized and $O(\log n)$ worst case time to calculate $h(n)$.

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_i \downarrow$$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_i \downarrow$$

Stronger than regular Monge property

$$a(n+1,j) + a(n,j+1) - a(n,j) - a(n+1,j+1)$$

$$= (\delta_j - \delta_{j+1})\beta_{n+1} \geq 0,$$

So Online Monge is special case of Monge

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_i \downarrow$$

Stronger than regular Monge property

$$a(n+1,j) + a(n,j+1) - a(n,j) - a(n+1,j+1)$$

$$= (\delta_j - \delta_{j+1})\beta_{n+1} \geq 0,$$

So Online Monge is special case of Monge

If problem has this stronger property, Theorem says that Monge speedup can be maintained in online problem variant.

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Occurs Quite Naturally

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Occurs Quite Naturally

$$H(i,d) = \min_{0 \leq j < i} \left(H(j, d-1) + w^{(d)}(j,i) \right)$$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Occurs Quite Naturally

$$H(i,d) = \min_{0 \leq j < i} \left(H(j,d-1) + w^{(d)}(j,i) \right)$$

Fix d $\quad \Rightarrow \quad a(i,j) = H(j,d-1) + w^{(d)}(j,i)$

$$\Rightarrow \quad a(i,j) - a(i-1,j) = w^{(d)}(j,i) - w^{(d)}(j,i-1)$$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Occurs Quite Naturally

$$H(i,d) = \min_{0 \leq j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$$

Fix d $\quad \Rightarrow \quad a(i,j) = H(j, d-1) + w^{(d)}(j,i)$

$$\Rightarrow \quad a(i,j) - a(i-1,j) = w^{(d)}(j,i) - w^{(d)}(j,i-1)$$

D-Medians on a Directed Line: $\quad w^{(d)}(j,i) = \sum_{l=j+1}^{i} w_l (v_l - v_{j+1})$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Occurs Quite Naturally

$$H(i,d) = \min_{0 \leq j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$$

Fix d $\quad \Rightarrow \quad a(i,j) = H(j, d-1) + w^{(d)}(j,i)$

$$\Rightarrow \quad a(i,j) - a(i-1,j) = w^{(d)}(j,i) - w^{(d)}(j,i-1)$$

D-Medians on a Directed Line: $\quad w^{(d)}(j,i) = \sum_{l=j+1}^{i} w_l (v_l - v_{j+1})$

$$a(i,j) - a(i-1,j) = w_i v_i + (-v_{j+1})\, w_i$$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Occurs Quite Naturally

$$H(i,d) = \min_{0 \leq j < i} \Big(H(j, d-1) + w^{(d)}(j,i) \Big)$$

Fix d $\quad \Rightarrow \quad a(i,j) = H(j, d-1) + w^{(d)}(j,i)$

$$\Rightarrow \quad a(i,j) - a(i-1,j) = w^{(d)}(j,i) - w^{(d)}(j,i-1)$$

D-Medians on a Directed Line: $\quad w^{(d)}(j,i) = \sum_{l=j+1}^{i} w_l (v_l - v_{j+1})$

$$a(i,j) - a(i-1,j) = \underbrace{w_i}_{c_i} v_i + \underbrace{(-v_{j+1})}_{\delta_j} \underbrace{w_i}_{\beta_i}$$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Occurs Quite Naturally

$$H(i,d) = \min_{0 \leq j < i} \left(H(j, d-1) + w^{(d)}(j,i) \right)$$

Fix d $\quad \Rightarrow \quad a(i,j) = H(j,d-1) + w^{(d)}(j,i)$

$$\Rightarrow \quad a(i,j) - a(i-1,j) = w^{(d)}(j,i) - w^{(d)}(j,i-1)$$

Wireless Mobile Paging $\qquad w^{(d)}(j,i) = i \left(\sum_{\ell=j+1}^{i} p_\ell \right)$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Occurs Quite Naturally

$$H(i,d) = \min_{0 \leq j < i} \left(H(j, d-1) + w^{(d)}(j,i) \right)$$

Fix d $\Rightarrow$ $a(i,j) = H(j, d-1) + w^{(d)}(j,i)$

$\Rightarrow$ $a(i,j) - a(i-1,j) = w^{(d)}(j,i) - w^{(d)}(j,i-1)$

Wireless Mobile Paging $\qquad w^{(d)}(j,i) = i \left(\sum_{\ell=j+1}^{i} p_\ell \right)$

$$a(i,j) - a(i-1,j) = ip_i + \sum_{t=j+1}^{i-1} p_t$$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad\qquad \beta_n \geq 0,\ \delta_j \downarrow$$

Occurs Quite Naturally

$$H(i,d) = \min_{0 \leq j < i} \left(H(j,d-1) + w^{(d)}(j,i) \right)$$

Fix d $\quad \Rightarrow \quad a(i,j) = H(j,d-1) + w^{(d)}(j,i)$

$$\Rightarrow \quad a(i,j) - a(i-1,j) = w^{(d)}(j,i) - w^{(d)}(j,i-1)$$

Wireless Mobile Paging $\qquad w^{(d)}(j,i) = i \left(\sum_{\ell=j+1}^{i} p_\ell \right)$

$$a(i,j) - a(i-1,j) = ip_i + \sum_{t=j+1}^{i-1} p_t = \underbrace{\left(ip_i + \sum_{t=1}^{j-1} p_j \right)}_{c_i} + \underbrace{\left(-\sum_{t=1}^{j} p_t \right)}_{\delta_j} \underbrace{1}_{\beta_i}$$

Online Monge:

$$a(n, j) - a(n-1, j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

$$h(n) = \min_{1 \leq j < n} a(n, j)$$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

$$h(n) = \min_{1 \leq j < n} a(n,j)$$

$\forall\, 1 \leq j \leq n \leq N$ define *lines* and *Lower Envelope*

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad L^n(x) = \min_{1 \leq j \leq n} L_j^n(x)$$

Online Monge:

$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n,$ $\qquad \beta_n \geq 0,\ \delta_j \downarrow$

$h(n) = \min_{1 \leq j < n} a(n,j)$

$\forall\, 1 \leq j \leq n \leq N$ define *lines* and *Lower Envelope*

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad L^n(x) = \min_{1 \leq j \leq n} L_j^n(x)$$

$$\Rightarrow \qquad h(n) = \min_{1 \leq j \leq n} L_j^n(0) = L^n(0).$$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \ge 0,\ \delta_j \downarrow$$

$$h(n) = \min_{1\le j<n} a(n,j)$$

$\forall\, 1 \le j \le n \le N$ define *lines* and *Lower Envelope*

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad L^n(x) = \min_{1\le j\le n} L_j^n(x)$$

$$\Rightarrow \qquad h(n) = \min_{1\le j\le n} L_j^n(0) = L^n(0).$$

Algorithm will maintain $L^n(x)$ for $x \in [0,\infty]$

Online Monge:

$$a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n, \qquad \beta_n \geq 0,\ \delta_j \downarrow$$

$$h(n) = \min_{1 \leq j < n} a(n,j)$$

$\forall\, 1 \leq j \leq n \leq N$ define *lines* and *Lower Envelope*

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad L^n(x) = \min_{1 \leq j \leq n} L_j^n(x)$$

$$\Rightarrow \qquad h(n) = \min_{1 \leq j \leq n} L_j^n(0) = L^n(0).$$

Algorithm will maintain $L^n(x)$ for $x \in [0, \infty]$

No line can appear on lower envelope more than once, so algorithm only has to keep track of $< n$ *breakpoints*. These will not change "much" from step to step

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad\qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

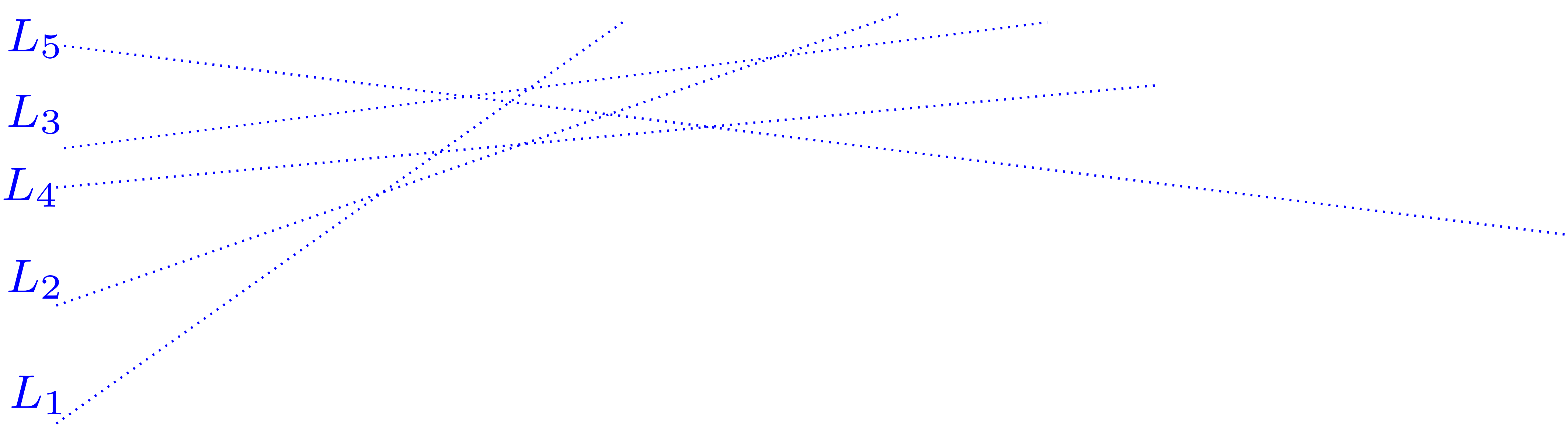
- The only data structure used is an array, called the *active-indices array*, $Z = (z_1, \ldots, z_t)$ for some $t \le n$.
- It stores, from left to right, the indices of the L_j^n that appear on L^n in the range $x \in [0, \infty)$.
- The slopes of the segments forming the lower envelope of a set of lines decreases as one sweeps from left to right. Since $\delta_1 > \delta_2 > \cdots > \delta_n$, we have $z_1 < z_2 < \cdots < z_t = n$

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad\qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

- The only data structure used is an array, called the *active-indices array*, $Z = (z_1, \ldots, z_t)$ for some $t \le n$.

- It stores, from left to right, the indices of the L_j^n that appear on L^n in the range $x \in [0, \infty)$.

- The slopes of the segments forming the lower envelope of a set of lines decreases as one sweeps from left to right. Since $\delta_1 > \delta_2 > \cdots > \delta_n$, we have $z_1 < z_2 < \cdots < z_t = n$

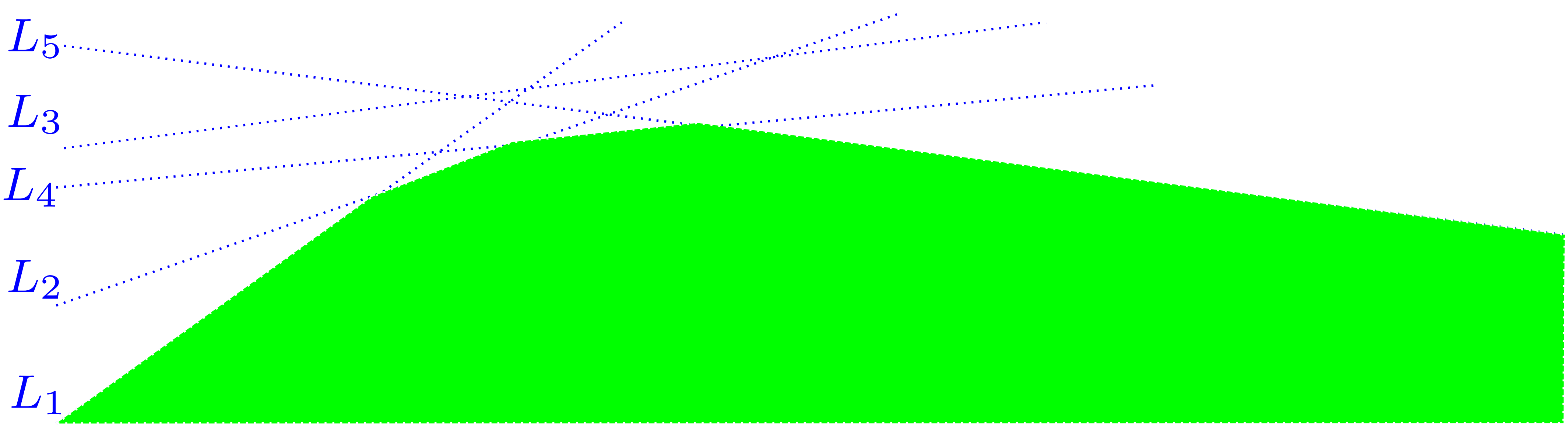


$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad\qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

- The only data structure used is an array, called the *active-indices array*, $Z = (z_1, \ldots, z_t)$ for some $t \le n$.

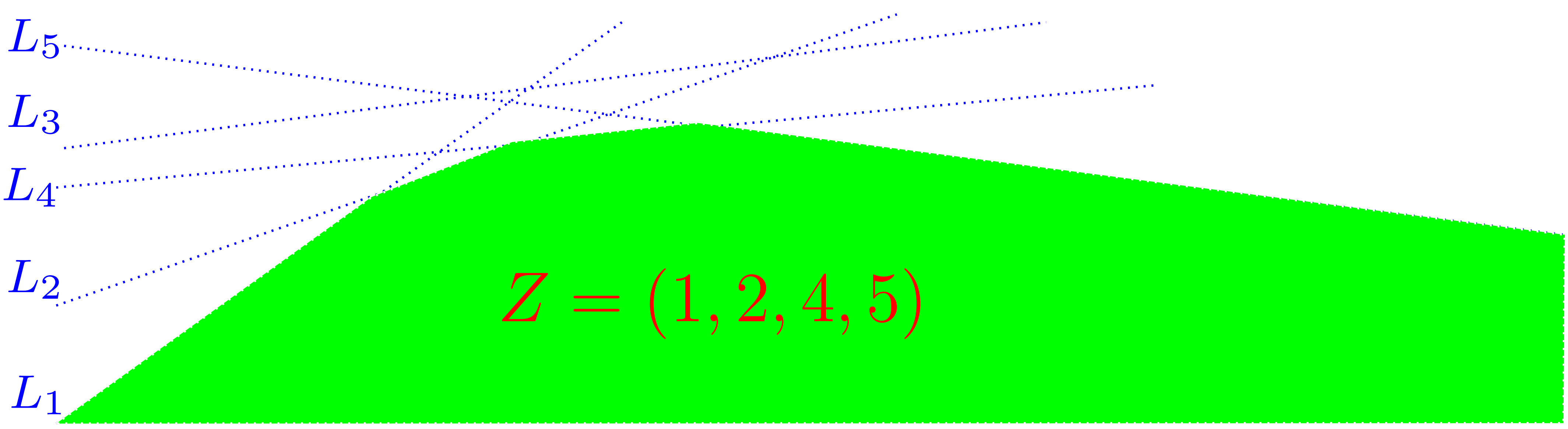
- It stores, from left to right, the indices of the L_j^n that appear on L^n in the range $x \in [0, \infty)$.

- The slopes of the segments forming the lower envelope of a set of lines decreases as one sweeps from left to right. Since $\delta_1 > \delta_2 > \cdots > \delta_n$, we have $z_1 < z_2 < \cdots < z_t = n$



$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

- The only data structure used is an array, called the *active-indices array*, $Z = (z_1, \ldots, z_t)$ for some $t \le n$.
- It stores, from left to right, the indices of the L_j^n that appear on L^n in the range $x \in [0, \infty)$.
- The slopes of the segments forming the lower envelope of a set of lines decreases as one sweeps from left to right. Since $\delta_1 > \delta_2 > \cdots > \delta_n$, we have $z_1 < z_2 < \cdots < z_t = n$



$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad\qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

To update lower envelope from $n-1$ to n

Recall $a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n$

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

To update lower envelope from $n-1$ to n
Recall $a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n$

Then $\forall\ 1 \le j \le n-1$.

$$\begin{aligned} L_j^n(x) &= [a(n,j) - \delta_j\, \beta_n] + \delta_j\,(x + \beta_n) \\ &= [a(n-1,j) + c_n] + \delta_j\,(x + \beta_n) \\ &= L_j^{n-1}(x + \beta_n) + c_n. \end{aligned}$$

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad\qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

To update lower envelope from $n-1$ to n
Recall $a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n$

Then $\forall\ 1 \le j \le n-1$.

$$\begin{aligned} L_j^n(x) &= [a(n,j) - \delta_j\, \beta_n] + \delta_j\,(x + \beta_n) \\ &= [a(n-1,j) + c_n] + \delta_j\,(x + \beta_n) \\ &= L_j^{n-1}(x + \beta_n) + c_n. \end{aligned}$$

So lower envelope for n is
  (a) lower envelope for $n-1$ shifted vertically and to right.
  (b) with new line L_n^n added

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad\qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

To update lower envelope from $n-1$ to n
Recall $a(n,j) - a(n-1,j) = c_n + \delta_j \beta_n$

Then $\forall\ 1 \le j \le n-1$.

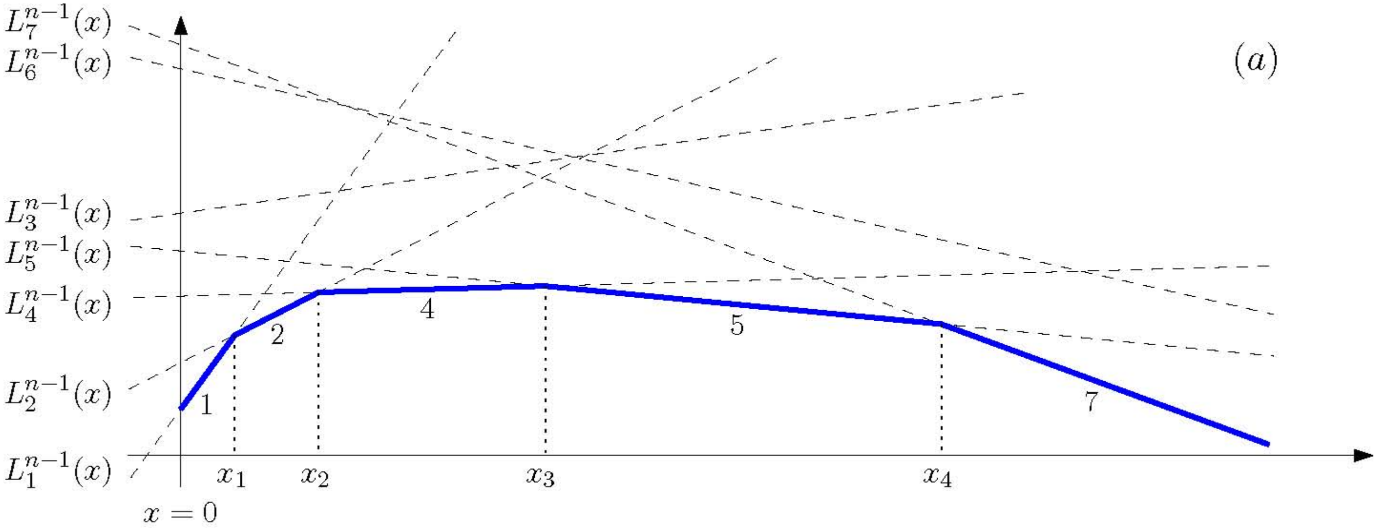
$$\begin{aligned} L_j^n(x) &= [a(n,j) - \delta_j\, \beta_n] + \delta_j\,(x + \beta_n) \\ &= [a(n-1,j) + c_n] + \delta_j\,(x + \beta_n) \\ &= L_j^{n-1}(x + \beta_n) + c_n. \end{aligned}$$

So lower envelope for n is

(a) lower envelope for $n-1$ shifted vertically and to right.

(b) with new line L_n^n added

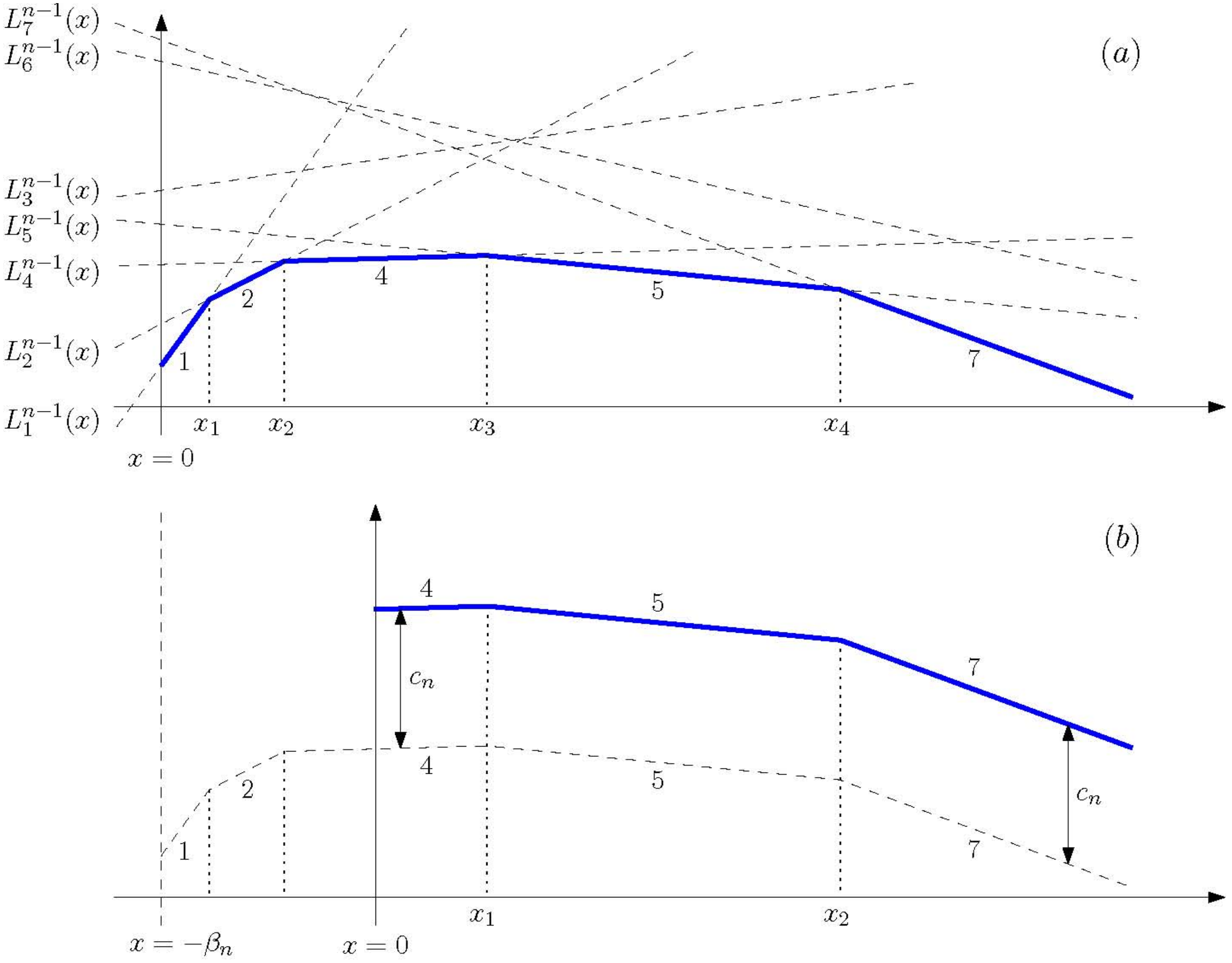
Note: Because $\delta_j \downarrow$, line L_n^n must be on lower envelope, and be rightmost segment on lower envelope



Lower env for lines

$$L_j^{n-1}(x): \quad 1 \le j < n$$

$$h(n-1) = \min_{1 \le j \le n-1} L_j^{n-1}(0)$$



Lower env for lines

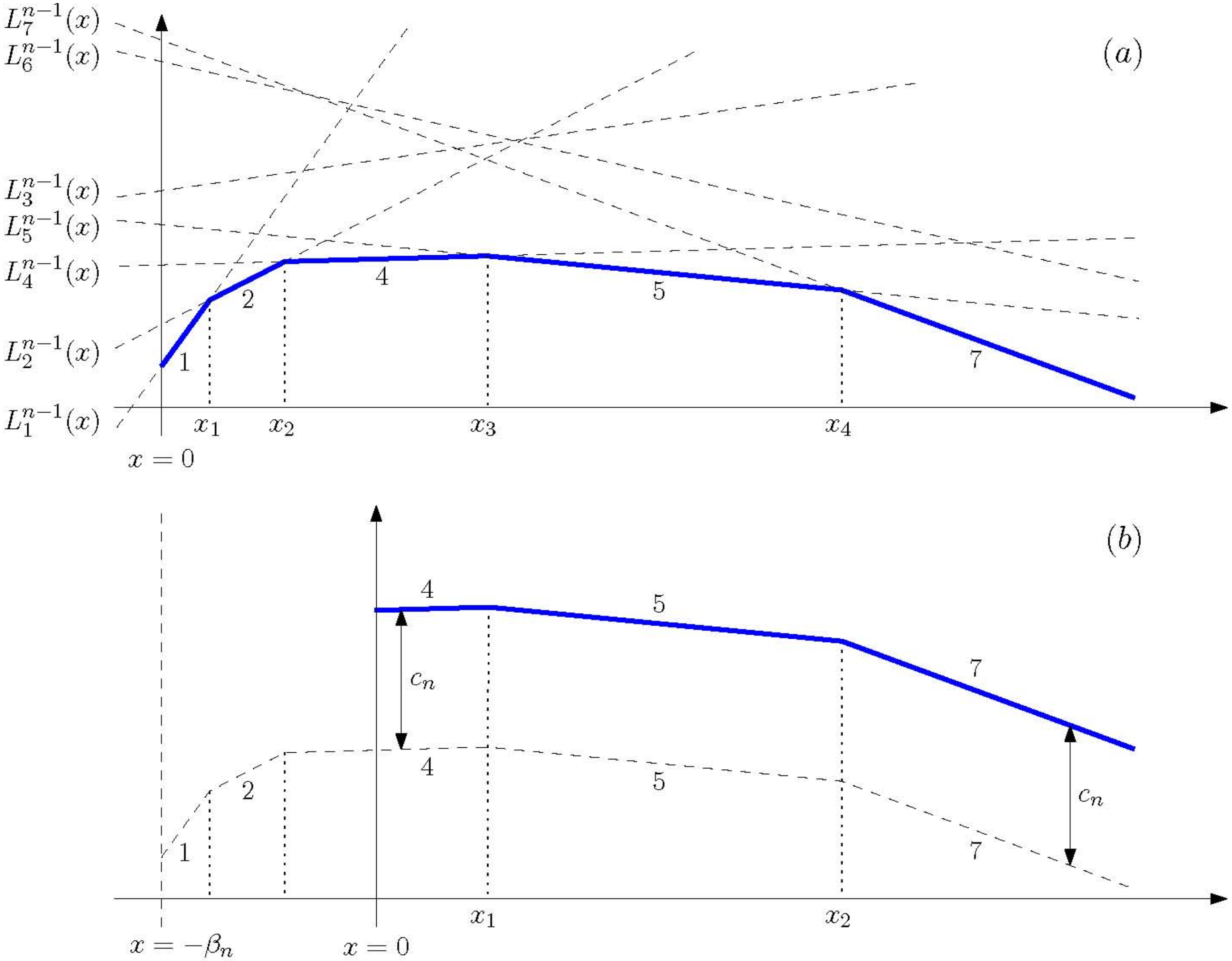
$L_j^{n-1}(x): \quad 1 \le j < n$

$$h(n-1) = \min_{1 \le j \le n-1} L_j^{n-1}(0)$$

Lower env for lines

$L_j^n(x) = L_j^{n-1}(x+\beta_n)+c_n$

$1 \le j < n$



Lower env for lines

$L_j^{n-1}(x): \quad 1 \leq j < n$

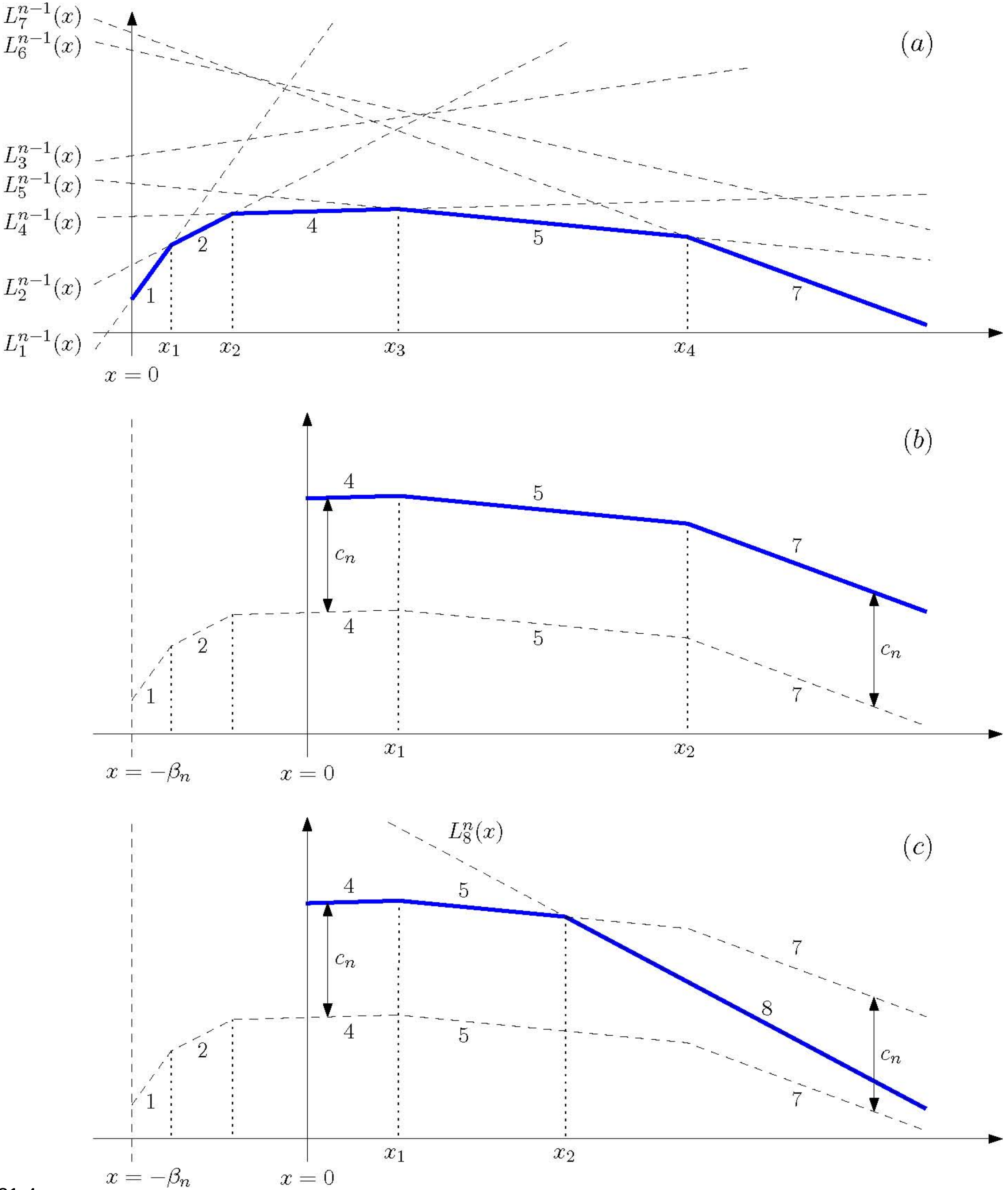
$$h(n-1) = \min_{1 \leq j \leq n-1} L_j^{n-1}(0)$$

Lower env for lines

$L_j^n(x) = L_j^{n-1}(x+\beta_n)+c_n$

$1 \leq j < n$

Note: lines shift up
axis shifts to right



Lower env for lines

$$L_j^{n-1}(x): \quad 1 \le j < n$$

$$h(n-1) = \min_{1 \le j \le n-1} L_j^{n-1}(0)$$

Lower env for lines

$$L_j^n(x) = L_j^{n-1}(x+\beta_n)+c_n$$

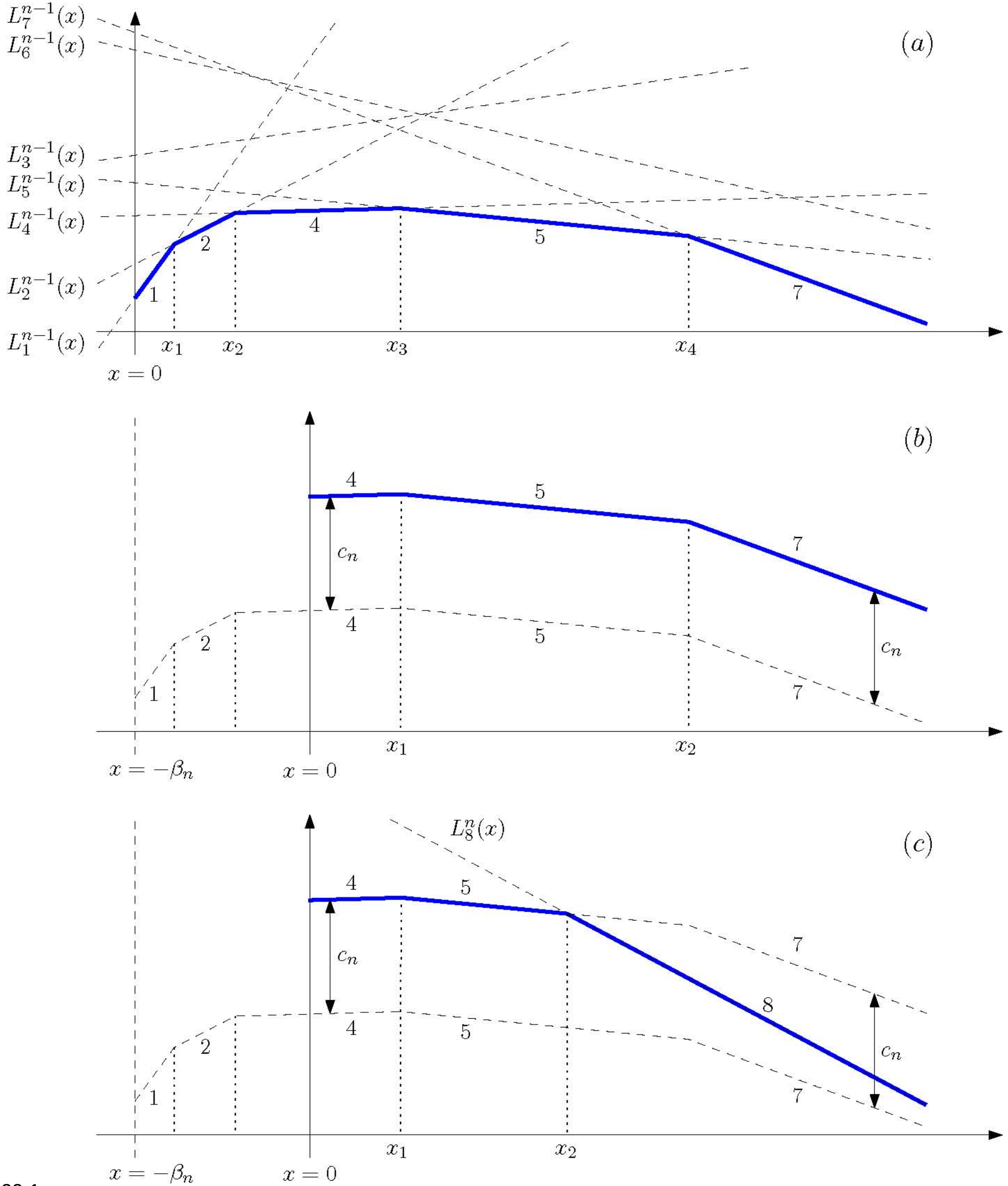
$$1 \le j < n$$

Note: lines shift up
axis shifts to right

Lower env for lines

$$L_j^n(x): \quad 1 \le j \le n$$

$$h(n) = \min_{1 \le j \le n} L_j^n(0)$$

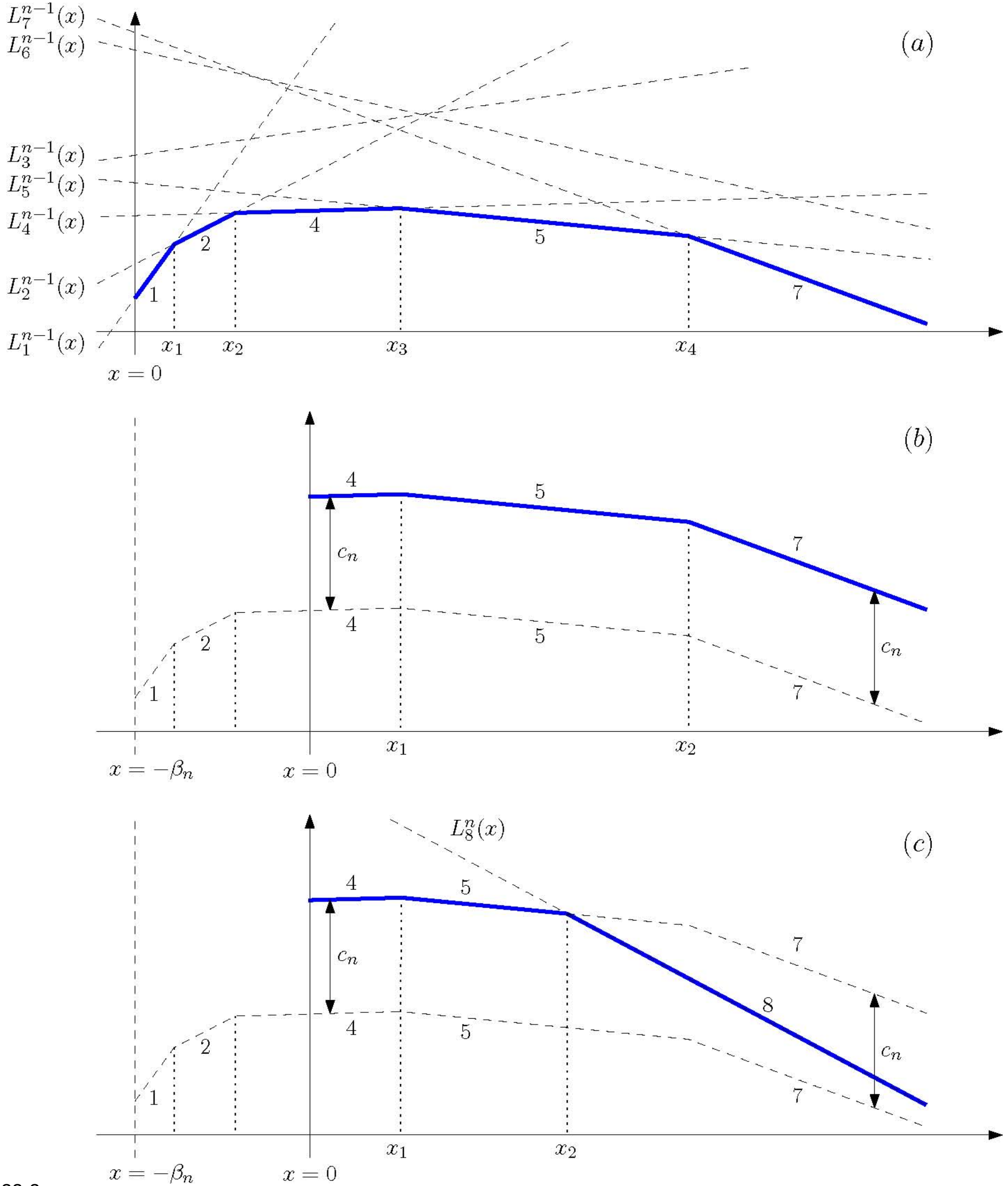


While moving from

$n = 7$ to $n = 8$

the indices of the active (lower envelope) lines change from

$\{1, 2, 4, 5, 7\}$



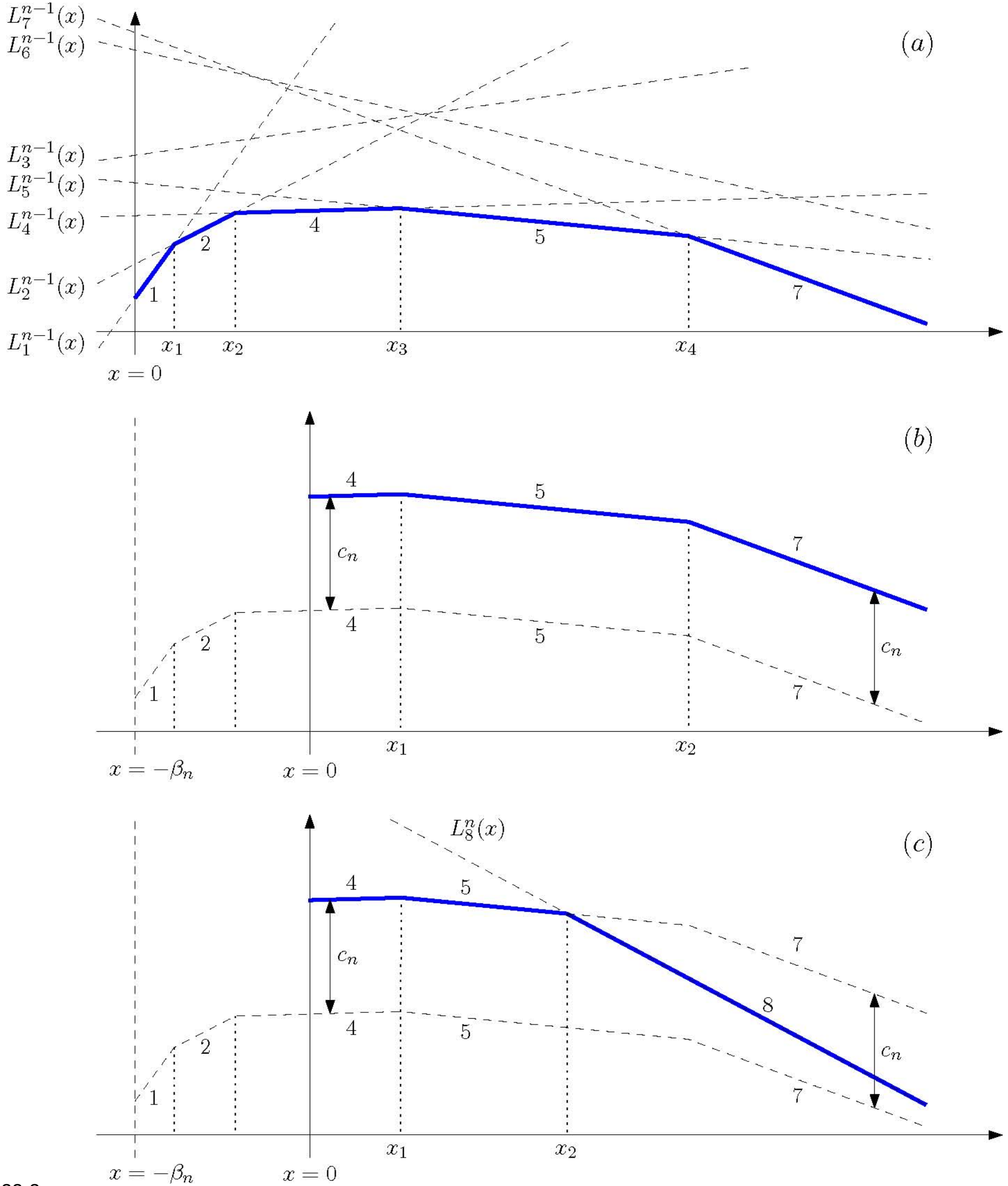
While moving from

$n = 7$ to $n = 8$

the indices of the active (lower envelope) lines change from

$\{1, 2, 4, 5, 7\}$ to

$\{4, 5, 7\}$



While moving from

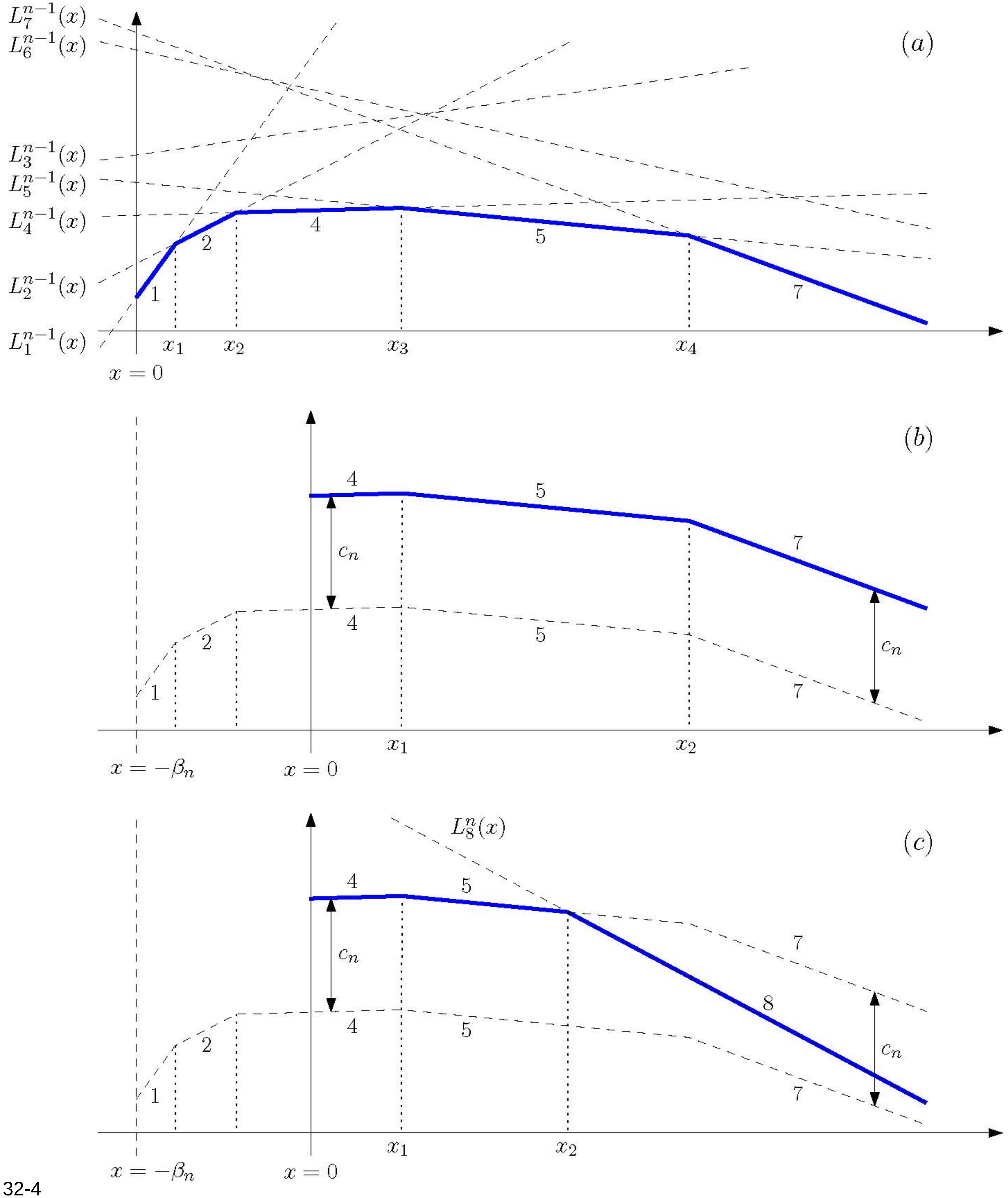
$n = 7$ to $n = 8$

the indices of the active (lower envelope) lines change from

$\{1, 2, 4, 5, 7\}$ to

$\{4, 5, 7\}$ to

$\{4, 5, 8\}$



While moving from

$n = 7$ to $n = 8$

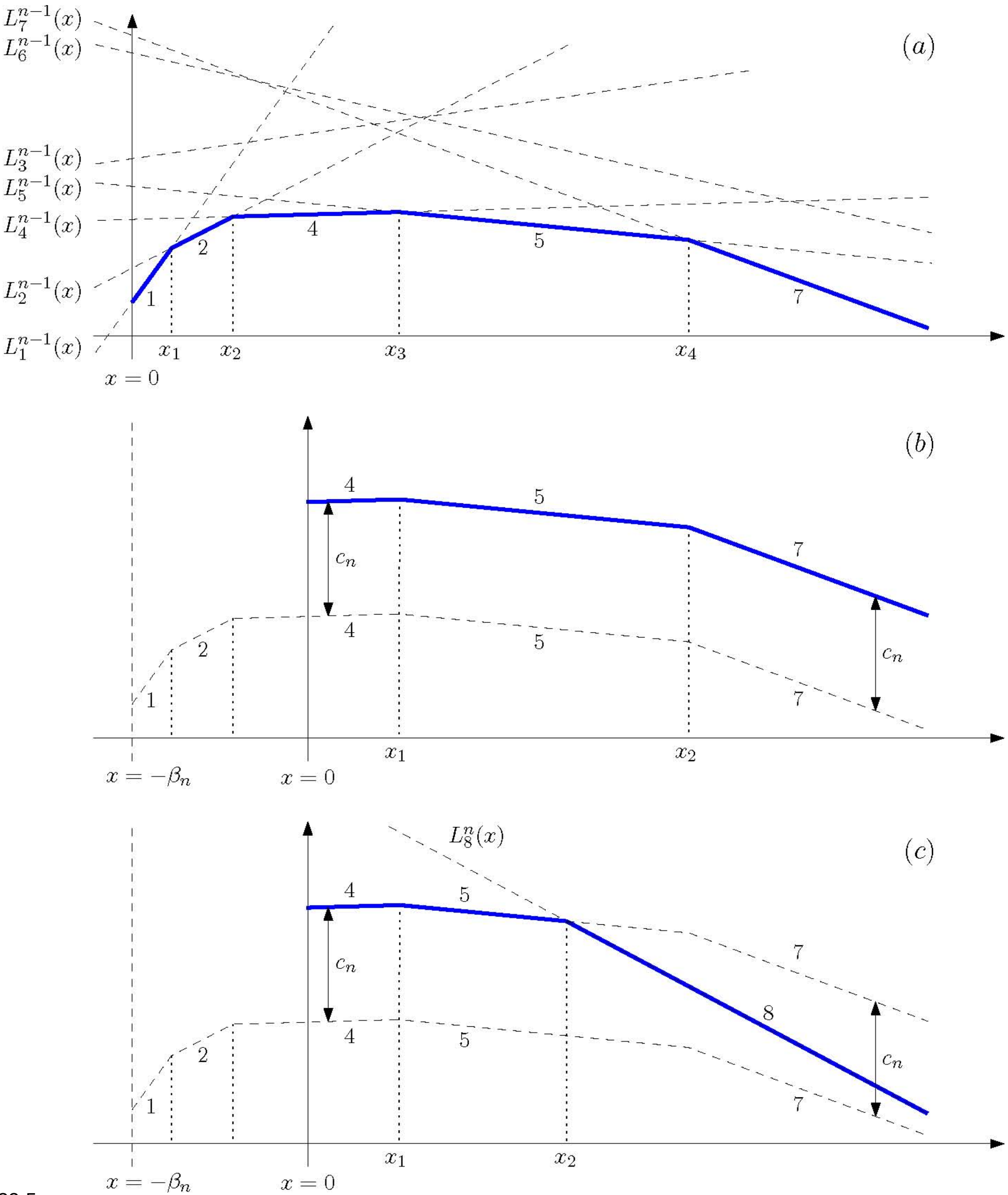
the indices of the active (lower envelope) lines change from

$\{\cancel{1}, \cancel{2}, 4, 5, 7\}$ to

$\{4, 5, 7\}$ to

$\{4, 5, 8\}$

We walk from left to chop off $\{1, 2\}$



While moving from

$n = 7$ to $n = 8$

the indices of the active (lower envelope) lines change from

$\{\cancel{1, 2}, 4, 5, 7\}$ to

$\{4, 5, 7\}$ to

$\{4, 5, 8\}$

We walk from left to chop off $\{1, 2\}$

And then add 8 from right, chopping off 7

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad\qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

Lower envelope for n is

(a) lower envelope for $n-1$ shifted vertically and to right.

(b) with new line L_n^n added

Note: Because $\delta_j \downarrow$, line L_n^n must be on lower envelope, and be rightmost segment on lower envelope

$L^n_j(x) = a(n, j) + \delta_j \cdot x$ $\qquad$ $L^n(x) = \min_{1 \le j \le n} L^n_j(x)$

Lower envelope for n is

(a) lower envelope for $n-1$ shifted vertically and to right.

Scan from left, chopping off line segments.

(b) with new line L^n_n added

Note: Because $\delta_j \downarrow$, line L^n_n must be on lower envelope, and be rightmost segment on lower envelope

$L_j^n(x) = a(n,j) + \delta_j \cdot x$ $\qquad$ $L^n(x) = \min_{1 \le j \le n} L_j^n(x)$

Lower envelope for n is

(a) lower envelope for $n-1$ shifted vertically and to right.

Scan from left, chopping off line segments.

(b) with new line L_n^n added

Note: Because $\delta_j \downarrow$, line L_n^n must be on lower envelope, and be rightmost segment on lower envelope

Scan from right to find line segments chopped off by L_n^n

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad L^n(x) = \min_{1 \le j \le n} L_j^n(x)$$

Lower envelope for n is

(a) lower envelope for $n-1$ shifted vertically and to right.

Scan from left, chopping off line segments.

(b) with new line L_n^n added

Note: Because $\delta_j \downarrow$, line L_n^n must be on lower envelope, and be rightmost segment on lower envelope

Scan from right to find line segments chopped off by L_n^n

Total amount of work per step is $O(1) + \#$ indices cut. Once a line (index) disappears from lower envelope it never reappears. Amortizing over all lines gives $O(1)$ cost per update.

$$L_j^n(x) = a(n,j) + \delta_j \cdot x \qquad\qquad L^n(x) = \min_{1\le j\le n} L_j^n(x)$$

Lower envelope for n is

(a) lower envelope for $n-1$ shifted vertically and to right.

Scan from left, chopping off line segments.

(b) with new line L_n^n added

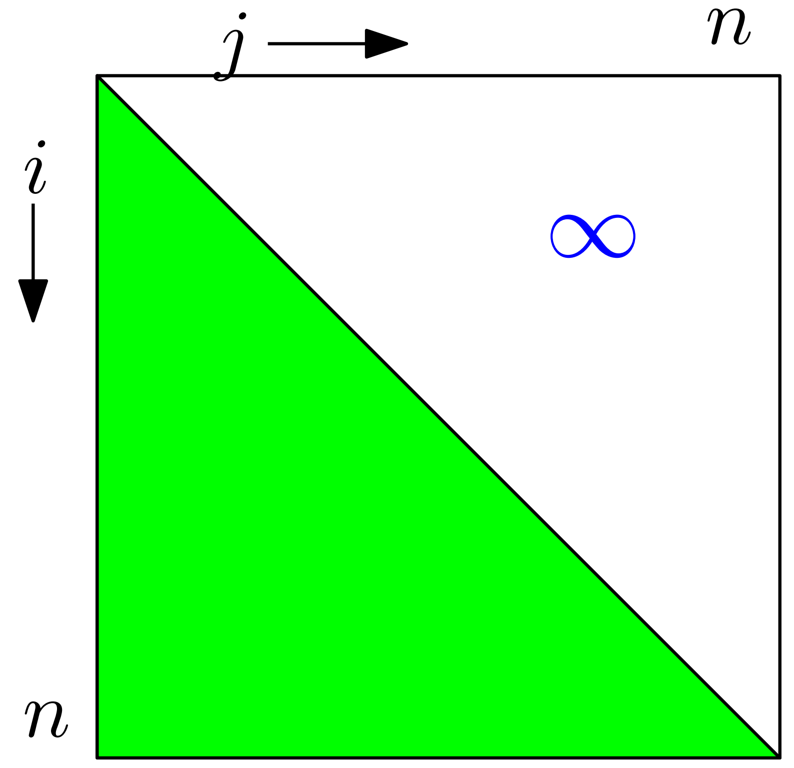
Note: Because $\delta_j \downarrow$, line L_n^n must be on lower envelope, and be rightmost segment on lower envelope

Scan from right to find line segments chopped off by L_n^n

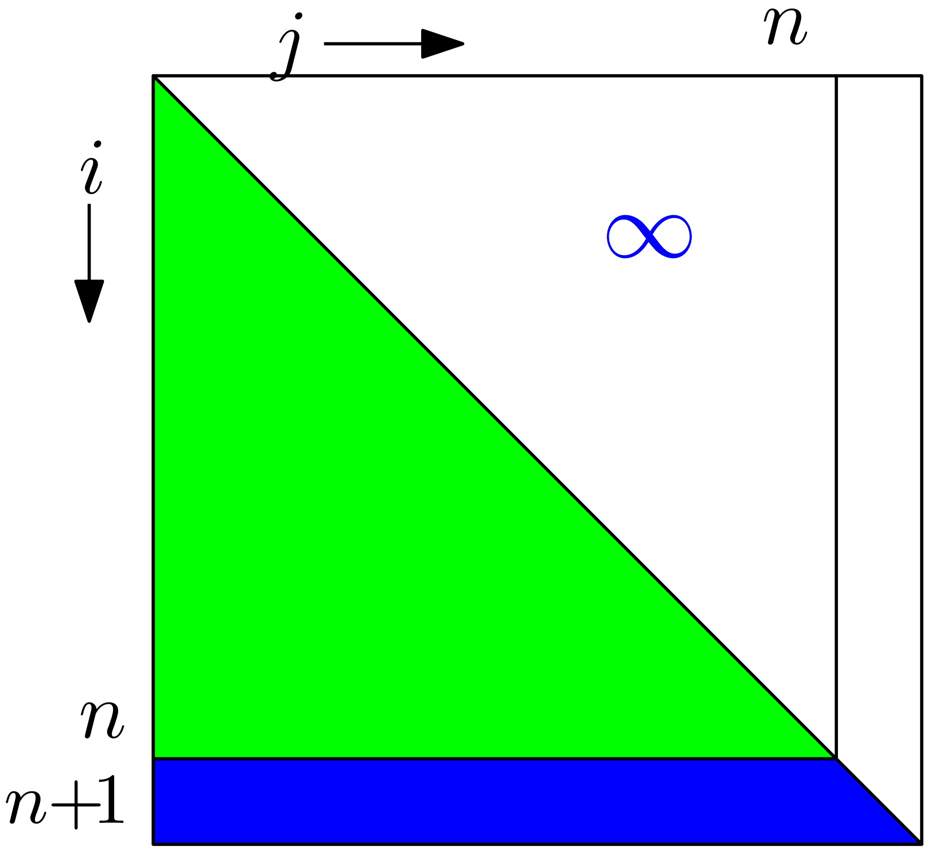
Total amount of work per step is $O(1) + \#$ indices cut. Once a line (index) disappears from lower envelope it never reappears. Amortizing over all lines gives $O(1)$ cost per update.

Can also use binary search to find "cut off points" in $O(\log n)$ worst case time

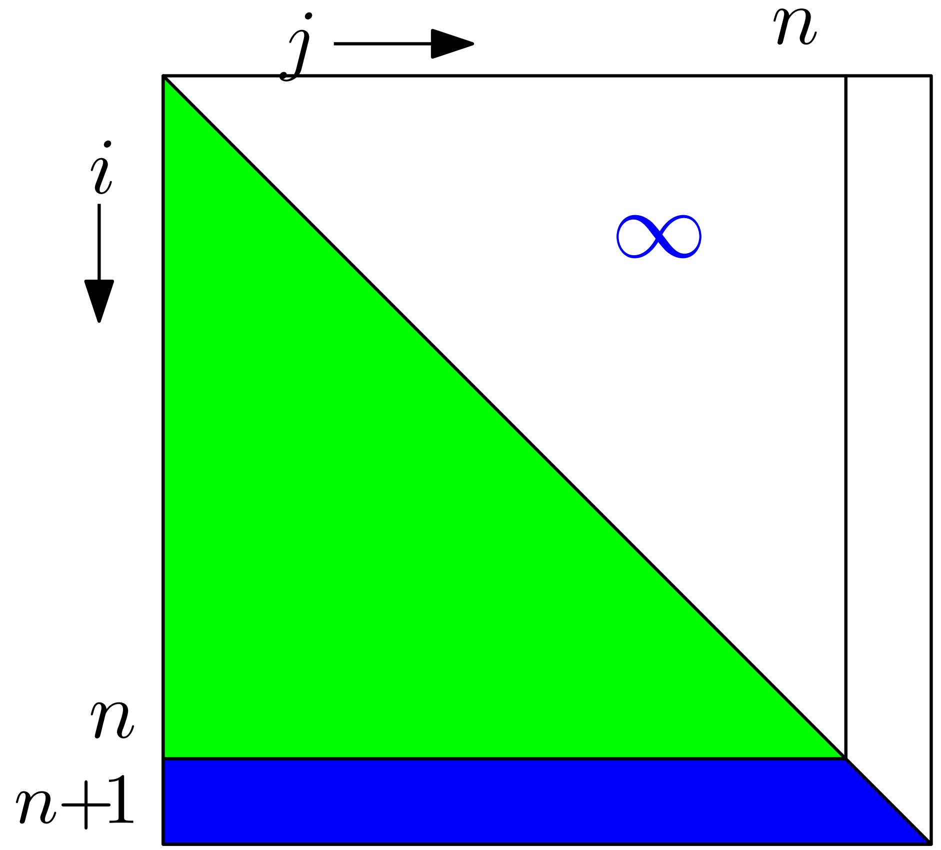
We just showed that for very special matrices $A = \{a_{i,j}\}$ the row minima can be found online, one row at a time, in $O(1)$ amortized and $O(\log n)$ worst-case time per step. The required condition was a very strong specialization of the Monge property.



We just showed that for very special matrices $A = \{a_{i,j}\}$ the row minima can be found online, one row at a time, in $O(1)$ amortized and $O(\log n)$ worst-case time per step. The required condition was a very strong specialization of the Monge property.



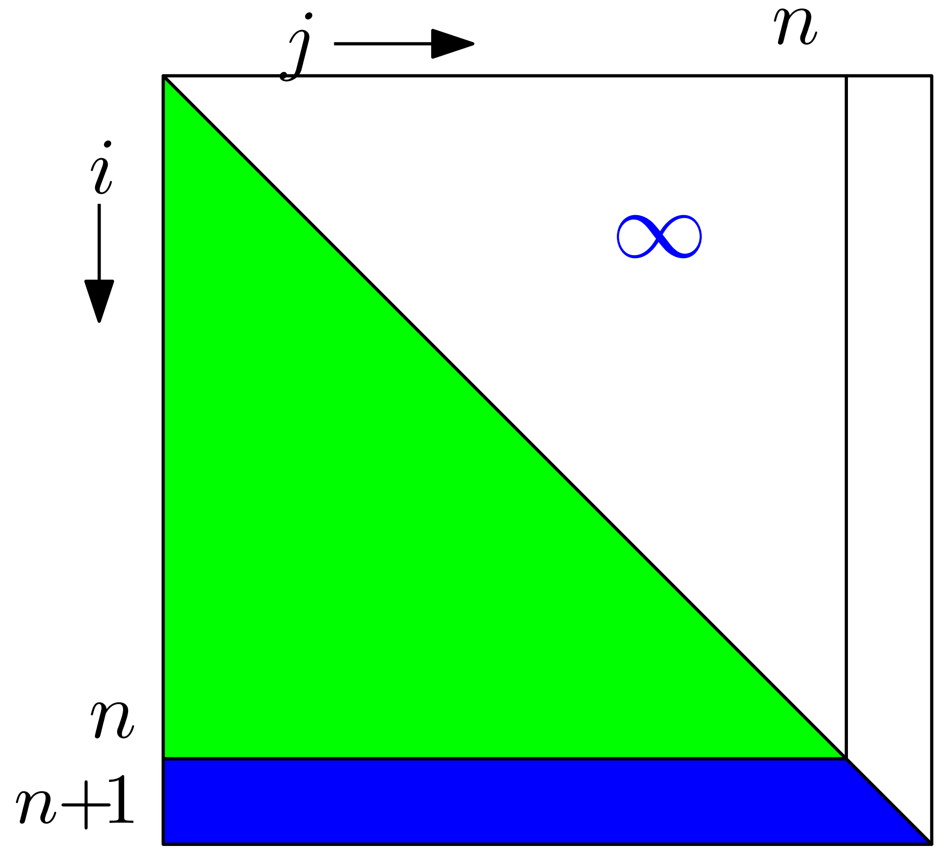
We just showed that for very special matrices $A = \{a_{i,j}\}$ the row minima can be found online, one row at a time, in $O(1)$ amortized and $O(\log n)$ worst-case time per step. The required condition was a very strong specialization of the Monge property.



Open Question

Are there weaker conditions that will permit $O(1)$ amortized updates?

We just showed that for very special matrices $A = \{a_{i,j}\}$ the row minima can be found online, one row at a time, in $O(1)$ amortized and $O(\log n)$ worst-case time per step. The required condition was a very strong specialization of the Monge property.



Open Question

Are there weaker conditions that will permit $O(1)$ amortized updates?

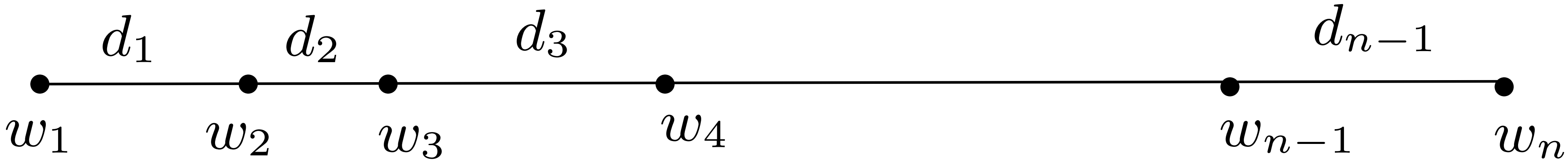
Can show that it's not possible for general Monge matrix

Outline

- Review of the Monge Speedup
- Saving Space While Saving Time
- Maintaining the Speedup in an Online Setting
- Thank You
  Questions?

Open Question

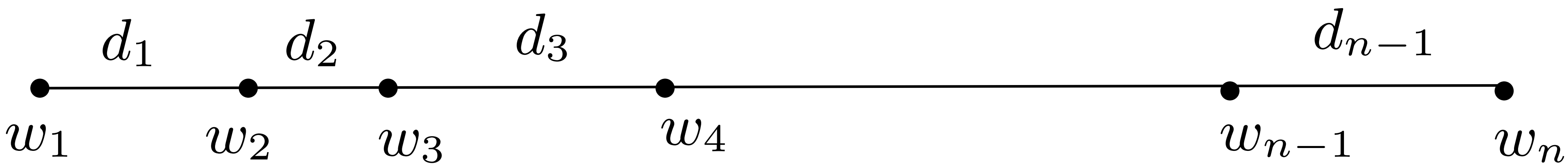
- Two-Sided Online K-Median on a Line



Identify k nodes as service centers. *Cost* of servicing request w_i, is w_i times distance from node i to nearest service center. Problem is to find location of k service centers that minimize total service cost.

Open Question

- Two-Sided Online K-Median on a Line

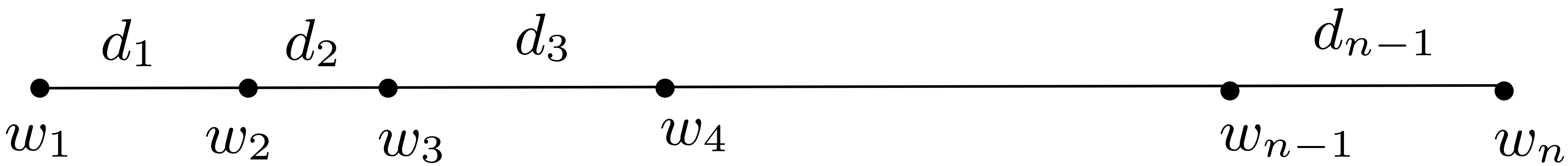


Identify k nodes as service centers. *Cost* of servicing request w_i, is w_i times distance from node i to nearest service center. Problem is to find location of k service centers that minimize total service cost.

- Naive DP: $O(kn^2)$
- Using Monge property: $O(kn)$
- Online, adding new element to right: Amortized $O(k)$

Open Question

- Two-Sided Online K-Median on a Line



Identify k nodes as service centers. *Cost* of servicing request w_i, is w_i times distance from node i to nearest service center. Problem is to find location of k service centers that minimize total service cost.

- Naive DP: $O(kn^2)$
- Using Monge property: $O(kn)$
- Online, adding new element to right: Amortized $O(k)$

Online Problem: Adding new elements to **right and left**.
Best known is $O(kn)$. Just as bad as reconstructing from scratch.
Is there a better way?