Improving Dynamic Programming

Mordecai Golin
Hong Kong UST

Dynamic Programming

DP creates a search space and calculates optimal cost for every item in the search space.

Optimal cost of larger items is based on optimal cost of smaller items.

Final Result: usually cost of largest item in search space.

Running time of DP algorithm, is time required to calculate all costs.

Dynamic Programming

DP creates a search space and calculates optimal cost for every item in the search space.

Optimal cost of larger items is based on optimal cost of smaller items.

Final Result: usually cost of largest item in search space.

Running time of DP algorithm, is time required to calculate all costs.

Chain Matrix Multiplication: Finding "cheapest" way to multiply matrices $A_1, \ldots, A_n$ where A_i is a $p_{i-1} \times p_i$ matrix.

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} m[i,k] + m[k+1,j] + p_{i-1} p_k p_j & \text{if } i > j \end{cases}$$

$m[i,j]$ is "best" way of multiplying $A_i, \ldots, A_j$

Dynamic Programming

DP creates a search space and calculates optimal cost for every item in the search space.

Optimal cost of larger items is based on optimal cost of smaller items.

Final Result: usually cost of largest item in search space.

Running time of DP algorithm, is time required to calculate all costs.

Chain Matrix Multiplication: Finding "cheapest" way to multiply matrices $A_1, \ldots, A_n$ where A_i is a $p_{i-1} \times p_i$ matrix.

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} m[i,k] + m[k+1,j] + p_{i-1}p_kp_j & \text{if } i > j \end{cases}$$

$m[i,j]$ is "best" way of multiplying $A_i, \ldots, A_j$

Want $m[1,n]$ and corresponding set of multiplications

Dynamic Programming

DP creates a search space and calculates optimal cost for every item in the search space.

Optimal cost of larger items is based on optimal cost of smaller items.

Final Result: usually cost of largest item in search space.

Running time of DP algorithm, is time required to calculate all costs.

Longest Common Subsequence: Find LCS of strings $X =< x, 1, \ldots, x_m >$, $Y =< y_1, \ldots, y_n >$.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } x_i = x_j \\ \max(c[i-1,j],\ c[i,j-1]) & \text{if } i,j > 0 \text{ and } x_i \neq x_j \end{cases}$$

$c[i,j]$ is length of LCS of $< x, 1, \ldots, x_i >$, $< y_1, \ldots, y_j >$.

Dynamic Programming

DP creates a search space and calculates optimal cost for every item in the search space.

Optimal cost of larger items is based on optimal cost of smaller items.

Final Result: usually cost of largest item in search space.

Running time of DP algorithm, is time required to calculate all costs.

Longest Common Subsequence: Find LCS of strings $X =< x, 1, \ldots, x_m >$, $Y =< y_1, \ldots, y_n >$.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = x_j \\ \max(c[i-1, j],\ c[i, j-1]) & \text{if } i, j > 0 \text{ and } x_i \neq x_j \end{cases}$$

$c[i,j]$ is length of LCS of $< x, 1, \ldots, x_i >$, $< y_1, \ldots, y_j >$.

Want $c[m,n]$ and corresponding LCS.

DP is taught to all CS undergrads.

Isn't it well understood? What's left to do?

DP is taught to all CS undergrads.

Isn't it well understood? What's left to do?

A LOT!:

There are techniques for *speeding up* DP computations by an order of magnitude.

Also techniques for *reducing space requirements* by an order of magnitude

DP is taught to all CS undergrads.

Isn't it well understood? What's left to do?

A LOT!:

There are techniques for *speeding up* DP computations by an order of magnitude.
Also techniques for *reducing space requirements* by an order of magnitude

Ad-hoc techniques, developed on problem-by-problem basis.
Individual techniques found wide applicability in areas outside original application.

DP is taught to all CS undergrads.

Isn't it well understood? What's left to do?

A LOT!:

There are techniques for *speeding up* DP computations by an order of magnitude.
Also techniques for *reducing space requirements* by an order of magnitude

Ad-hoc techniques, developed on problem-by-problem basis.
Individual techniques found wide applicability in areas outside original application.

New speedups are still being found, still on ad-hoc basis.
Crying need for a general theory of speedups, that can be referenced by application users.

In this talk, will combine

- one well-known time speedup:
  Monge Property + SMAWK algorithm and
- one basic $\Theta(n)$ space improvement
  (Hirschberg 1975)

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

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

$0 \le i \le n$ $\Theta(Dn^2)$ 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) $$H(i,d) = \min_{0 \le j < i} \Big(H(j,d-1) + w(j,i) \Big)$$

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

$0 \le d \le D$

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

+ Monge Property

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

$0 \le i \le n$

~~$\Theta(n^2)$ time~~

$\Theta(n)$ time

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

$0 \le i \le n$

$0 \le d \le D$

~~$\Theta(Dn^2)$ time~~

$\Theta(Dn)$ time

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

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

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

$0 \leq i \leq n$ $\quad n^2 \longrightarrow n$

$0 \leq d \leq D$ $\quad Dn^2 \longrightarrow Dn$

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

$0 \le i \le n$ $n^2 \rightarrow n$

$0 \le d \le D$ $Dn^2 \rightarrow Dn$

Calculating $H(n, D)$ requires only $O(n)$ space.

Note that storing the table uses $\Theta(Dn)$ space,
where D could be quite large.

Naive method of constructing solution from DP table,
requires backtracking through table
requires storing entire DP table
$\Rightarrow \Theta(Dn)$ space.

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

$0 \le i \le n$ $\quad n^2 \to n$

$0 \le d \le D$ $\quad Dn^2 \to Dn$

Calculating $H(n, D)$ requires only $O(n)$ space.

Note that storing the table uses $\Theta(Dn)$ space,
where D could be quite large.

Naive method of constructing solution from DP table,
requires backtracking through table
requires storing entire DP table
$\Rightarrow \Theta(Dn)$ space.

Will see how to reduce this to $O(n)$ space.

Outline

- The Monge Speedup

- Saving Space While Saving Time

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 $\forall i \le i'$, $\text{RM}_M(i) \le \text{RM}_M(i')$.

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 $\forall i \le i'$, $\text{RM}_M(i) \le \text{RM}_M(i')$.

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 $\forall i \leq i'$, $\text{RM}_M(i) \leq \text{RM}_M(i')$.
- 2×2 monotone matrices have form

2	4
4	5

2	3
5	3

7	1
2	2

~~7~~	~~1~~
~~2~~	~~3~~

- 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 $O(n)$.
  - See http://www.cs.ust.hk/mjg_lib/Classes/COMP572_Fall07/Notes/SMAWK.pdf for proof

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 $O(n)$.
  - See http://www.cs.ust.hk/mjg_lib/Classes/COMP572_Fall07/Notes/SMAWK.pdf for proof
- 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.

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 $\forall i \leq i'$ 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

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 $\forall i \leq i'$ 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.
- $\Rightarrow$ Only need to prove Monge property for adjacent rows and columns.

An Example of a Monge Matrix

An Example of a Monge Matrix

From http://en.wikipedia.org/wiki/Monge_array

$$\begin{bmatrix} 10 & 17 & 13 & 28 & 23 \\ 17 & 22 & 16 & 29 & 23 \\ 24 & 28 & 22 & 34 & 24 \\ 11 & 13 & 6 & 17 & 7 \\ 45 & 44 & 32 & 37 & 23 \\ 36 & 33 & 19 & 21 & 6 \\ 75 & 66 & 51 & 53 & 34 \end{bmatrix}$$

To see that it's Monge, only need to check the 24 instances of

$$M_{i,j} + M_{i+1,j+1} \leq M_{i+1,j} + M_{i,j+1}$$

An Example of a Monge Matrix

From http://en.wikipedia.org/wiki/Monge_array

$$\begin{bmatrix} 10 & 17 & 13 & 28 & 23 \\ 17 & 22 & 16 & 29 & 23 \\ 24 & 28 & 22 & 34 & 24 \\ 11 & 13 & 6 & 17 & 7 \\ 45 & 44 & 32 & 37 & 23 \\ 36 & 33 & 19 & 21 & 6 \\ 75 & 66 & 51 & 53 & 34 \end{bmatrix}$$

To see that it's Monge, only need to check the 24 instances of

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

e.g., $10 + 22 \le 17 + 17$

An Example of a Monge Matrix

From http://en.wikipedia.org/wiki/Monge_array

$$\begin{bmatrix} 10 & 17 & 13 & 28 & 23 \\ 17 & 22 & 16 & 29 & 23 \\ 24 & 28 & 22 & 34 & 24 \\ 11 & 13 & 6 & 17 & 7 \\ 45 & 44 & 32 & 37 & 23 \\ 36 & 33 & 19 & 21 & 6 \\ 75 & 66 & 51 & 53 & 34 \end{bmatrix}$$

To see that it's Monge, only need to check the 24 instances of

$$M_{i,j} + M_{i+1,j+1} \leq M_{i+1,j} + M_{i,j+1}$$

e.g., $10 + 22 \leq 17 + 17$

Since it's Monge, it's Totally Monotone, so the SMAWK algorithm can find the row minima in time linear in the perimeter (not area) or the matrix!

An Example of a Monge Matrix

From http://en.wikipedia.org/wiki/Monge_array

10	17	13	28	23
17	22	**16**	29	23
24	28	**22**	34	24
11	13	**6**	17	7
45	44	**32**	37	23
36	33	19	21	**6**
75	66	51	53	**34**

To see that it's Monge, only need to check the 24 instances of

$$M_{i,j} + M_{i+1,j+1} \leq M_{i+1,j} + M_{i,j+1}$$

e.g., $10 + 22 \leq 17 + 17$

Since it's Monge, it's Totally Monotone, so the SMAWK algorithm can find the row minima in time linear in the perimeter (not area) or the matrix!

An Example of a Monge Matrix

From http://en.wikipedia.org/wiki/Monge_array

10	17	13	28	23
17	22	16	29	23
24	28	22	34	24
11	13	6	17	7
45	44	32	37	23
36	33	19	21	6
75	66	51	53	34

To see that it's Monge, only need to check the 24 instances of

$$M_{i,j} + M_{i+1,j+1} \leq M_{i+1,j} + M_{i,j+1}$$

e.g., $10 + 22 \leq 17 + 17$

Since it's Monge, it's Totally Monotone, so the SMAWK algorithm can find the row minima in time linear in the perimeter (not area) or the matrix!

Monge (or Total Monotonicity) seems an esoteric condition. In reality, it occurs *very* often.

Finding row minima can be used as a DP primitive.
$\Rightarrow$ the SMAWK algorithm can be used to speed up many DPs.

Using The Monge Property

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

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

Using The Monge Property

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

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

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

Using The Monge Property

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

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

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

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

Using The Monge Property

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

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

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

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

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

Using The Monge Property

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

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

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

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

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

Given $H(*,d-1)$, SMAWK finds all $H(*,d)$ in $O(n)$ time;

Using The Monge Property

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

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

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

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

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

Given $H(*, d-1)$, SMAWK finds all $H(*, d)$ in $O(n)$ time;

$$H(*,0) \overset{O(n)}{\Rightarrow} H(*,1) \overset{O(n)}{\Rightarrow} H(*,2) \overset{O(n)}{\Rightarrow} \cdots \overset{O(n)}{\Rightarrow} H(*,d)$$

Using The Monge Property

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

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

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

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

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

Given $H(*, d-1)$, SMAWK finds all $H(*, d)$ in $O(n)$ time;

$$H(*,0) \stackrel{O(n)}{\Rightarrow} H(*,1) \stackrel{O(n)}{\Rightarrow} H(*,2) \stackrel{O(n)}{\Rightarrow} \cdots \stackrel{O(n)}{\Rightarrow} H(*,d)$$

So, $O(Dn)$ time to calculate $H(n,d)$ and we are done!

Examples of

$i \leq n,\ d \leq D$

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

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

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

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

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

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

Examples of

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

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

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

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

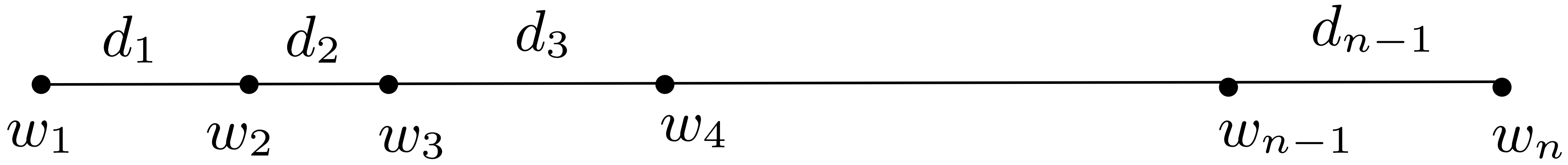
  $H(n-1, D)$ is cost of min-cost D-limited code

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

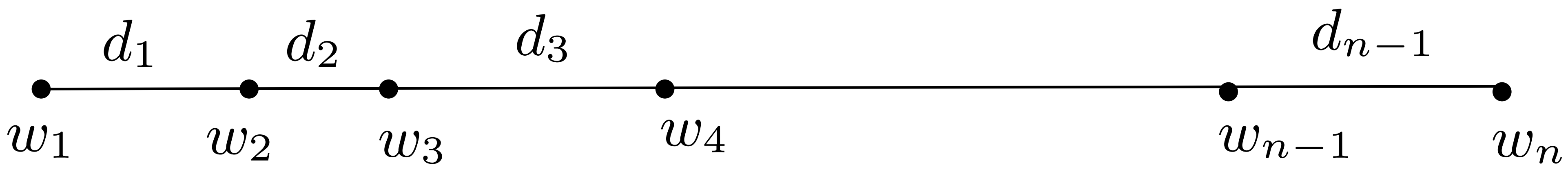
  $w(j,i) = i\left(\sum_{\ell=j+1}^{i} p_\ell\right)$

  $H(n, D)$ 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*



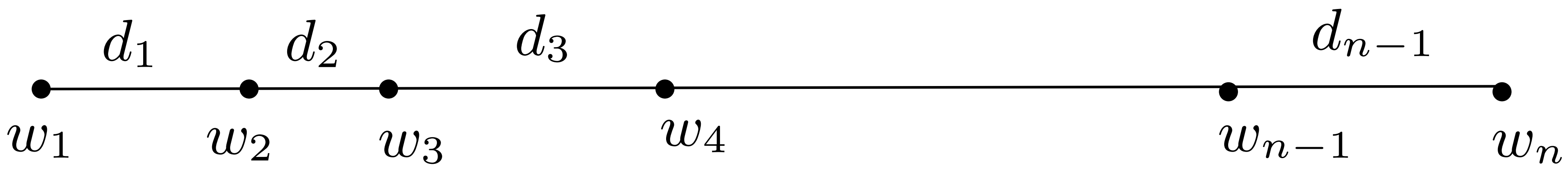
Identify D nodes as service centers.

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.

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

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

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

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

- Length Limited Huffman Codes

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

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

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

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

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

- Length Limited Huffman Codes

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

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

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

All these $w(j,i) = w_{j,i}$ 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

Given a DP in the form

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

in which, the $w(j,i)$ 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

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

in which, the $w(j,i)$ 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:

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

D-Length-Limited Huffman Coding

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

Context:

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

D-Length-Limited Huffman Coding

(*) $w(j,i) = S_{2j-i}$ 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:

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

D-Length-Limited Huffman Coding

(*) $w(j,i) = S_{2j-i}$ 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:

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

D-Length-Limited Huffman Coding

(*) $w(j,i) = S_{2j-i}$ 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.

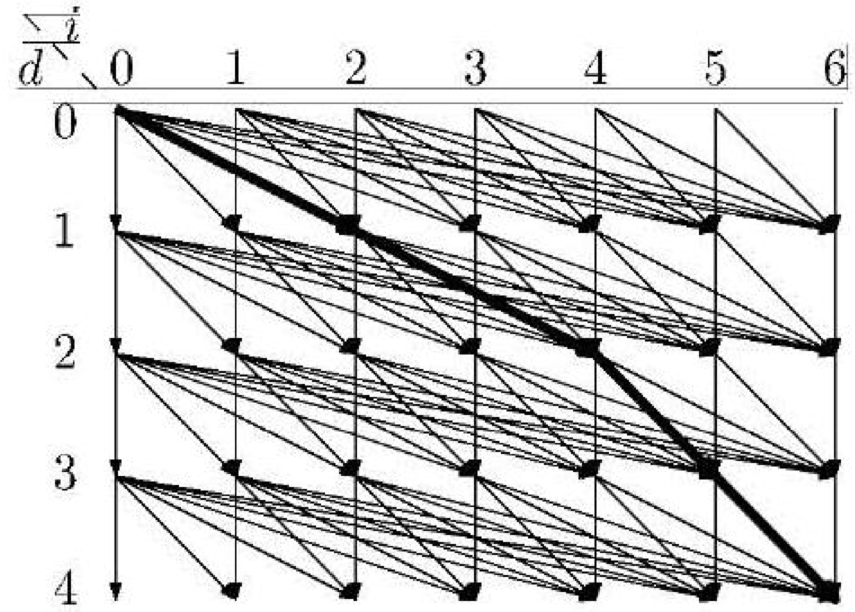
Would like to reduce space for (*) down to $\Theta(n)$

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

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

Alternative Interpretation:

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

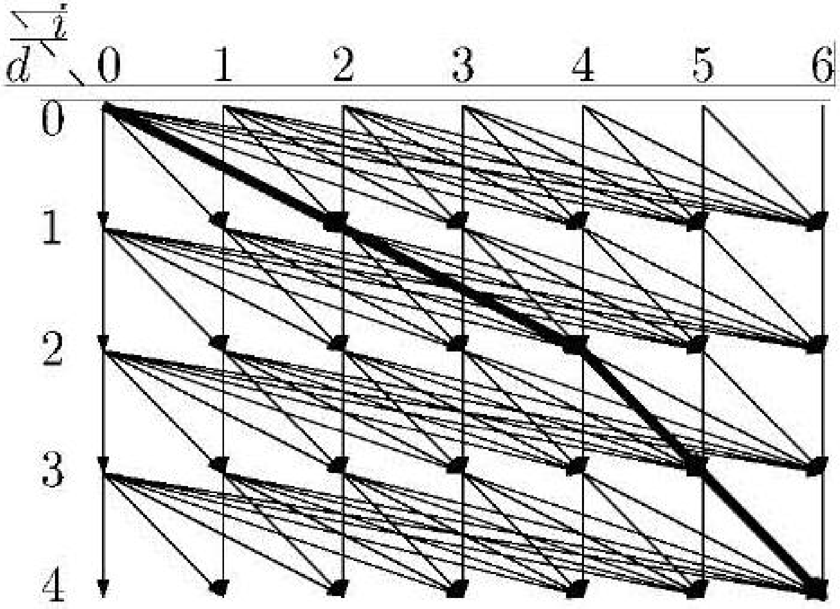


$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w(j,i) \Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

Alternative Interpretation:

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

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

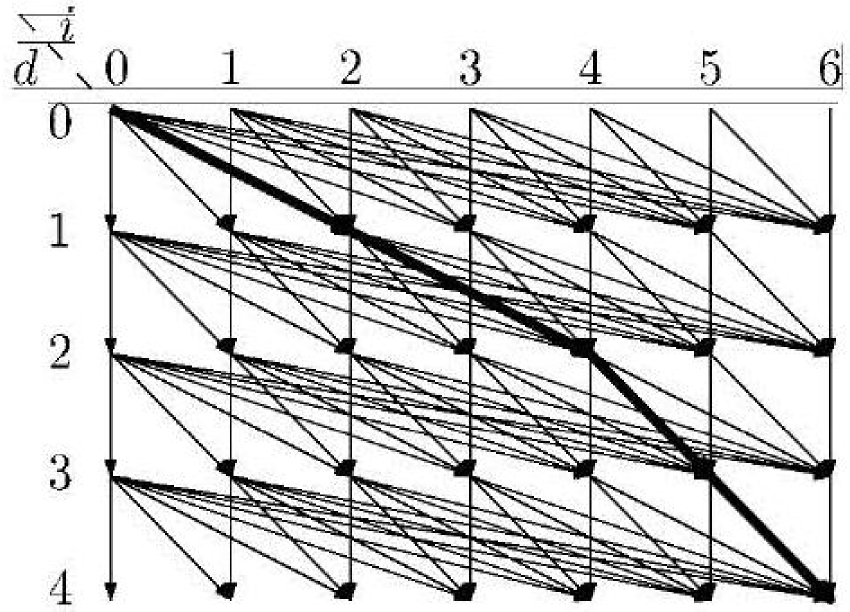


$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w(j,i) \Big) \qquad \begin{matrix} 0 \le i \le n \\ 0 \le d \le D \end{matrix}$$

Alternative Interpretation:

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

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



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

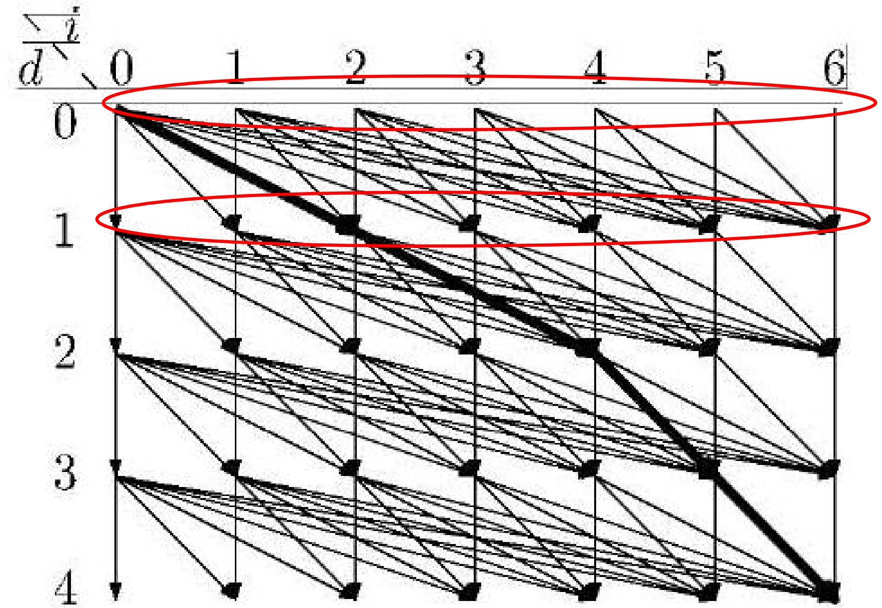
Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

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

Alternative Interpretation:

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

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



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

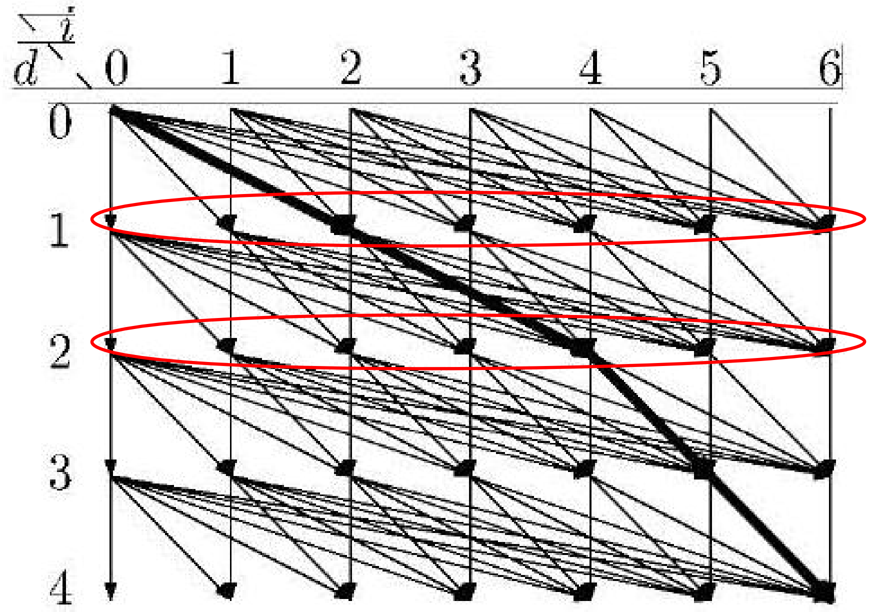
Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

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

Alternative Interpretation:

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

$$w\big(\ (d-1,j) \to (d,i)\ \big) = w(j,i)$$



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

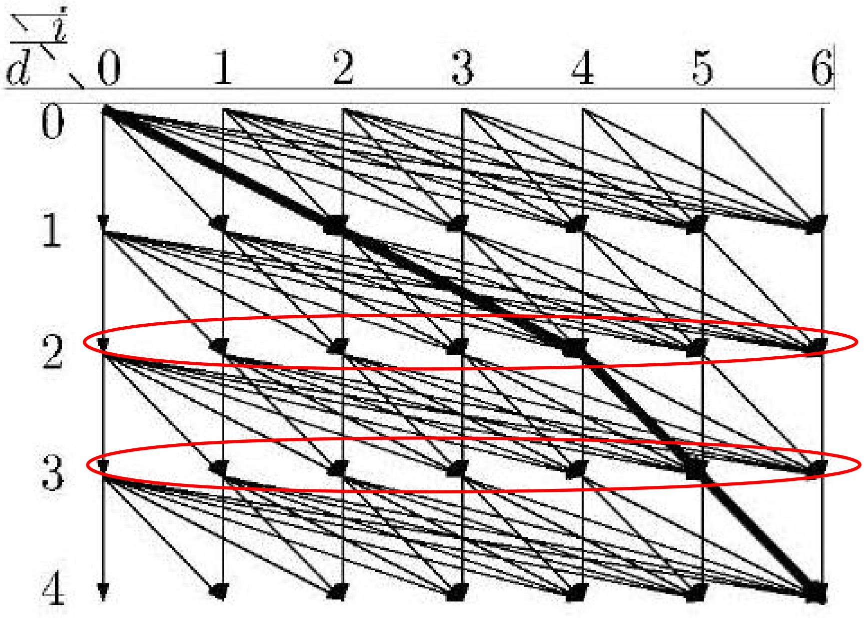
Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

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

Alternative Interpretation:

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

$$w\big(\ (d-1, j) \to (d, i)\ \big) = w(j,i)$$



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

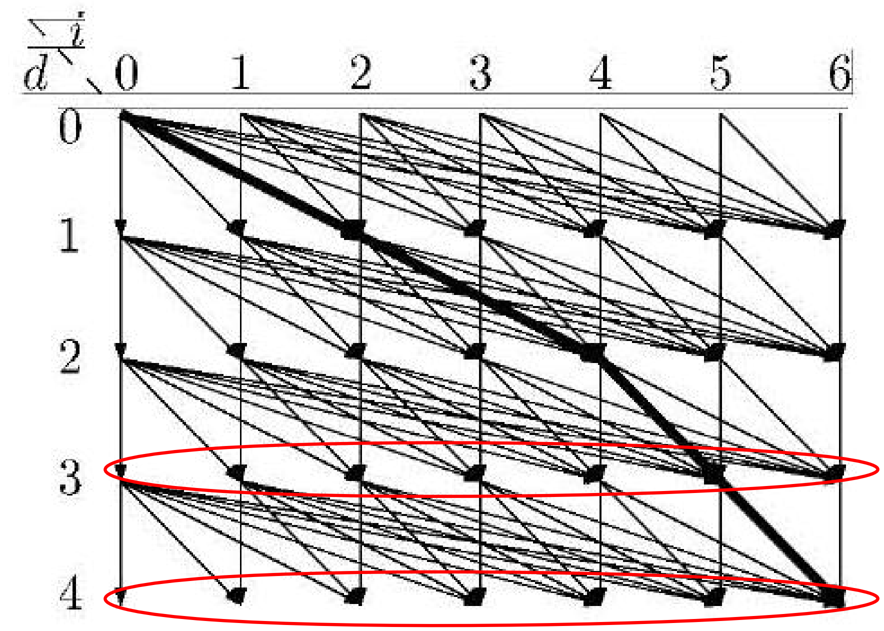
Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

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

Alternative Interpretation:

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

$$w\big(\ (d-1, j) \to (d, i)\ \big) = w(j,i)$$



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

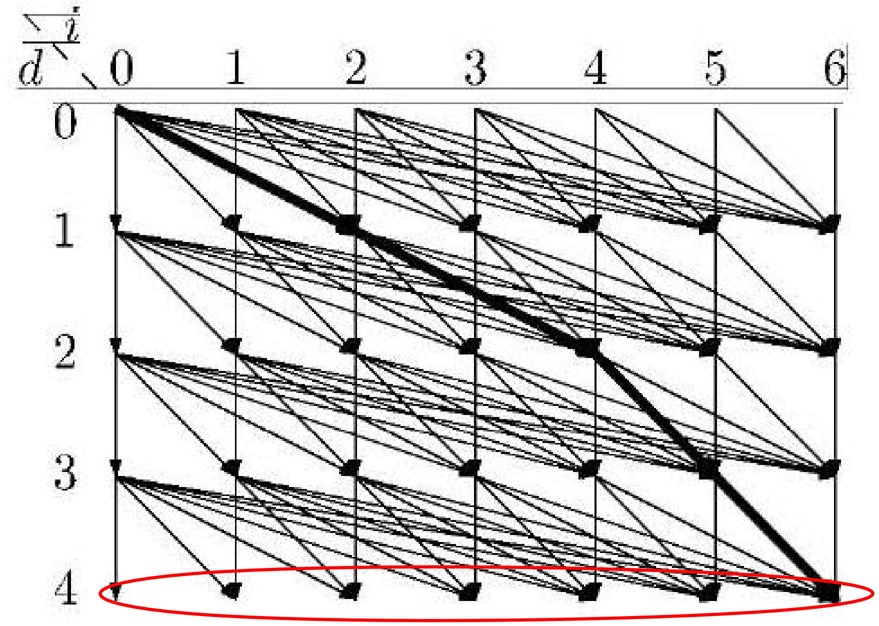
Given row $H(\cdot, d-1)$, SMAWK calculates row $H(\cdot, d)$ in $O(n)$ time. By throwing away uneeded rows, can calculate $H(\cdot, D)$ in $O(nD)$ time and $O(D)$ space.

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

Alternative Interpretation:

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

$$w\Big(\; (d-1, j) \to (d,i) \; \Big) = w(j,i)$$



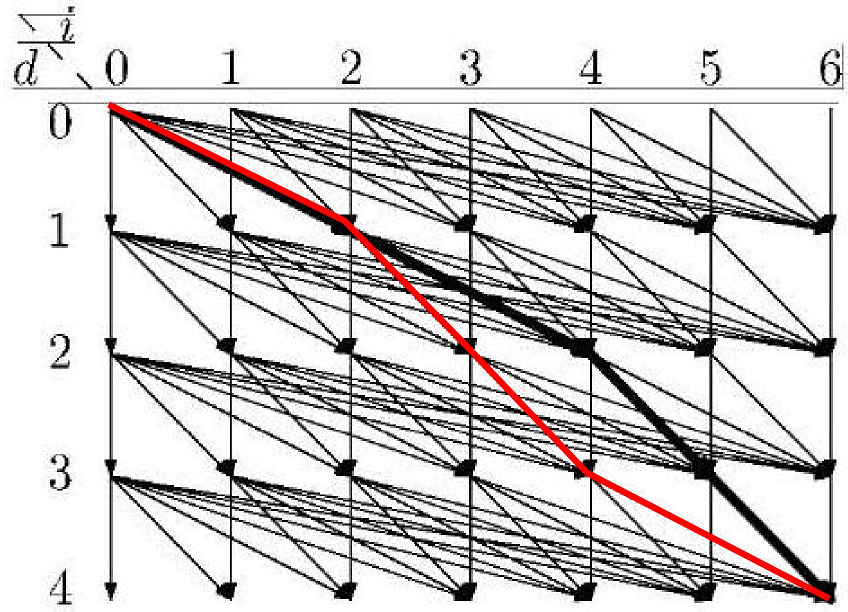
$H(i,d) =$ cost of min-cost path from $(0,0)$ to (d,i).

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.

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

Alternative Interpretation:

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



$$w\big(\; (d-1, j) \to (d, i) \; \big) = w(j,i)$$

$H(i,d) =$ 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

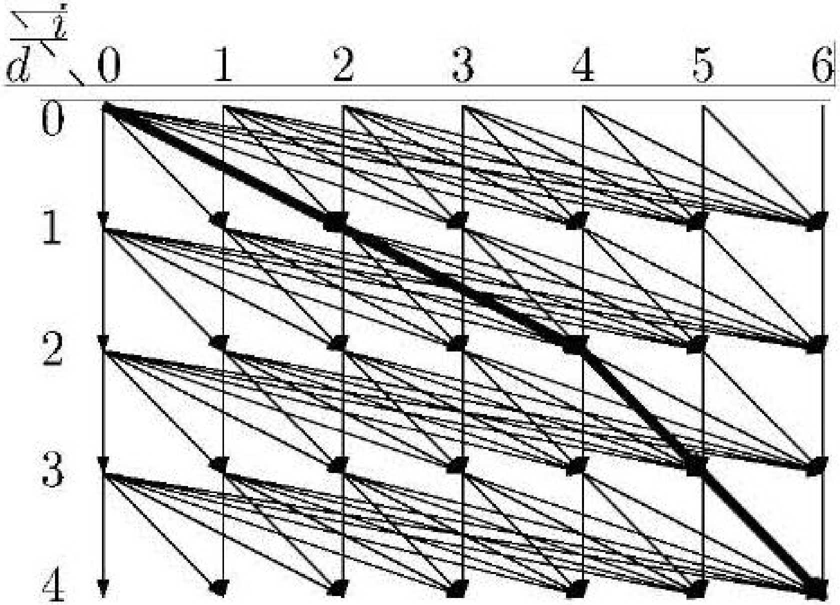
$$H(i,d) = \min_{0 \le j < i} \Big(H(j, d-1) + w(j,i) \Big) \qquad \begin{array}{l} 0 \le i \le n \\ 0 \le d \le D \end{array}$$

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

Modification of idea due to

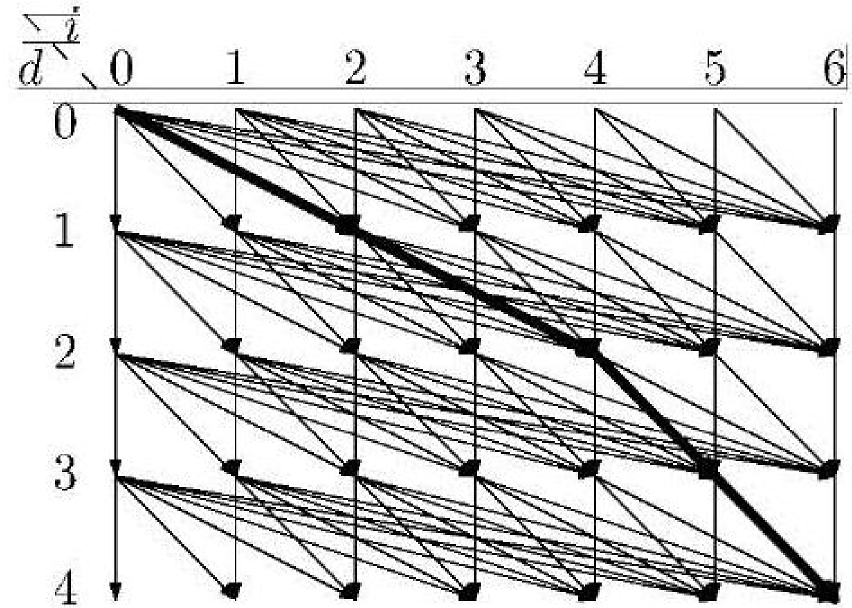
Hirschberg ('75)

Munro & Ramirez ('82)



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

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

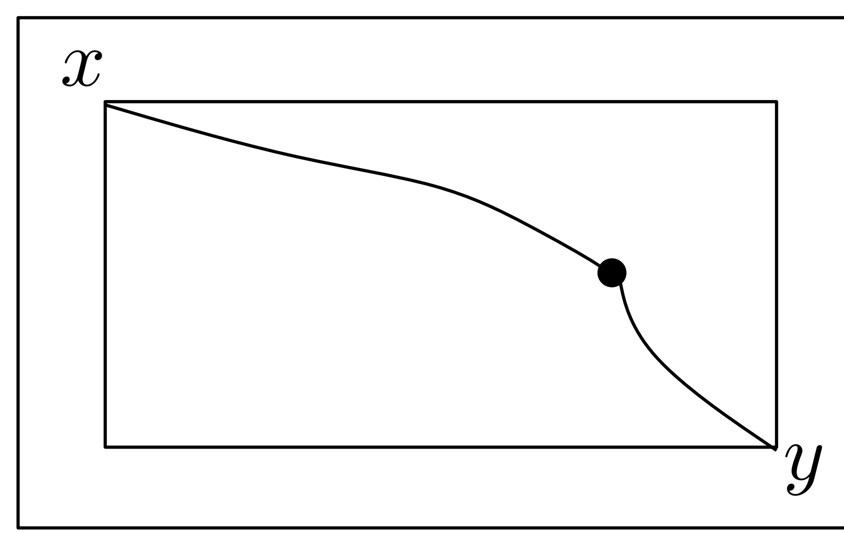


Modification of idea due to

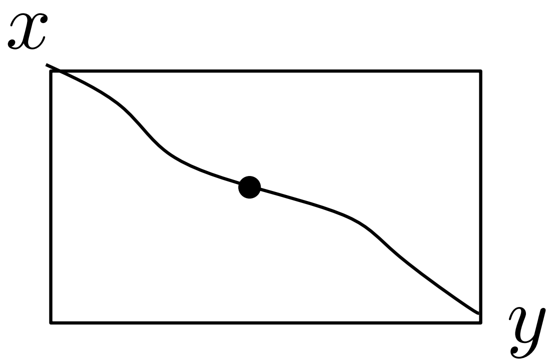
Hirschberg ('75)

Munro & Ramirez ('82)

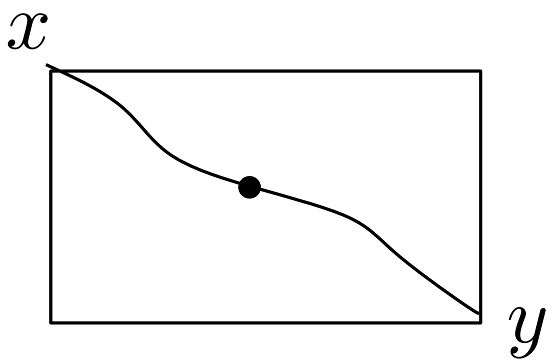
Let y be below and to the right of x. Assume existence of an oracle $Mid(x,y)$ that returns a midpoint (hop distance) on some min-cost x-y path.



$Mid(x, y)$ 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 $(x \to y)$

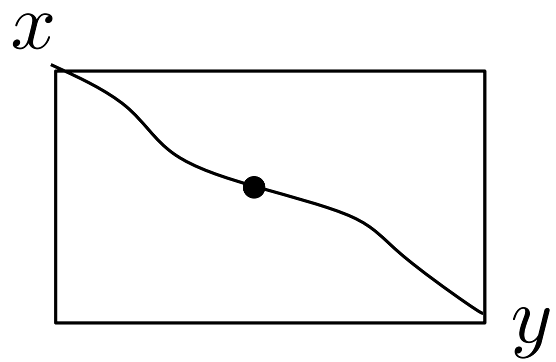
else

$z = Mid(x, y)$

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 $y_d = x_{d+1}$

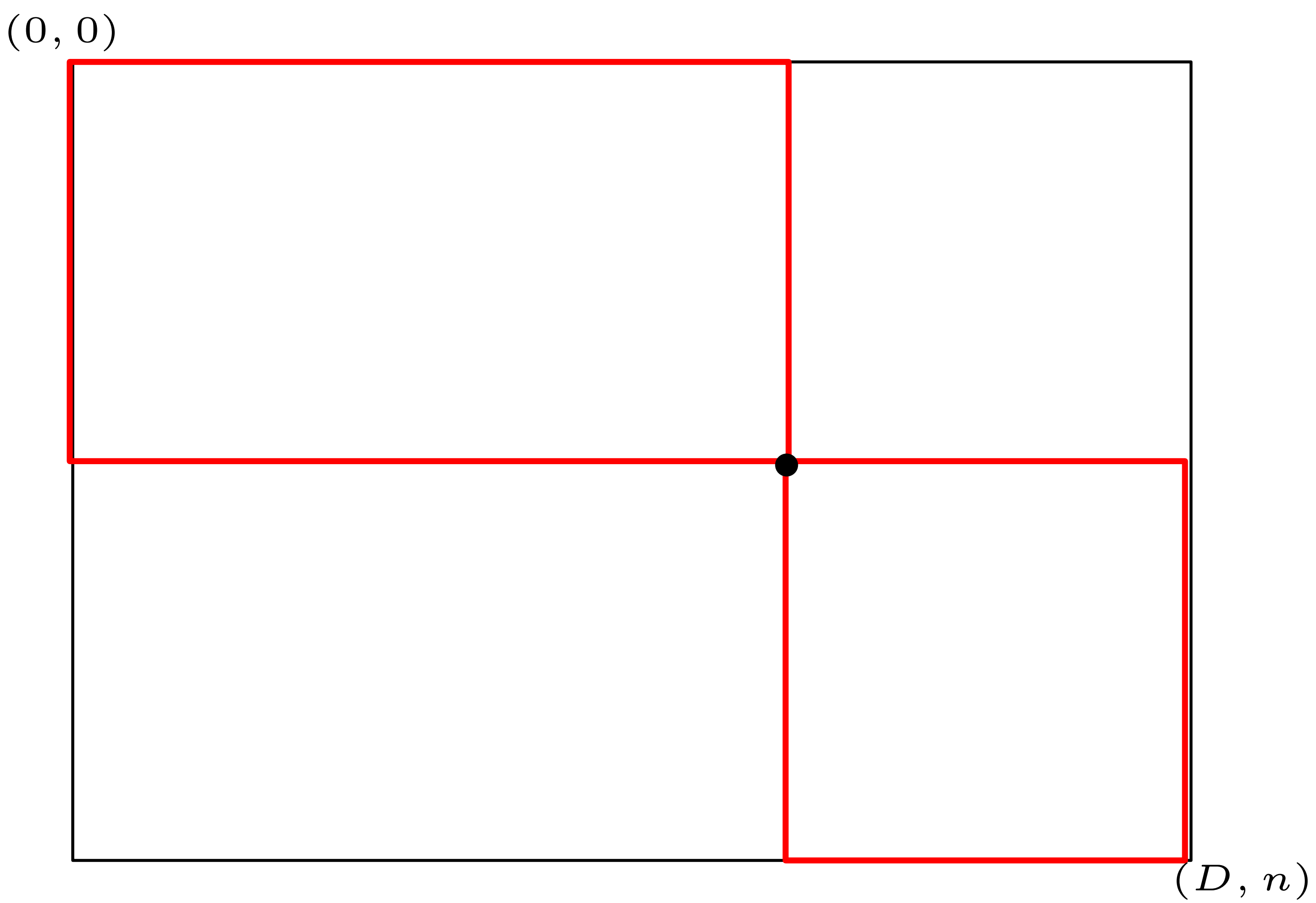
return $(x \to y)$

else

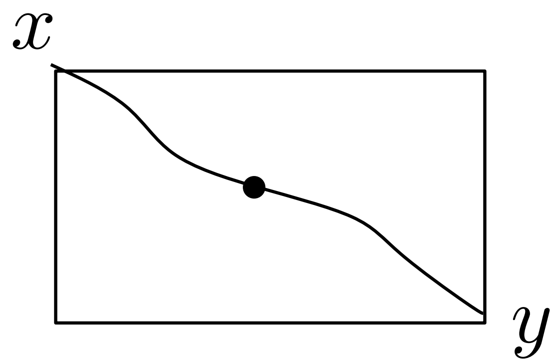
$z = Mid(x, y)$

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 $y_d = x_{d+1}$

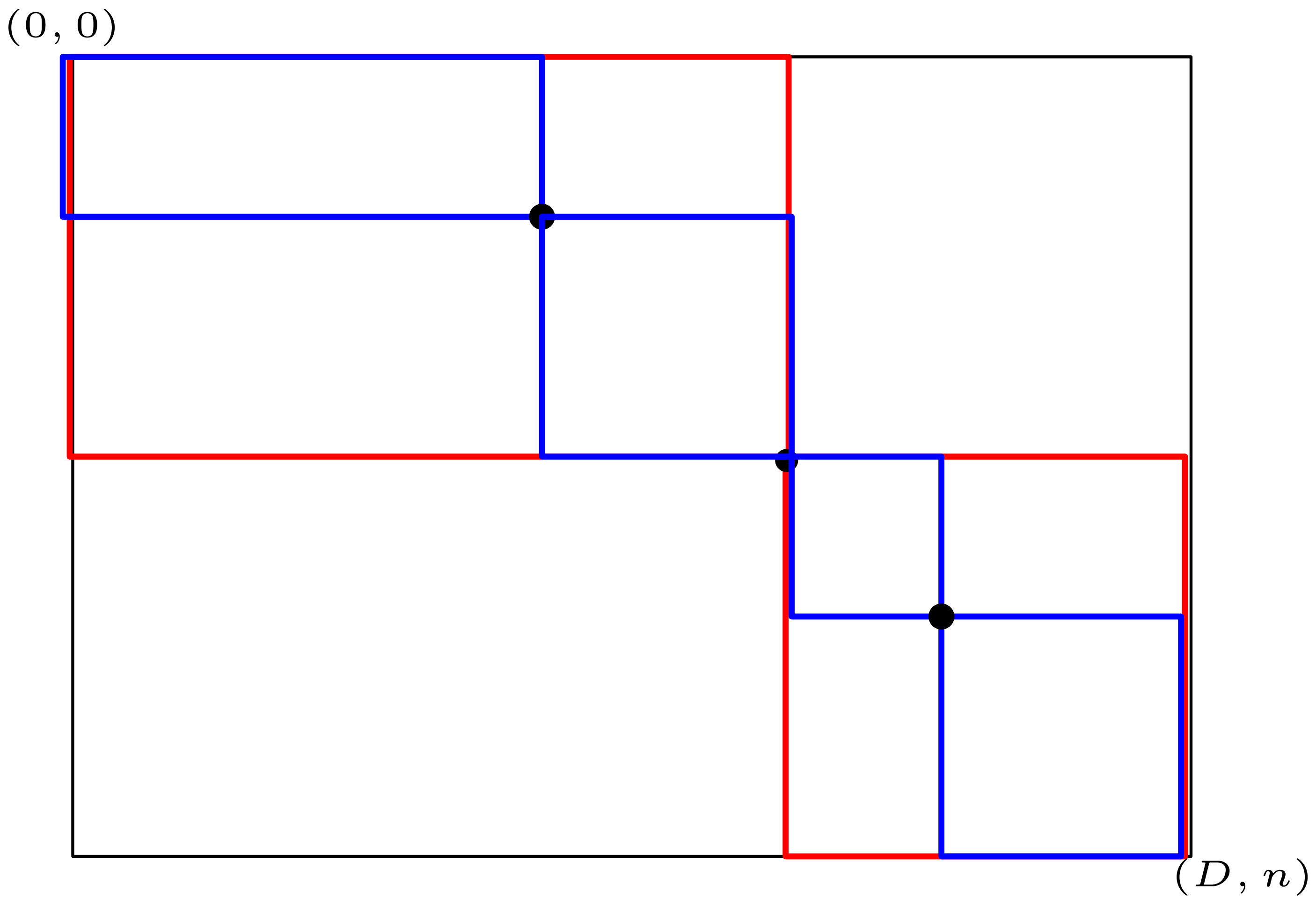
return $(x \to y)$

else

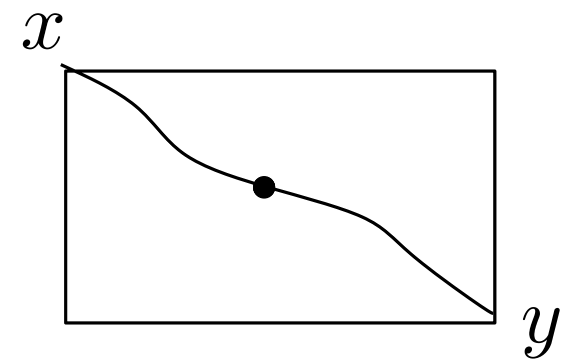
$z = Mid(x, y)$

Buildpath(x,z)

Buildpath(z,y)



$Mid(x, y)$ 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 $y_d = x_{d+1}$

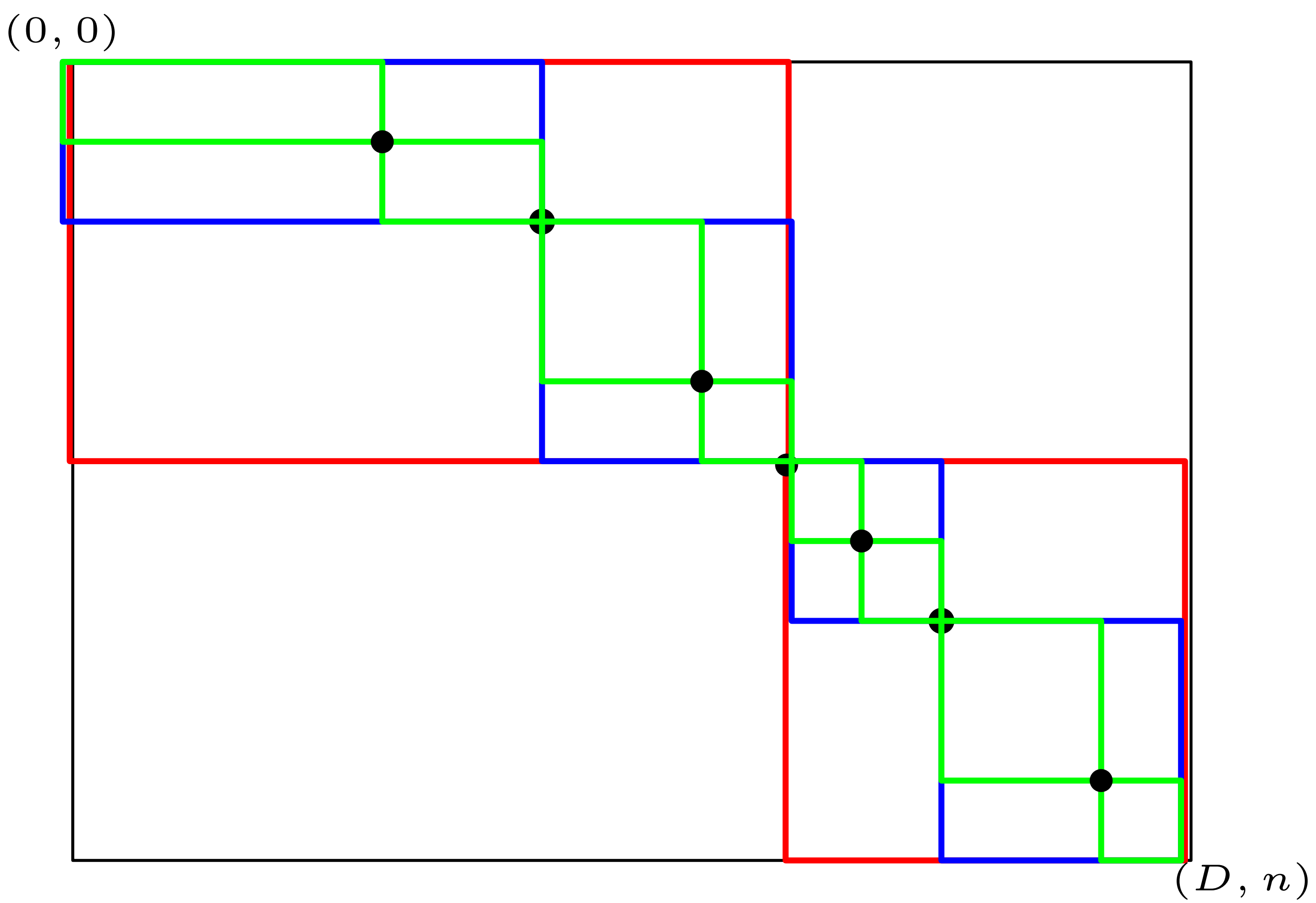
return $(x \to y)$

else

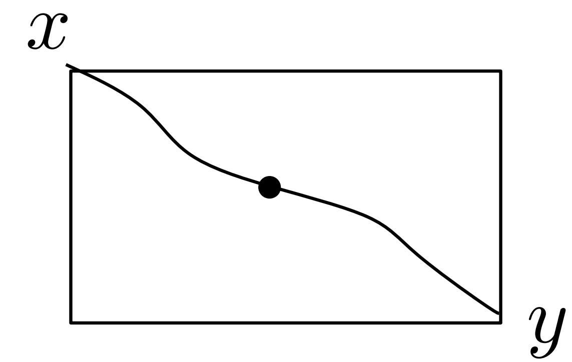
$z = Mid(x, y)$

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 $y_d = x_{d+1}$

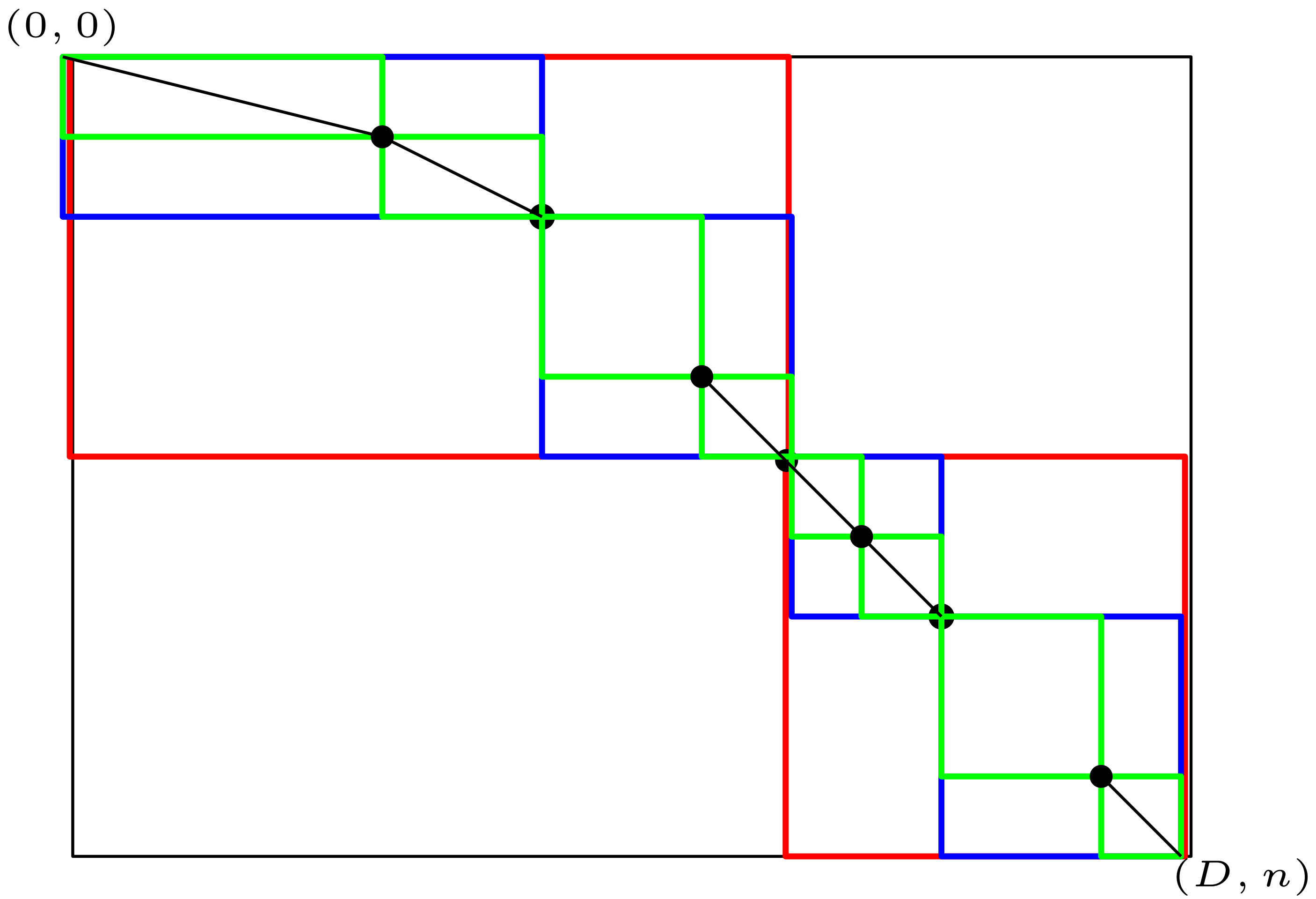
return $(x \rightarrow y)$

else

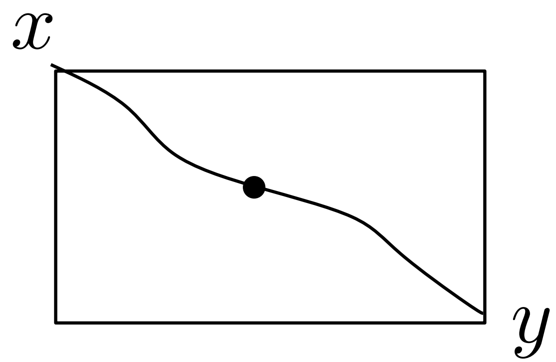
$z = Mid(x, y)$

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

<u>Buildpath(x,y)</u>

If $y_d = x_{d+1}$

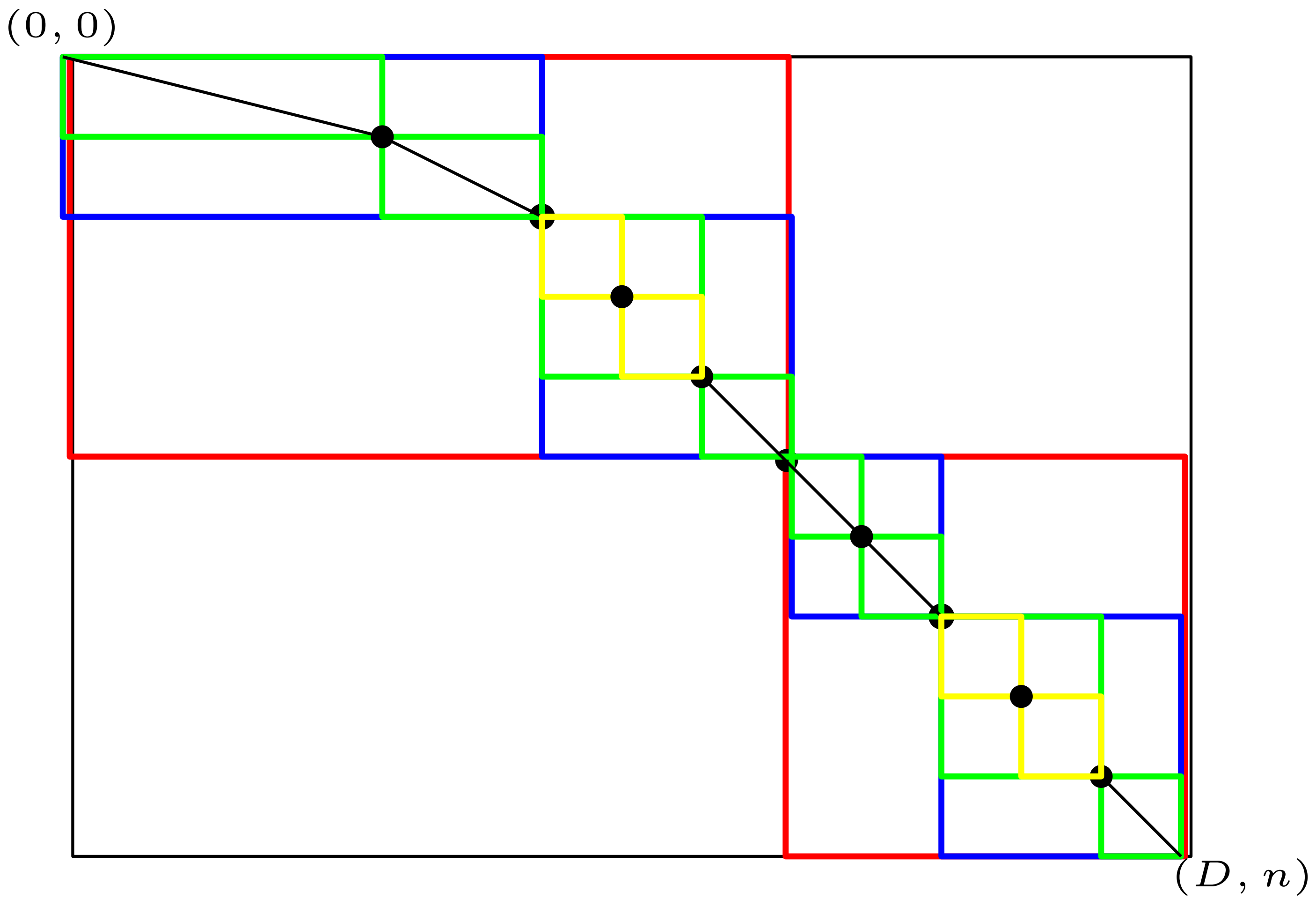
return $(x \to y)$

else

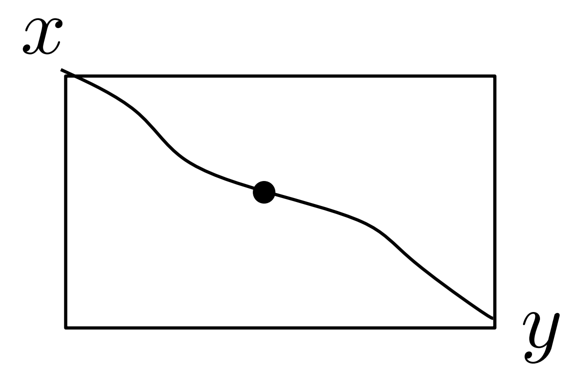
$z = Mid(x, y)$

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 $y_d = x_{d+1}$

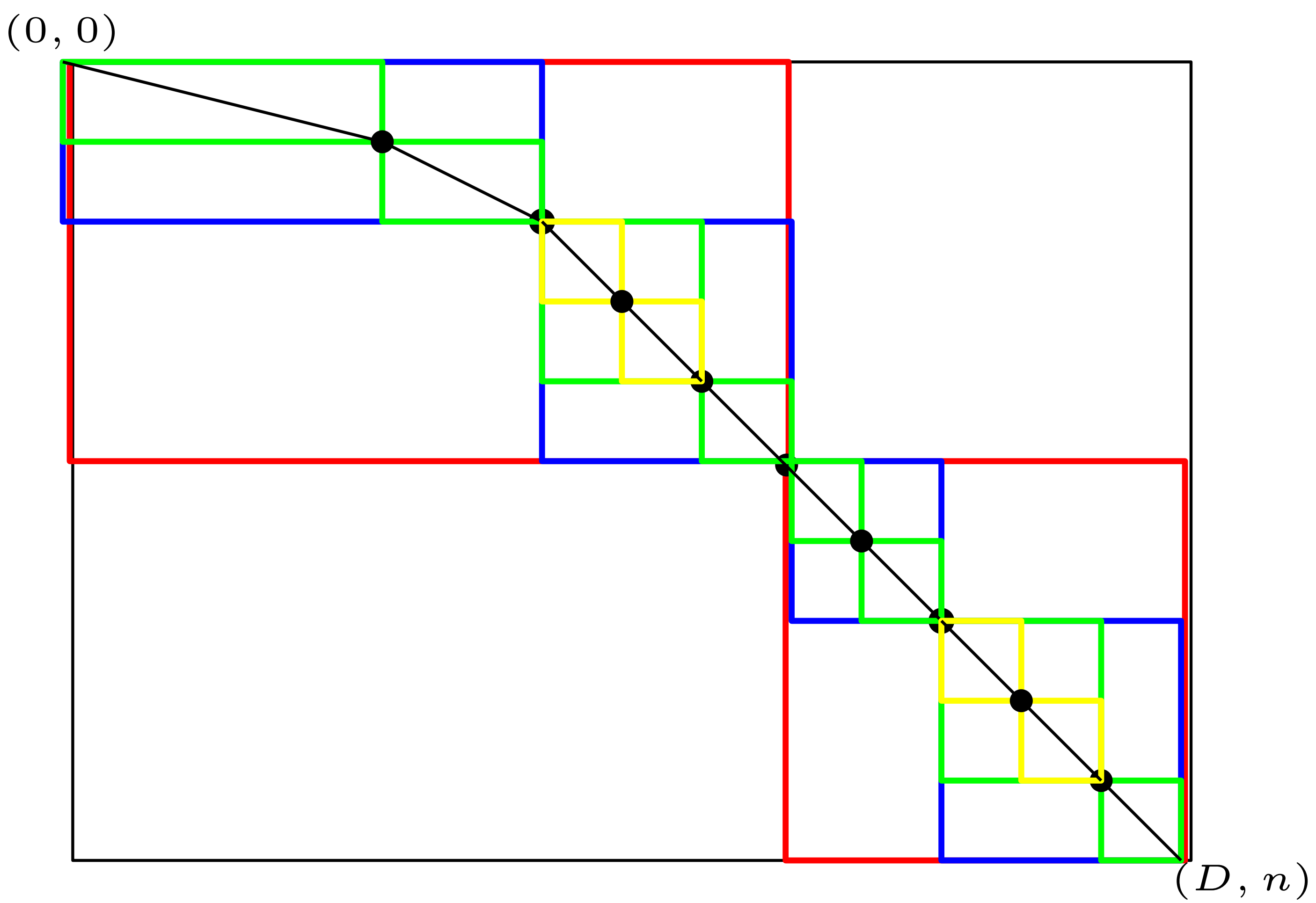
return $(x \to y)$

else

$z = Mid(x, y)$

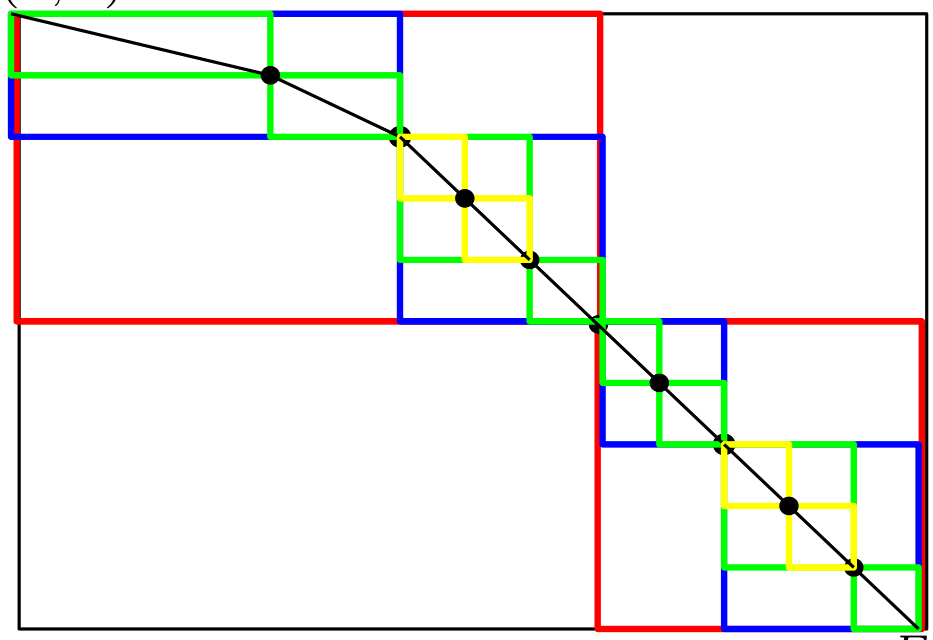
Buildpath(x,z)

Buildpath(z,y)



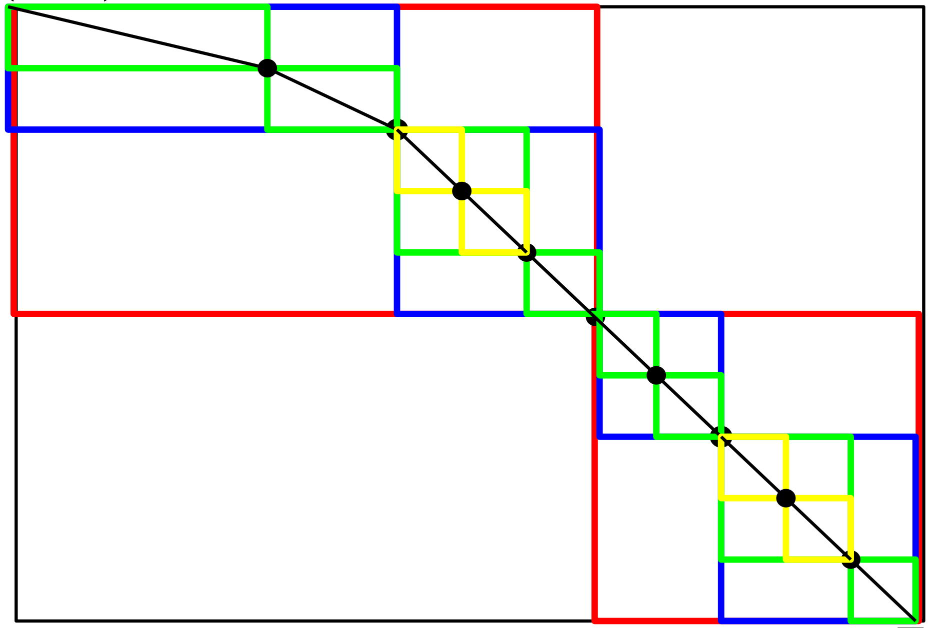
Buildpath(x,y)

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



Buildpath(x,y)

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

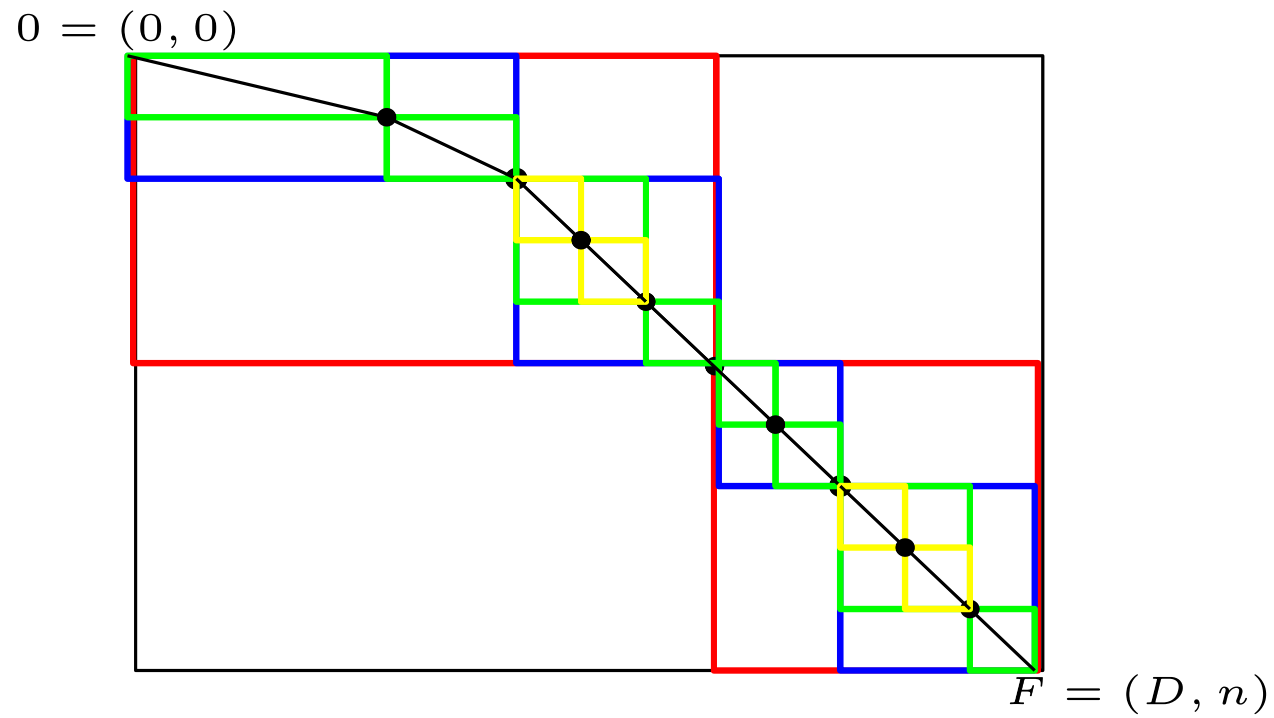


Lemma: If $Mid(x, y)$ uses $O(D + n)$ space

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

Buildpath(x,y)

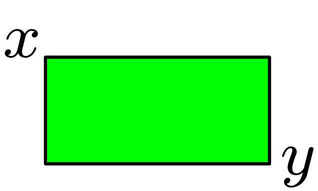
If $y_d = x_{d+1}$
  return $(x \rightarrow y)$
else
  $z = Mid(x, y)$
  Buildpath(x,z)
  Buildpath(z,y)



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

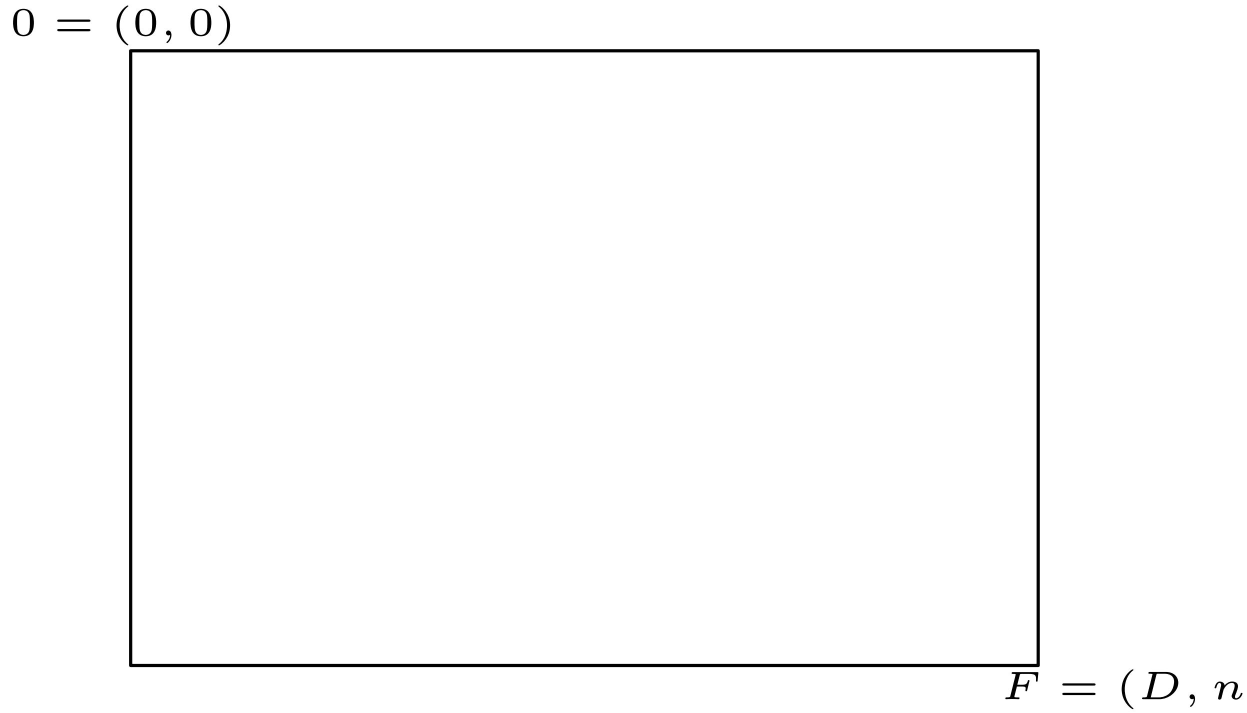


If $Mid(x, y)$ uses $O(Area(x, y))$ time

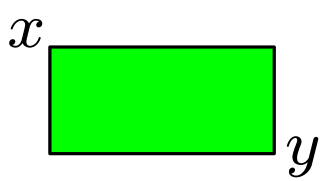
$\Longrightarrow$ Buildpath(0,F) uses $O(Dn)$ time

Buildpath(x,y)

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

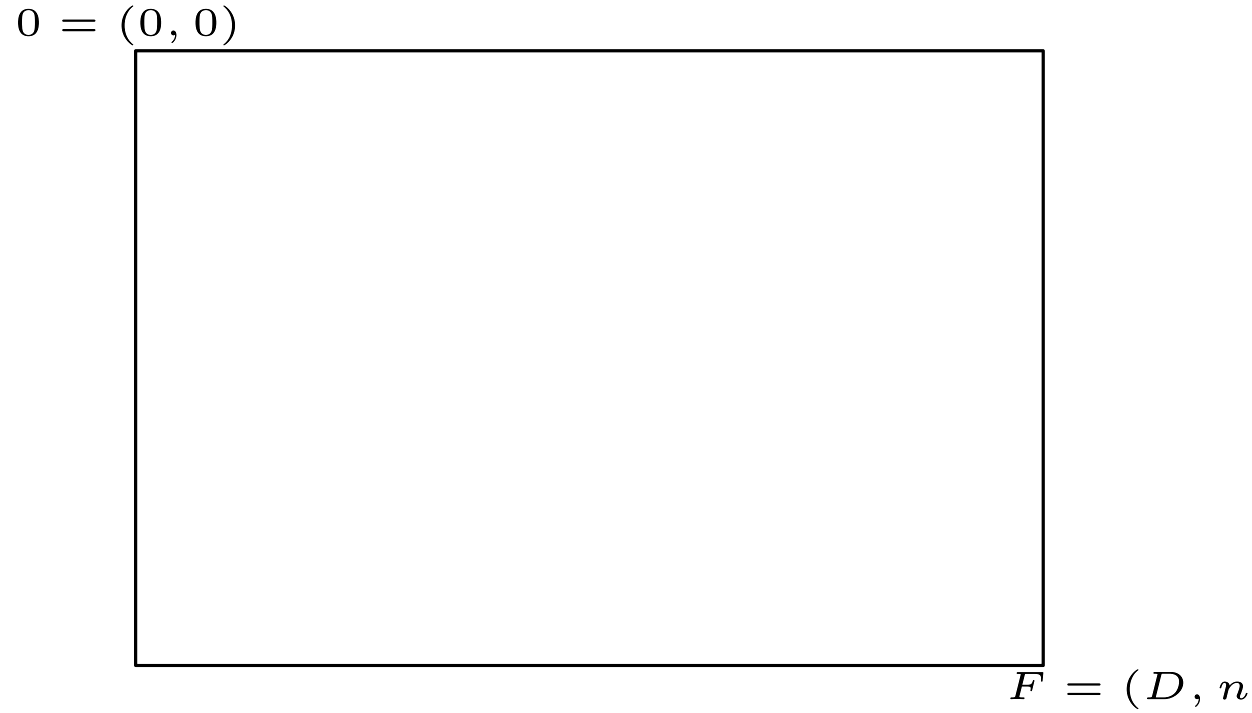


Lemma: Let $Area(x,y)$ be area of x,y box



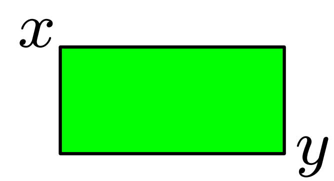
If $Mid(x,y)$ uses $O(Area(x,y))$ time

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



Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \rightarrow y)$
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

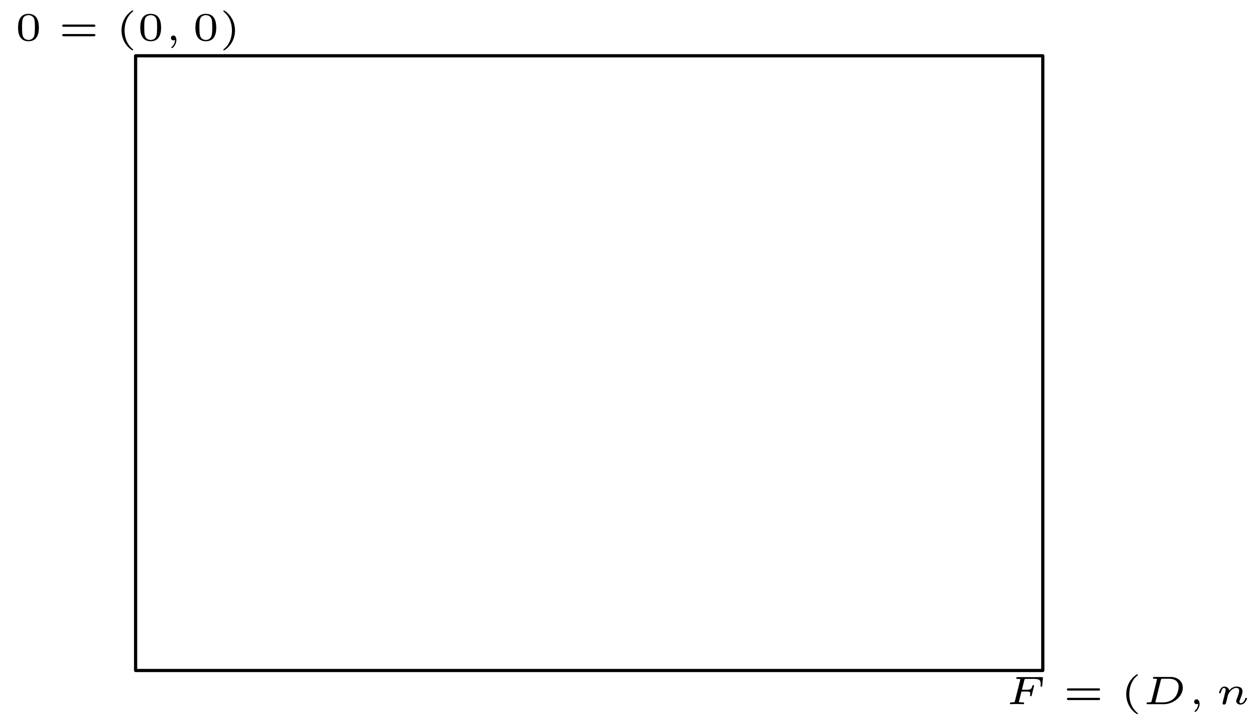
Proof: Rectangles at recursion level i are height $\leq D/2^i$

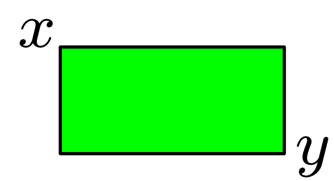
$\Longrightarrow$ Total work at level i is $\leq nD/2^i$

$\Longrightarrow$ Total work $\leq$

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

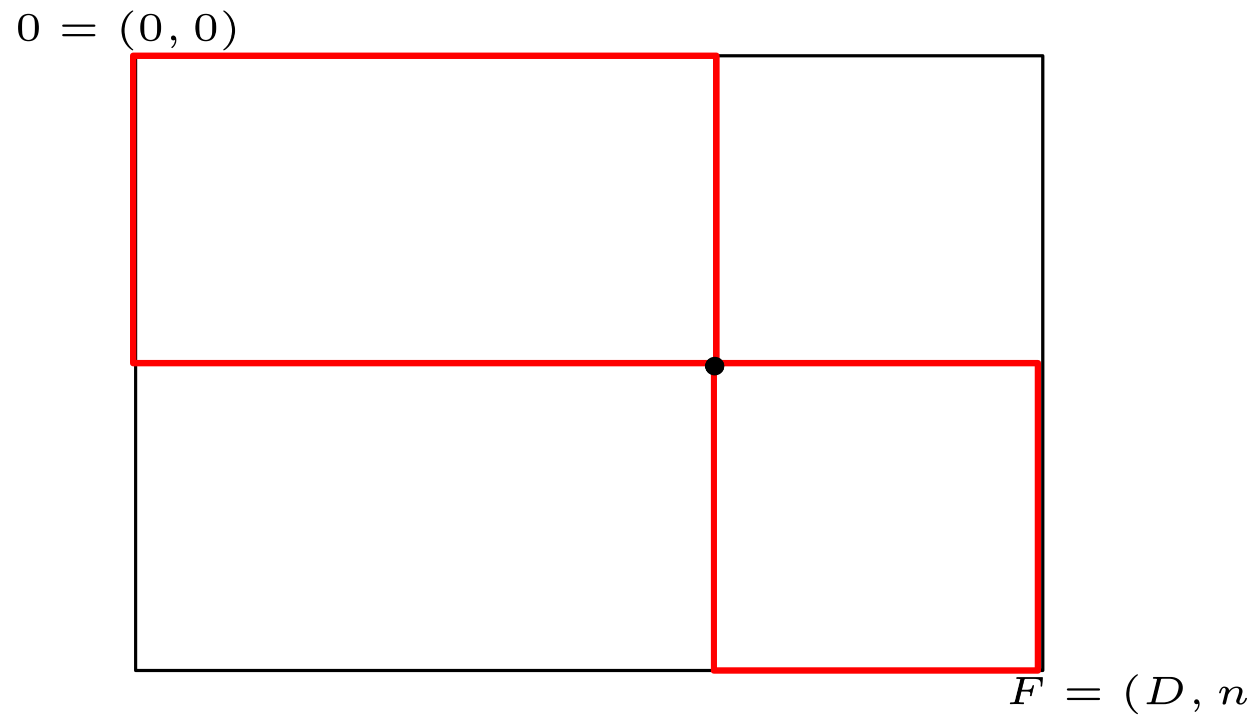
Proof: Rectangles at recursion level i are height $\leq D/2^i$

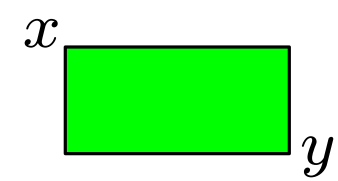
$\Longrightarrow$ Total work at level i is $\leq nD/2^i$

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

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

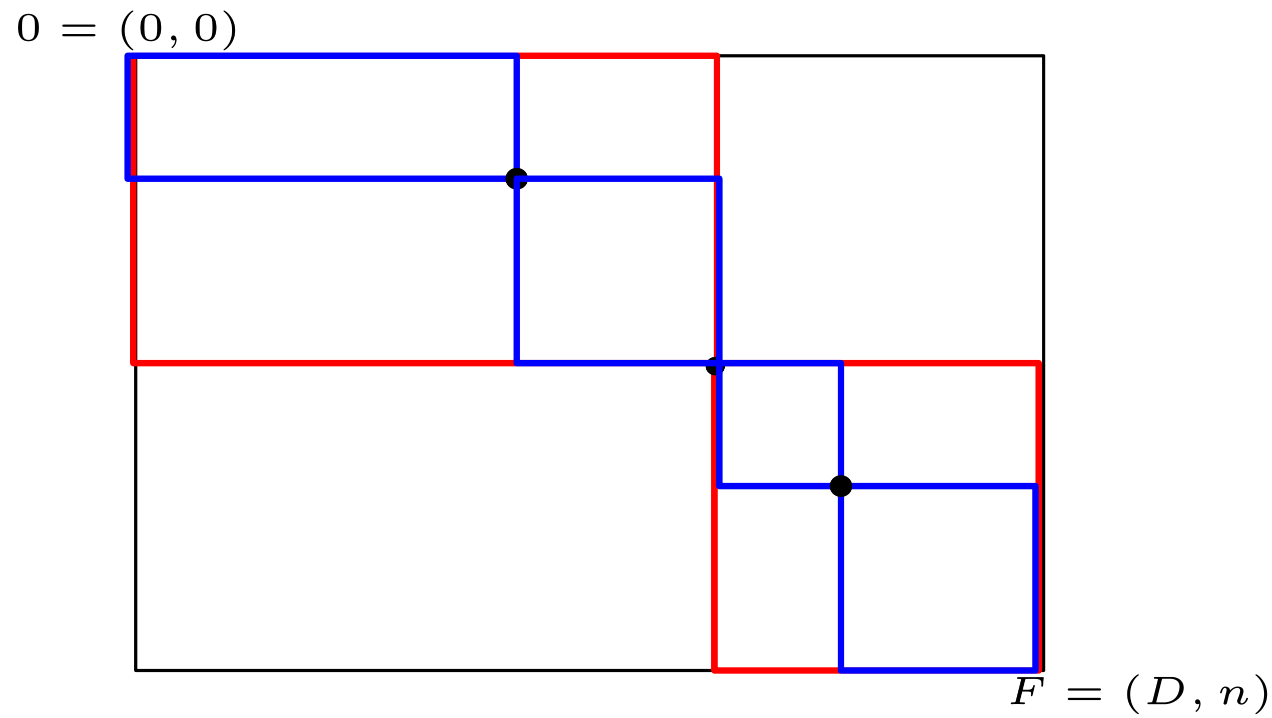
Proof: Rectangles at recursion level i are height $\leq D/2^i$

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

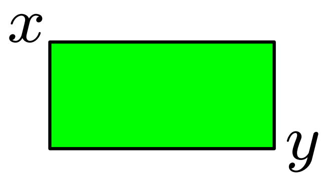
$\Longrightarrow$ Total work $\leq n \left(\frac{D}{2^0} + \frac{D}{2^1} \right.$

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



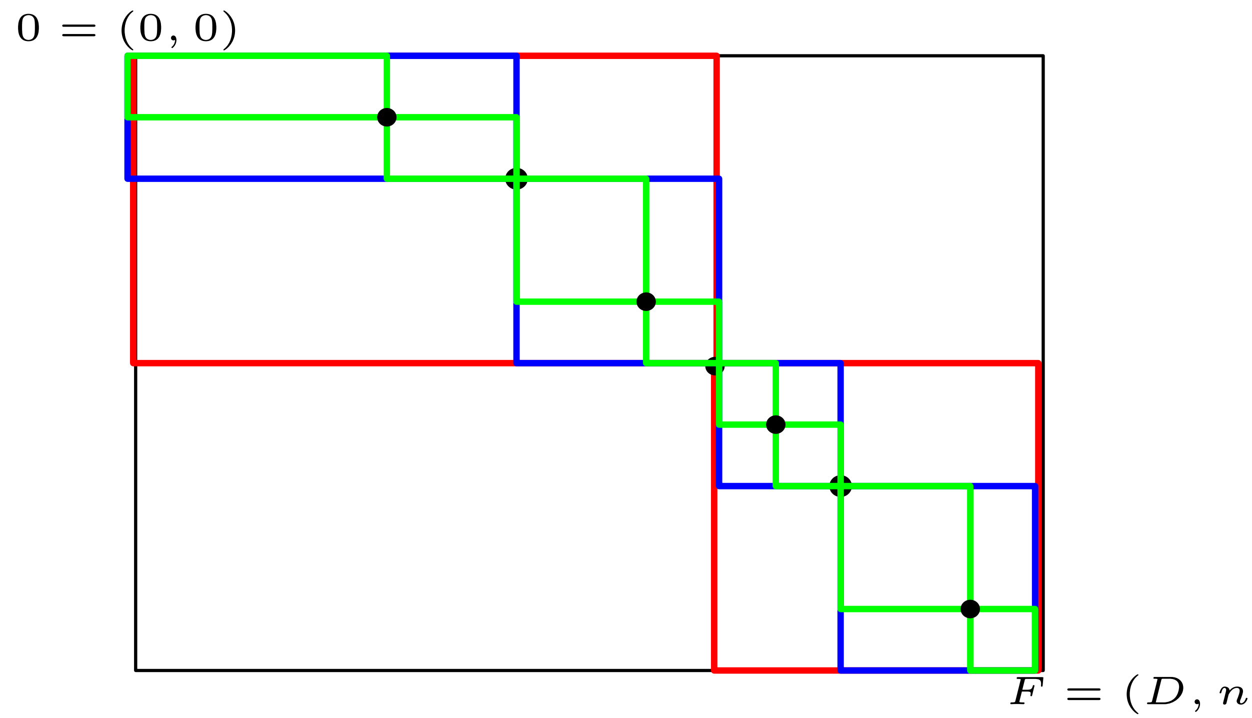
If $Mid(x,y)$ uses $O(Area(x,y))$ time

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

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

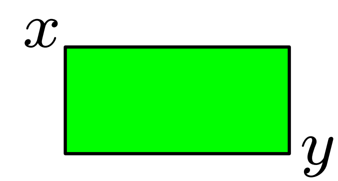
$\Longrightarrow$ Total work at level i is $\leq nD/2^i$

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



Buildpath(x,y)

If $y_d = x_{d+1}$
  return $(x \rightarrow y)$
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

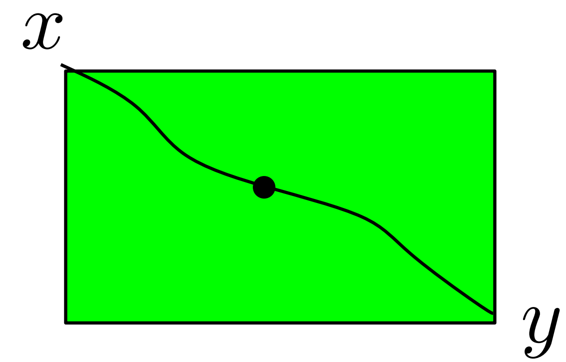
$\Longrightarrow$ Buildpath(0,F) uses $O(Dn)$ time

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

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).

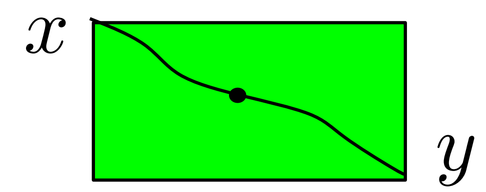
(a) Hirschberg ('75)

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

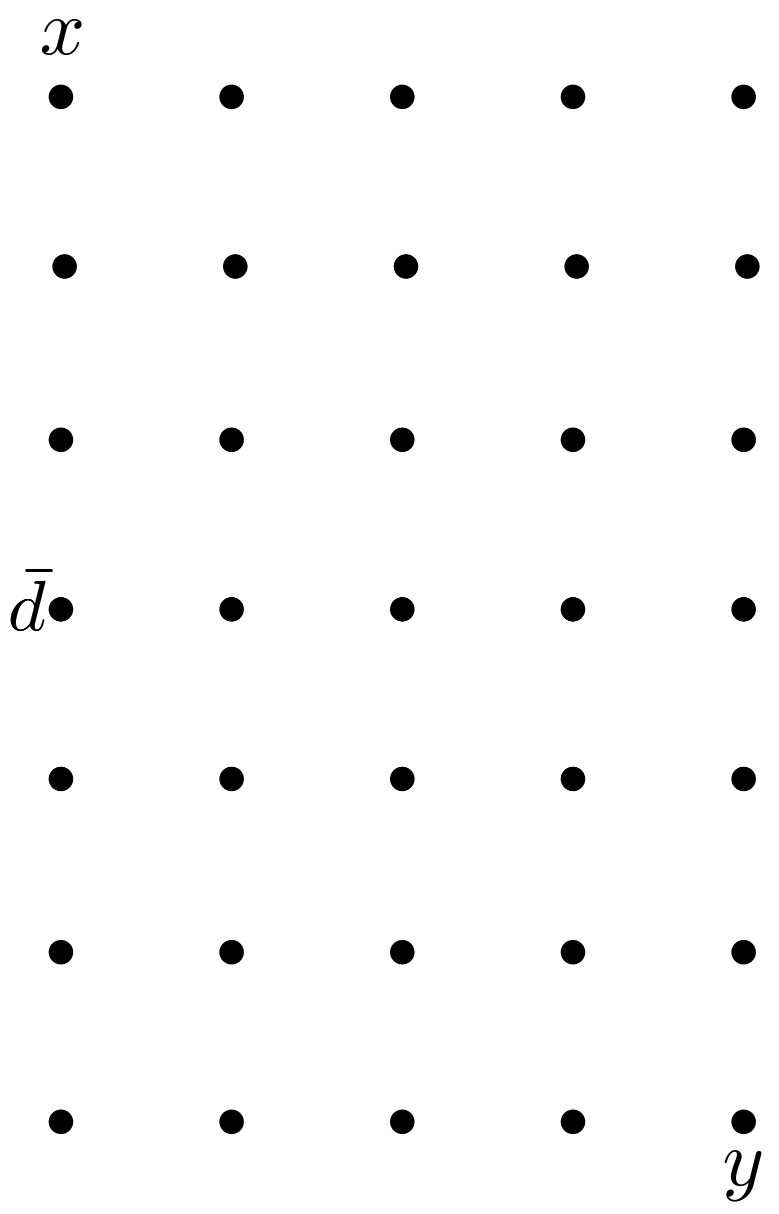
(b) Munro & Ramirez ('82)

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

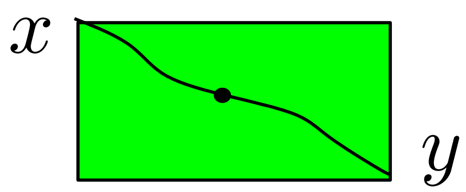
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



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.

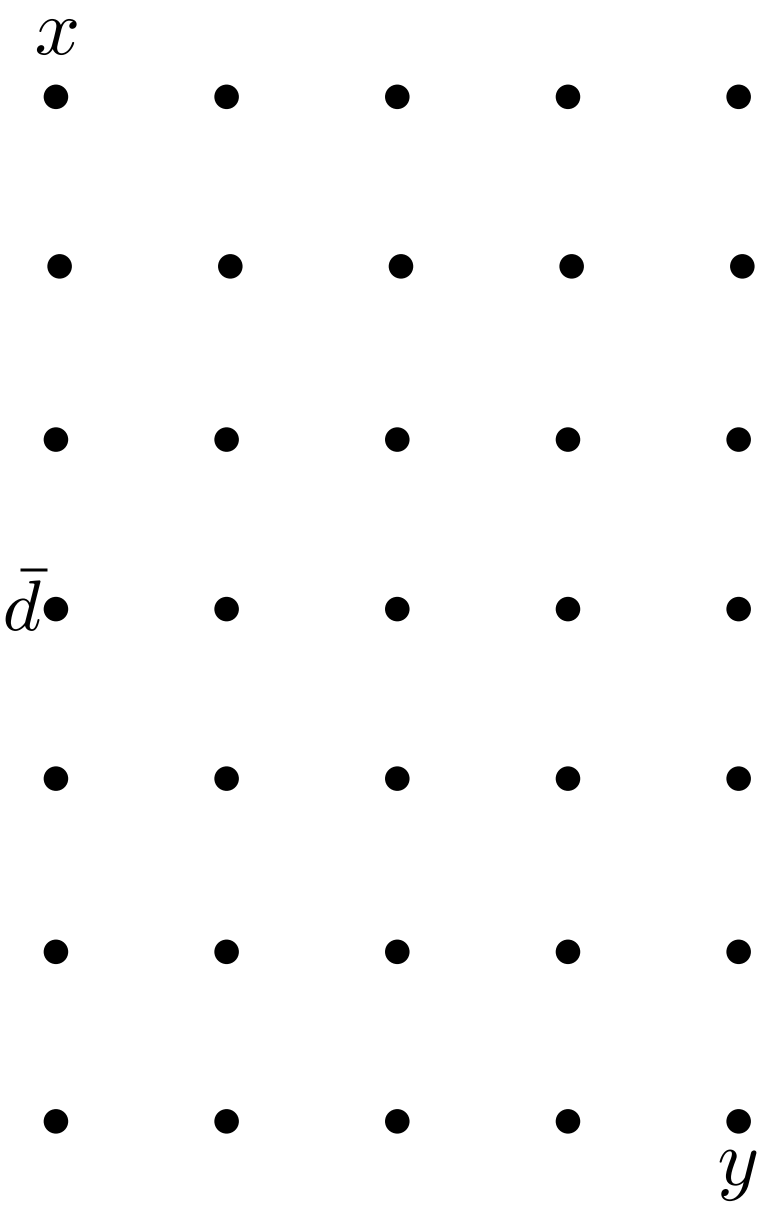


Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time

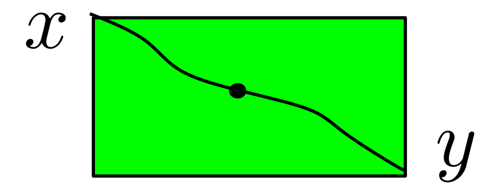


For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

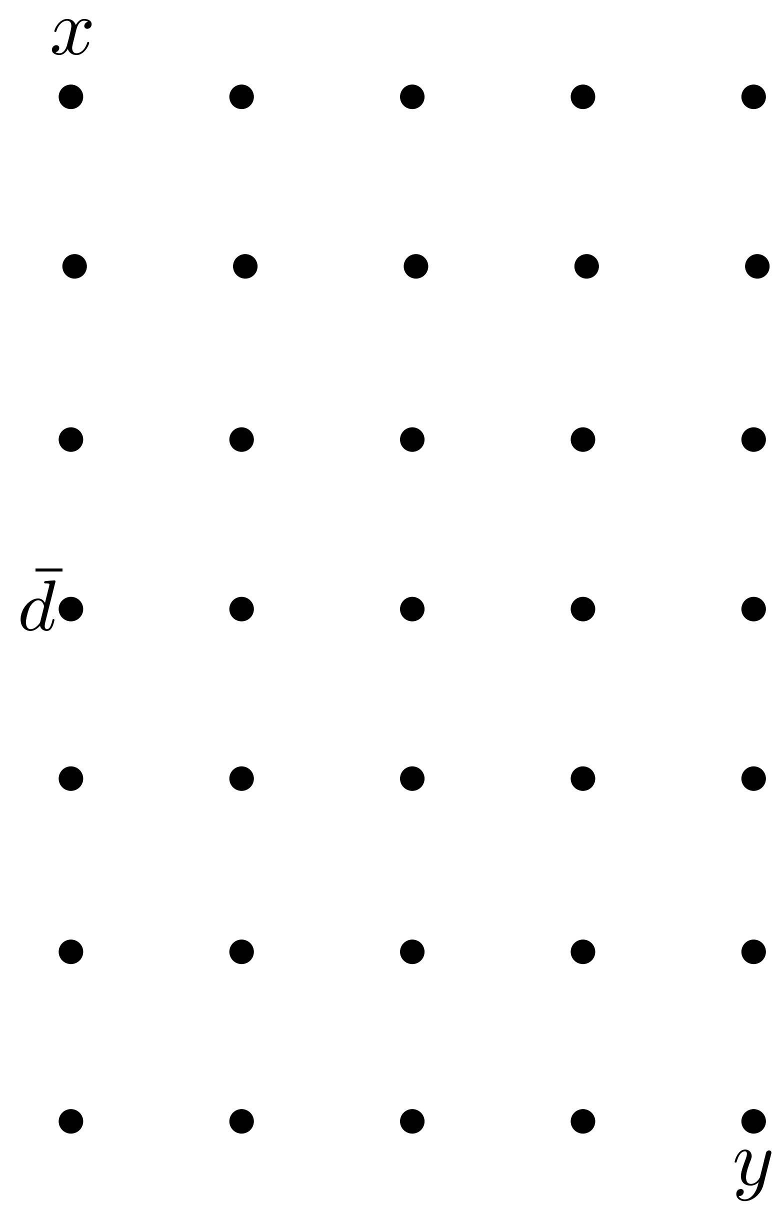
If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.



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



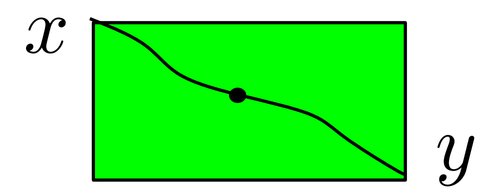
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



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 $d-1$.

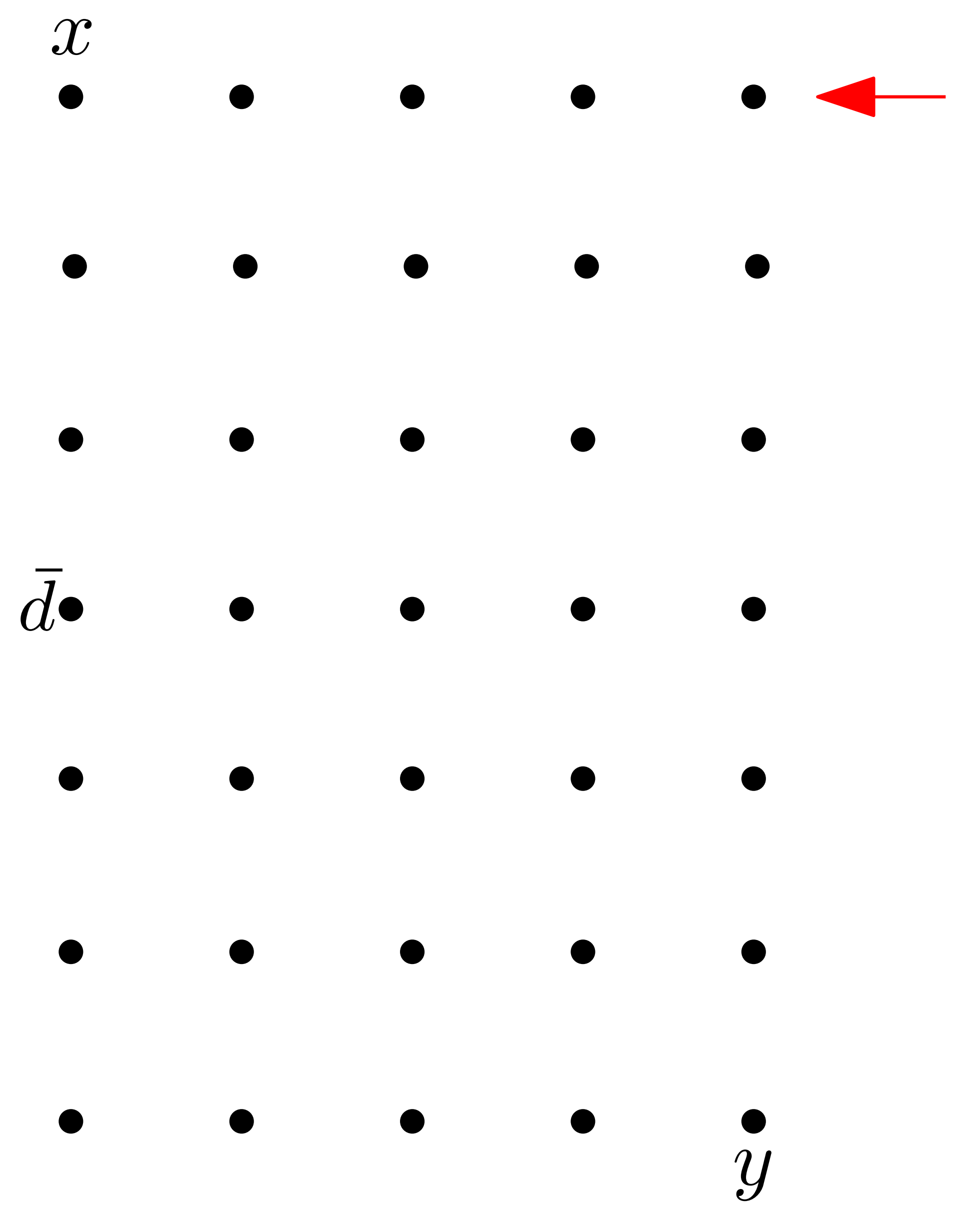
Implementing $Mid(x,y)$ in $O(D+n)$ space and $Area(x,y)$ time



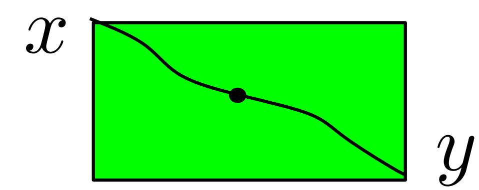
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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 $d-1$.



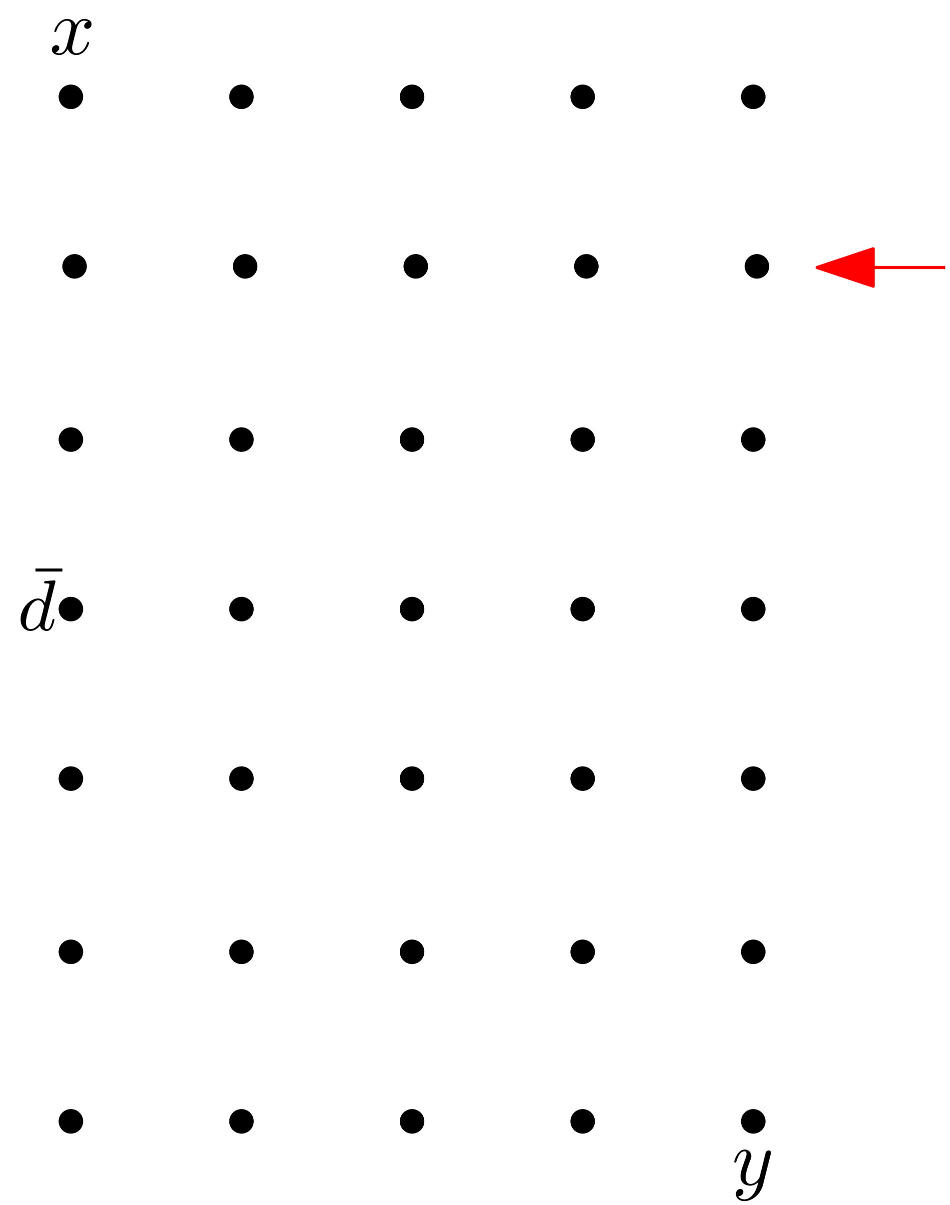
Implementing $Mid(x,y)$ in $O(D+n)$ space and $Area(x,y)$ time



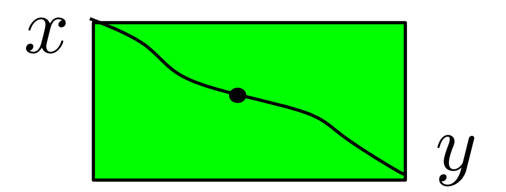
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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.

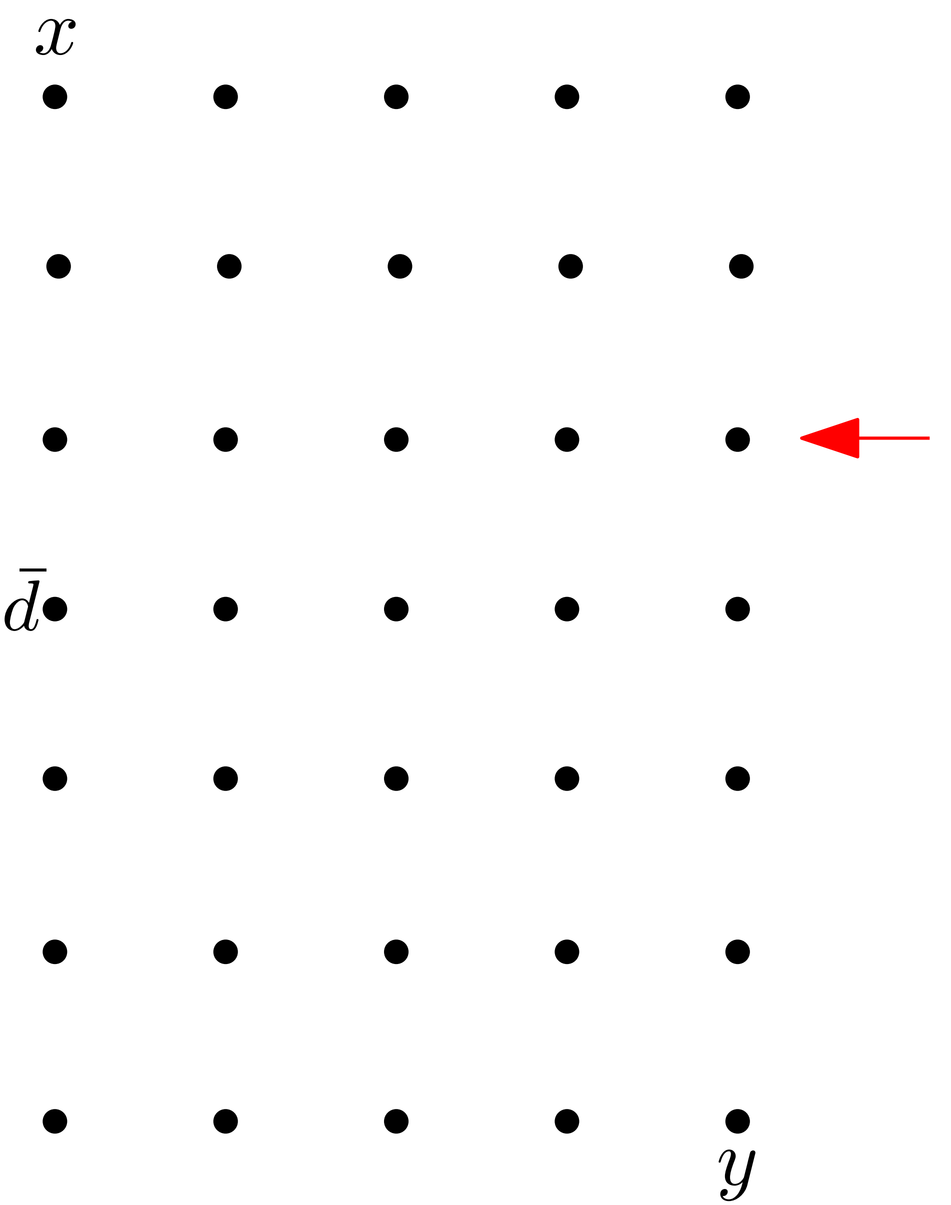
All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d-1$.



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



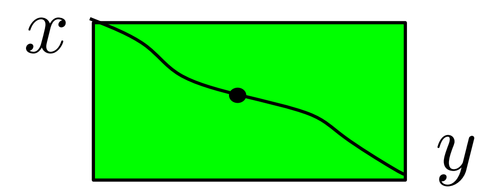
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



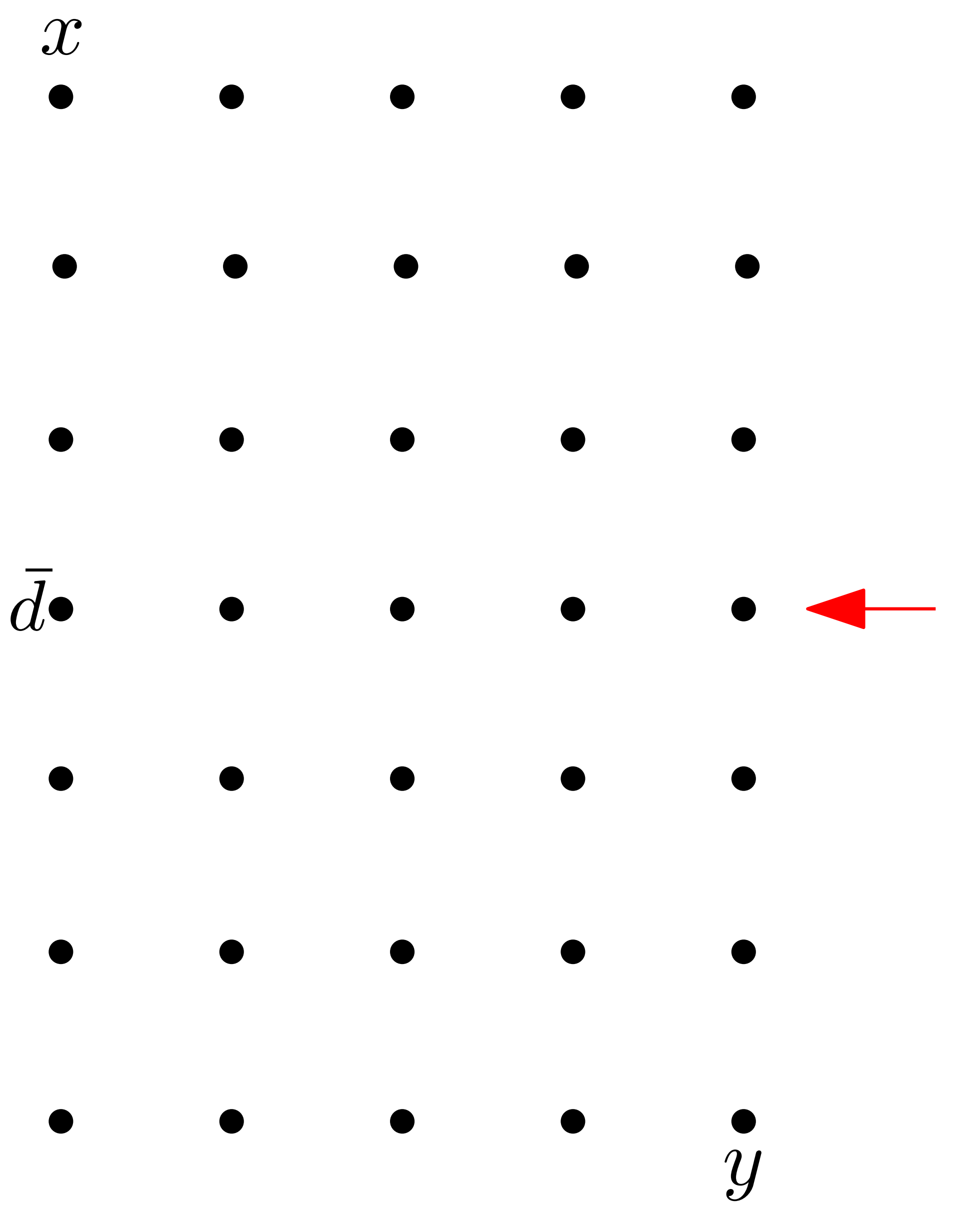
If $z_d = \bar{d}$, $P(z) = z$.
If $z_d > \bar{d}$, then $P(z) = P(pred(z))$ where $pred(z)$ is predecessor of z on min cost path.

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



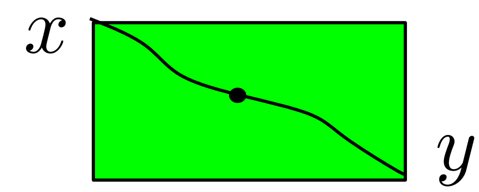
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



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 $d - 1$.

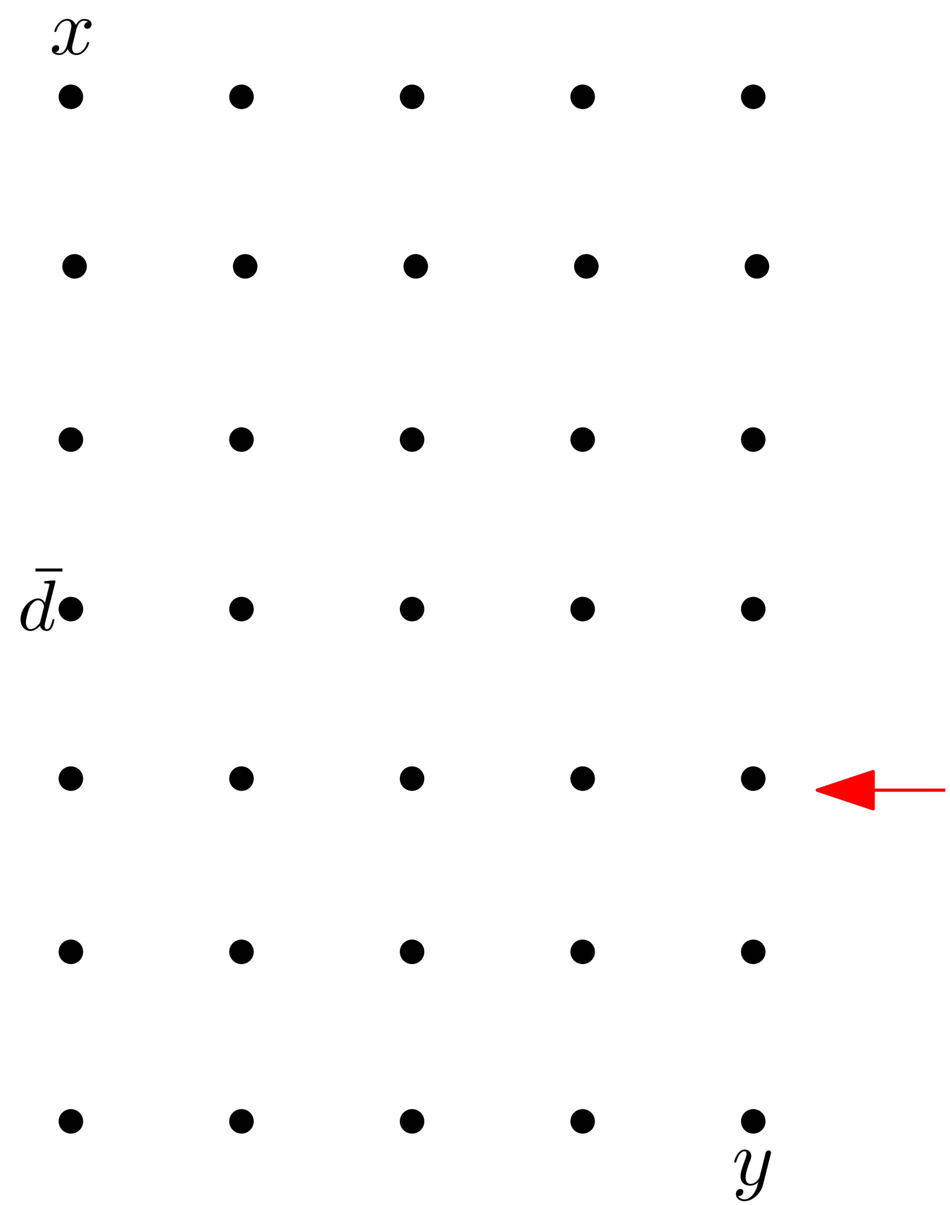
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



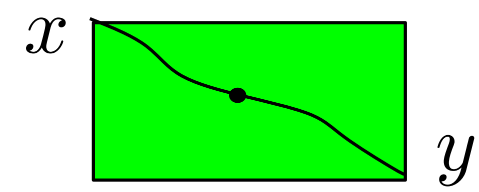
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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 $d - 1$.



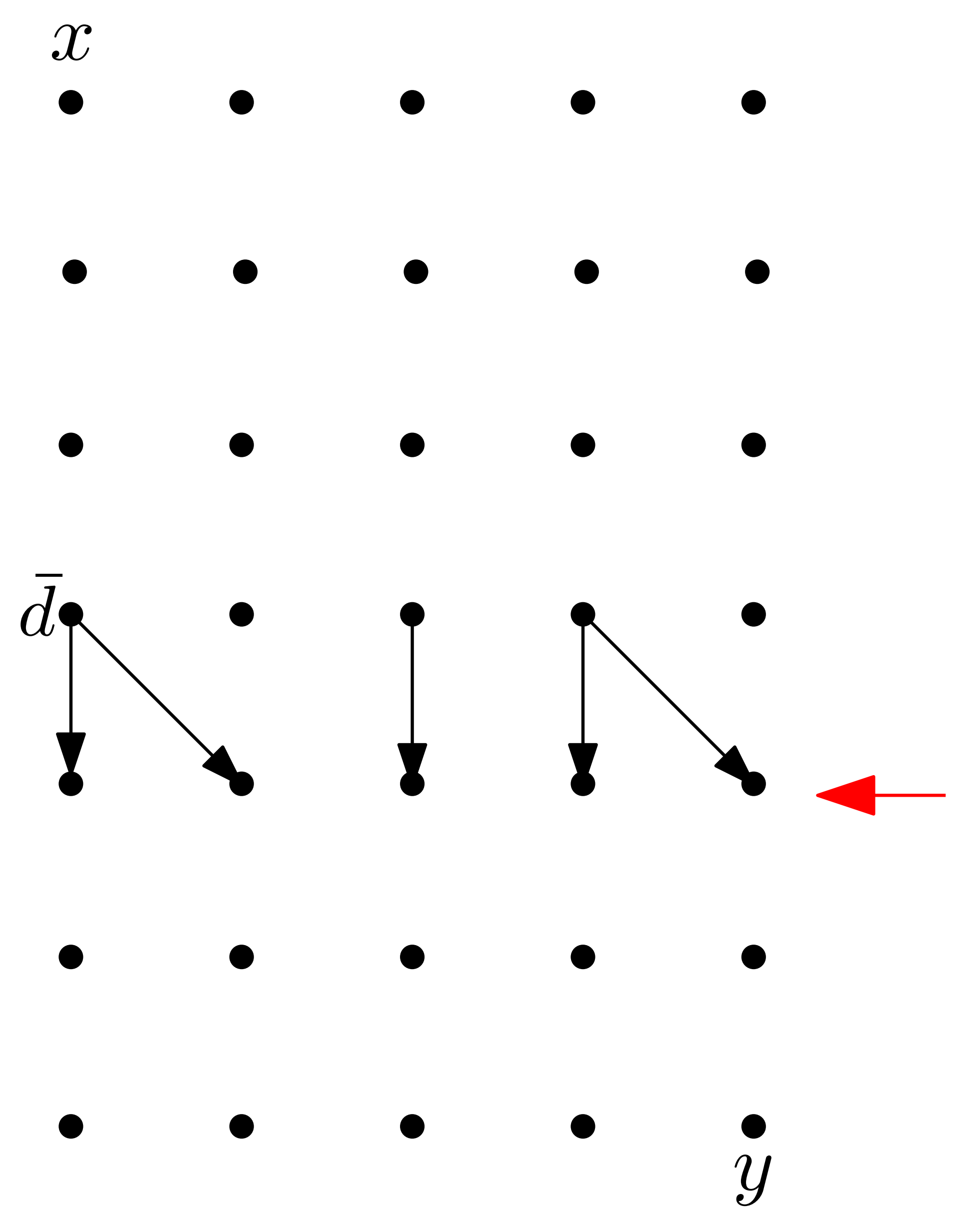
Implementing $Mid(x,y)$ in $O(D+n)$ space and $Area(x,y)$ time



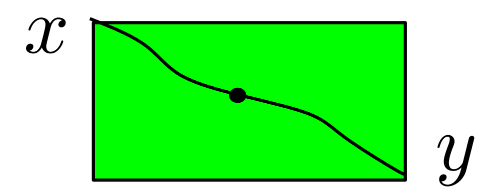
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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 $d-1$.



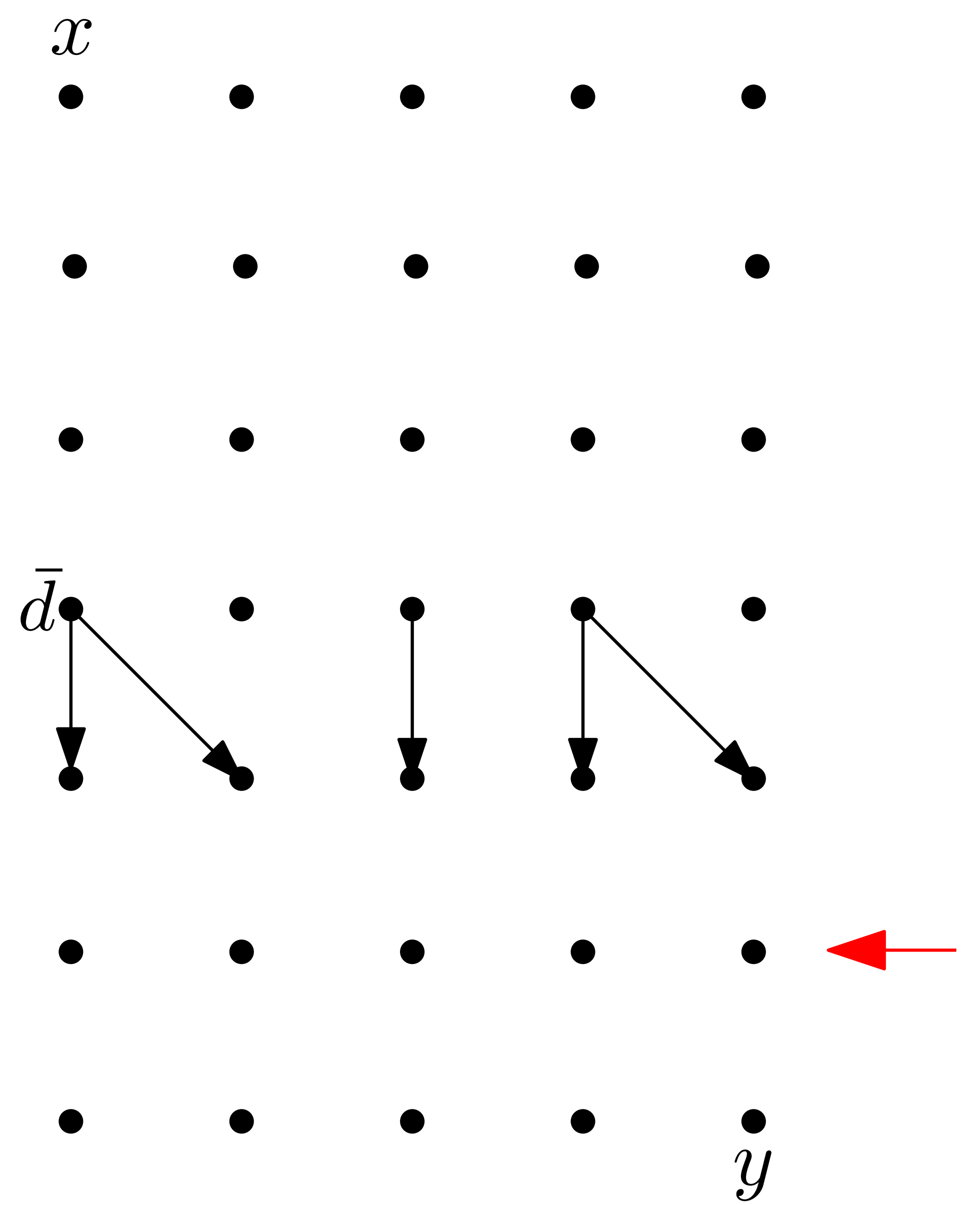
Implementing $Mid(x,y)$ in $O(D+n)$ space and $Area(x,y)$ time



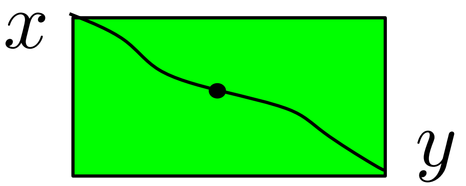
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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 $d-1$.



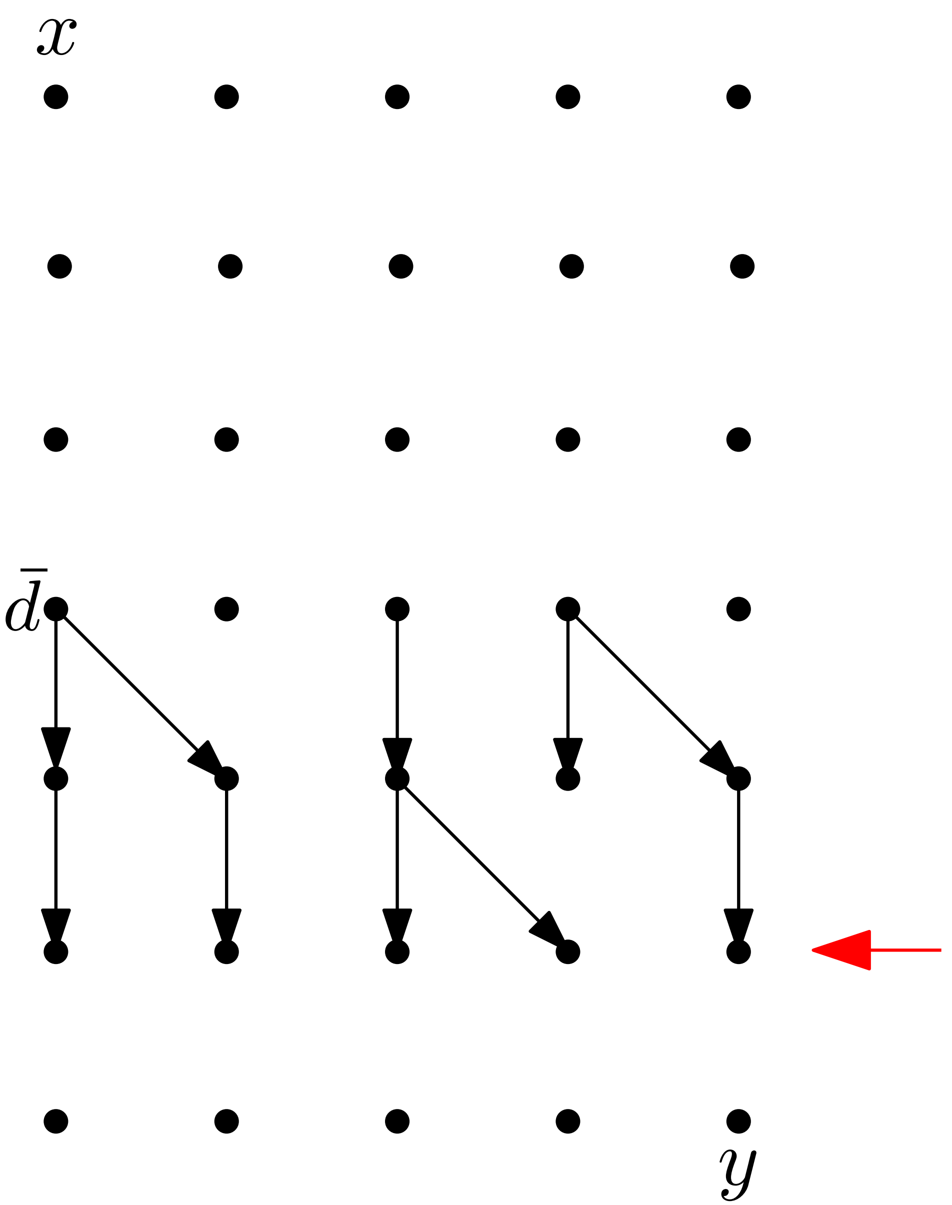
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



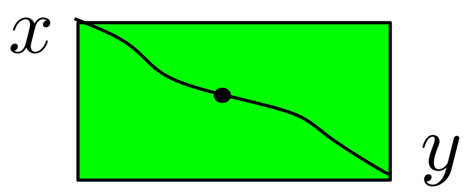
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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.

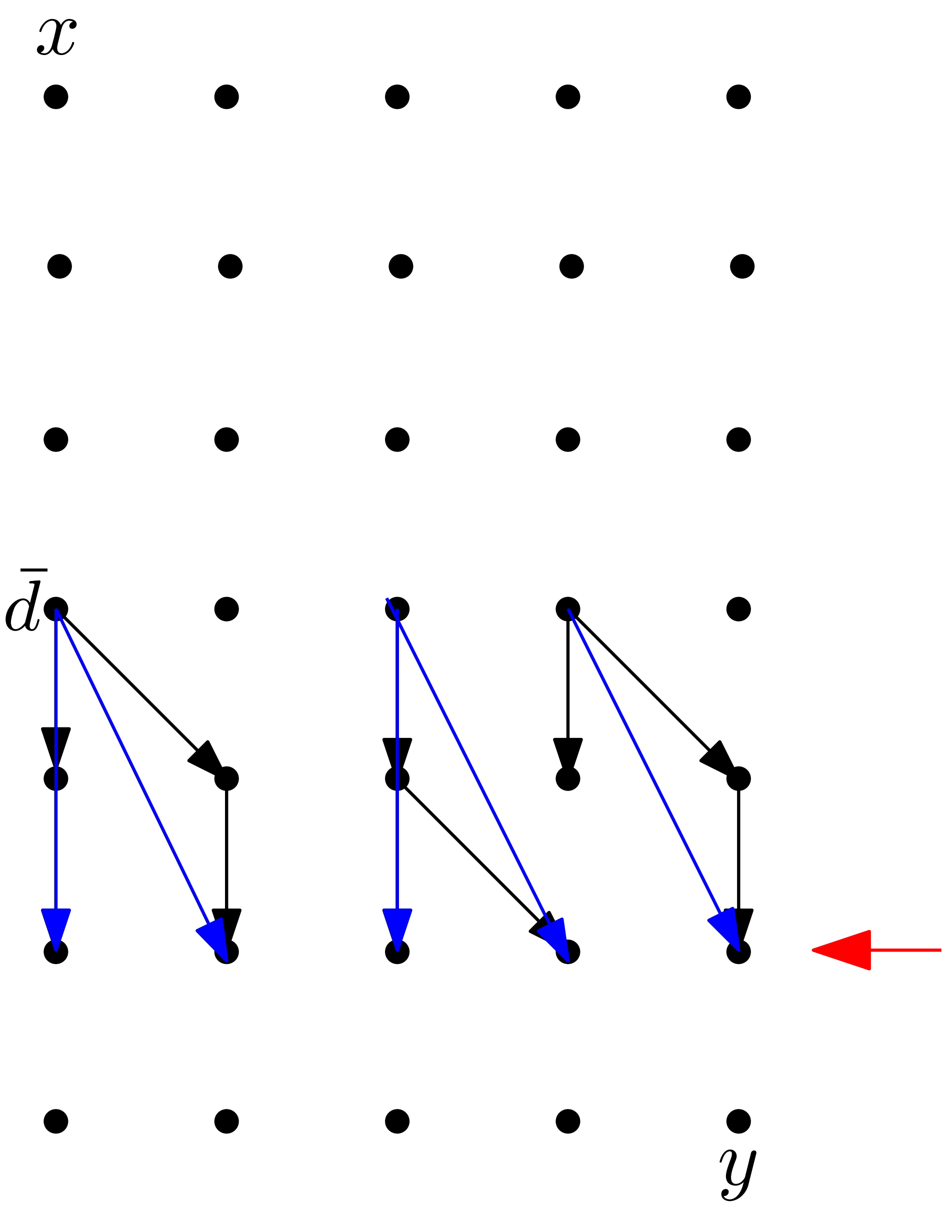
All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.



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



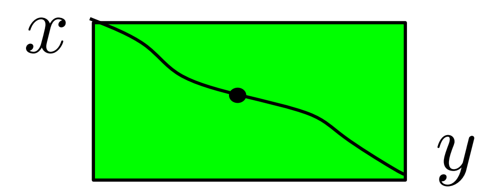
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



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 $d - 1$.

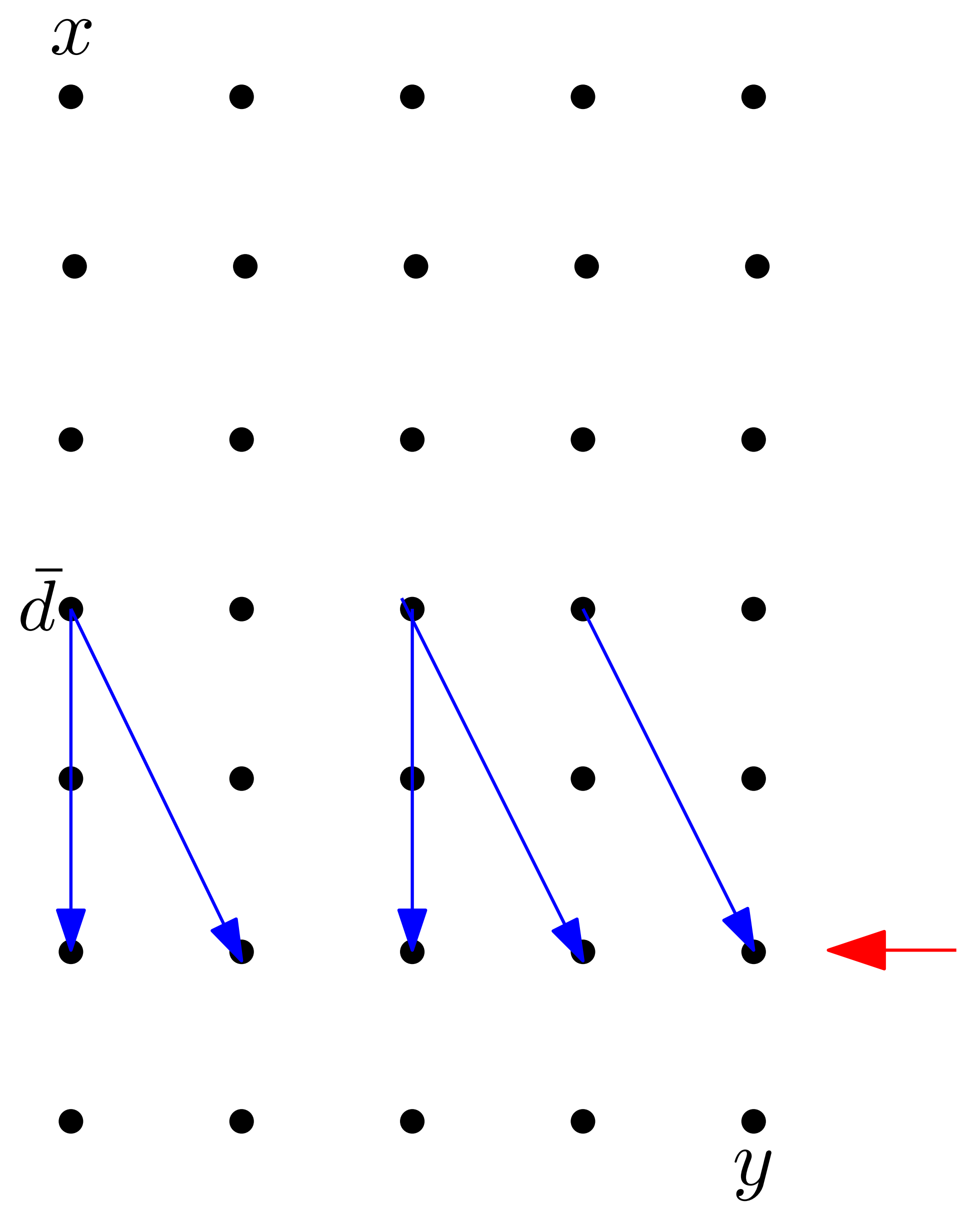
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



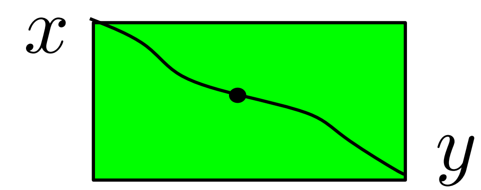
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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 $d - 1$.



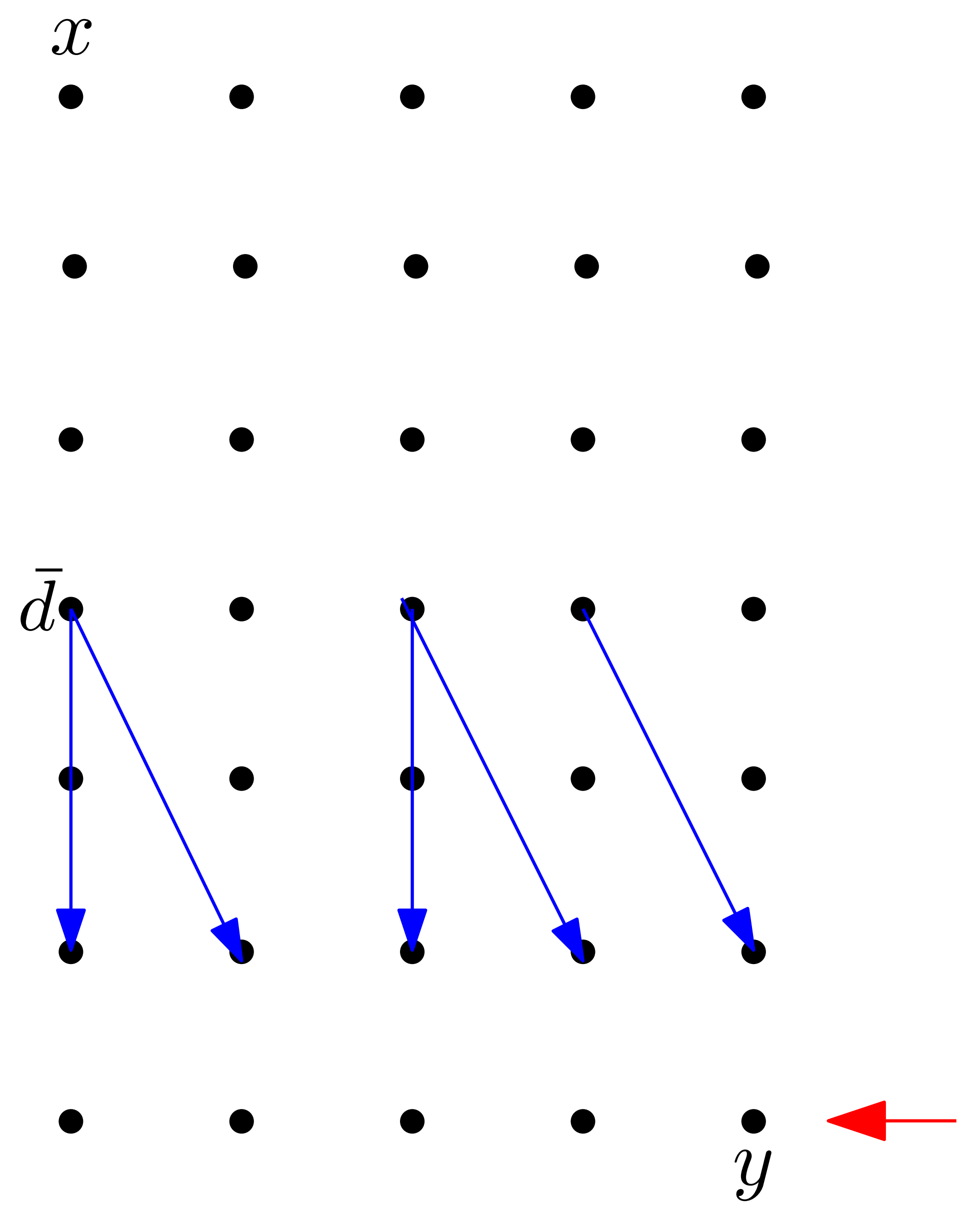
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



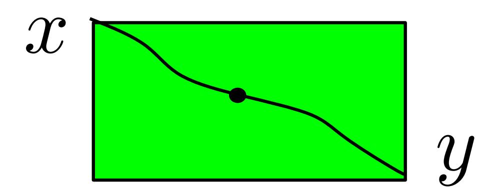
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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 $d - 1$.



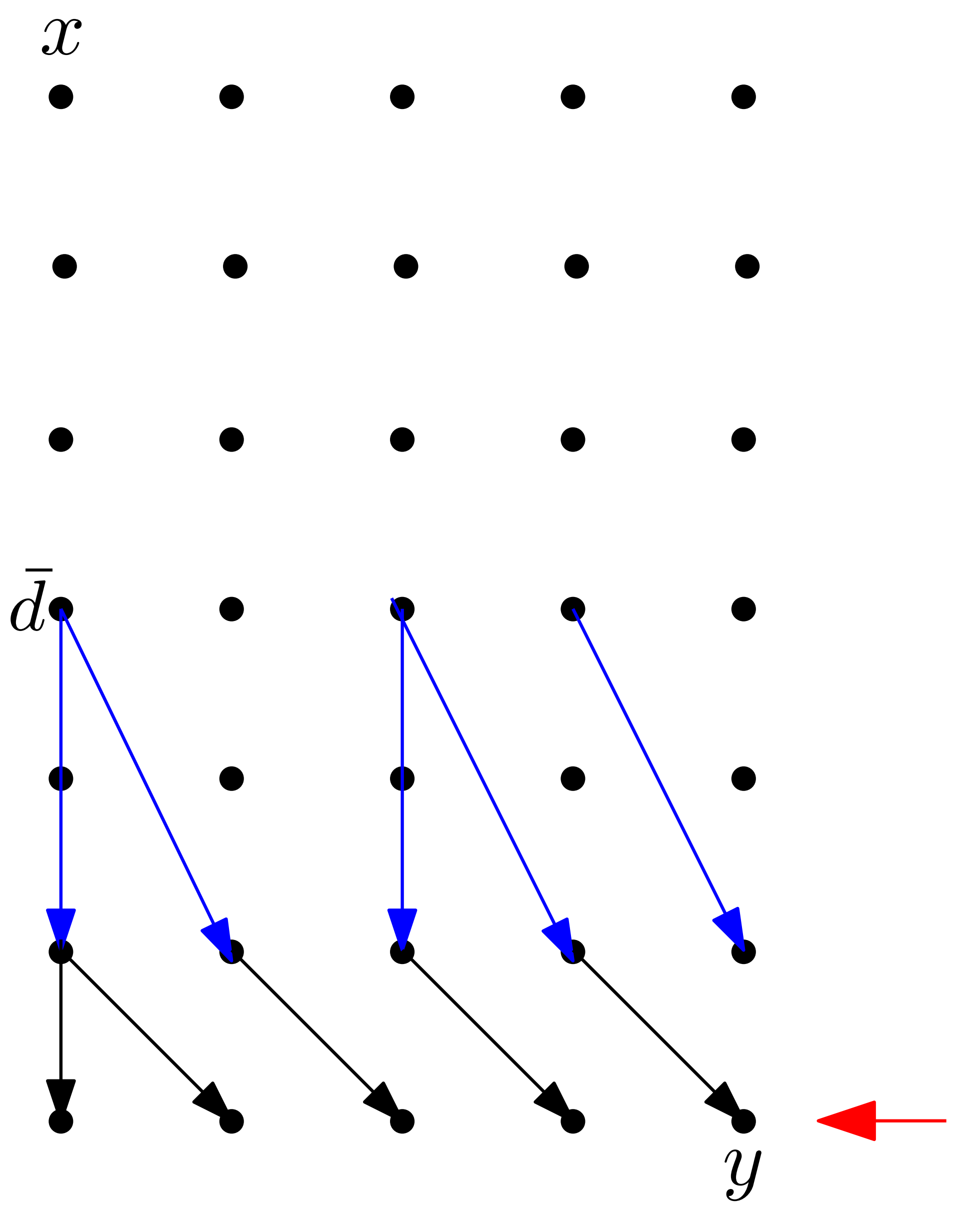
Implementing $Mid(x, y)$ in $O(D+n)$ space and $Area(x, y)$ time



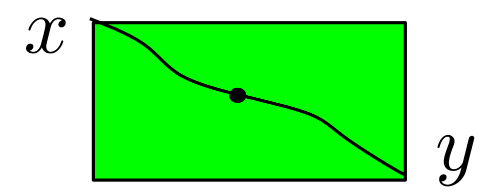
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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 $d-1$.



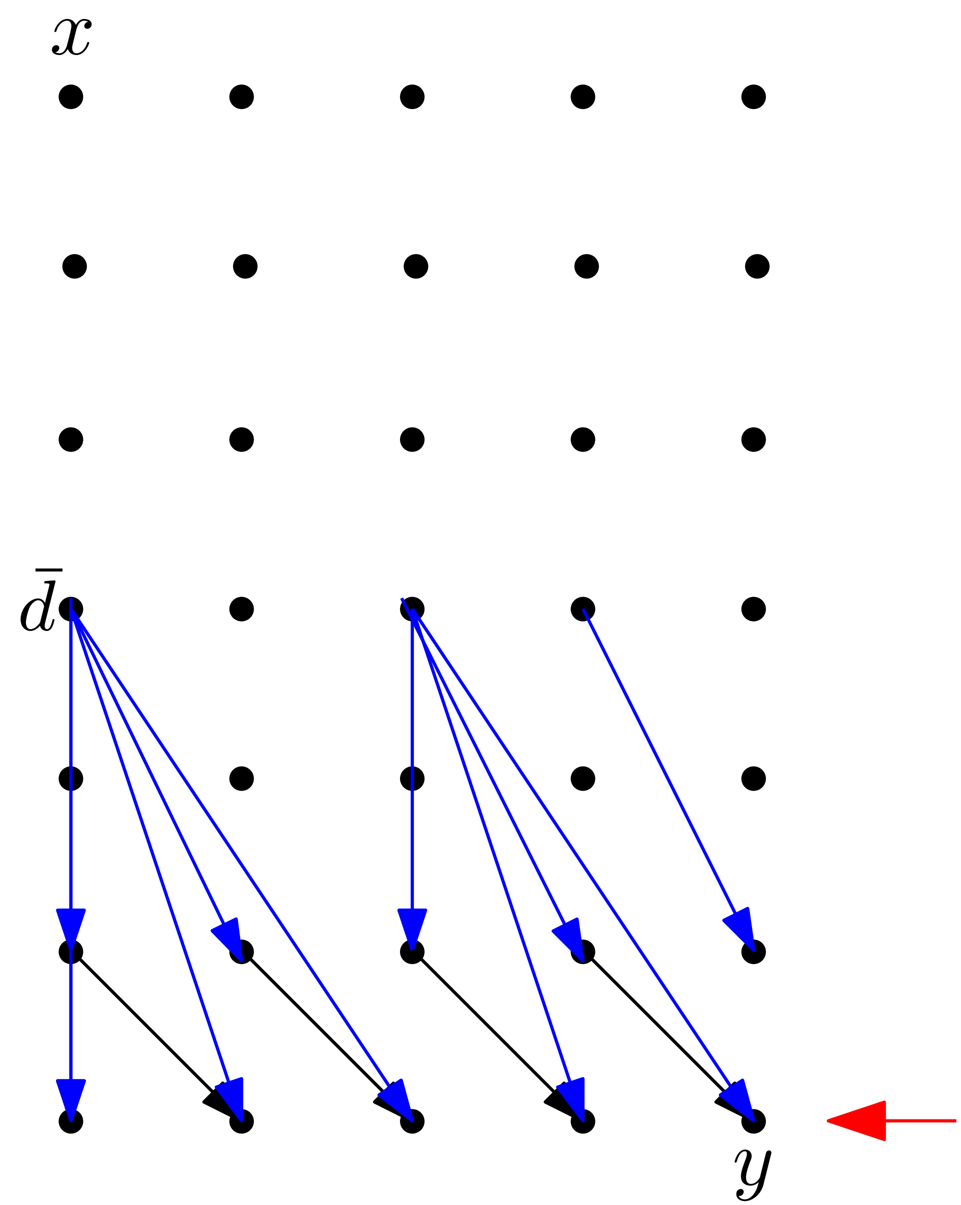
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



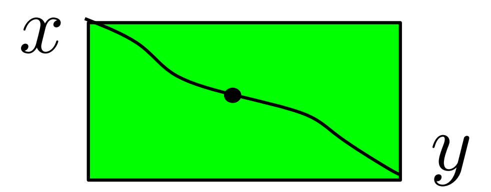
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.

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.

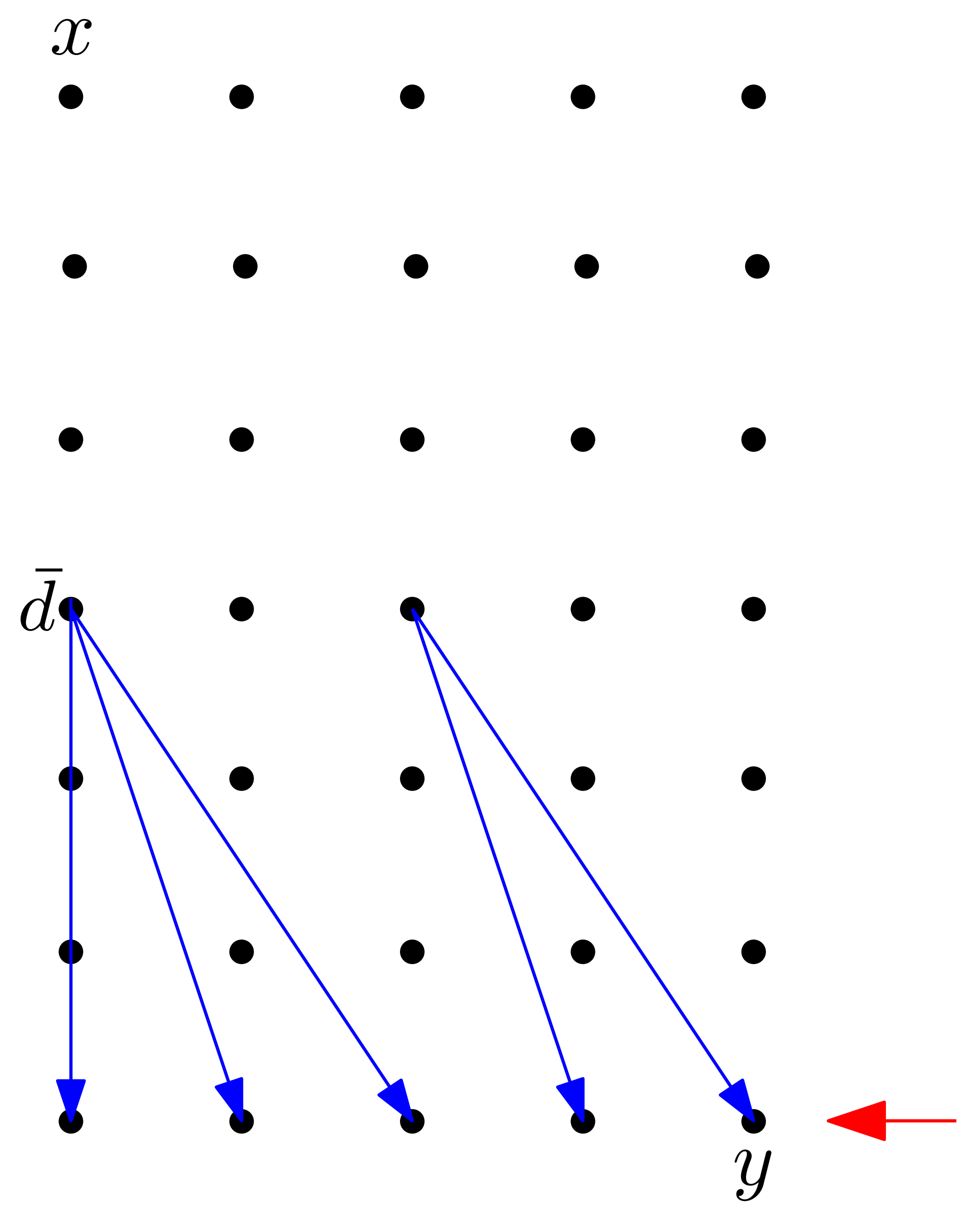
All of the $C(z)$ and $P(z)$ on level d can be calculated in $O(y_d - x_d)$ time (Monge property) using only knowledge of $C(z')$ and $P(z')$ where z' on level $d - 1$.



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



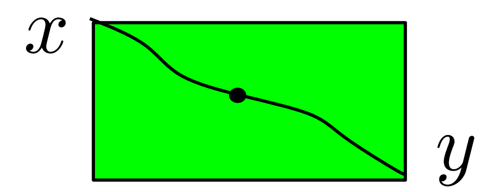
For every z, let $C(z)$ be min cost path distance from x to z.
For $z_d \geq \bar{d}$, let $P(z)$ be a point on level $\bar{d}$ lying on some min-cost path.



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 $d - 1$.

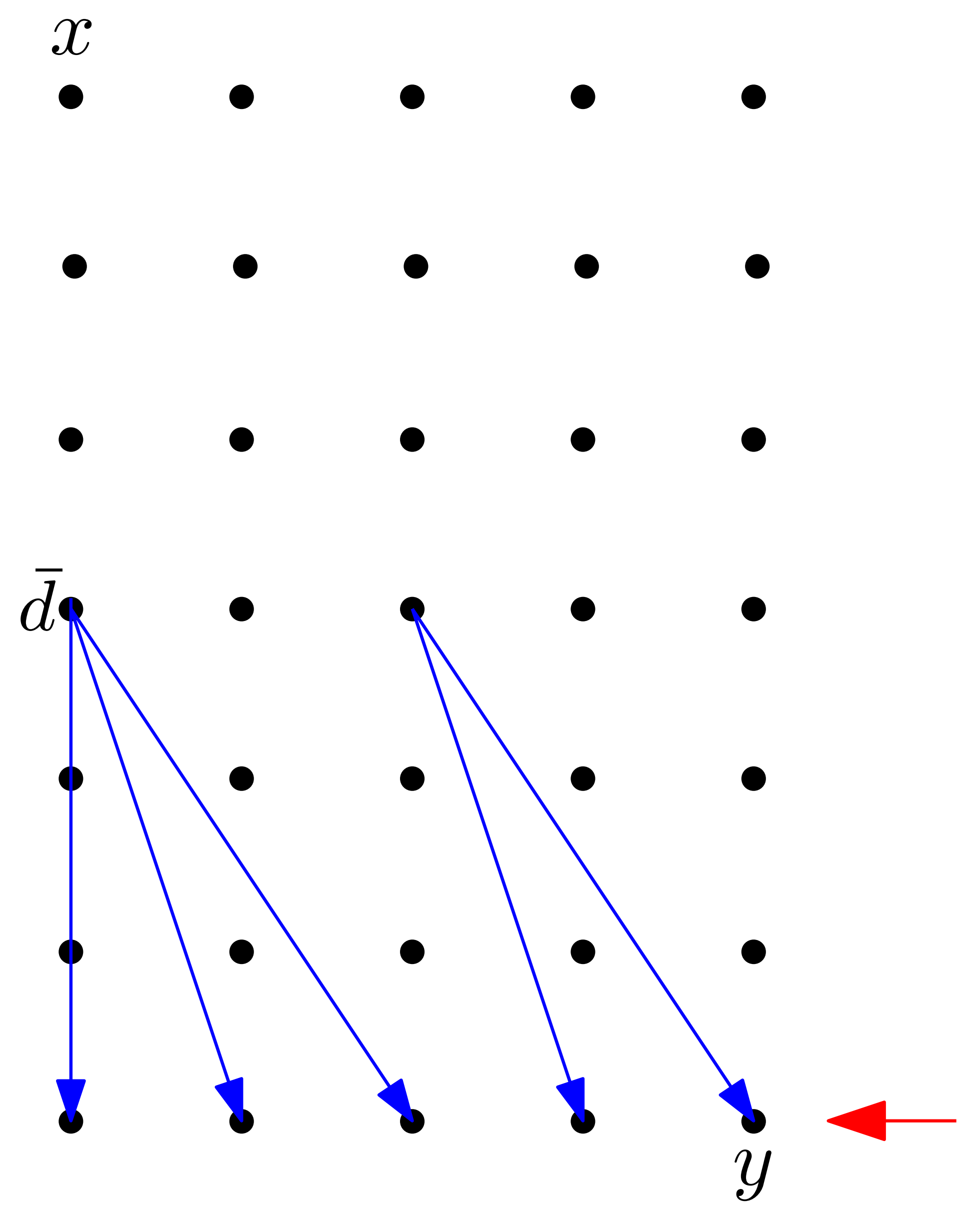
Implementing $Mid(x, y)$ in $O(D + n)$ space and $Area(x, y)$ time



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 $d - 1$.



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

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

$\Rightarrow$ $Buildpath(x, y)$ uses $O(D + n)$ space and $O(Area(x, y))$ time

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

$\Rightarrow Buildpath(x, y)$ uses $O(D + n)$ space and $O(Area(x, y))$ time

$\Rightarrow Buildpath((0, 0), (n, D))$ uses $O(D+n)$ space and $O(Dn)$ time

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

$\Rightarrow$ $Buildpath(x, y)$ uses $O(D + n)$ space and $O(Area(x, y))$ time

$\Rightarrow$ $Buildpath((0, 0), (n, D))$ uses $O(D+n)$ space and $O(Dn)$ time

$\Rightarrow$ can calculate value of $H(n, D)$ defined by

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

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

$\Rightarrow$ $Buildpath(x,y)$ uses $O(D+n)$ space and $O(Area(x,y))$ time

$\Rightarrow$ $Buildpath((0,0),(n,D))$ uses $O(D+n)$ space and $O(Dn)$ time

$\Rightarrow$ can calculate value of $H(n,D)$ defined by

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

using $O(D+n)$ space and $O(Dn)$ time

Outline

- Review of the Monge Speedup
- Saving Space While Saving Time
- Conclusion

Conclusion

We just saw one technique for reducing time in dynamic programming and another for reducing space.

There are *many* such DP improvement techniques.

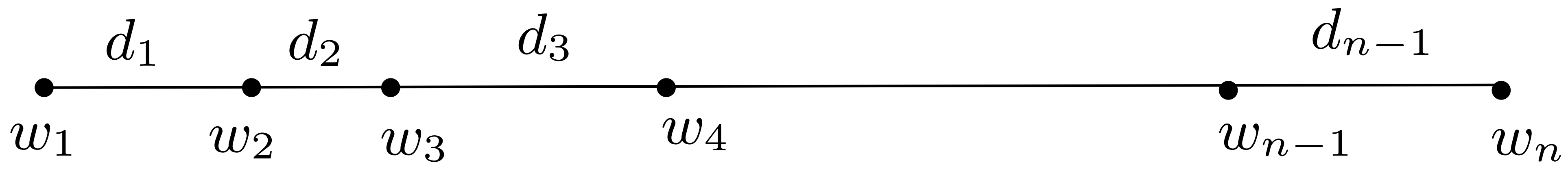
The problem is that they're they are all ad-hoc techniques, primarily known to specialists.

Need to develop a general theory of DP improvements, especially speedups, that is accessible to "users".

Goal is a recipe book that DP designers can check to see how to speed up their application-specific problems.

Open Question

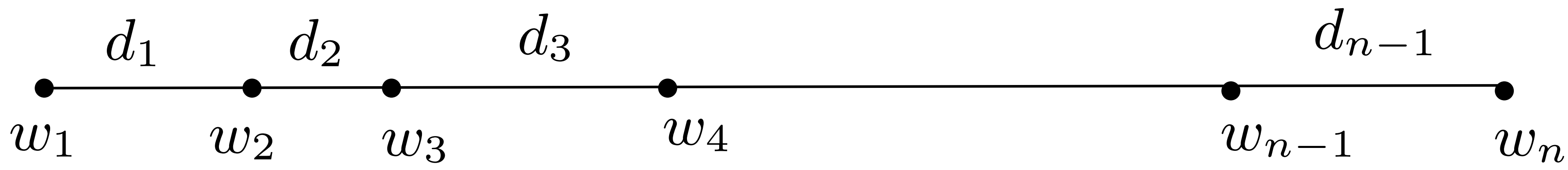
- Two-Sided Online K-Median on a Line



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

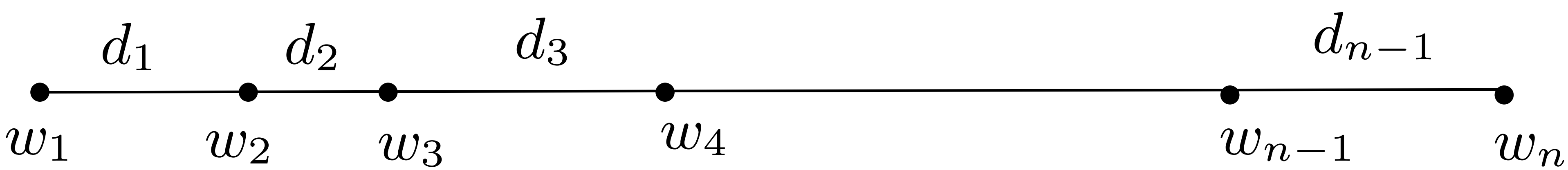


Identify k nodes as service centers. *Cost* of servicing request w_i, is w_i times distance from node i to nearest service center. Problem is to find location of k service centers that minimize total service cost.

- 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?