

CS 466

Introduction to Bioinformatics

Lecture 2

Mohammed El-Kebir

August 27, 2021



Course Announcements

Instructor:

- Mohammed El-Kebir (melkebir)
- Office hours: Wednesdays, 3:30-4:30pm in Siebel 3216 / Zoom

Zoom:

- <https://go.cs.illinois.edu/CS466>

Piazza / Gradescope:

- <https://piazza.com/illinois/fall2021/cs466>
- Access code: 'X3KPBE'

TA:

- Leah Weber (leahlw2), office hours: Mondays, 4-5pm via Zoom or in person 3rd floor Siebel by elevator
- Yuanyuan Qi (yq7), office hours: Fridays, 9-10am via Zoom or in person 3rd floor Siebel by elevator

Outline

1. Change problem
2. Review of running time analysis
3. Edit distance
4. Review elementary graph theory
5. Manhattan Tourist problem
6. Longest/shortest paths in DAGs

Reading:

- Jones and Pevzner. Chapters 2.7-2.9 and 6.1-6.4
- Lecture notes

The Change Problem

Change Problem: Given amount $M \in \mathbb{N} \setminus \{0\}$ and coins $\mathbf{c} = (c_1, \dots, c_n) \in \mathbb{N}^n$
s.t. $c_n = 1$ and $c_i \geq c_{i+1}$ for all $i \in [n - 1] = \{1, \dots, n - 1\}$,
find $\mathbf{d} = (d_1, \dots, d_n) \in \mathbb{N}^n$ s.t. (i) $M = \sum_{i=1}^n c_i d_i$ and (ii) $\sum_{i=1}^n d_i$ is minimum

- Suppose we have $n = 3$ coins:

$$\mathbf{c} = (\text{7 cent} , \text{3 cent} , \text{1 cent})$$

- What is the minimum number of coins needed to make change for $M = 9$ cents?
- Answer: $(d_1, \dots, d_n) = (1, 0, 2)$ thus $1 + 0 + 2 = 3$ coins.

The Change Problem – Four Algorithms

GreedyChange(M, c_1, \dots, c_n)

1. **for** $i \leftarrow 1$ **to** n
2. $d_i \leftarrow \lfloor M/c_i \rfloor$
3. $M \leftarrow M - d_i c_i$

ExhaustiveChange(M, c_1, \dots, c_n)

1. **for** $(d_1, \dots, d_n) \in \{0, \dots, \lfloor M/c_1 \rfloor\} \times \dots \times \{0, \dots, \lfloor M/c_n \rfloor\}$
2. **if** $\sum_{i=1}^n c_i d_i = M$
3. **return** (d_1, \dots, d_n)

RecursiveChange(M, c_1, \dots, c_n)

1. **if** $M = 0$
2. **return** 0
3. $\text{bestNumCoins} \leftarrow \infty$
4. **for** $i \leftarrow 1$ **to** n
5. **if** $M \geq c_i$
6. $\text{numCoins} \leftarrow \text{RecursiveChange}(M - c_i, c_1, \dots, c_n)$
7. **if** $\text{numCoins} + 1 < \text{bestNumCoins}$
8. $\text{bestNumCoins} \leftarrow \text{numCoins} + 1$
9. **return** bestNumCoins

DPChange(M, c_1, \dots, c_n)

1. **for** $m \leftarrow 1$ **to** M
2. $\text{minNumCoins}[m] \leftarrow \infty$
3. **for** $i \leftarrow 1$ **to** n
4. $\text{minNumCoins}[c_i] \leftarrow 1$
5. **for** $m \leftarrow 1$ **to** M
6. **for** $i \leftarrow 1$ **to** n
7. **if** $m > c_i$
8. $\text{minNumCoins}[m] \leftarrow \min(1 + \text{minNumCoins}[m - c_i], \text{minNumCoins}[m])$
9. **return** $\text{minNumCoins}[M]$

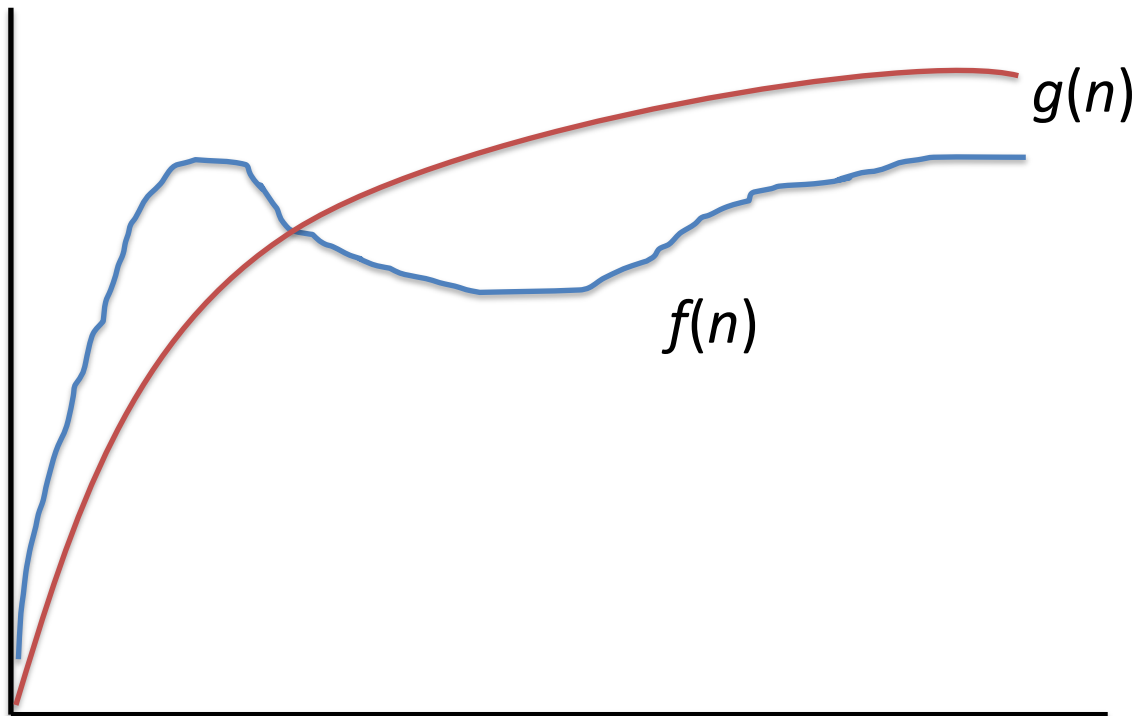
Four Different Algorithms

Technique	Correct?	Efficient?
Greedy algorithm [GreedyChange]	no	yes
Exhaustive enumeration [ExhaustiveChange]	yes	no
Recursive algorithm [RecursiveChange]	yes	no
Dynamic programming [DPChange]	yes	yes

Question: How to assess efficiency?

Running Time Analysis

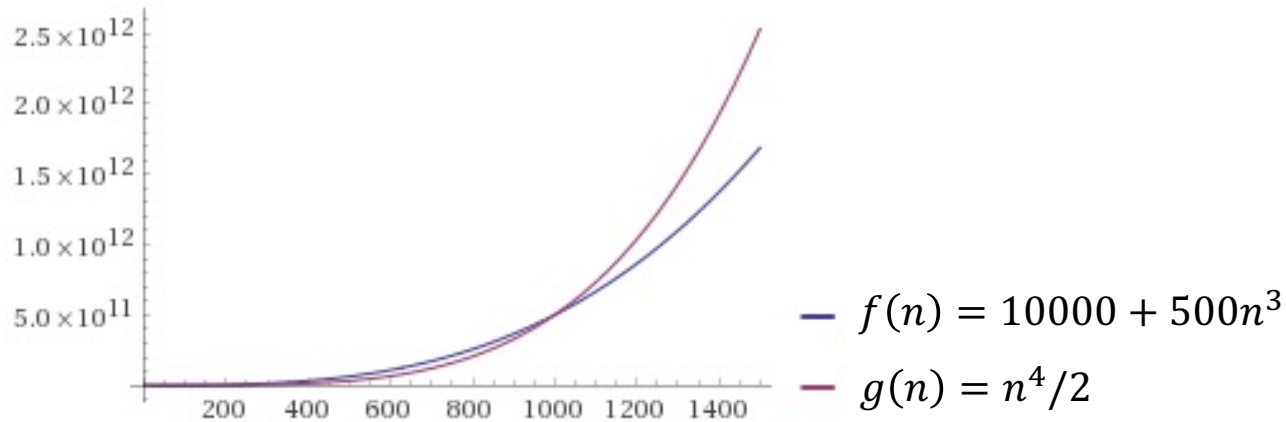
- The **running time** of an algorithm A for problem Π is the maximum number of steps that A will take on any instance of size $n = |X|$
- **Asymptotic running time** ignores constant factors using Big O notation



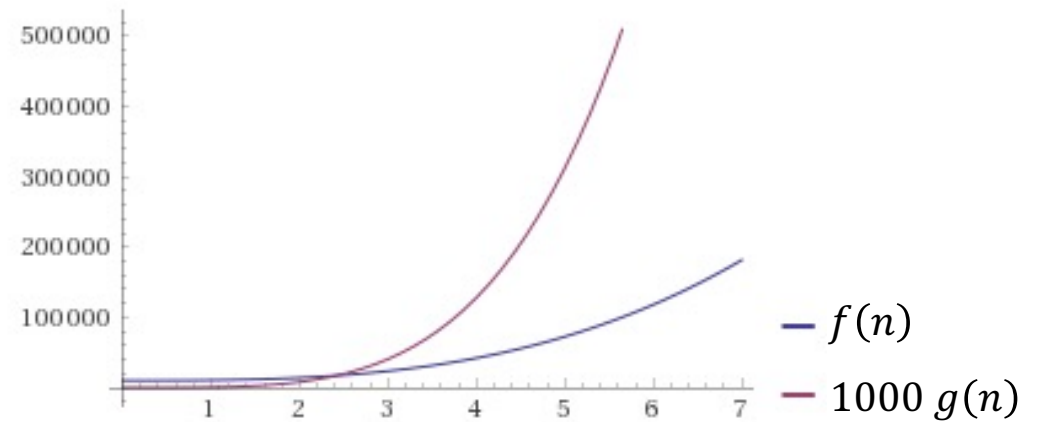
$f(n)$ is $O(g(n))$ provided there exists $c > 0$ and $n_0 \geq 0$ such that $f(n) \leq c g(n)$ for all $n \geq n_0$

Running Time Analysis – Example

$f(n)$ is $O(g(n))$ provided there exists $c > 0$ and $n_0 \geq 0$ such that
 $f(n) \leq c g(n)$ for all $n \geq n_0$



Computed by Wolfram|Alpha



Computed by Wolfram|Alpha

Pick $c = 1000$ and $n_0 = 3$. Then, $f(n) \leq c g(n)$ for all $n \geq n_0$.

The Change Problem – Running Time Analysis

GreedyChange(M, c_1, \dots, c_n)

1. **for** $i \leftarrow 1$ **to** n
2. $d_i \leftarrow \lfloor M/c_i \rfloor$
3. $M \leftarrow M - d_i c_i$

DPChange(M, c_1, \dots, c_n)

1. **for** $m \leftarrow 1$ **to** M
2. minNumCoins[m] $\leftarrow \infty$
3. **for** $i \leftarrow 1$ **to** n
4. minNumCoins[c_i] $\leftarrow 1$
5. **for** $m \leftarrow 1$ **to** M
6. **for** $i \leftarrow 1$ **to** n
7. **if** $m > c_i$
8. minNumCoins[m] $\leftarrow \min(1 + \text{minNumCoins}[m - c_i], \text{minNumCoins}[m])$
9. **return** minNumCoins[M]

Number of operations:

- Line 2: $3 = O(1)$
- Line 3: $3 = O(1)$
- Total: $6n = O(n)$

Number of operations:

- Lines 1-2: $O(M)$
- Lines 3-4: $O(n)$
- Lines 5-8: $O(Mn)$
- Total: $O(M) + O(n) + O(Mn) = O(Mn)$

Running Time Analysis – Guidelines

- $O(n^a) \subset O(n^b)$ for any positive constants $a < b$
- For any constants $a, b > 0$ and $c > 1$,
 $O(a) \subset O(\log n) \subset O(n^b) \subset O(c^n)$
- We can multiply to learn about other functions. For any constants $a, b > 0$ and $c > 1$,
 $O(an) = O(n) \subset O(n \log n) \subset O(n n^b) = O(n^{b+1}) \subset O(nc^n)$
- Base of the logarithm is a constant and can be ignored. For any constants $a, b > 1$,
 $O(\log_a n) = O(\log_b n / \log_b a) = O(1/(\log_b a) \log_b n) = O(\log_b n)$

Running Time Analysis – Guidelines

- $O(n^a) \subset O(n^b)$ for any positive constants $a < b$

- For any constants $a, b > 0$ and $c > 1$,

$$O(a) \subset O(\log n) \subset O(n^b) \subset O(c^n)$$

- We can multiply to learn about other functions. For any constants $a, b > 0$ and $c > 1$,

$$O(an) = O(n) \subset O(n \log n) \subset O(n n^b) = O(n^{b+1}) \subset O(nc^n)$$

- Base of the logarithm is a constant and can be ignored. For any constants $a, b > 1$,

$$O(\log_a n) = O(\log_b n / \log_b a) = O(1/(\log_b a) \log_b n) = O(\log_b n)$$

Big Oh	Name
$O(1)$	Constant
$O(\log n)$	Logarithmic
$O(n)$	Linear
$O(n^2)$	Quadratic
$O(n^c) = O(\text{poly}(n))$	Polynomial
$O(2^{\text{poly}(n)})$	Exponential

Running Time Analysis – More Examples

Question: What is $O\left(\binom{n}{k}\right)$?

Running Time Analysis – More Examples

Question: What is $O\left(\binom{n}{k}\right)$?

- For constant $k > 0$ it holds that $\binom{n}{k} = O(n^k)$

- Recall that $n! = \prod_{i=1}^n i$

Question: What is $O(n!)$?

Running Time Analysis – More Examples

Question: What is $O\left(\binom{n}{k}\right)$?

- For constant $k > 0$ it holds that $\binom{n}{k} = O(n^k)$

- Recall that $n! = \prod_{i=1}^n i$

Question: What is $O(n!)$?

Stirling's approximation: $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n = \sqrt{2\pi} \frac{\sqrt{n}}{\exp(n)} n^n = O(n^n) = O(2^{n \log n})$
(*): $\sqrt{n} / \exp(n) < 1$ for all $n > 0$ (*)

Question: Is $n^n = O(n!)$?

Running Time Analysis – More Examples

Question: What is $O\left(\binom{n}{k}\right)$?

- For constant $k > 0$ it holds that $\binom{n}{k} = O(n^k)$

- Recall that $n! = \prod_{i=1}^n i$

Question: What is $O(n!)$?

Stirling's approximation: $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n = \sqrt{2\pi} \frac{\sqrt{n}}{\exp(n)} n^n = O(n^n) = O(2^{n \log n})$
(*): $\sqrt{n} / \exp(n) < 1$ for all $n > 0$ (*)

Question: Is $n^n = O(n!)$?

Question: What is $O(\log(n!))$?

Alignment

An **alignment** between two strings \mathbf{v} (of m characters) and \mathbf{w} (of n characters) is a two row matrix where the first row contains the characters of \mathbf{v} in order, the second row contains the characters of \mathbf{w} in order, and spaces may be interspersed throughout each.

Input

\mathbf{v} : KITTEN ($m = 6$)
 \mathbf{w} : SITTING ($n = 7$)

Output

\mathbf{v} :	K	-	I	T	T	E	N	-
\mathbf{w} :	S	I	-	T	T	I	N	G

Question: Is this a good alignment?

Answer: Count the number of insertion, deletions, substitutions.

Alignment

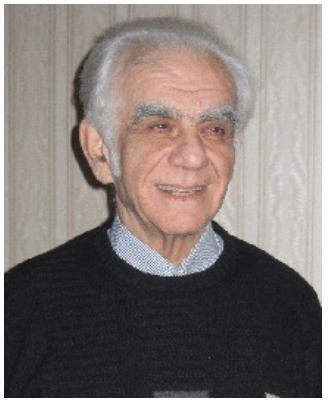
An **alignment** between two strings \mathbf{v} (of m characters) and \mathbf{w} (of n characters) is a two row matrix where the first row contains the characters of \mathbf{v} in order, the second row contains the characters of \mathbf{w} in order, and spaces may be interspersed throughout each.

Input				Output						
\mathbf{v} : KITTEN	$(m = 6)$	\mathbf{v} :	K	-	I	T	T	E	N	-
\mathbf{w} : SITTING	$(n = 7)$	\mathbf{w} :	S	I	-	T	T	I	N	G

Question: Is this a good alignment?

Answer: Count the number of insertion, deletions, substitutions.

Edit Distance [Levenshtein, 1966]



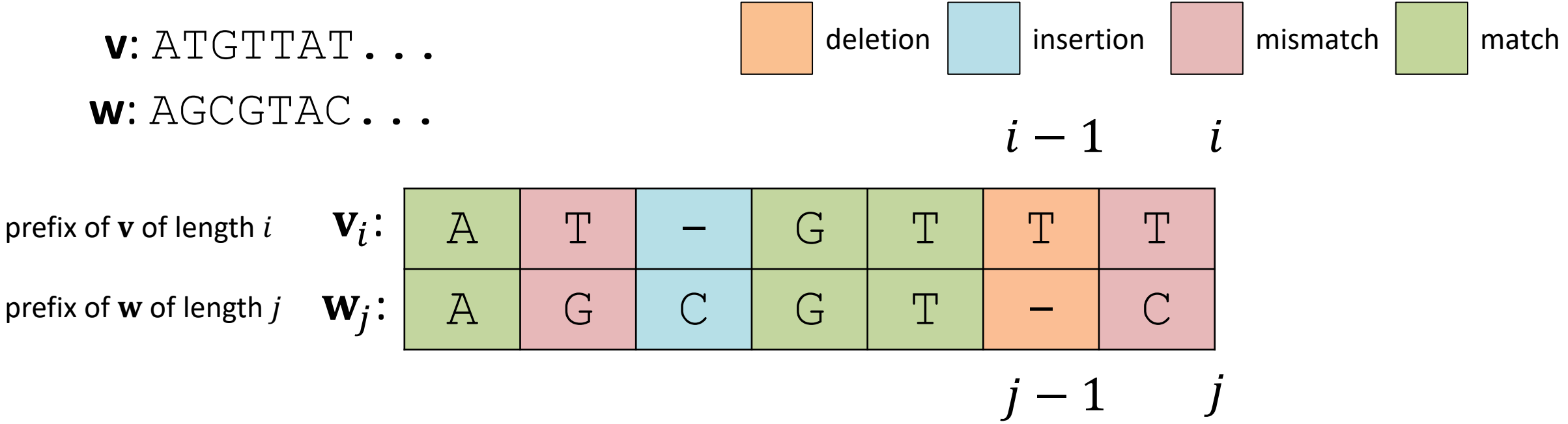
Elementary operations: insertion, deletions and substitutions of single characters

Edit Distance problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$, compute the minimum number $d(\mathbf{v}, \mathbf{w})$ of elementary operations to transform \mathbf{v} into \mathbf{w} .

$$d(\mathbf{cat}, \mathbf{car}) = 1 \quad d(\mathbf{cat}, \mathbf{ate}) = 2 \quad d(\mathbf{cat}, \mathbf{are}) = 3$$

Computing Edit Distance

Edit Distance problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$, compute the minimum number $d(\mathbf{v}, \mathbf{w})$ of elementary operations to transform \mathbf{v} into \mathbf{w} .



Optimal substructure:
 Edit distance obtained from edit distance of prefix of string.

Computing Edit Distance – Optimal Substructure

$d[i, j]$ is the edit distance of \mathbf{v}_i and \mathbf{w}_j ,
where \mathbf{v}_i is prefix of \mathbf{v} of length i and \mathbf{w}_j is prefix of \mathbf{w} of length j

Deletion: $d[i, j] = d[i - 1, j] + 1$
Extend by a character in \mathbf{v}

...	\mathbf{v}_i
...	–

Insertion: $d[i, j] = d[i, j - 1] + 1$
Extend by a character in \mathbf{w}

...	–
...	\mathbf{w}_j

Mismatch: $d[i, j] = d[i - 1, j - 1] + 1$
Extend by a character in \mathbf{v} and \mathbf{w}

...	\mathbf{v}_i
...	\mathbf{w}_j

Match: $d[i, j] = d[i - 1, j - 1]$
Extend by a character in \mathbf{v} and \mathbf{w}

...	\mathbf{v}_i
...	\mathbf{w}_j

Computing Edit Distance – Recurrence

$d[i, j]$ is the edit distance of \mathbf{v}_i and \mathbf{w}_j ,
 where \mathbf{v}_i is prefix of \mathbf{v} of length i and \mathbf{w}_j is prefix of \mathbf{w} of length j

$$d[i, j] = \min \begin{cases} d[i - 1, j] + 1, \\ d[i, j - 1] + 1, \\ d[i - 1, j - 1] + 1, & \text{if } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } v_i = w_j. \end{cases}$$

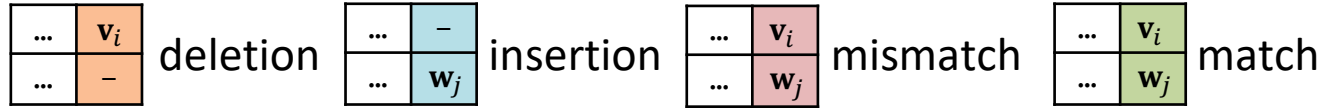
...	\mathbf{v}_i
...	-
...	-
...	\mathbf{w}_j
...	\mathbf{v}_i
...	\mathbf{w}_j
...	\mathbf{v}_i
...	\mathbf{w}_j

Computing Edit Distance – Recurrence

$d[i, j]$ is the edit distance of \mathbf{v}_i and \mathbf{w}_j ,
where \mathbf{v}_i is prefix of \mathbf{v} of length i and \mathbf{w}_j is prefix of \mathbf{w} of length j

$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

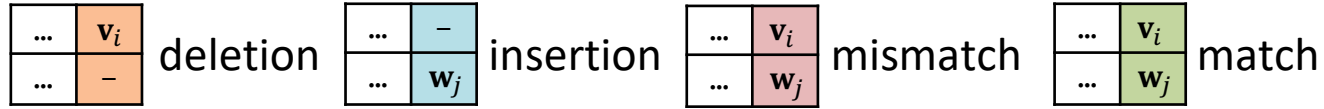
Computing Edit Distance – Dynamic Programming



$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

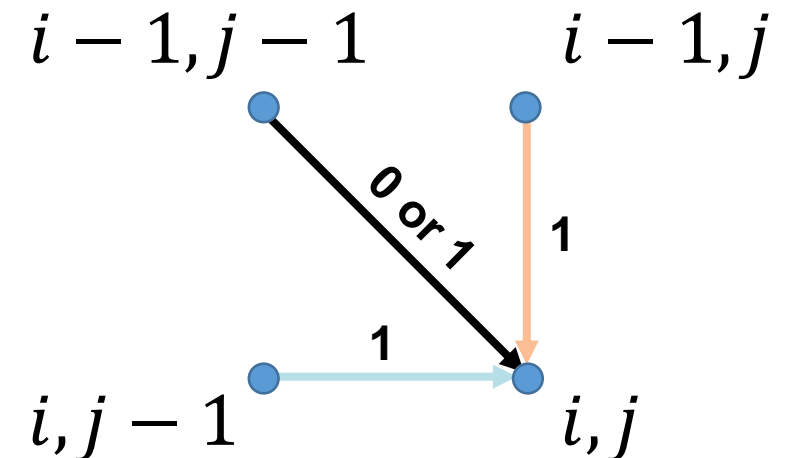
	W	A	T	C	G	
V		0	1	2	3	4
0						
A						
T						
G						
T						

Computing Edit Distance – Dynamic Programming

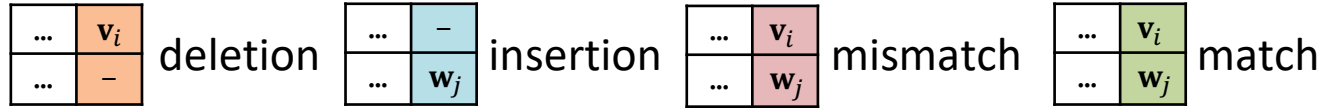


$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G
V	0	1	2	3	4
0	0				
A	1				
T	2				
G	3				
T	4				



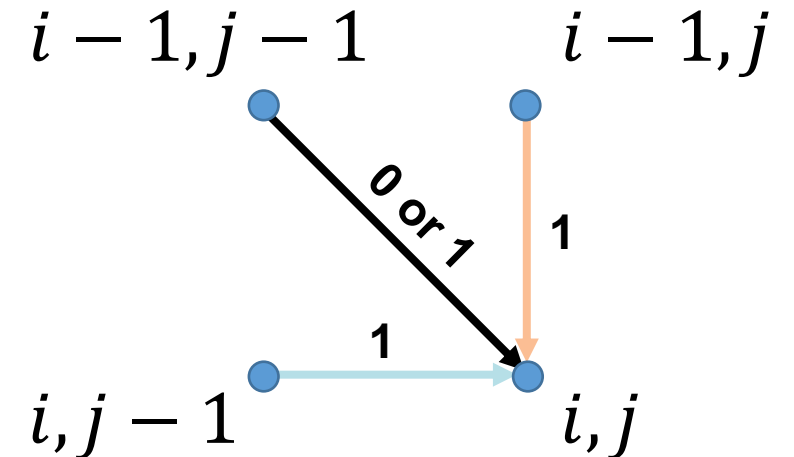
Computing Edit Distance – Dynamic Programming



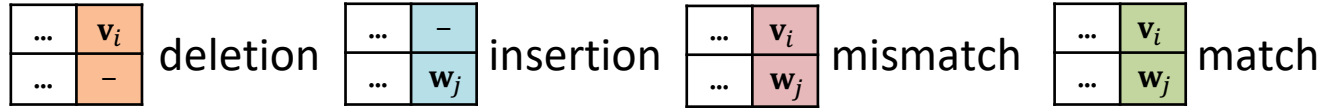
$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G
V	0	1	2	3	4
A	1				
T	2				
G	3				
T	4				

The table shows the edit distance between the prefix 'V' of the first string and prefixes of the second string 'WATCG'. The first column and first row are shaded. Light blue arrows point from (0,0) to (0,1), (0,1) to (0,2), (0,2) to (0,3), and (0,3) to (0,4). Orange arrows point from (0,0) to (1,0), (1,0) to (2,0), (2,0) to (3,0), and (3,0) to (4,0).



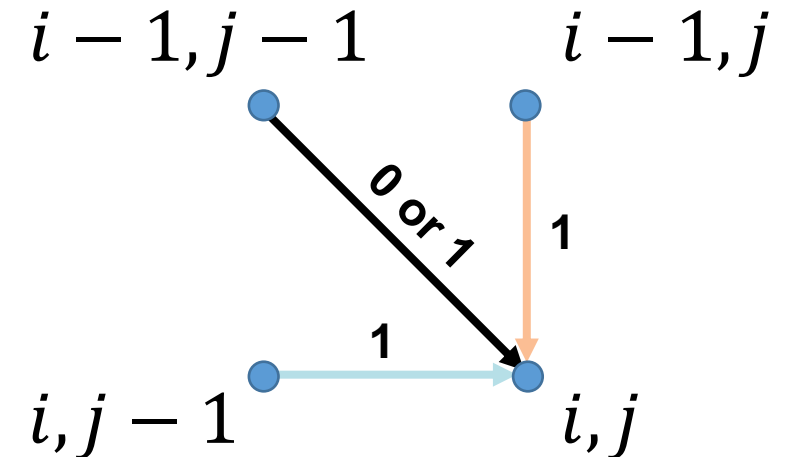
Computing Edit Distance – Dynamic Programming



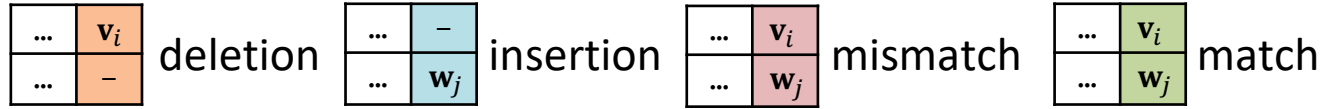
$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G	
V		0	1	2	3	4
0	0	1	2	3	4	
A	1	?				
T	2					
G	3					
T	4					

The table shows the edit distance between the prefix "V" and prefixes of "WATCG". The first column (index 0) shows the distance for the empty prefix of "W". The first row (index 0) shows the distance for the empty prefix of "V". Blue arrows indicate the path from (0,0) to (0,4). Orange arrows indicate the path from (0,0) to (1,0). A black arrow points from (0,1) to (1,1), which contains a question mark.

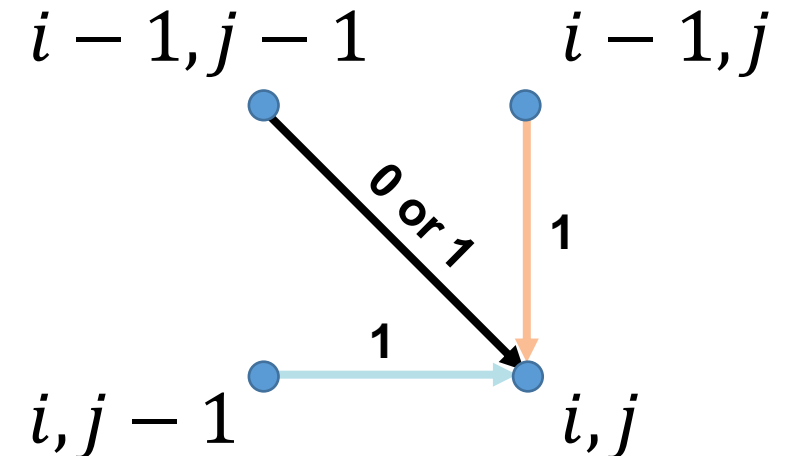


Computing Edit Distance – Dynamic Programming

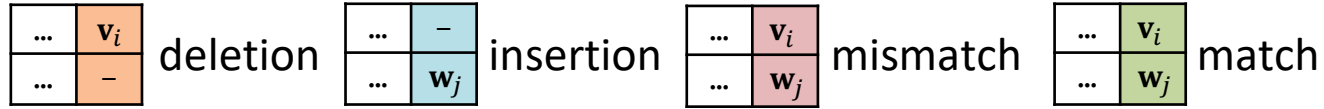


$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G	
V		0	1	2	3	4
0	0	1	2	3	4	
A	1	1	0			
T	2	2				
G	3	3				
T	4	4				



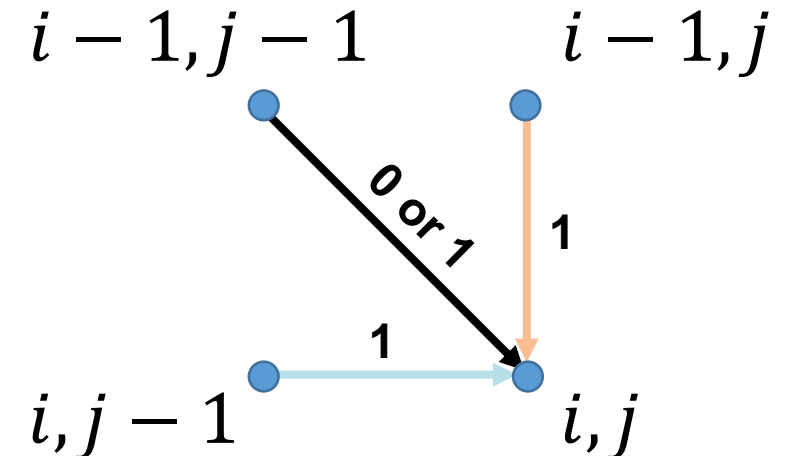
Computing Edit Distance – Dynamic Programming



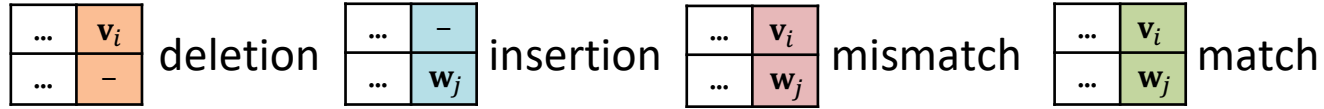
$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G	
V		0	1	2	3	4
A	0	1	0			
T	1	2	?			
G	2	3				
T	3	4				

The table shows the edit distance between the prefix "V" and prefixes of "WATCG". The cell at (T, A) contains a question mark, indicating the current step in the DP process. Arrows indicate the path of computation: blue arrows for horizontal moves (insertion), orange arrows for vertical moves (deletion), and a green arrow for a diagonal move (match). A black arrow points to the question mark cell, showing the three possible transitions from its neighbors.



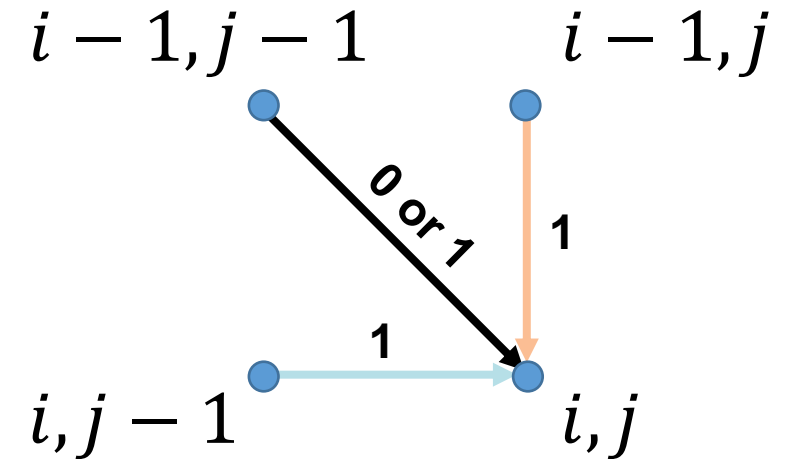
Computing Edit Distance – Dynamic Programming



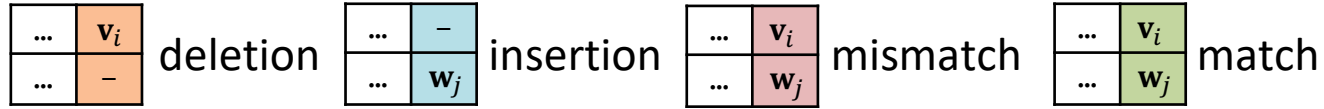
$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G	
V		0	1	2	3	4
0	0	1	2	3	4	
A	1	1	0			
T	2	2	1			
G	3	3				
T	4	4				

The table shows the edit distance between the prefix 'V' and prefixes of 'WATCG'. The first row and column are shaded. Blue arrows show the path from (0,0) to (0,4). Orange arrows show the path from (0,0) to (4,0). A green arrow points from (0,1) to (1,2).



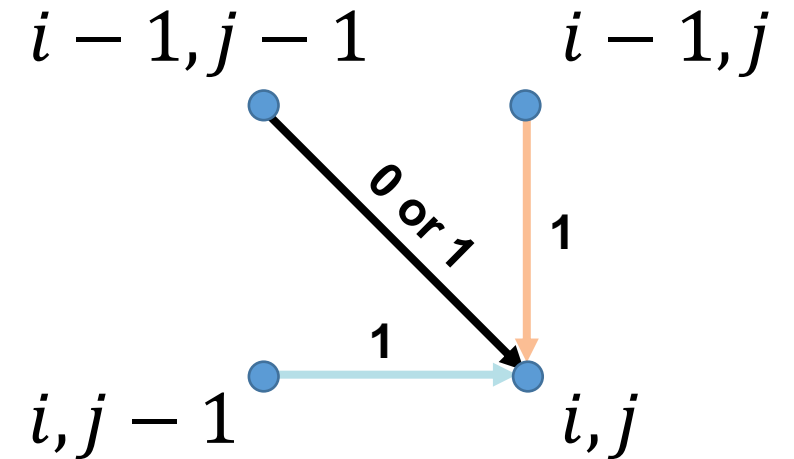
Computing Edit Distance – Dynamic Programming



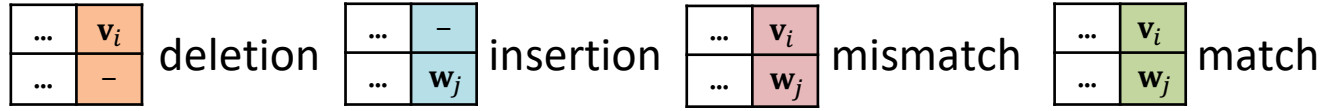
$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G	
V		0	1	2	3	4
0	0	1	2	3	4	
A	1					
T	2					
G	3					
T	4					

The table shows the edit distance between the prefix 'V' and prefixes of 'WATCG'. Light blue arrows indicate the path for the edit distance of 0 (deletion of 'V'). Red arrows indicate paths for other edit distances: solid red arrows for insertions (e.g., 'A' to 'VA'), dashed red arrows for mismatches (e.g., 'T' to 'VA'), and solid red arrows for matches (e.g., 'G' to 'VAG').



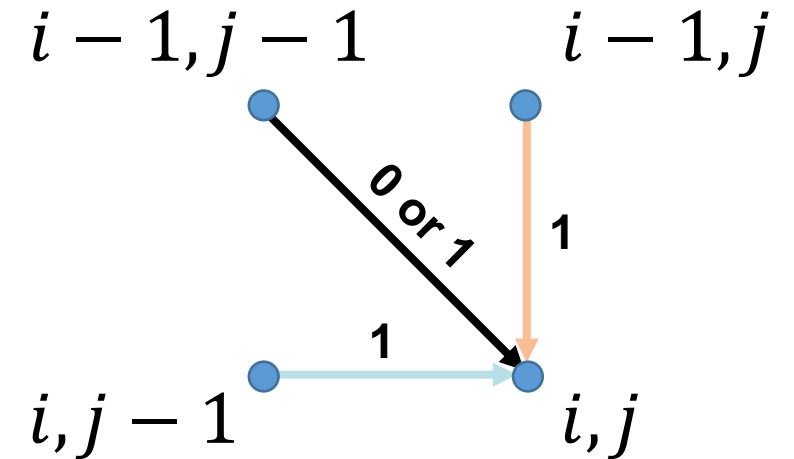
Computing Edit Distance – Dynamic Programming



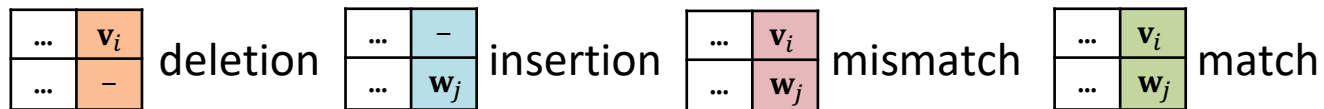
$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G
V	0	1	2	3	4
A	1				
T	2				
G	3				
T	4				

The table shows the edit distance between the prefix 'V' and prefixes of 'WATCG'. Light blue arrows show the path from (0,0) to (0,4). Red arrows show transitions from (0,1) to (1,1), (1,2), (2,2), (3,2), (4,2). Dashed red arrows show transitions from (1,2) to (2,1), (2,3), (3,3), (4,3).



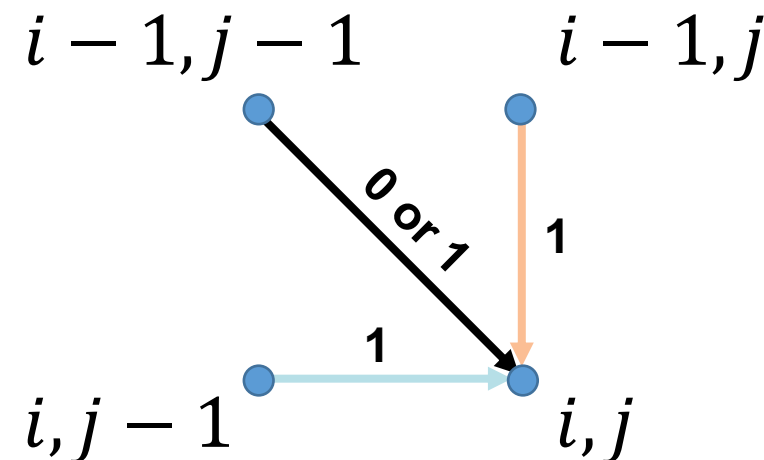
Computing Edit Distance – Dynamic Programming



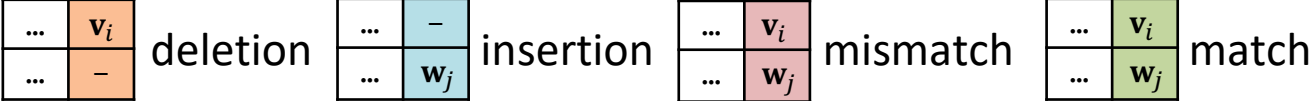
$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G
V	0	1	2	3	4
A	1				
T	2				
G	3				
T	4				

The table shows the edit distance between the prefix 'V' and prefixes of 'WATCG'. The first row and column are shaded gray. Light blue arrows show the path from (0,0) to (0,4). Red arrows show transitions from (0,1) to (1,1), (1,2), (2,2), (3,2), (4,2) and from (0,2) to (1,3), (2,3), (3,3), (4,3).

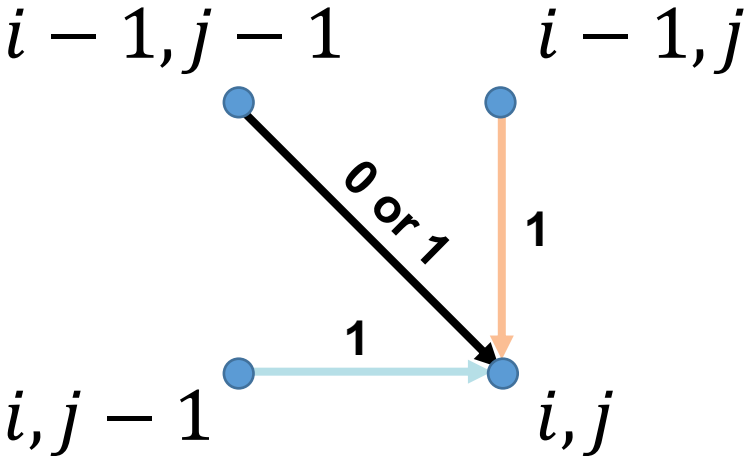


Computing Edit Distance – Dynamic Programming

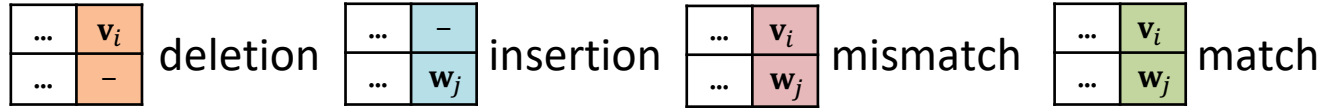


$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G	
V		0	1	2	3	4
0	0	1	2	3	4	
A	1	0	1	2	3	
T	2	1	0	1	2	
G	3	2	1	1	1	
T	4	3	2	2	2	

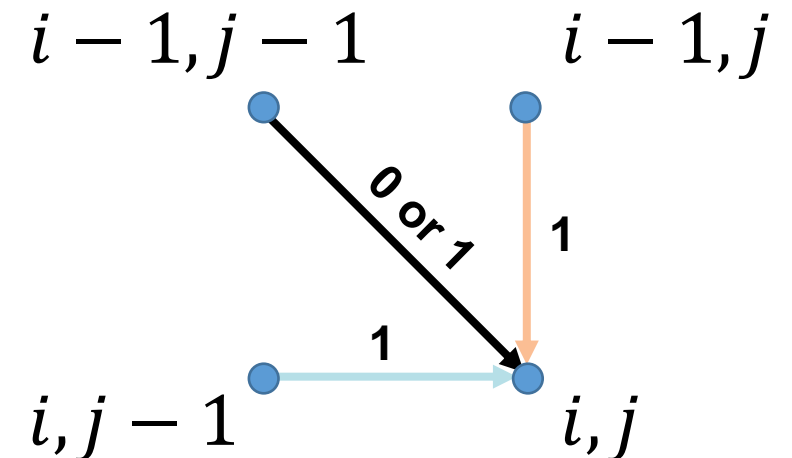


Computing Edit Distance – Dynamic Programming

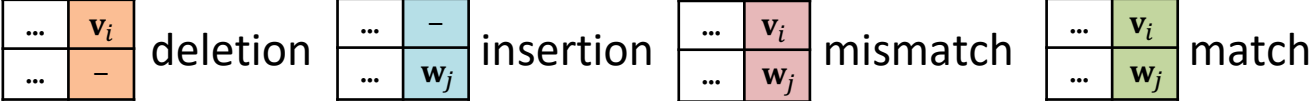


$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G
V	0	1	2	3	4
A	1	0	1	2	3
T	2	1	0	1	2
G	3	2	1	1	1
T	4	3	2	2	2



Computing Edit Distance – Dynamic Programming



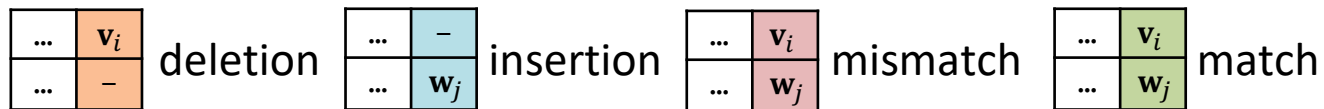
$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

		W	A	T	C	G
V		0	1	2	3	4
A	0	0	1	2	3	4
T	1	1	0	1	2	3
G	2	2	1	0	1	2
T	3	3	2	1	1	1
T	4	4	3	2	2	2

A	T	-	G	T
A	T	C	G	-

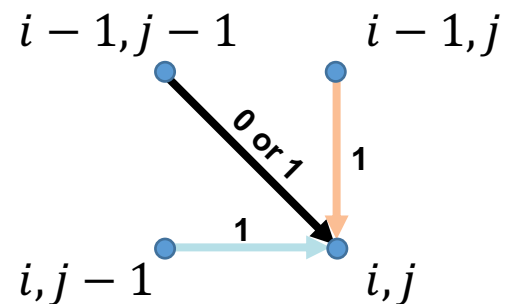
A	T	G	T
A	T	C	G

Computing Edit Distance – Running Time



$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G
V	0	1	2	3	4
A	1	0	1	2	3
T	2	1	0	1	2
G	3	2	1	1	1
T	4	3	2	2	2

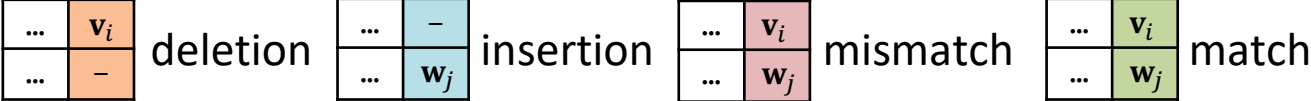


For each $(m + 1) \times (n + 1)$ entry:

- 3 addition operations
- 1 comparison operation
- 1 minimum operation

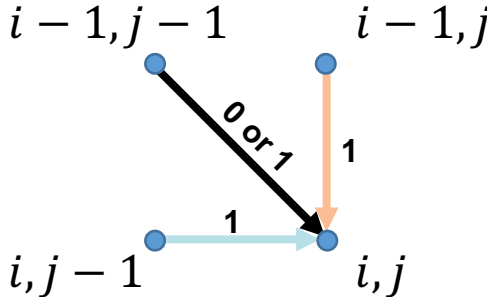
Running time: $O(mn)$ time

Computing Edit Distance – Running Time



$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i - 1, j] + 1, & \text{if } i > 0, \\ d[i, j - 1] + 1, & \text{if } j > 0, \\ d[i - 1, j - 1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i - 1, j - 1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	A	T	C	G
V	0	1	2	3	4
A	1	0	1	2	3
T	2	1	0	1	2
G	3	2	1	1	1
T	4	3	2	2	2

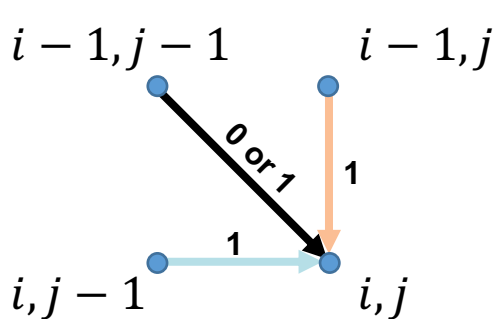
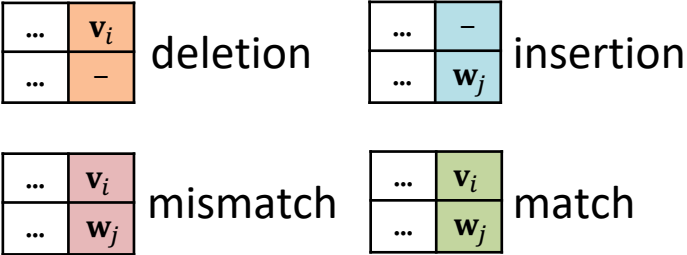


For each $(m + 1) \times (n + 1)$ entry:

- 3 addition operations
- 1 comparison operation
- 1 minimum operation

Running time: $O(mn)$ time

Computing Edit Distance – Your turn!



$$d[i, j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1, j] + 1, & \text{if } i > 0, \\ d[i, j-1] + 1, & \text{if } j > 0, \\ d[i-1, j-1] + 1, & \text{if } i > 0, j > 0 \text{ and } v_i \neq w_j, \\ d[i-1, j-1], & \text{if } i > 0, j > 0 \text{ and } v_i = w_j. \end{cases}$$

	W	C	A	R
V				
C				
A				
T				

$d(\mathbf{cat}, \mathbf{car}) =$

	W	A	T	E
V				
C				
A				
T				

$d(\mathbf{cat}, \mathbf{ate}) =$

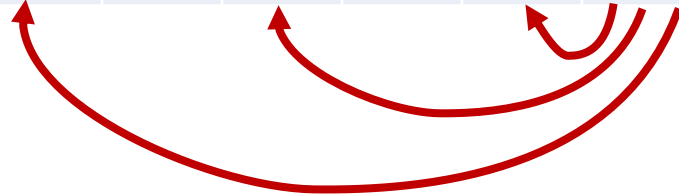
	W	A	R	E
V				
C				
A				
T				

$d(\mathbf{cat}, \mathbf{are}) =$

Change Problem and Edit distance

Make M cents using minimum number of 1, 3 and 5 cent coins.

Value	1	2	3	4	5	6	7
Min # coins	1	2	1	2	1	2	3

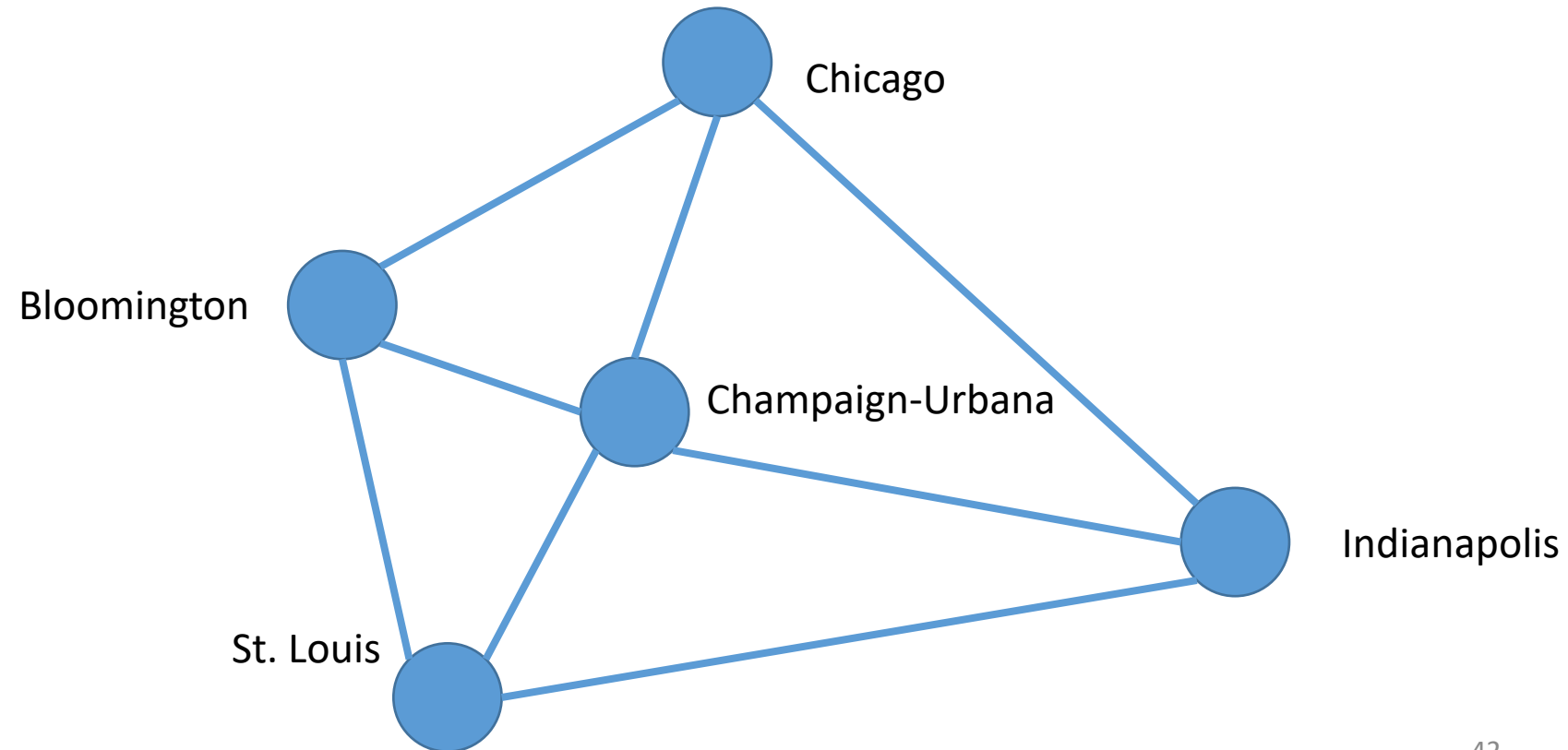


	W	A	T	C	G
V					
0	0	1	2	3	4
A	1	0	1	2	3
T	2	1	0	1	2
G	3	2	1	1	1
T	4	3	2	2	2

- Both have optimal substructure and can be solved using dynamic programming
 - These are examples of a more general problem!

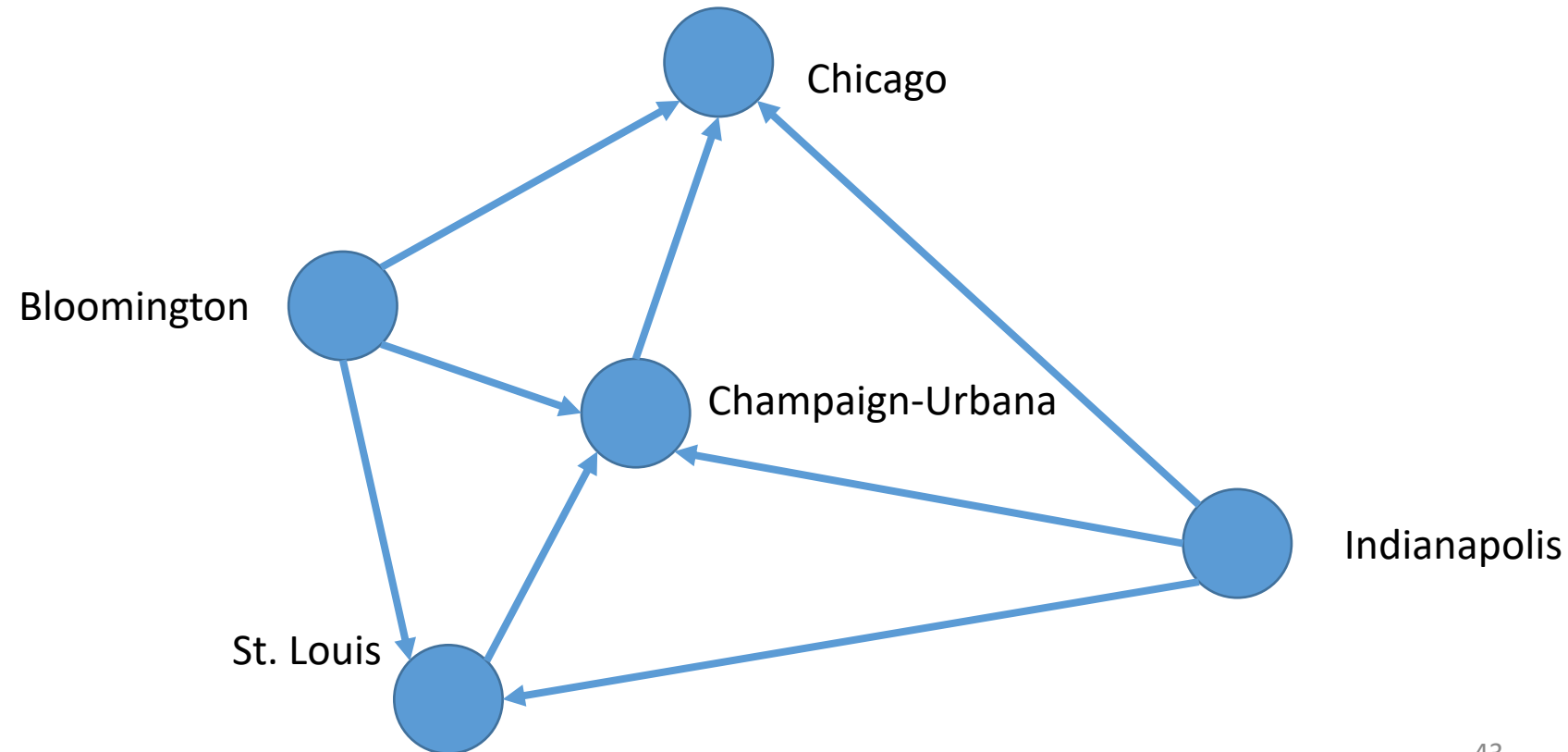
Review of Graph Theory

- Graph $G = (V, E)$
- Vertices $V = \{v_1, \dots, v_n\}$
- Edges $E = \{(v_i, v_j), \dots\}$



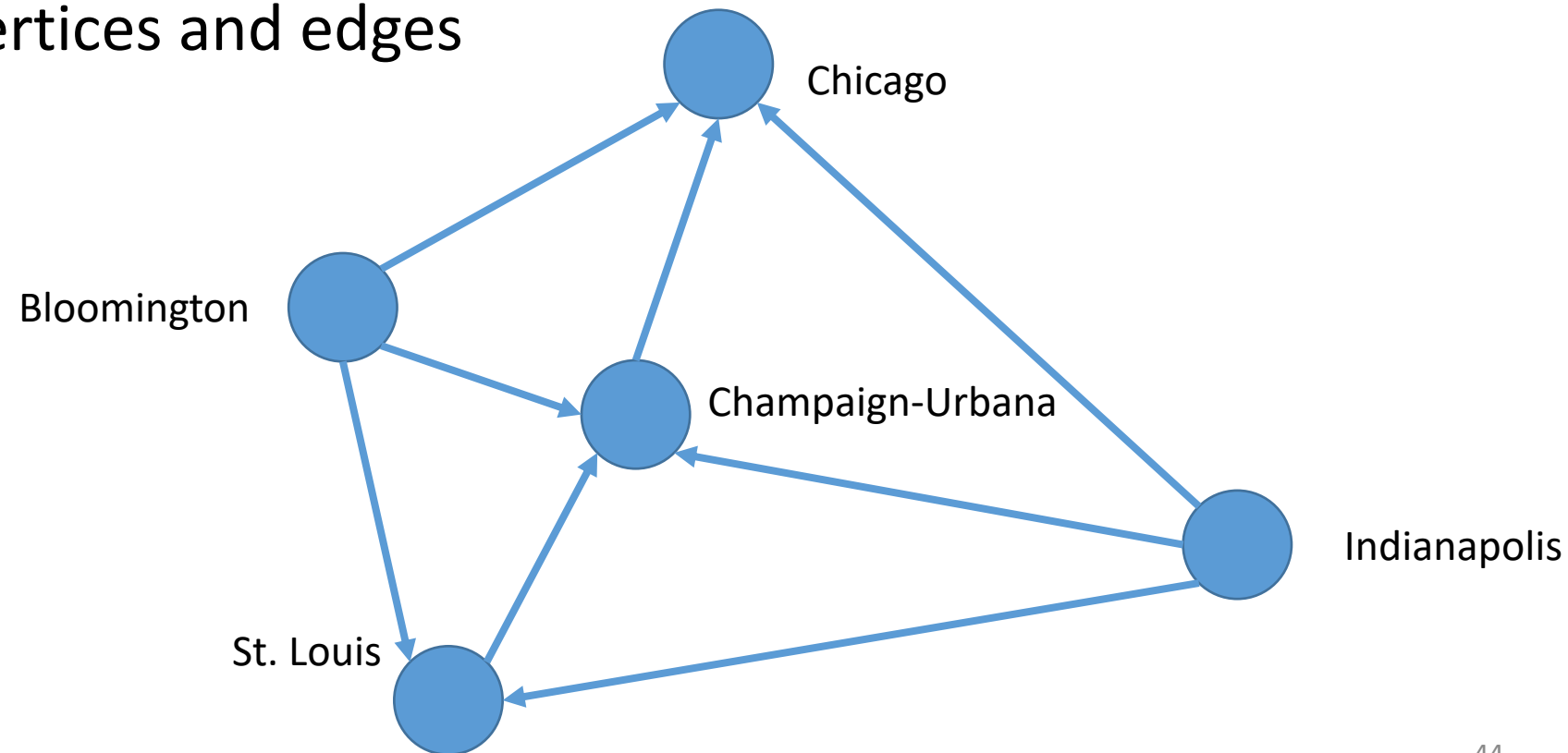
Review of Graph Theory

- Directed Graph $G = (V, E)$
- Vertices $V = \{v_1, \dots, v_n\}$
- Directed edges $E = \{(v_i, v_j), \dots\}$



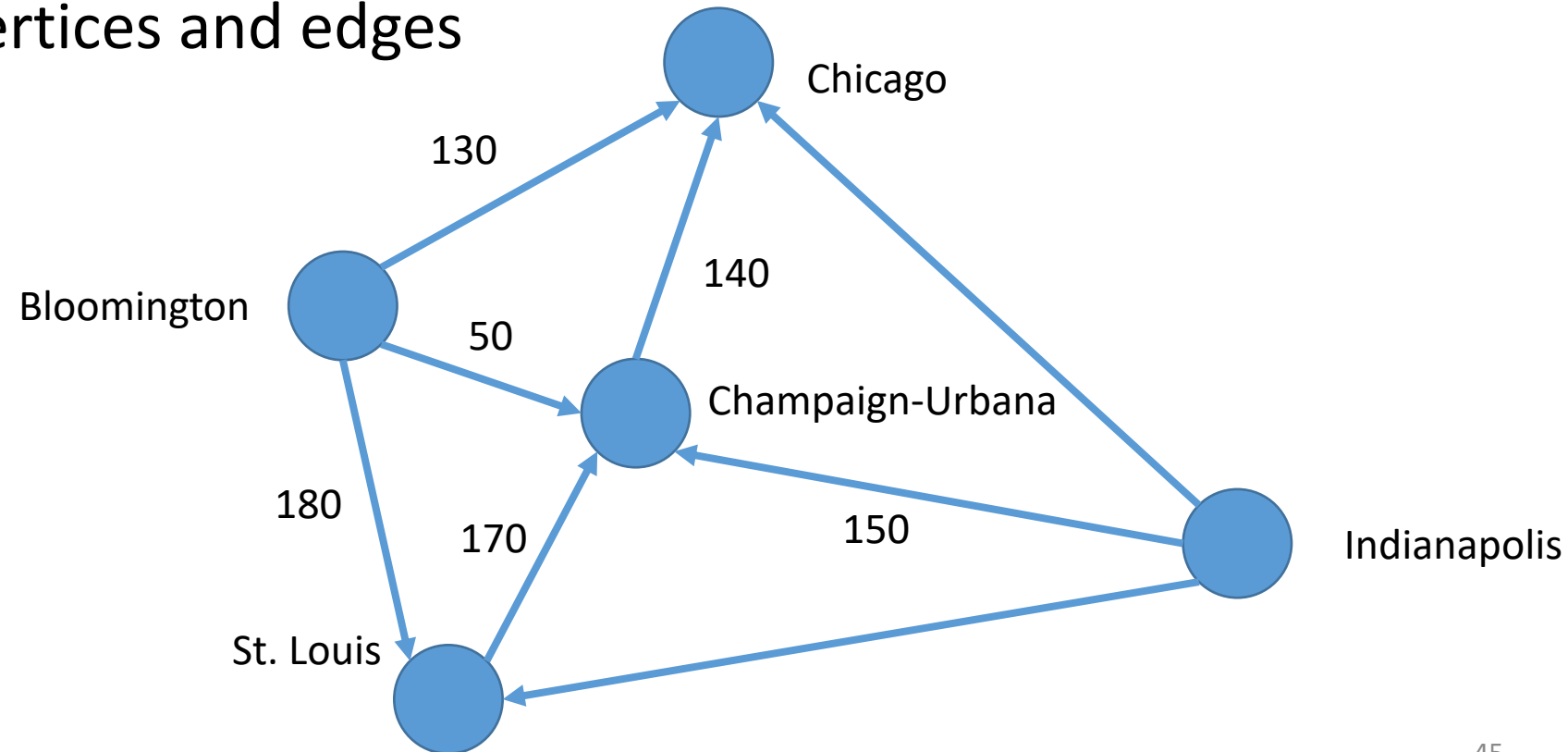
Review of Graph Theory

- Directed Graph $G = (V, E)$
- Vertices $V = \{v_1, \dots, v_n\}$
- Directed edges $E = \{(v_i, v_j), \dots\}$
- Path is a sequence of vertices and edges that connect them



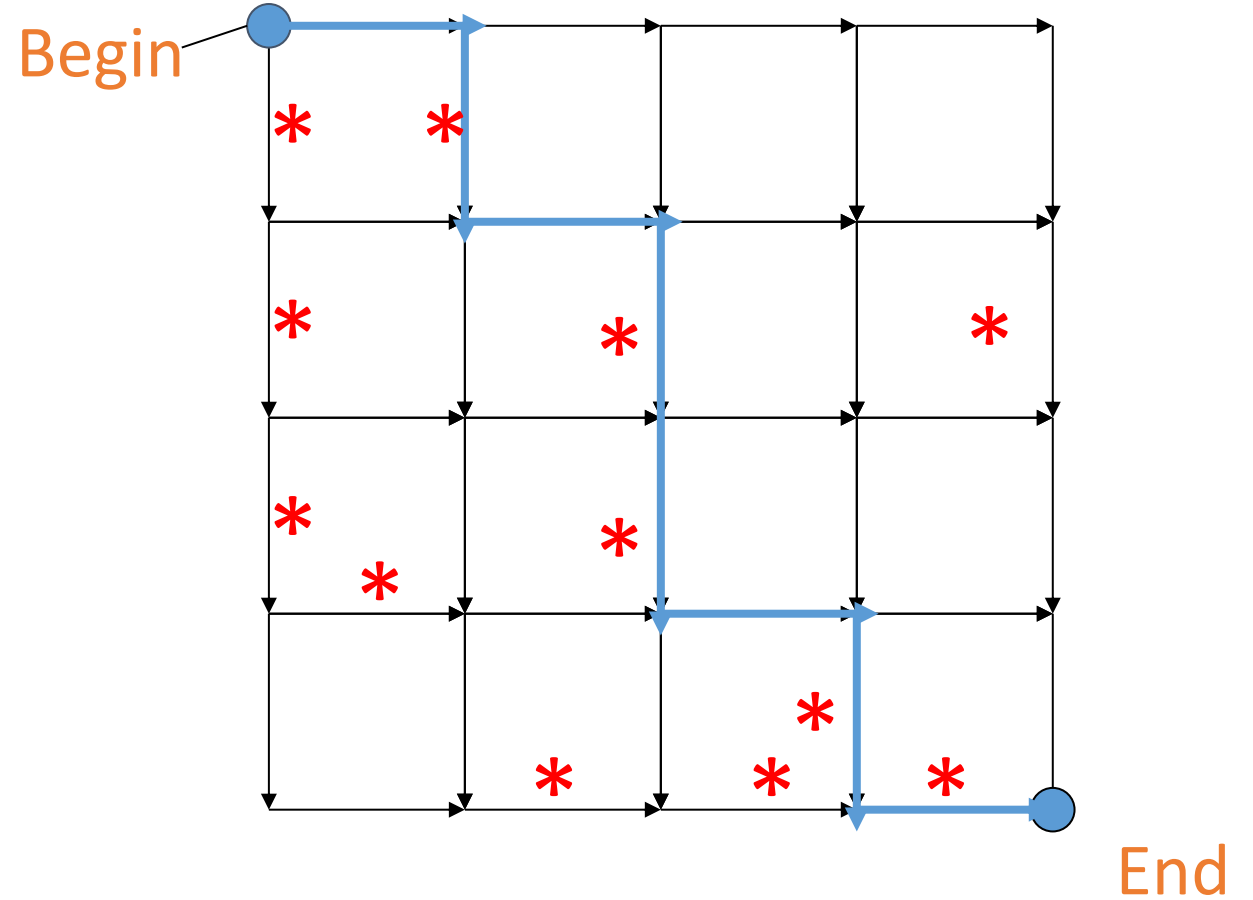
Review of Graph Theory

- Directed Graph $G = (V, E)$
- Vertices $V = \{v_1, \dots, v_n\}$
- Directed edges $E = \{(v_i, v_j), \dots\}$
- Path is a sequence of vertices and edges that connect them
- Edges can be weighted



Manhattan Tourist Problem

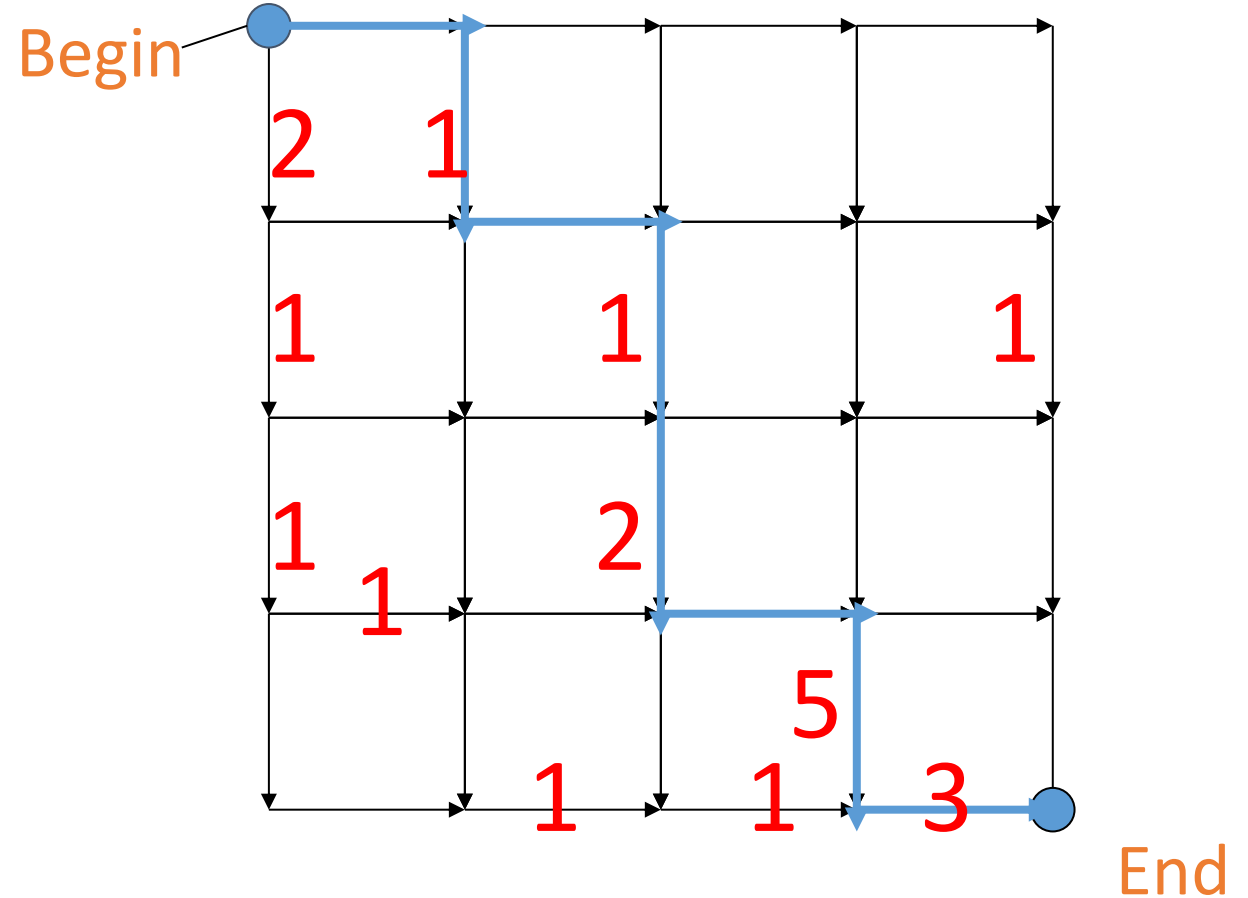
A tourist in Manhattan wants to visit the maximum number of attractions (*) by traveling on a path (only eastward and southward) from start to end



Manhattan Tourist Problem

A tourist in Manhattan wants to visit the maximum number of attractions (*) by traveling on a path (only eastward and southward) from start to end

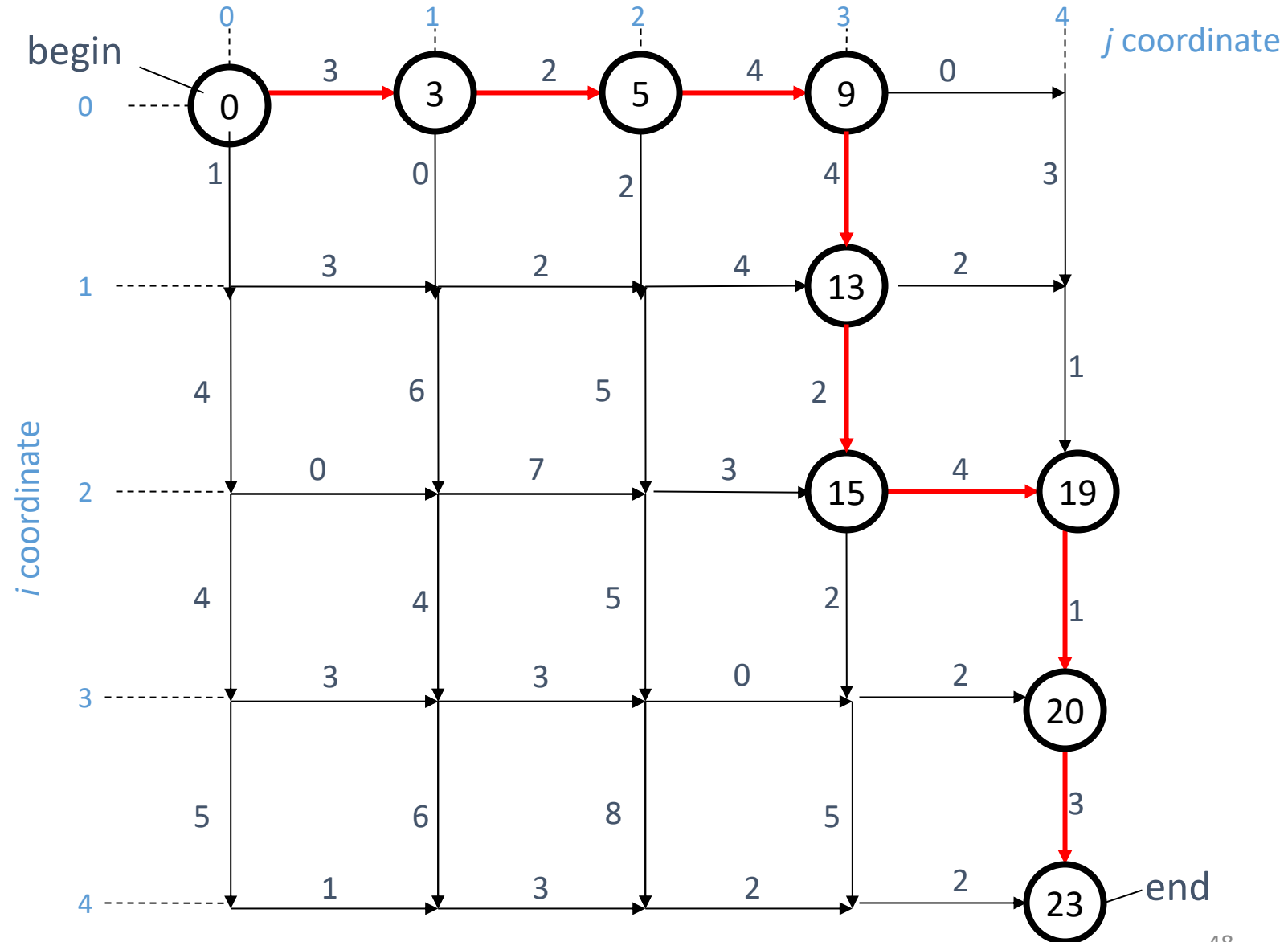
May be more than 1 attraction on a street.
Add weights!



Manhattan Tourist Problem

Manhattan Tourist Problem (MTP):

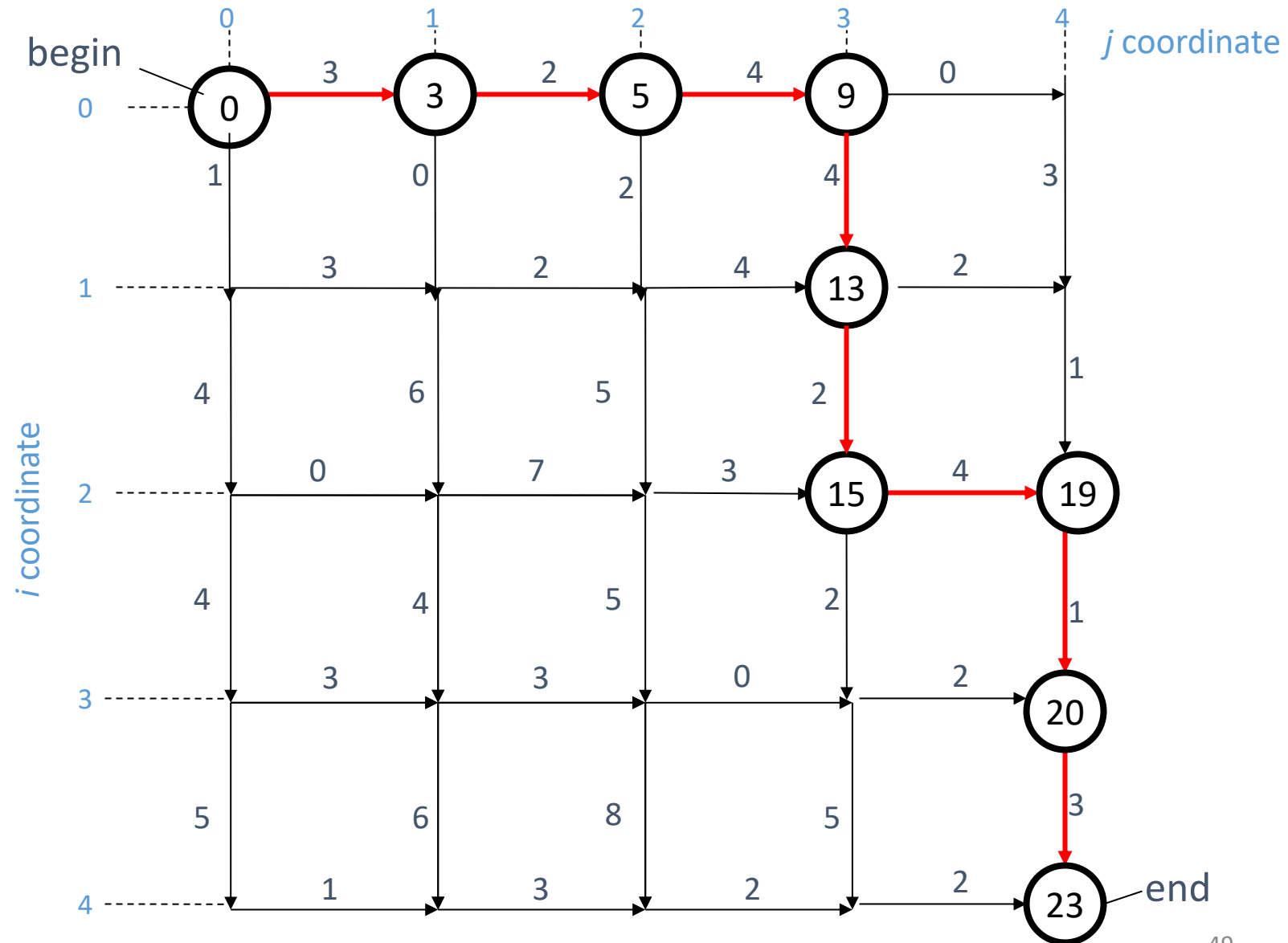
Given a weighted, directed grid graph G with two vertices “begin” and “end”, find the maximum weight path in G from “begin” to “end”.



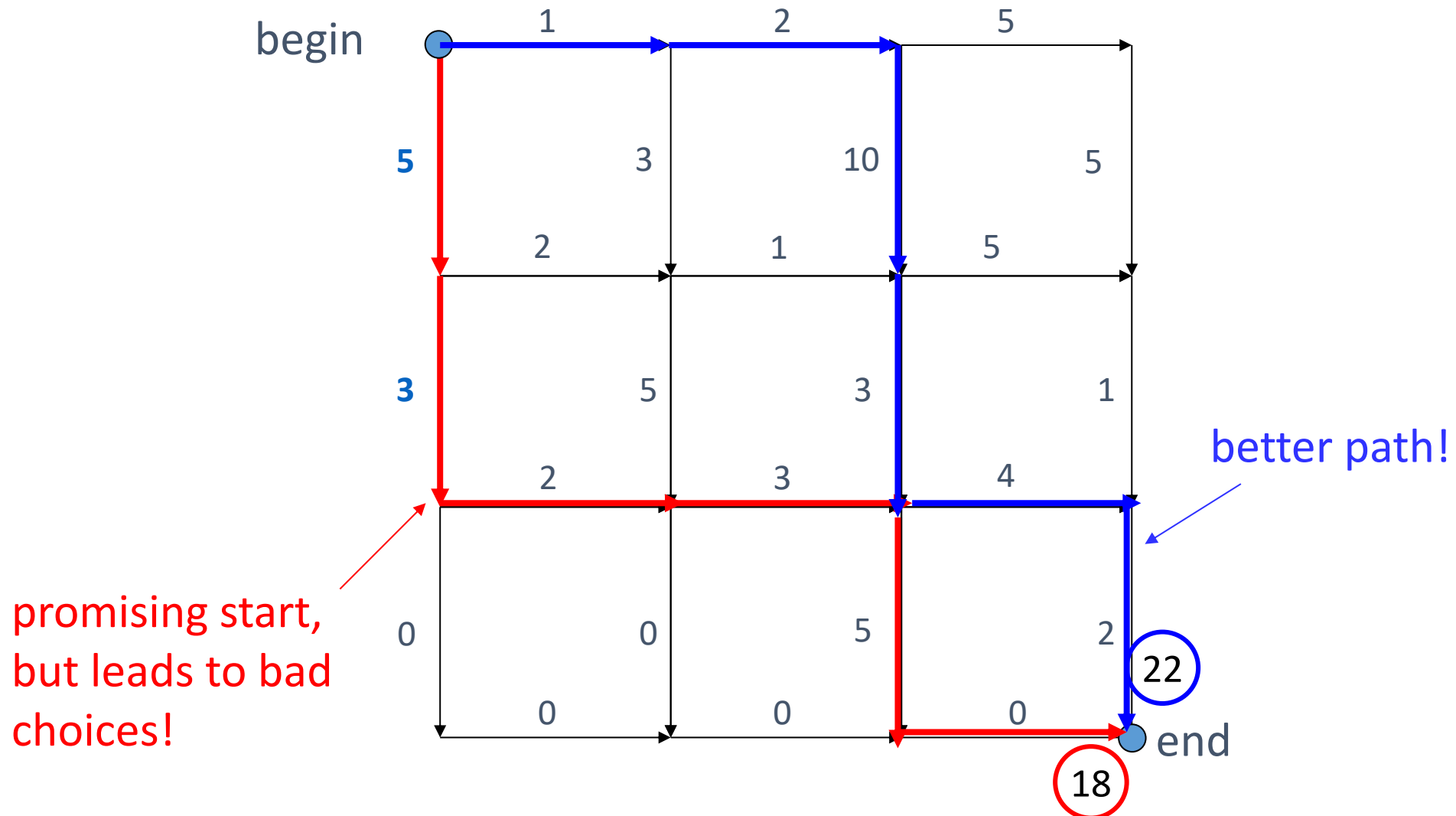
Manhattan Tourist Problem – Exhaustive Algorithm

Check all paths

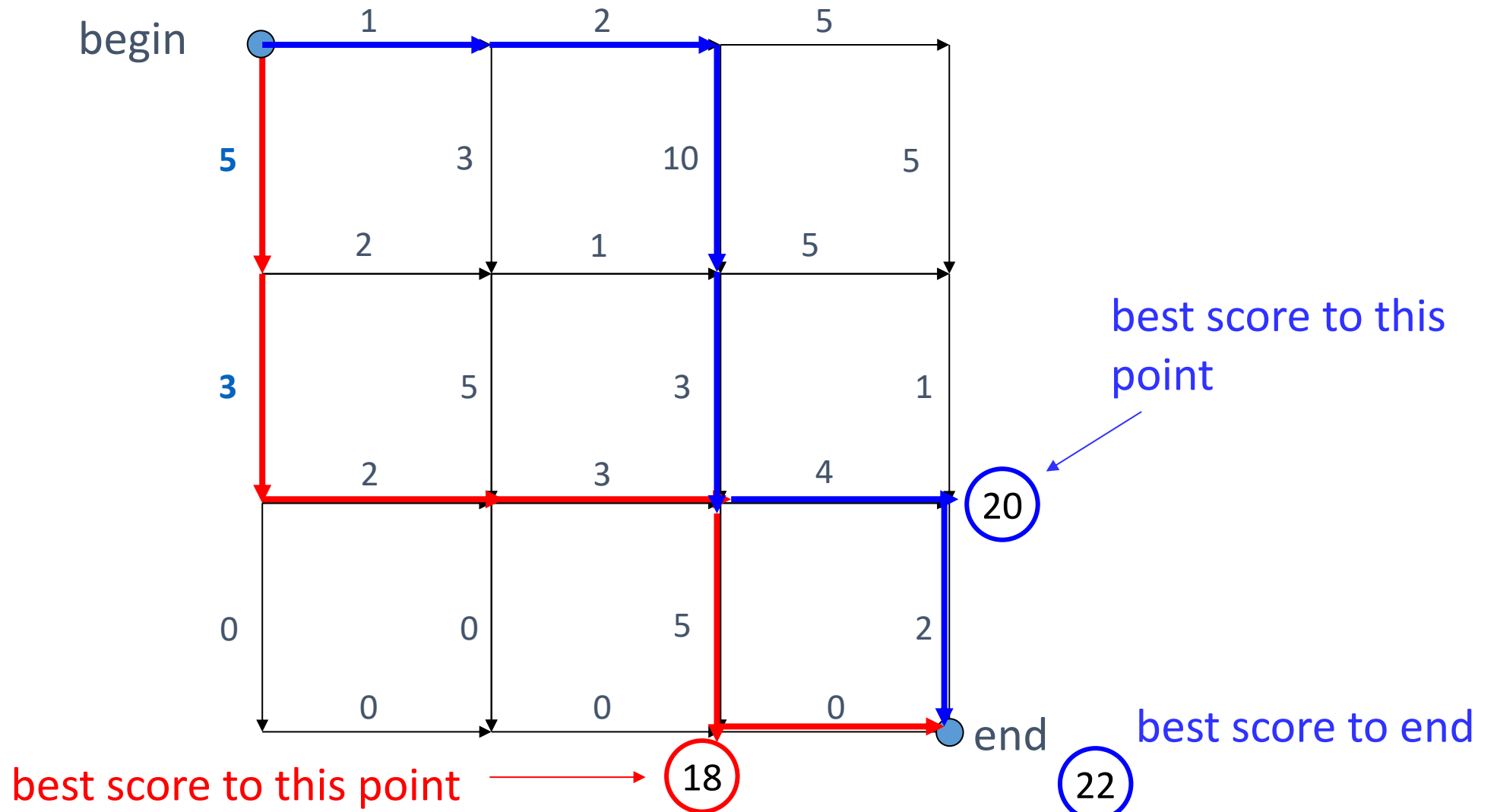
Question:
How many paths?



Manhattan Tourist Problem – Greedy Algorithm



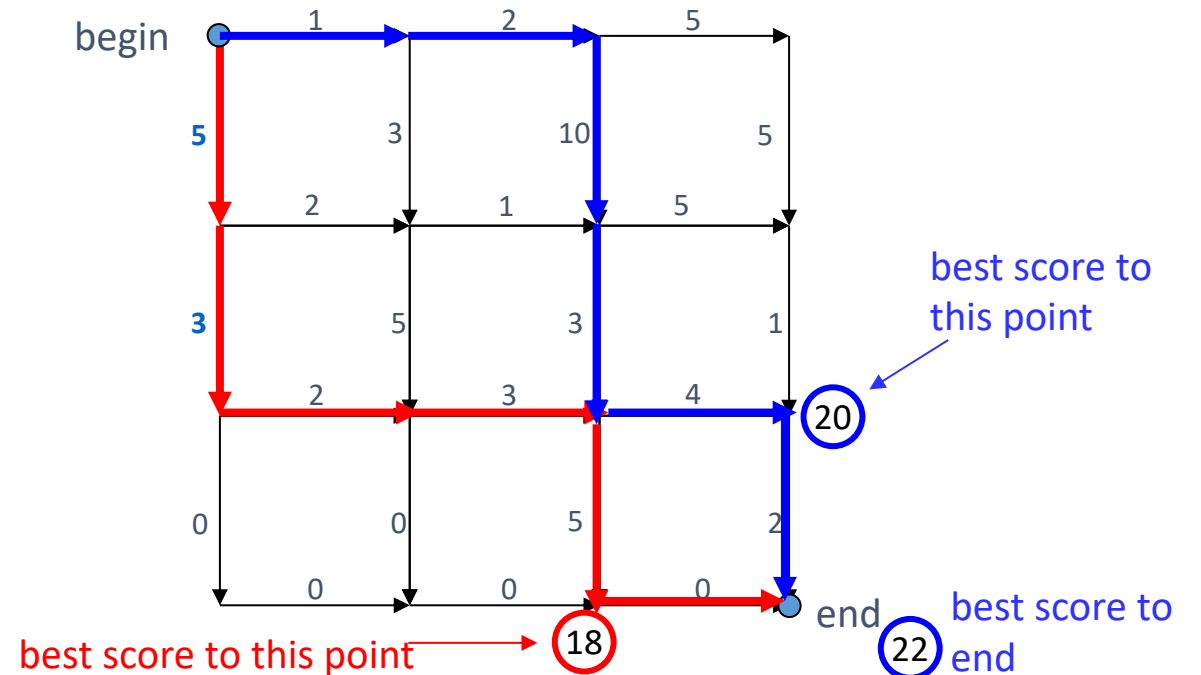
Manhattan Tourist Problem – Optimal Substructure



Manhattan Tourist Problem – Optimal Substructure

$s[i, j]$ is the best score for path to coordinate (i, j)

Question: What is the recurrence?

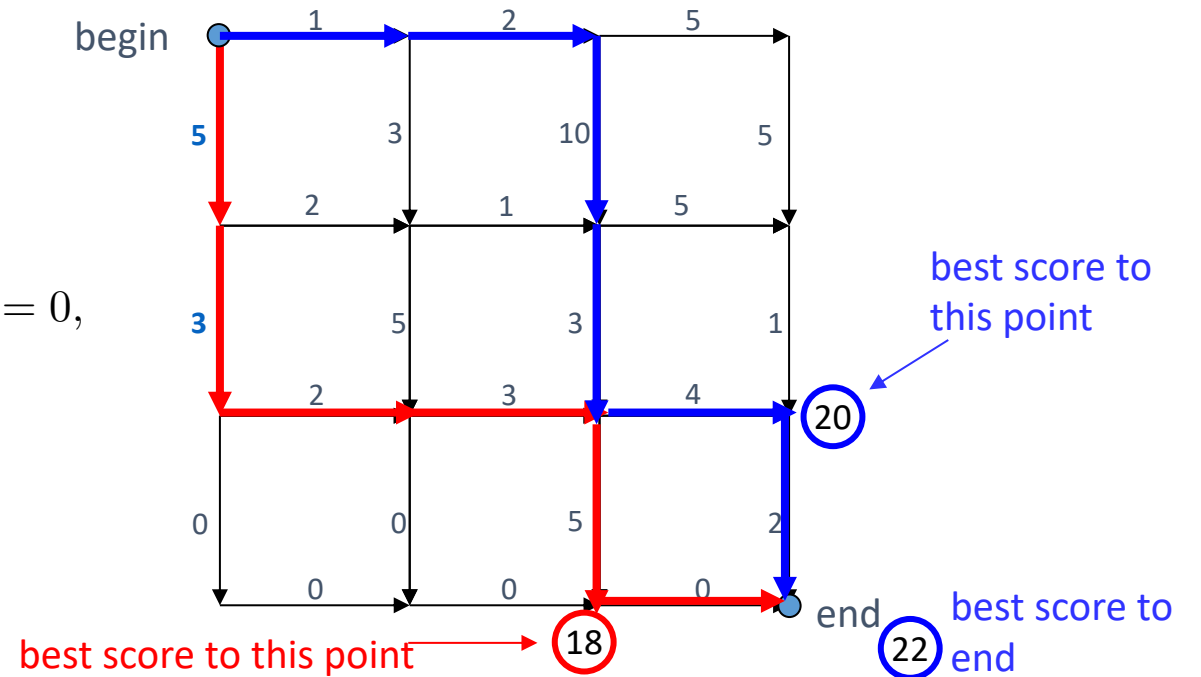


- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

Manhattan Tourist Problem – Optimal Substructure

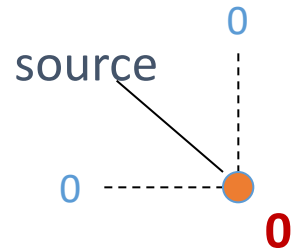
$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] + w[(i - 1, j), (i, j)] & \text{if } i > 0, \\ s[i, j - 1] + w[(i, j - 1), (i, j)] & \text{if } j > 0. \end{cases}$$



- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

MTP – Solving Recurrence using Dynamic Programming

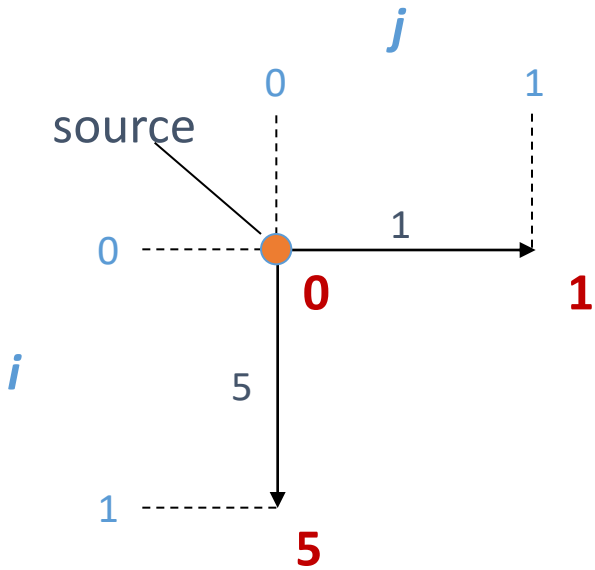


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] + w[(i - 1, j), (i, j)] & \text{if } i > 0, \\ s[i, j - 1] + w[(i, j - 1), (i, j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

MTP – Solving Recurrence using Dynamic Programming

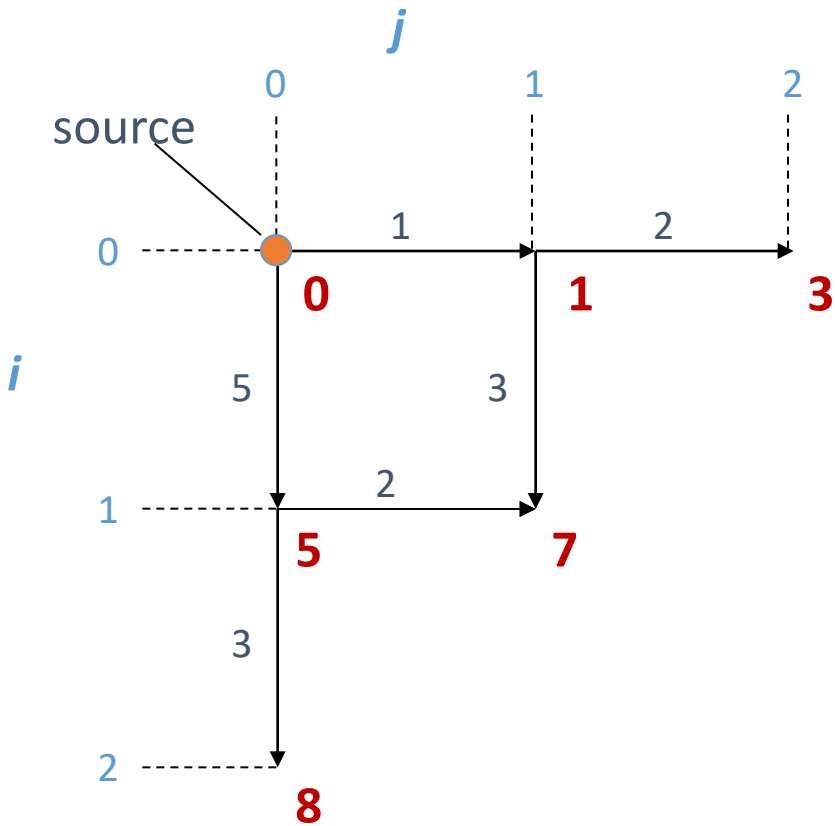


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] + w[(i - 1, j), (i, j)] & \text{if } i > 0, \\ s[i, j - 1] + w[(i, j - 1), (i, j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

MTP – Solving Recurrence using Dynamic Programming

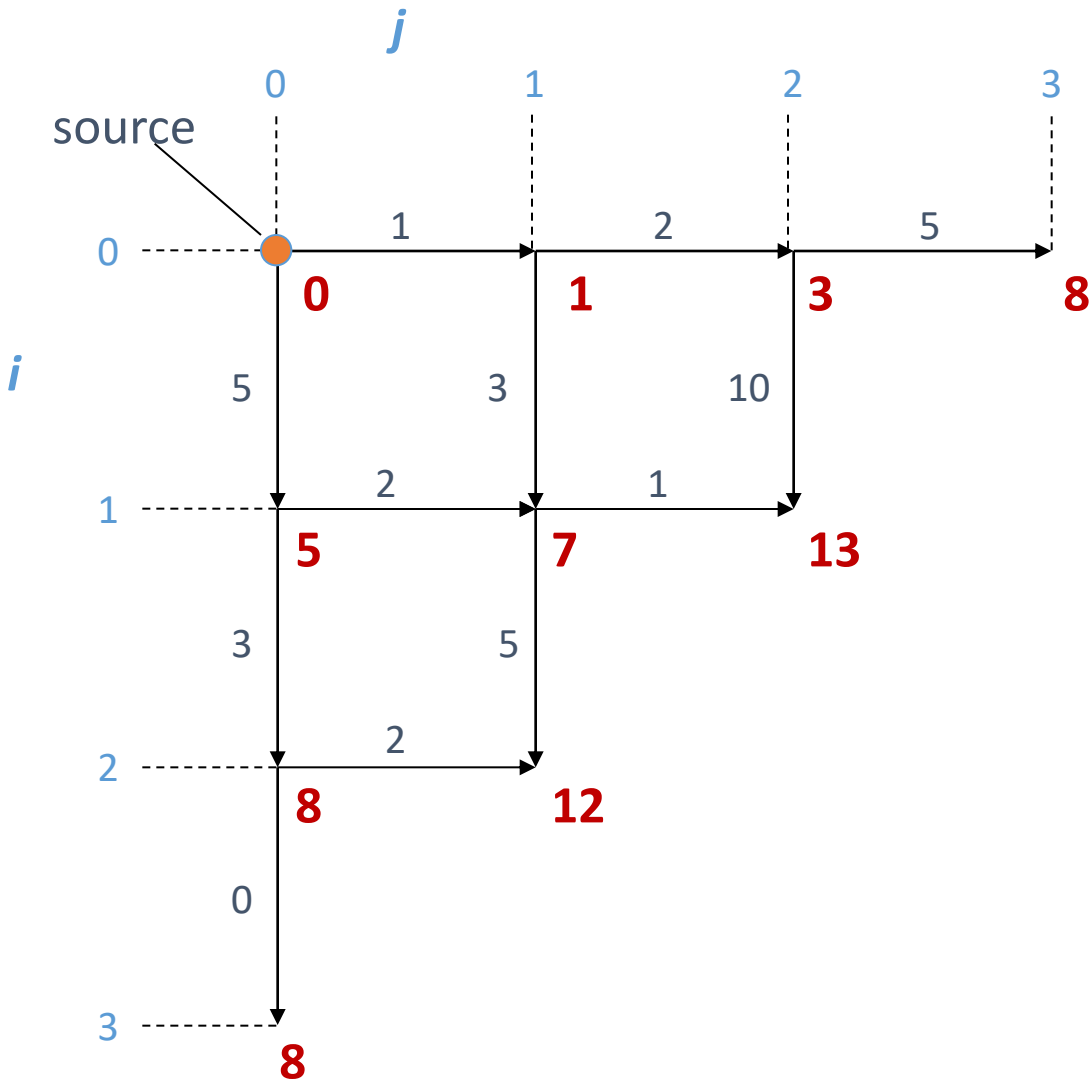


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] + w[(i - 1, j), (i, j)] & \text{if } i > 0, \\ s[i, j - 1] + w[(i, j - 1), (i, j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

MTP – Solving Recurrence using Dynamic Programming

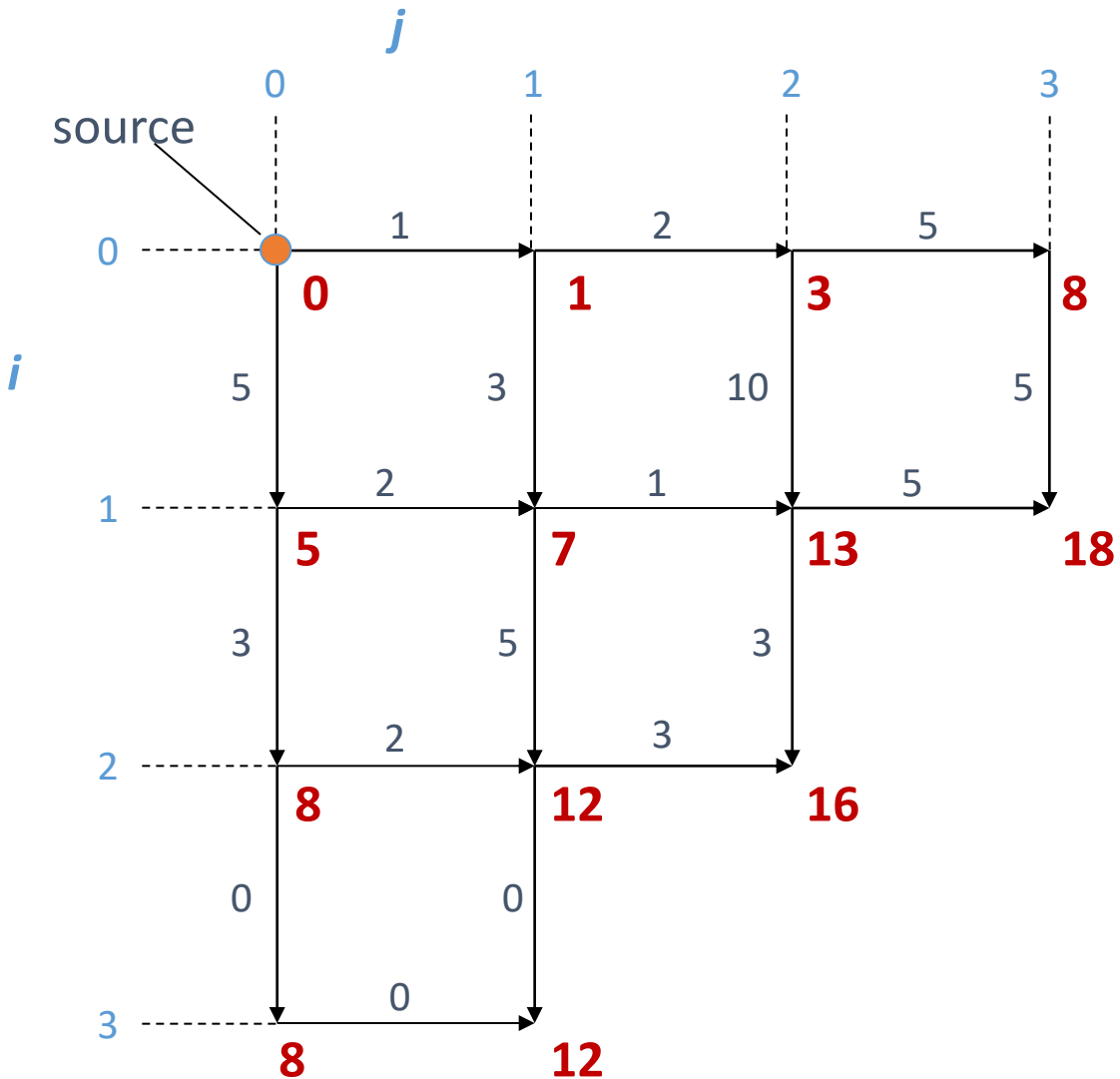


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] + w[(i - 1, j), (i, j)] & \text{if } i > 0, \\ s[i, j - 1] + w[(i, j - 1), (i, j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

MTP – Solving Recurrence using Dynamic Programming

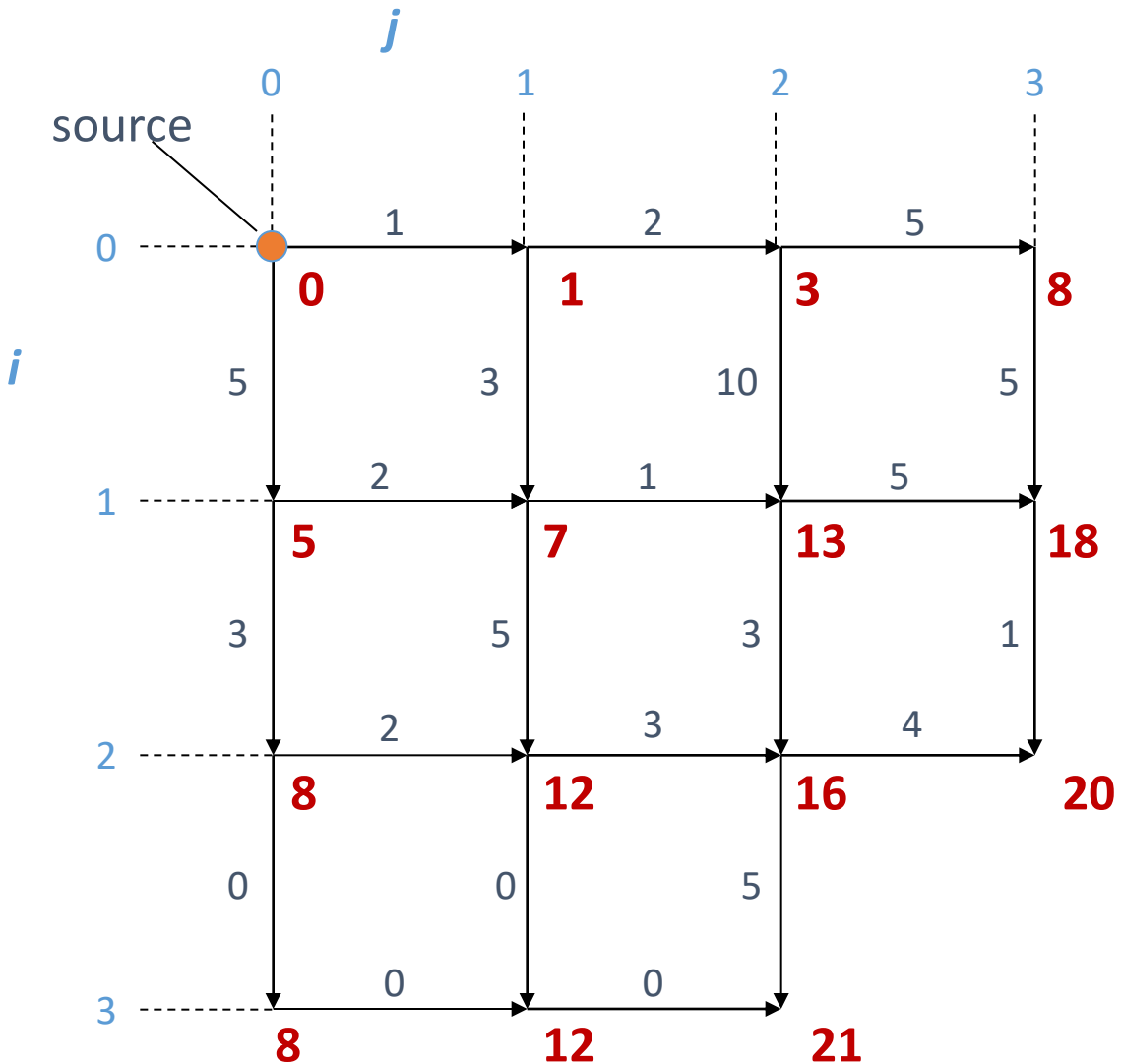


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] + w[(i - 1, j), (i, j)] & \text{if } i > 0, \\ s[i, j - 1] + w[(i, j - 1), (i, j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

MTP – Solving Recurrence using Dynamic Programming

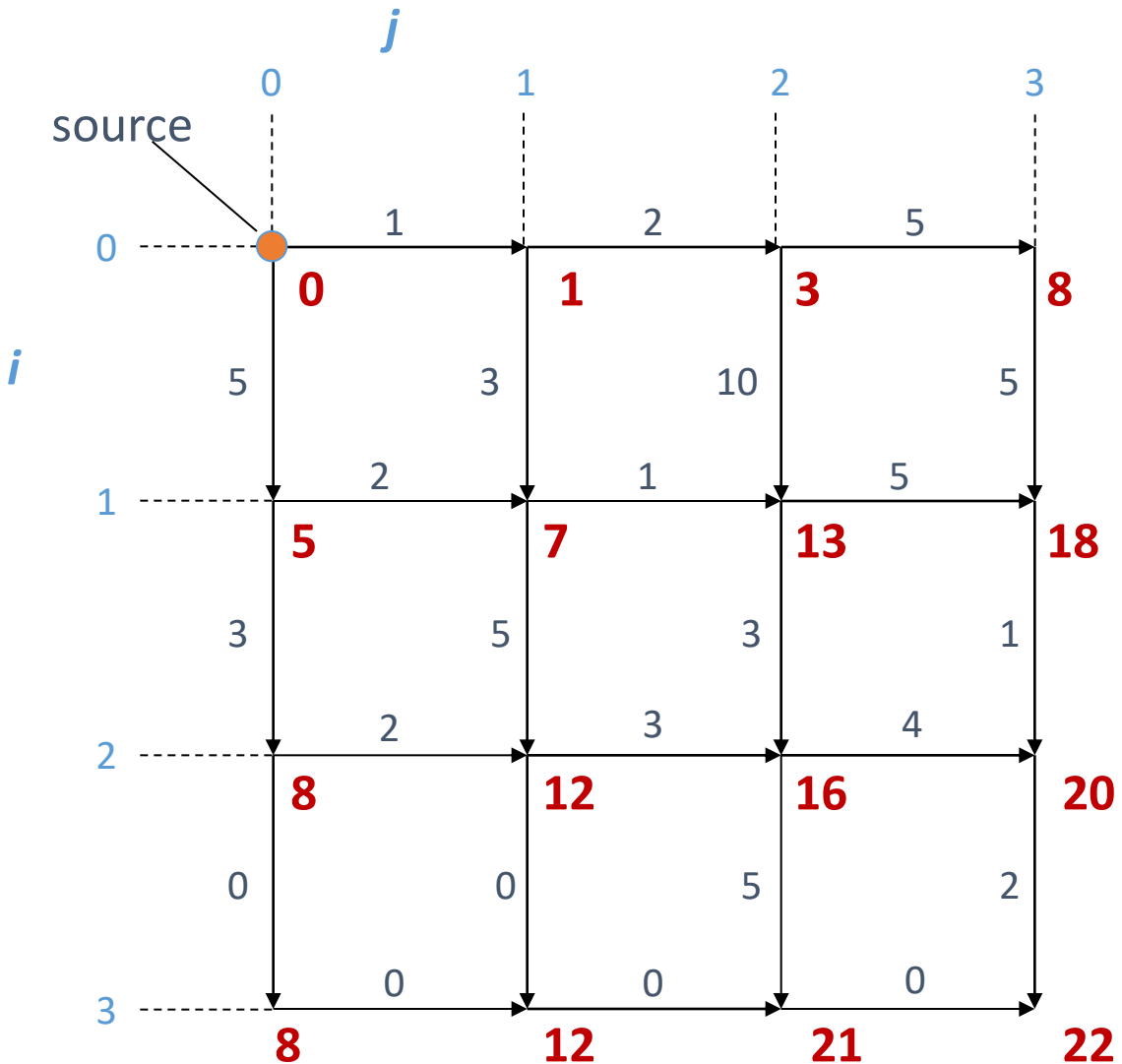


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] + w[(i - 1, j), (i, j)] & \text{if } i > 0, \\ s[i, j - 1] + w[(i, j - 1), (i, j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

MTP – Solving Recurrence using Dynamic Programming

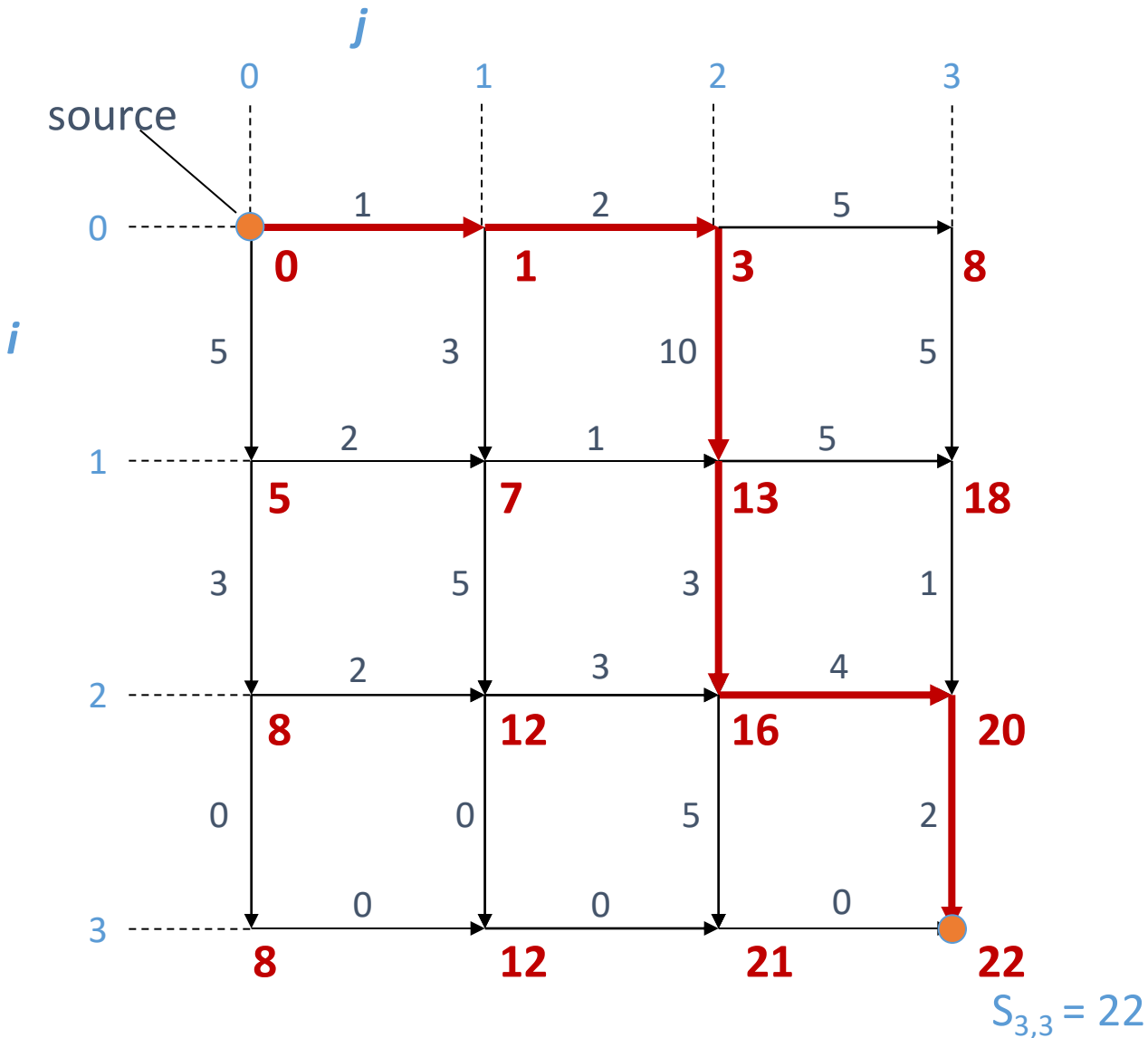


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] + w[(i - 1, j), (i, j)] & \text{if } i > 0, \\ s[i, j - 1] + w[(i, j - 1), (i, j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

MTP – Solving Recurrence using Dynamic Programming



$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i, j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i - 1, j] + w[(i - 1, j), (i, j)] & \text{if } i > 0, \\ s[i, j - 1] + w[(i, j - 1), (i, j)] & \text{if } j > 0. \end{cases}$$

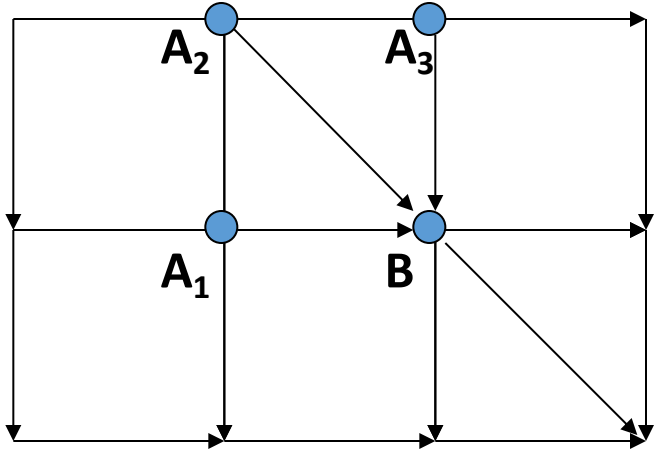
- $w[(i - 1, j), (i, j)]$ weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ weight of street between $(i, j - 1)$ and (i, j)

Let m be the number of rows and n be the number of columns.

Running time: $O(mn)$

Question: Implementation?

Manhattan Is Not a Perfect Grid

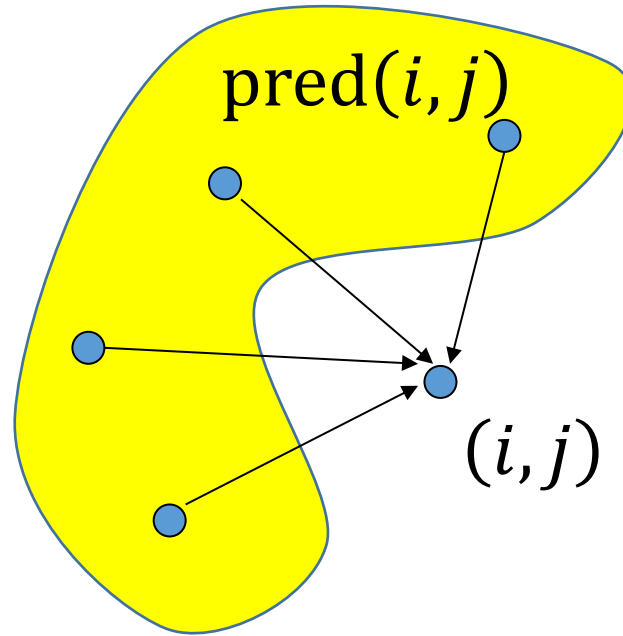


What about diagonals?

$$s[B] = \max \begin{cases} s[A_1] + w[A_1, B], \\ s[A_2] + w[A_2, B], \\ s[A_3] + w[A_3, B]. \end{cases}$$

Manhattan Is Not a Perfect Grid, It's a Directed Graph

$G = (V, E)$ is a directed acyclic graph (DAG) with nonnegative edges weights $w : E \rightarrow \mathbb{R}^+$



Each edge is evaluated once: $O(|E|)$ time

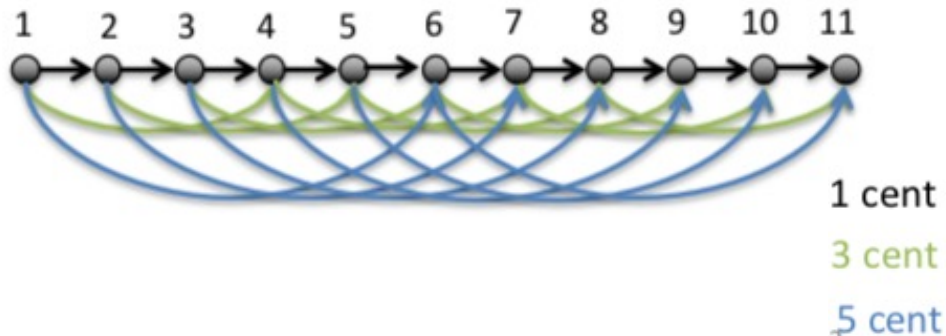
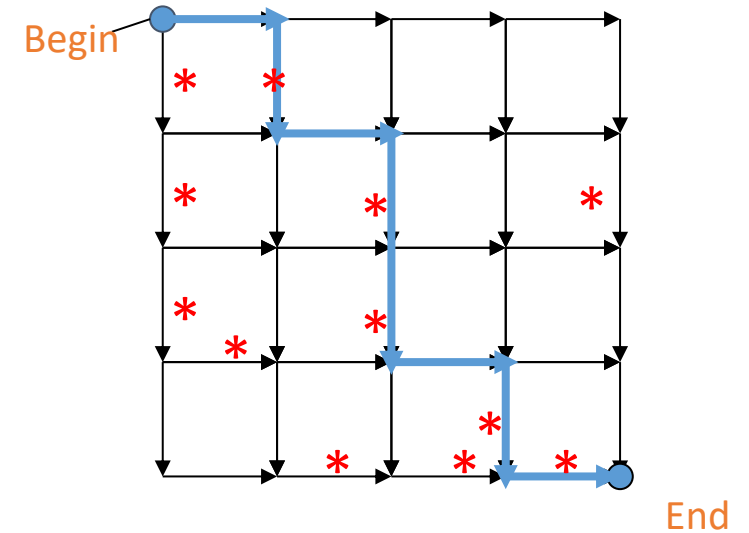
$$s[0, 0] = 0$$

$$s[i, j] = \max_{(i', j') \in \text{pred}(i, j)} \{s[i', j'] + w[(i', j'), (i, j)]\}$$

Dynamic Programming as a Graph Problem

Manhattan Tourist Problem:

Every path in directed graph is a possible tourist path. Find **maximum weight path**.
Running time: $O(mn) = O(|E|)$

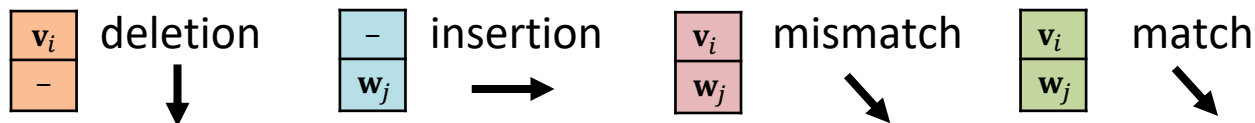


Change Problem: Make M cents using minimum number of coins $\mathbf{c} = (1, 3, 5)$. Every path in directed graph is a possible change. Find **shortest path**.
Running time: $O(Mn) = O(|E|)$

What About the Edit Distance Problem?

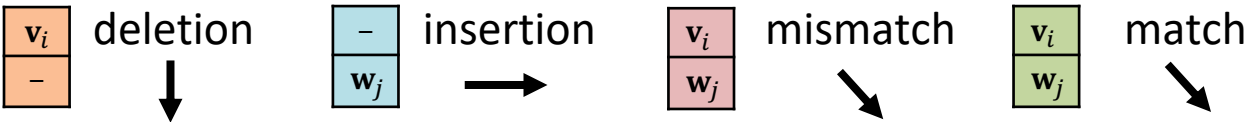
	W	A	T	C	G	
V		0	1	2	3	4
0						
A	1					
T	2					
G	3					
T	4					

Edit Distance problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$, compute the minimum number $d(\mathbf{v}, \mathbf{w})$ of elementary operations to transform \mathbf{v} into \mathbf{w} .



What About the Edit Distance Problem?

		W	A	T	C	G
V		0	1	2	3	4
	0	0	0	0	0	0
A	1	0	0	0	0	0
T	2	0	0	0	0	0
G	3	0	0	0	0	0
T	4	0	0	0	0	0



Edit Distance problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$, compute the minimum number $d(\mathbf{v}, \mathbf{w})$ of elementary operations to transform \mathbf{v} into \mathbf{w} .

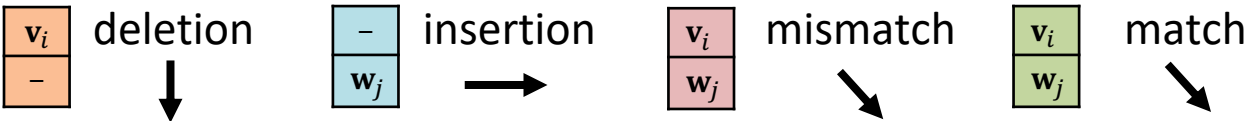
Edit graph is a weighed, directed grid graph $G = (V, E)$ with source vertex $(0, 0)$ and target vertex (m, n) . Each edge (i, j) has weight $[i, j]$ corresponding to edit cost: deletion (1), insertion (1), mismatch (1) and match (0).

What About the Edit Distance Problem?

		W	A	T	C	G
V		0	1	2	3	4
0		0	1	2	3	4
A	1	0	1	2	3	4
T	2	0	1	2	3	4
G	3	0	1	2	3	4
T	4	0	1	2	3	4

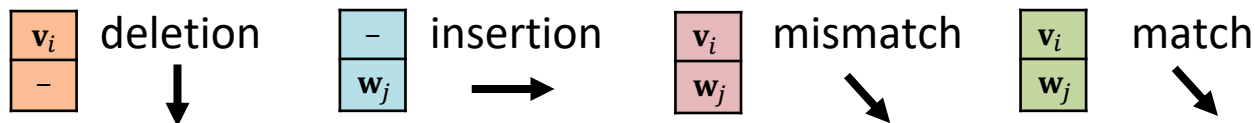
Alignment is a path from $(0, 0)$ to (m, n)

Edit graph is a weighed, directed grid graph $G = (V, E)$ with source vertex $(0, 0)$ and target vertex (m, n) . Each edge (i, j) has weight $[i, j]$ corresponding to edit cost: deletion (1), insertion (1), mismatch (1) and match (0).



What About the Edit Distance Problem?

	W	A	T	C	G
V	0	1	2	3	4
A	1	0	1	2	3
T	2	1	0	1	2
G	3	2	1	0	1
T	4	3	2	1	0



Edit Distance problem: Given edit graph $G = (V, E)$, with edge weights $c : E \rightarrow \{0,1\}$. Find shortest path from $(0, 0)$ to (m, n) .

Alignment is a path from $(0, 0)$ to (m, n)

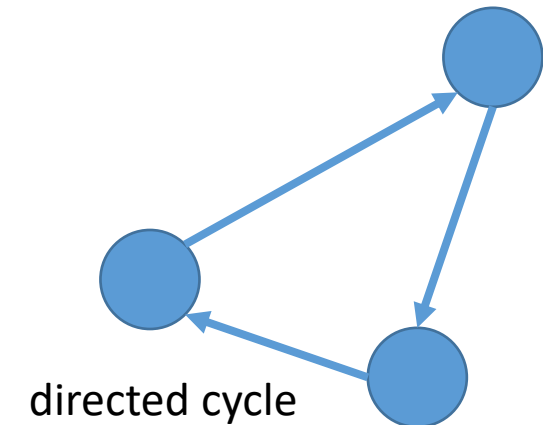
Edit graph is a weighed, directed grid graph $G = (V, E)$ with source vertex $(0, 0)$ and target vertex (m, n) . Each edge (i, j) has weight $[i, j]$ corresponding to edit cost: deletion (1), insertion (1), mismatch (1) and match (0).

Shortest Path vs Longest Path

- Change graph, edit graph and the MTP grid are **directed graphs G** .
- Change problem and Edit Distance problem are **minimization** problems.
- Find **shortest path** in **G** from source to sink.
- Manhattan Tourist problem is a **maximization** problem.
- Find **longest path** in **G** from source to sink.

Shortest Path vs Longest Path

- Shortest path in directed graphs can be found efficiently (Dijkstra, Bellman-Ford, Floyd-Warshall algorithms)
- Longest path in direct graphs *cannot* be found efficiently (NP-hard).
- Change graph, edit graph and MTP grid graph are **directed acyclic graphs (DAGs)**.
- No directed cycles.
- Longest path problem in a DAG can be solved efficiently by dynamic programming



Question: What's the relation between absence of directed cycles and optimal substructure?

Weighted Edit Distance

$d[i, j]$ is the edit distance of \mathbf{v}_i and \mathbf{w}_j ,
 where \mathbf{v}_i is prefix of \mathbf{v} of length i and \mathbf{w}_j is prefix of \mathbf{w} of length j

$$d[i, j] = \min \begin{cases} d[i-1, j] + 1, & \text{deletion} \\ d[i, j-1] + 1, & \text{insertion} \\ d[i-1, j-1] + 1, & \text{mismatch} \\ d[i-1, j-1], & \text{if } v_i = w_j. \end{cases}$$

...	\mathbf{v}_i
...	-

...	-
...	\mathbf{w}_j

...	\mathbf{v}_i
...	\mathbf{w}_j

...	\mathbf{v}_i
...	\mathbf{w}_j

Replace +1 with different penalties for different types of edits.

Summary

1. Change problem
2. Review of running time analysis
3. Edit distance
4. Review elementary graph theory
5. Manhattan Tourist problem
6. Longest/shortest paths in DAGs

Reading:

- Jones and Pevzner. Chapters 2.7-2.9 and 6.1-6.4
- Lecture notes

Sources

- CS 362 by Layla Oesper (Carleton College)
- CS 1810 by Ben Raphael (Brown/Princeton University)
- An Introduction to Bioinformatics Algorithms book (Jones and Pevzner)
- <http://bioalgorithms.info/>