Selected Topics on Dynamic Programming

Dr. 何明昕，He Mingxin, Max

program07 @ yeah.net

with Subject：(A2E|A3E|A4E|A9|A10|A11) + *ID* + *Name*: *TOPIC*

Sakai: CS102A_MX_fall2020_

计算机程序设计基础A

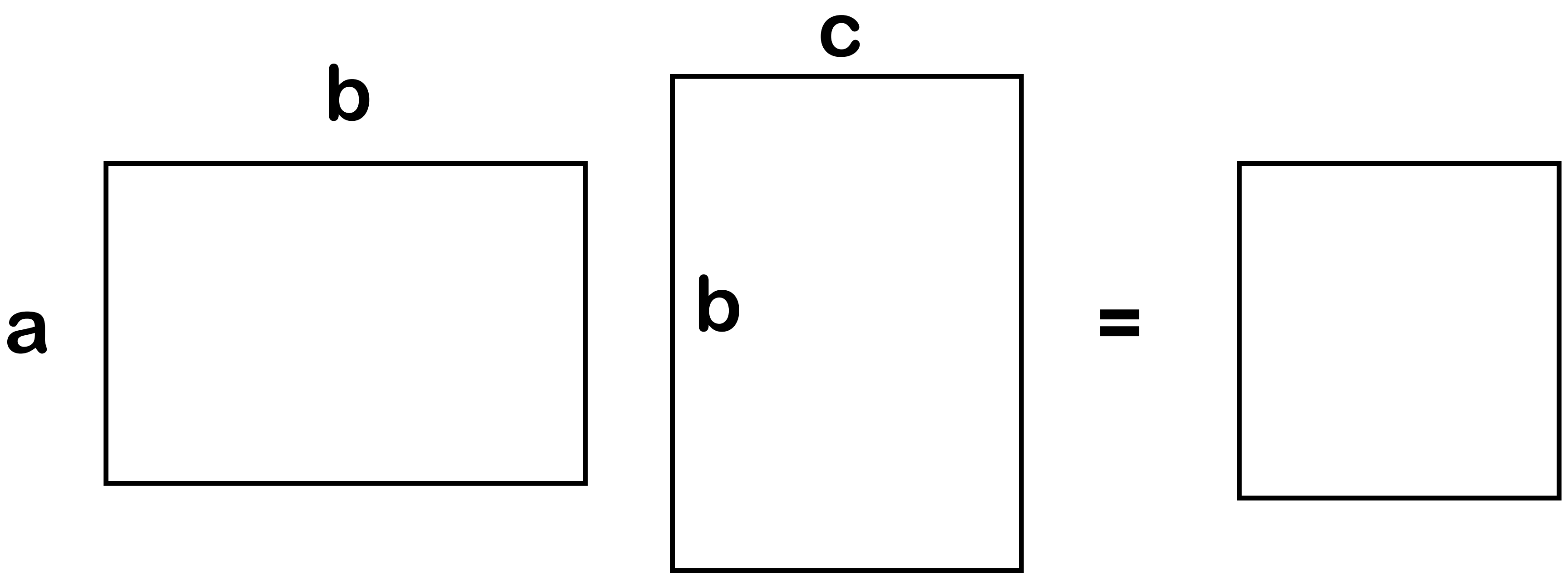
Introduction to Computer Programming A

Contents

- Matrix Chain Multiplication
- Coins Change Problem
- Manhattan Tourist Problem
- Power of DNA Sequence Comparison
- Edit Distance and Alignments
- Longest Common Subsequences

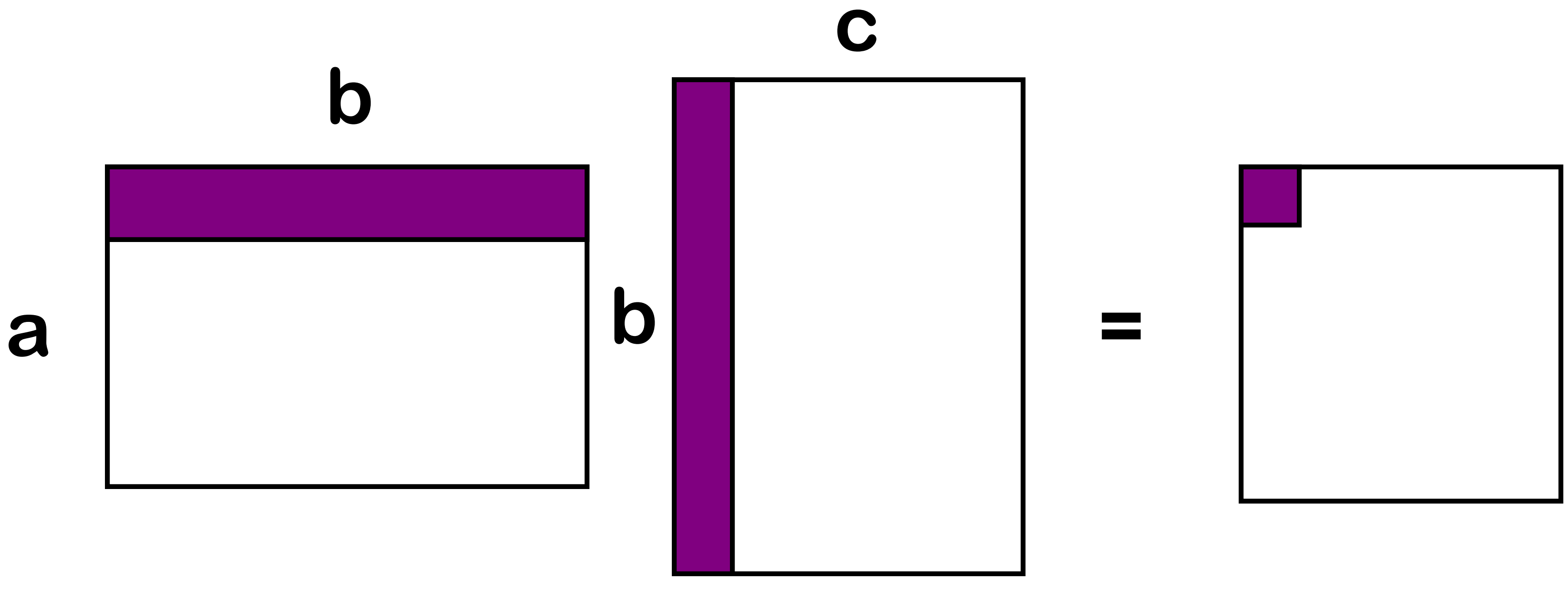
Matrix Chain Multiplication

matrix multiplication



A = a x b matrix
B = b x c matrix

Multiplying the Matrix



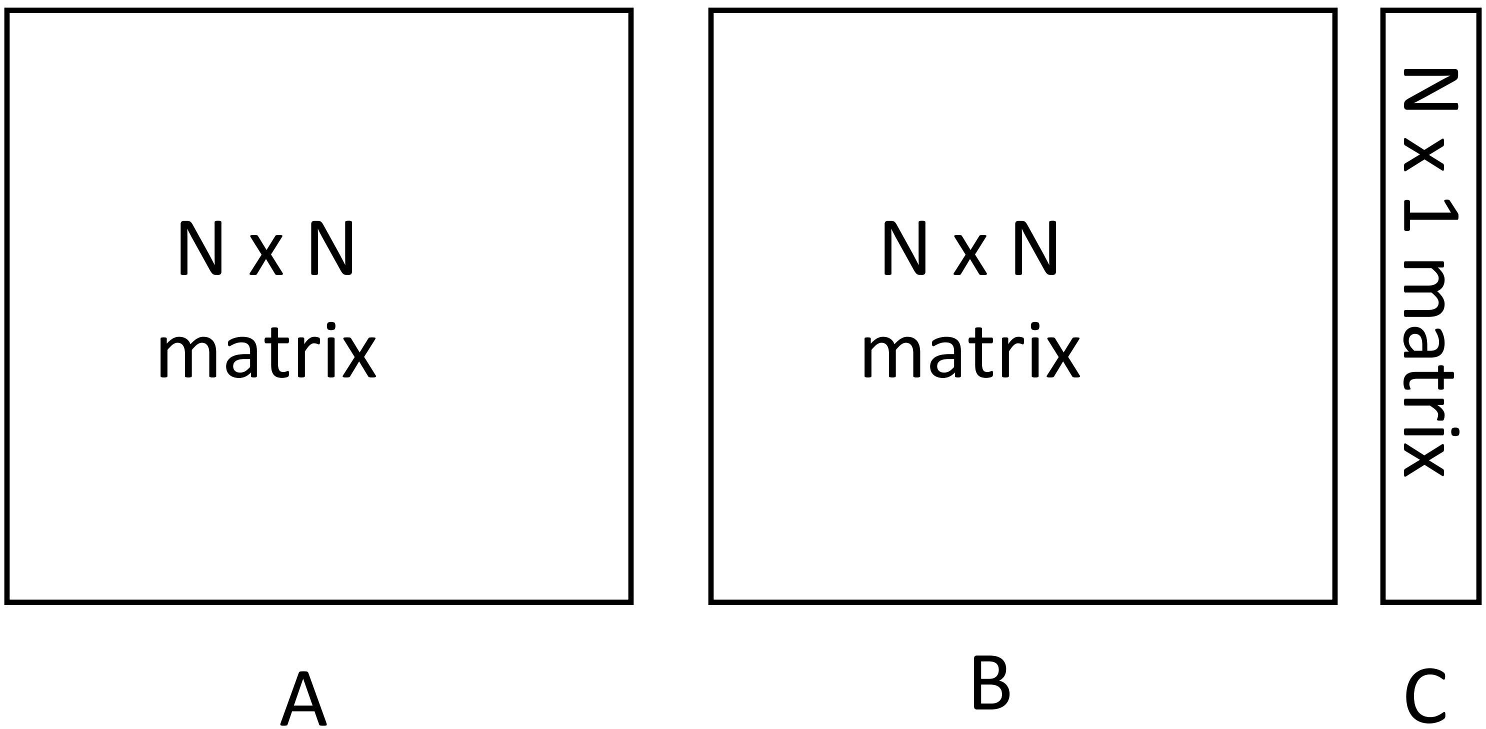
Time used = $\Theta(abc)$

Naïve Method

```
for (i = 1; i <= a; i++) {
  for (j = 1; i <= c; j++) {
     sum = 0;
     for (k = 1; k <= b; k++) {
       sum += A[i][k] * B[k][j];
     }
     C[i][j] = sum;
  }
}
```

O(abc)

Matrix Chain Multiplication



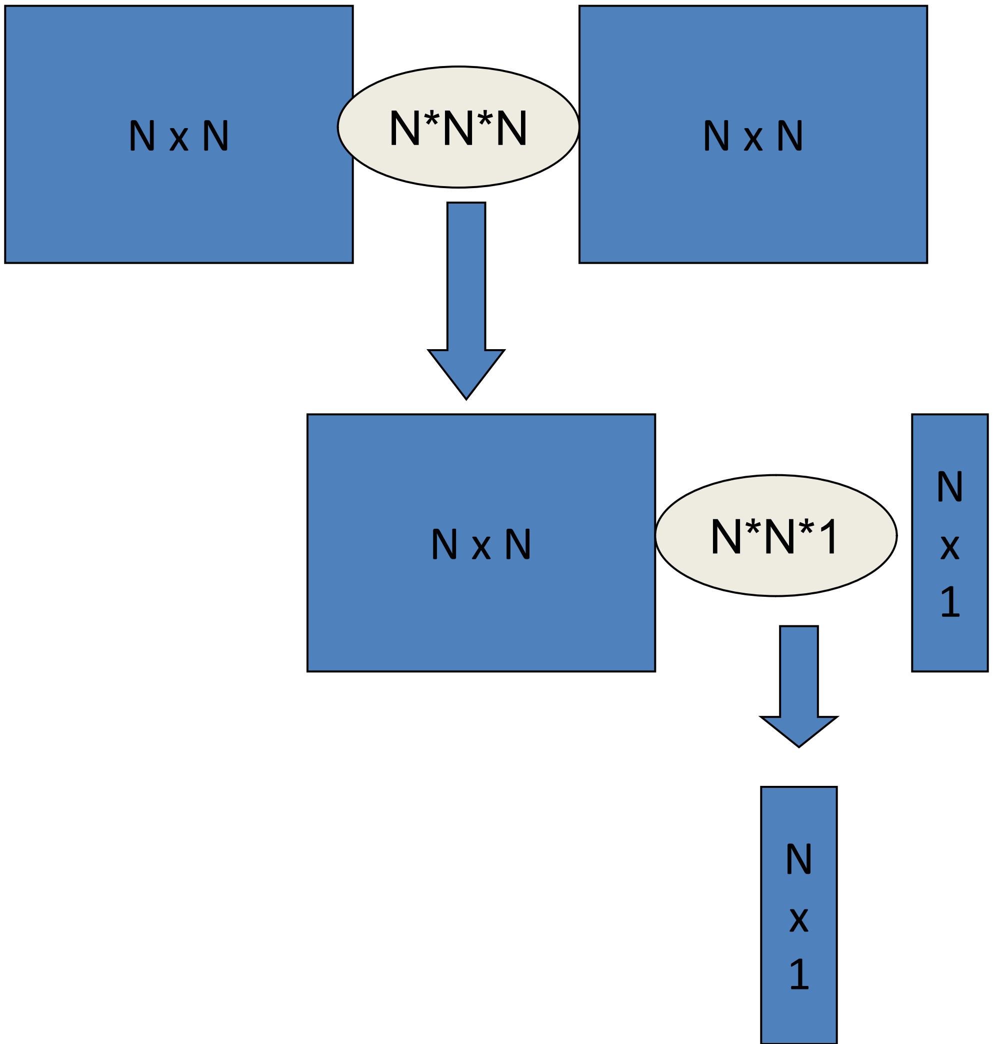
How to compute ABC ?

Matrix Multiplication

- ABC = (AB)C = A(BC)
- (AB)C differs from A(BC)?
  - Same result, different efficiency

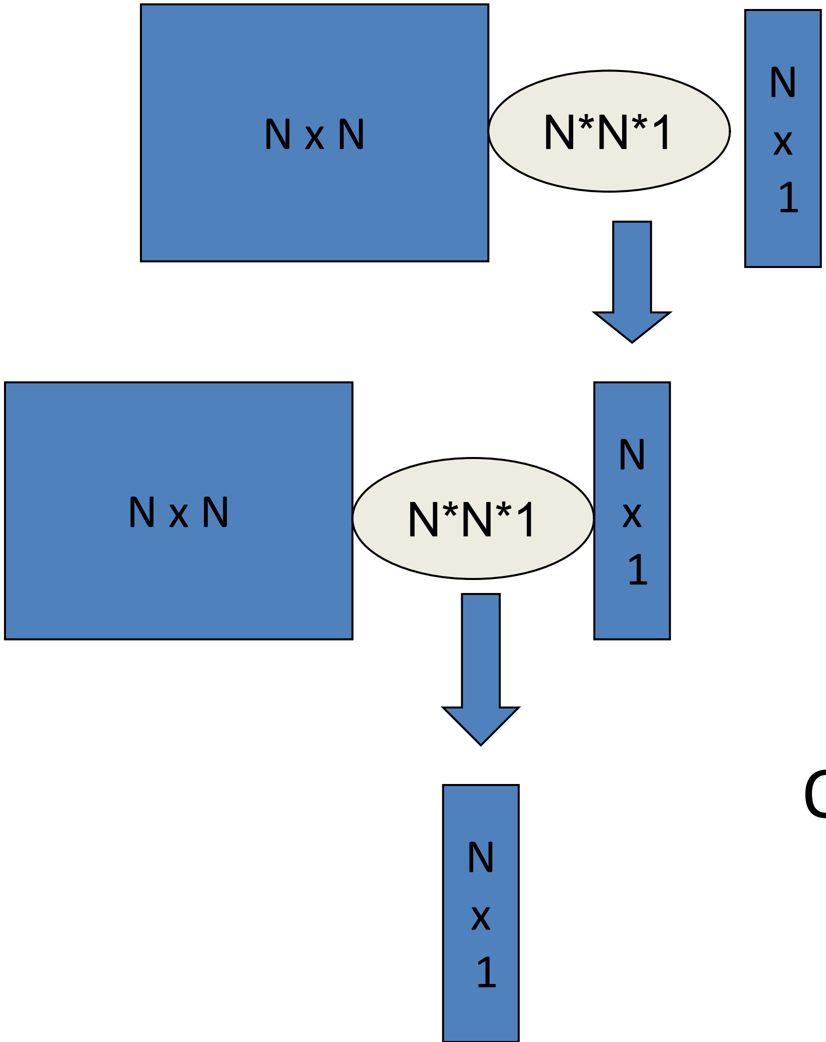
- What is the cost of (AB)C?
- What is the cost of A(BC)?

(AB)C



$O(N^3) + O(N^2)$

A(BC)



$O(N^2) + O(N^2)$

The Problem

- Input: $a_1, a_2, a_3, \ldots., a_n$
  - n-1 matrices of sizes
  - $a_1 \times a_2$ B_1
  - $a_2 \times a_3$ B_2
  - $a_3 \times a_4$ B_3
  - ....
  - $a_{n-1} \times a_n$ B_{n-1}
- Output
  - The order of multiplication
  - How to parenthesize the chain

Example

INPUT

- $a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$
- $10 \times 5 \times 1 \times 5 \times 10 \times 2$

$B_1 \quad B_2 \quad B_3 \quad B_4 \quad B_5$

Possible Output

$((B_1B_2)(B_3B_4))B_5$

$(B_1B_2)((B_3B_4)B_5)$

$(B_1((B_2B_3)B_4))B_5$

And much more…

Consider the Output

What do

$(B_1B_2)((B_3B_4)B_5)$

$(B_1B_2)(B_3(B_4B_5))$

have in common?

What do

$((B_1B_2)(B_3B_4))B_5$

$(((B_1B_2)B_3)B_4))B_5$

have in common?

Solving $B_1\ B_2\ B_3\ B_4\ \ldots\ B_{n-1}$

Min cost of

$B_1\ (B_2\ B_3\ B_4\ \ldots\ B_{n-1})$
$(B_1\ B_2)\ (B_3\ B_4\ \ldots\ B_{n-1})$
$(B_1\ B_2\ B_3)\ (B_4\ \ldots\ B_{n-1})$
…
$(B_1\ B_2\ B_3\ B_4\ \ldots)\ B_{n-1}$

Solving $B_1\ B_2\ B_3\ B_4 \ldots B_{n-1}$

Min cost of

$B_1\ (B_2\ B_3\ B_4 \ldots B_{n-1})$ Sub problem

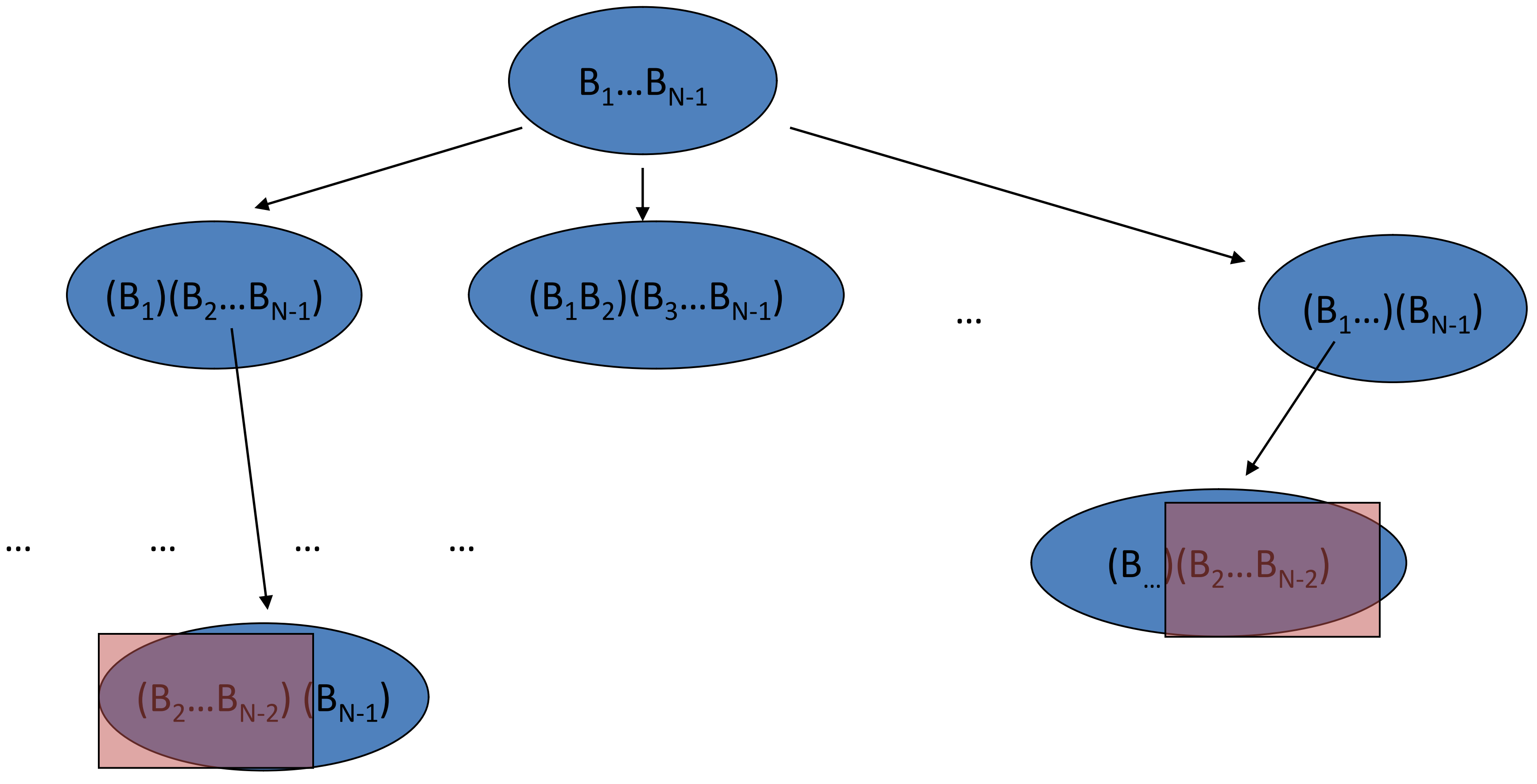
$(B_1\ B_2)\ (B_3\ B_4 \ldots B_{n-1})$ Sub problem Sub problem

$(B_1\ B_2\ B_3)\ (B_4 \ldots B_{n-1})$ Sub problem Sub problem

…

$(B_1\ B_2\ B_3\ B_4 \ldots)\ B_{n-1}$ Sub problem

Matrix Chain Multiplication



Deriving the Recurrent

- mcm(l,r)
  - The least cost to multiply $B_l \ldots B_r$
- The solution is mcm(1,n-1)

The Recurrence

- Initial Case
  - mcm(x,x) = 0
  - mcm(x,x+1) = a[x] * a[x+1] * a[x+2]

The Recurrence

`mcm(l,r)`
`= min of`

$$\texttt{mcm(l,r)} = \min \text{ of } \begin{cases} \text{min cost of } B_l\,(B_{l+1}\; B_{l+2}\; B_{l+3}\; \ldots\; B_r\,) & + a_l \bullet a_{l+1} \bullet a_{r+1} \\ \text{min cost of } (B_l\; B_{l+1})\,(B_{l+2}\; B_{l+3}\; \ldots\; B_r\,) & + a_l \bullet a_{l+2} \bullet a_{r+1} \\ \text{min cost of } (B_l\; B_{l+1}\; B_{l+2})\,(B_{l+3}\; \ldots\; B_r\,) & + a_l \bullet a_{l+3} \bullet a_{r+1} \\ \ldots & \\ \text{min cost of } (B_l\; B_{l+1}\; B_{l+2}\; B_{l+3}\; \ldots)\,B_r & + a_l \bullet a_r \bullet a_{r+1} \end{cases}$$

The Recurrence

mcm(l,r)

= min of

- 0 + mcm(l+1,r) + $a_l \bullet a_{l+1} \bullet a_{r+1}$
- mcm(l,l+1) + mcm(l+2,r) + $a_l \bullet a_{l+2} \bullet a_{r+1}$
- mcm(l,l+2) + mcm(l+3,r) + $a_l \bullet a_{l+3} \bullet a_{r+1}$
- …
- mcm(l,r-1) + 0 + $a_l \bullet a_r \bullet a_{r+1}$

Matrix Chain Multiplication

```
int mcm(int l, int r) {
   if (l < r) {
      minCost = MAX_INT;
      for (int i = l; i < r; i++) {
         my_cost = mcm(l,i) + mcm(i+1,r) + (a[l] * a[i+1] * a[r+1]);
         minCost = min(my_cost, minCost);
      }
      return minCost;
   } else {
      return 0;
   }
}
```

Using bottom-up DP

- Design the table

- `M[i,j]` = the best solution (min cost) for multiplying $B_i \ldots B_j$

- The solution is at `M[1,n-1]`

What is `M[i,j]`?

- Trivial case
  - What is m[x,x] ?
  - No multiplication, m[x,x] = 0

What is `M[i,j]`?

- Simple case
  - What is m[x,x+1] ?
  - $B_x B_{x+1}$
  - Only one solution = $a_x * a_{x+1} * a_{x+2}$

What is `M[i,j]`?

- General case
  - What is m[x,x+k] ?
  - $B_x B_{x+1} B_{x+2} \ldots B_{x+k}$

$$
\text{min of}
\begin{cases}
B_x\ (B_{x+1}\ B_{x+2}\ B_{x+3}\ \ldots\ B_{x\text{-}k}\ ) & + a_x \bullet a_{x+1} \bullet a_{x+k+1} \\
(B_x\ B_{x+1})\ (B_{x+2}\ B_{x+3}\ \ldots\ B_{x+k}\ ) & + a_x \bullet a_{x+2} \bullet a_{x+k+1} \\
(B_x\ B_{x+1}\ B_{x+2})\ (B_{x+3}\ \ldots\ B_{x+k}\ ) & + a_x \bullet a_{x+3} \bullet a_{x+k+1} \\
\ldots & \\
(B_x\ B_{x+1}\ B_{x+2}\ B_{x+3}\ \ldots)\ B_{x+k} & + a_x \bullet a_{x+k} \bullet a_{x+k+1}
\end{cases}
$$

Filling the Table

M[1,1]

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

M[1,6]
solution

Filling the Table

Trivial case

	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Filling the Table

Arbitrary case

	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Filling the Table

Arbitrary case

	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Plus $a_1 \bullet a_2 \bullet a_6$

Filling the Table

Arbitrary case

	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Plus $a_1 \bullet a_3 \bullet a_6$

Filling the Table

Arbitrary case

	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Plus $a_1 \bullet a_4 \bullet a_6$

Filling the Table

Arbitrary case

	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Plus $a_1 \bullet a_5 \bullet a_6$

Filling the Table

	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5						
6						0

Filling the Table

	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Example

- $a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$
- 10 x 5 x 1 x 5 x 10 x 2

$B_1 \quad B_2 \quad B_3 \quad B_4 \quad B_5$

Example

a_1 a_2 a_3 a_4 a_5 a_6
10 x 5 x 1 x 5 x 10 x 2

	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$\boxed{10 \times 5 \times 1} \times 5 \times 10 \times 2$

	1	2	3	4	5
1	0	50			
2		0			
3			0		
4				0	
5					0

Example

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 10 \times & \boxed{5 \times 1 \times 5} & & & \times 10 \times & 2 \end{array}$$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0		
4				0	
5					0

Example

a_1 a_2 a_3 a_4 a_5 a_6

$10 \times 5 \times \boxed{1 \times 5 \times 10} \times 2$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0	50	
4				0	
5					0

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$10 \times 5 \times 1 \times \boxed{5 \times 10 \times 2}$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0	50	
4				0	100
5					0

Example

$$\begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 10 \times & 5 \times & 1 \times & 5 \times & 10 \times & 2 \end{array}$$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0	50	
4				0	100
5					0

Example

a_1 a_2 a_3 a_4 a_5 a_6

$\boxed{10} \times \boxed{5} \times 1 \times \boxed{5} \times 10 \times 2$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0	50	
4				0	100
5					0

Option 1 = 0 + 25 + 10 x 5 x 5 = 275

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$\boxed{10} \times 5 \times \boxed{1} \times \boxed{5} \times 10 \times 2$

	1	2	3	4	5
1	0	50			
2		0	25		
3			0	50	
4				0	100
5					0

Option 2 = 50 + 0 + 10 x 1 x 5 = 100 minimal

Example

a_1 a_2 a_3 a_4 a_5 a_6
10 x 5 x 1 x 5 x 10 x 2

	1	2	3	4	5
1	0	50	100 (2)		
2		0	25		
3			0	50	
4				0	100
5					0

(2) means that the minimal solution is by dividing at B_2

Option 2 = 50 + 0 + 10 x 1 x 5 = 100 minimal

Example

a_1 a_2 a_3 a_4 a_5 a_6

$10 \times \boxed{5} \times \boxed{1} \times 5 \times \boxed{10} \times 2$

	1	2	3	4	5
1	0	50	100 (2)		
2		0	25		
3			0	50	
4				0	100
5					0

Option 1 = 0+ 50 + 5 x 1 x 10 = 100

Example

a_1 a_2 a_3 a_4 a_5 a_6
10 x $\boxed{5}$ x 1 x $\boxed{5}$ x $\boxed{10}$ x 2

	1	2	3	4	5
1	0	50	100 (2)		
2		0	25		
3			0	50	
4				0	100
5					0

Option 1 = 25+ 0 + 5 x 5 x 10 = 275

Example

a_1 a_2 a_3 a_4 a_5 a_6
10 x 5 x 1 x 5 x 10 x 2

	1	2	3	4	5
1	0	50	100 (2)		
2		0	25	100 (2)	
3			0	50	
4				0	100
5					0

Option 1 is better

Example

a_1 a_2 a_3 a_4 a_5 a_6
10 x 5 x 1 x 5 x 10 x 2

	1	2	3	4	5
1	0	50	100 (2)		
2		0	25	100 (2)	
3			0	50	70 (4)
4				0	100
5					0

Example

a_1 a_2 a_3 a_4 a_5 a_6
10 x 5 x 1 x 5 x 10 x 2

	1	2	3	4	5
1	0	50	100 (2)	200 (2)	
2		0	25	100 (2)	
3			0	50	70 (4)
4				0	100
5					0

Example

a_1 a_2 a_3 a_4 a_5 a_6

10 x 5 x 1 x 5 x 10 x 2

	1	2	3	4	5
1	0	50	100 (2)	200 (2)	
2		0	25	100 (2)	80 (2)
3			0	50	70 (4)
4				0	100
5					0

Example

a_1 a_2 a_3 a_4 a_5 a_6
10 x 5 x 1 x 5 x 10 x 2

	1	2	3	4	5
1	0	50	100 (2)	200 (2)	140 (2)
2		0	25	100 (2)	80 (2)
3			0	50	70 (4)
4				0	100
5					0

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$10 \times 5 \times 1 \times 5 \times 10 \times 2$

$B_1 \quad B_2 \quad B_3 \quad B_4 \quad B_5$

	1	**2**	**3**	**4**	**5**
1	0	50	100 (2)	200 (2)	140 (2)
2		0	25	100 (2)	80 (2)
3			0	50	70 (4)
4				0	100
5					0

$(B_1 \cdot B_2) \cdot ((B_3 \cdot B_4) \cdot B_5)$

50 20 50 20

Example

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

$10 \times 5 \times 1 \times 5 \times 10 \times 2$

$B_1 \quad B_2 \quad B_3 \quad B_4 \quad B_5$

	1	2	3	4	5
1	0	50	100 (2)	200 (2)	**140** (2)
2		0	25	100 (2)	80 (2)
3			0	50	70 (4)
4				0	100
5					0

$(B_1 \cdot B_2) \cdot ((B_3 \cdot B_4) \cdot B_5)$

50 20 50 20

70

Solution : 140

The Coins Change Problem

- ***Intention:*** Changing an amount of money M into the smallest number of coins from denominations c = (c1, c2, . . . , cd).
- Using greedy algorithm to solve this some times gave out incorrect results.
- Brute-Force algorithm though correct was very slow.
- Good idea is to use Dynamic Programming.

Illustration of A Recursive Approach.

- Suppose you need to make change for 77 cents and the only coin denominations available are 1, 3, and 7 cents.

$$bestNumCoins_M = \min \begin{cases} bestNumCoins_{M-1} + 1 \\ bestNumCoins_{M-3} + 1 \\ bestNumCoins_{M-7} + 1 \end{cases}$$

- Best combination for 77 − 1 = 76 cents, plus a 1-cent coin;

A More General Solution

In the more general case of d denominations $\mathbf{c} = (c_1, \ldots, c_d)$:

$$bestNumCoins_M = \min \begin{cases} bestNumCoins_{M-c_1} + 1 \\ bestNumCoins_{M-c_2} + 1 \\ \vdots \\ bestNumCoins_{M-c_d} + 1 \end{cases}$$

Algorithm

```
RecursiveChange(M, c, d)
  if M = 0
      return 0
  bestNumCoins ← ∞
  for i ← 1 to d
      if M ≥ c_i
          numCoins ← RecursiveChange(M − c_i, c, d)
          if numCoins + 1 < bestNumCoins
              bestNumCoins ← numCoins + 1
  return bestNumCoins
```

What Happens in Recursion

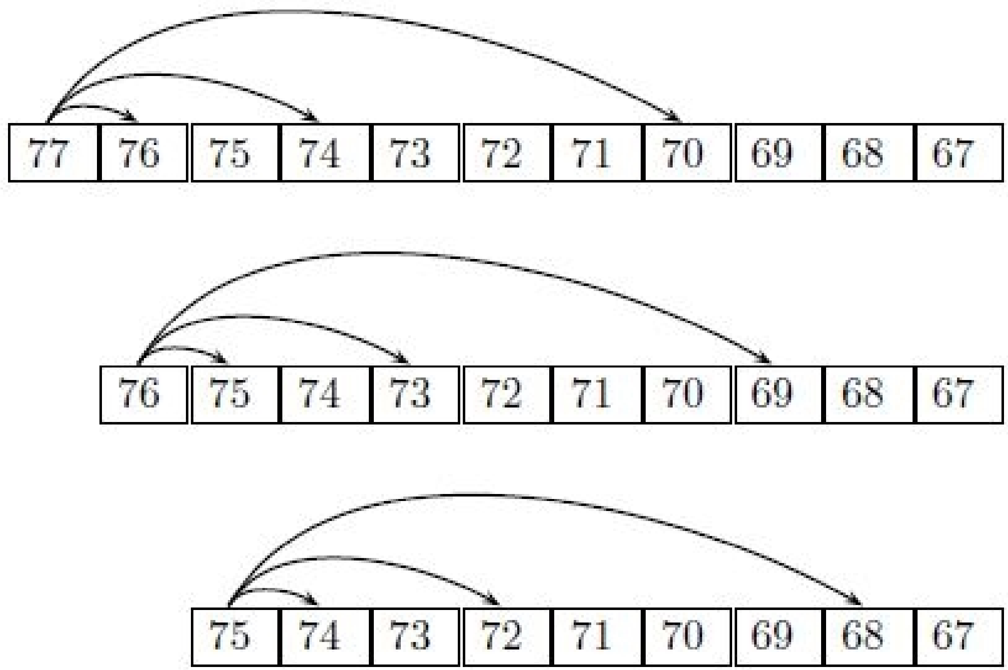


Figure 6.1 The relationships between optimal solutions in the Change problem. The smallest number of coins for 77 cents depends on the smallest number of coins for 76, 74, and 70 cents; the smallest number of coins for 76 cents depends on the smallest number of coins for 75, 73, and 69 cents, and so on.

Disaster

- **Turns impractical**: This algorithm needs a very big fix as it is going to consume a lot of time as the problem size and the number of denominations increase.
- Under these circumstances we are motivated to use Dynamic Programming .

Dynamic Programming

- This allows us to leverage previously computed solutions to form solutions to larger problems and avoid all this re computation.
- All we really need to do is use the fact that the solution for M relies on solutions for M – c1, M – c2, and so on, and then reverse the order in which we solve the problem.

Algorithm

```
DPChange(M, c, d)
  bestNumCoins_0 ← 0
  for m ← 1 to M
      bestNumCoins_m ← ∞
      for i ← 1 to d
          if m ≥ c_i
              if bestNumCoins_{m−c_i} + 1 < bestNumCoins_m
                  bestNumCoins_m ← bestNumCoins_{m−c_i} + 1
  return bestNumCoins_M
```

Complexity Has Improved

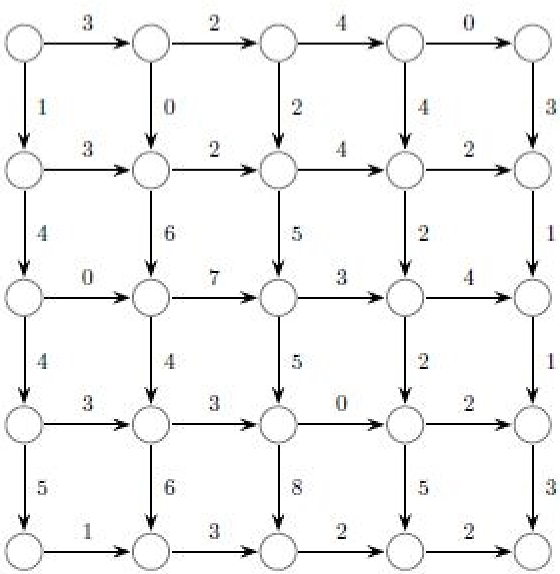
- Recursive Approach Complexity was $O(M^d)$ Does not appear to be any easy way to remedy this situation.
- Yet the DPCHANGE algorithm provides a simple $O(Md)$ solution.

The Manhattan Tourist Problem

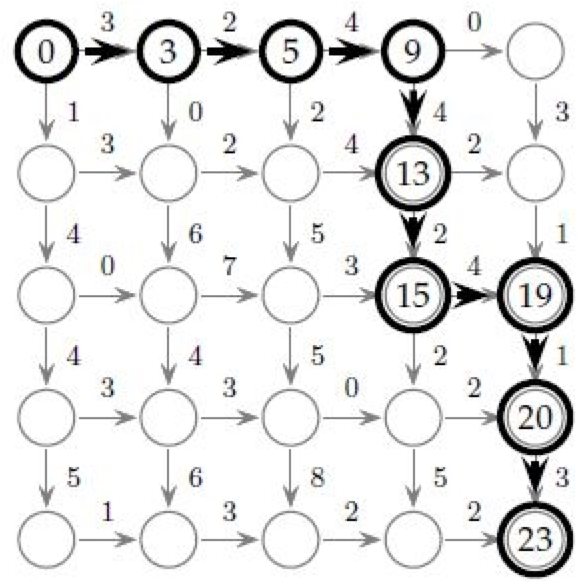
Intention: Find a longest path in a weighted grid.
Input: A weighted grid G with two distinguished vertices: a source and a sink.
Output: A longest path in G from source to sink.

Illustration

INPUT



SOLUTION

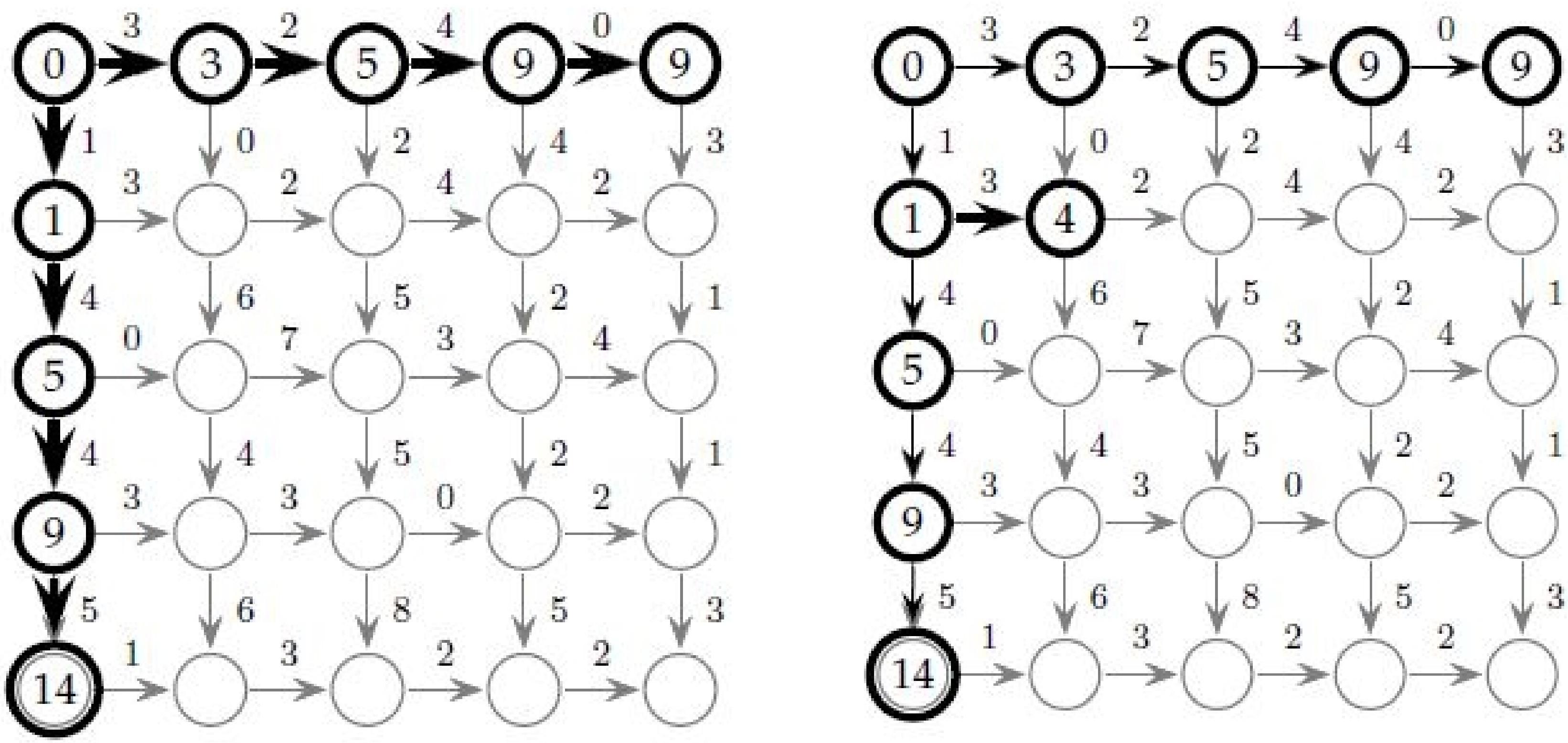


Using Dynamic Programming

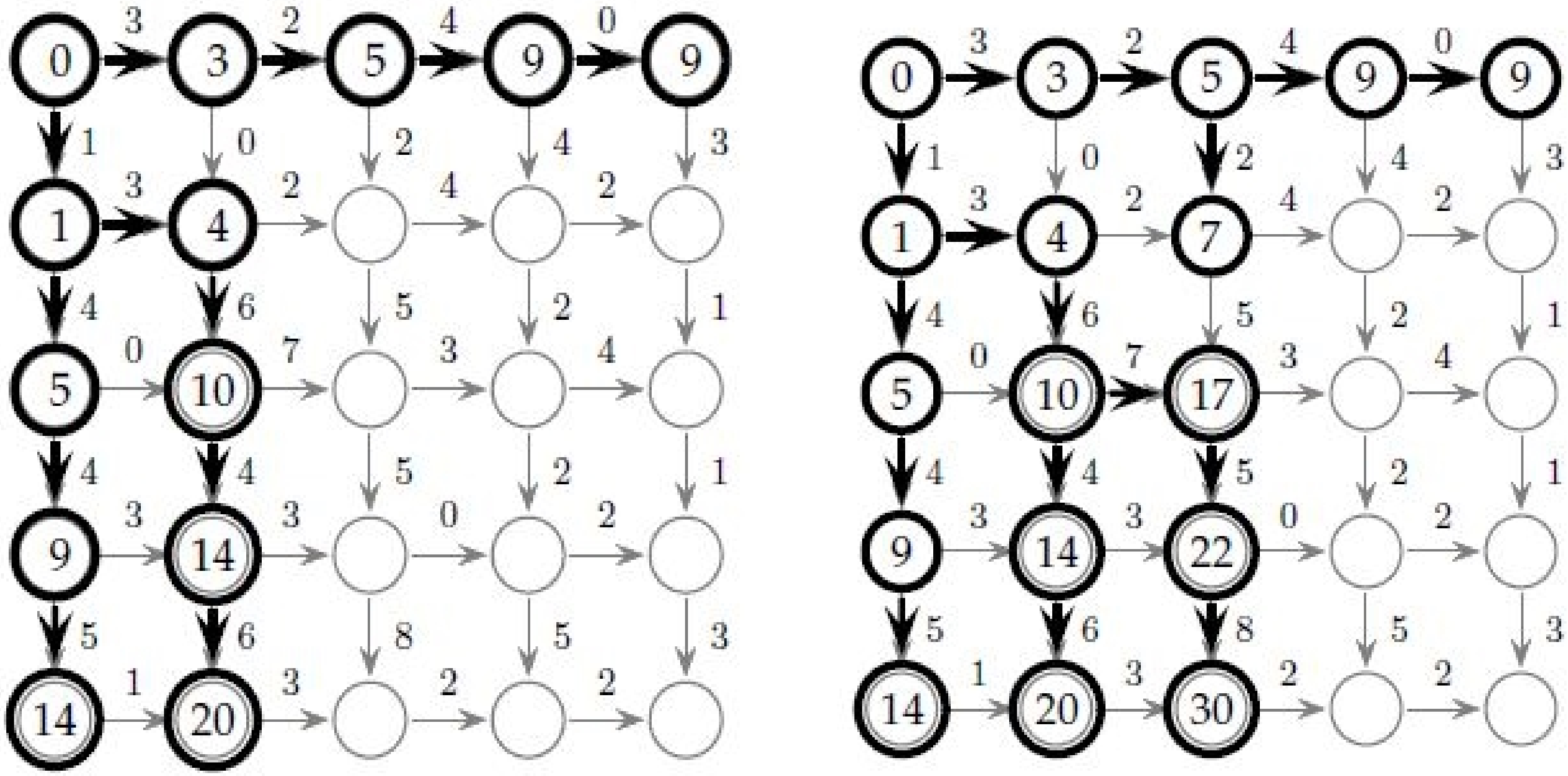
We solve a more general problem: find the longest path from source to an arbitrary vertex (i, j)

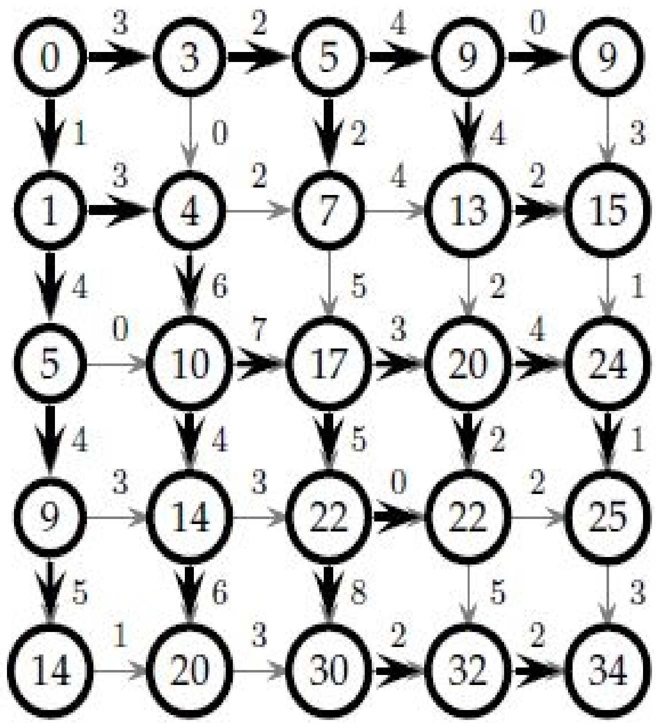
MANHATTANTOURIST($\overset{\downarrow}{\mathbf{w}}, \overset{\rightarrow}{\mathbf{w}}, n, m$)

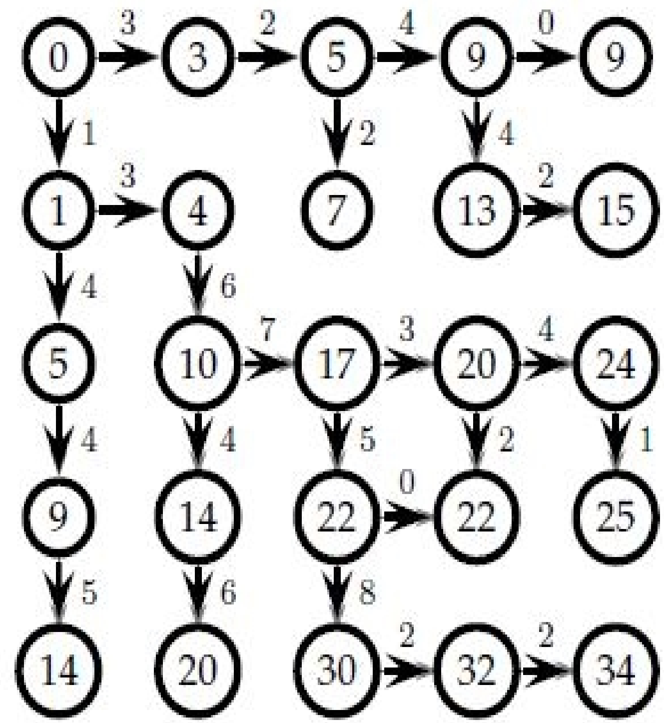
$s_{0,0} \leftarrow 0$
for $i \leftarrow 1$ **to** n
    $s_{i,0} \leftarrow s_{i-1,0} + \overset{\downarrow}{w}_{i,0}$
for $j \leftarrow 1$ **to** m
    $s_{0,j} \leftarrow s_{0,j-1} + \vec{w}_{0,j}$
for $i \leftarrow 1$ **to** n
    for $j \leftarrow 1$ **to** m
        $s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} + \overset{\downarrow}{w}_{i,j} \\ s_{i,j-1} + \vec{w}_{i,j} \end{cases}$
return $s_{n,m}$



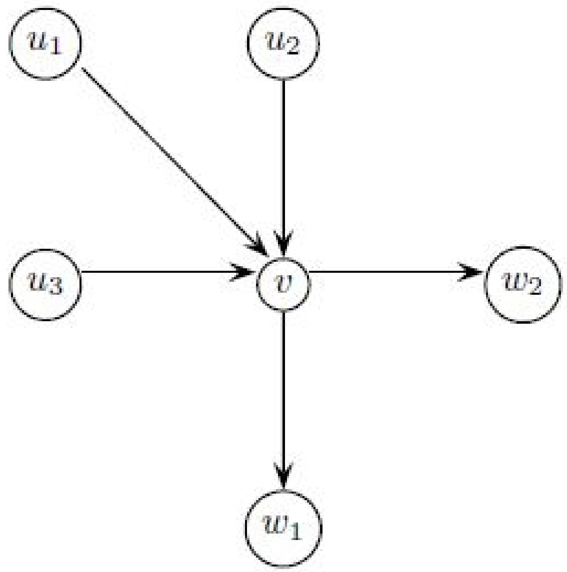
0 3 5 9 9
1 5 9 14
3 2 4 0
1 0 2 4 3
3 2 4 2
4 6 5 2 1
0 7 3 4
4 4 5 2 1
3 3 0 2
5 6 8 5 3
1 3 2 2
4




0 3 5 9 9
1 4 7 13 15
5 10 17 20 24
9 14 22 22 25
14 20 30 32 34


0 3 5 9 9
1 4 7 13 15
5 10 17 20 24
9 14 22 22 25
14 20 30 32 34

Longest Path Computation



Suppose a vertex v has indegree 3, and the set of predecessors of v is $\{u_1, u_2, u_3\}$ (figure 6.6). The longest path to v can be computed as follows:

$$s_v = \max \begin{cases} s_{u_1} + \text{ weight of edge from } u_1 \text{ to } v \\ s_{u_2} + \text{ weight of edge from } u_2 \text{ to } v \\ s_{u_3} + \text{ weight of edge from } u_3 \text{ to } v \end{cases}$$

Power of DNA Sequence Comparison

- Cancer-causing gene matched a normal gene involved in growth and development called platelet-derived growth factor (PDGF).
- A good gene doing the right thing at the wrong time.
- DNA or RNA sequence Detection

Edit Distance and Alignments

- **Mutation in DNA** is an evolutionary process: DNA replication errors cause substitutions, insertions, and deletions of nucleotides, leading to “edited” DNA texts.
- Difficult to find the ith symbol in one DNA sequence corresponds to the ith symbol in the other.

Edit Distance

Edit distance between two strings is the minimum number of editing operations needed to transform one string into another,

Edit Operations

- Insertion of a symbol
- Deletion of a symbol
- Substitution of one symbol for another

How Do We Do It?

TGCATAT

↓ delete last T

TGCATA

↓ delete last A

TGCAT

↓ insert A at the front

ATGCAT

↓ substitute C for G in the third position

ATCCAT

↓ insert a G before the last A

ATCCGAT

Ordering of DNA Strings

A	T	-	G	T	T	A	T	-
A	T	C	G	T	-	A	-	C

Convention
Match: Same letter in each row
Mismatch: Different letter in each row

Indels
Insertions: Columns containing a space in the top row
Deletions: Columns containing a space in the bottom row

Numbering the Sequences

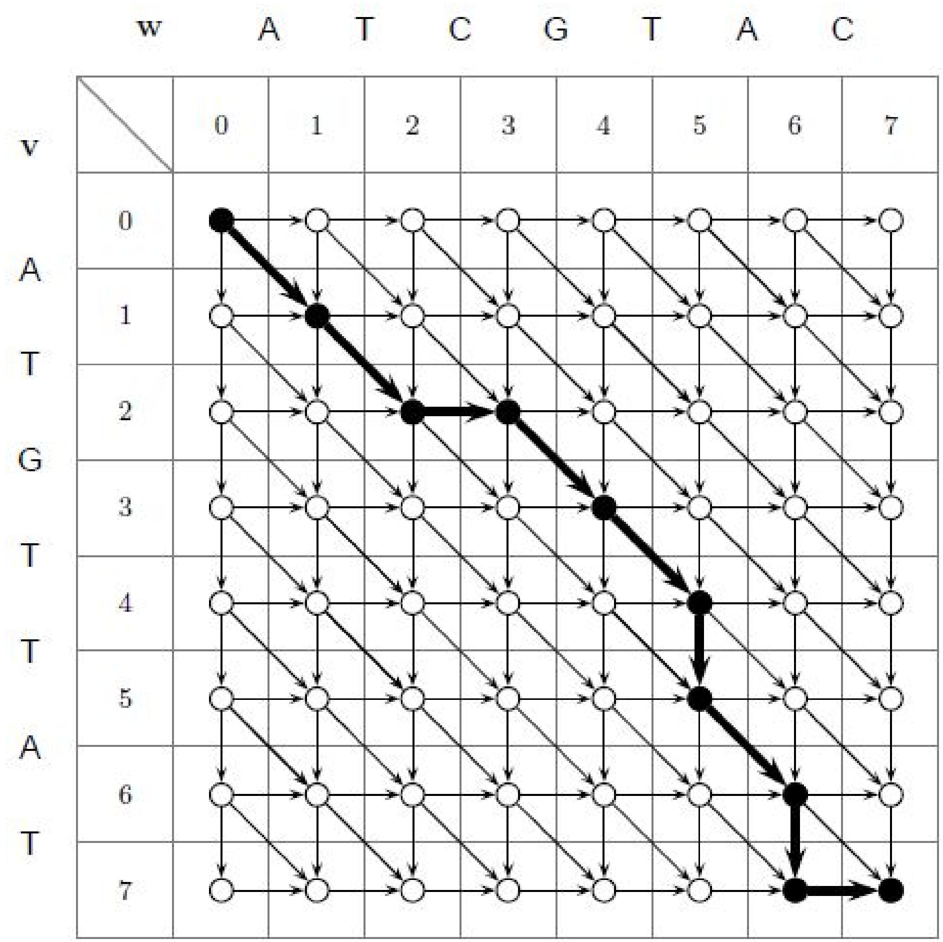
Numbering the DNA Strings v and w

$$
\begin{array}{cccccccccccc}
 & & 0 & 1 & 2 & 2 & 3 & 4 & 5 & 6 & 7 & 7 \\
\mathrm{v} & = & & A & T & - & G & T & T & A & T & - \\
 & & & | & | & & | & | & & | & & \\
\mathrm{w} & = & & A & T & C & G & T & - & A & - & C \\
 & & 0 & 1 & 2 & 3 & 4 & 5 & 5 & 6 & 6 & 7
\end{array}
$$

Resulting Alignment Matrix

$$
\begin{pmatrix} 0 \\ 0 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\begin{pmatrix} 2 \\ 2 \end{pmatrix}\begin{pmatrix} 2 \\ 3 \end{pmatrix}\begin{pmatrix} 3 \\ 4 \end{pmatrix}\begin{pmatrix} 4 \\ 5 \end{pmatrix}\begin{pmatrix} 5 \\ 5 \end{pmatrix}\begin{pmatrix} 6 \\ 6 \end{pmatrix}\begin{pmatrix} 7 \\ 6 \end{pmatrix}\begin{pmatrix} 7 \\ 7 \end{pmatrix}
$$

Alignment Grid



↘ ↘ → ↘ ↘ ↓ ↘ ↓ →
A T - G T T A T -
A T C G T - A - C

Longest Common Subsequences

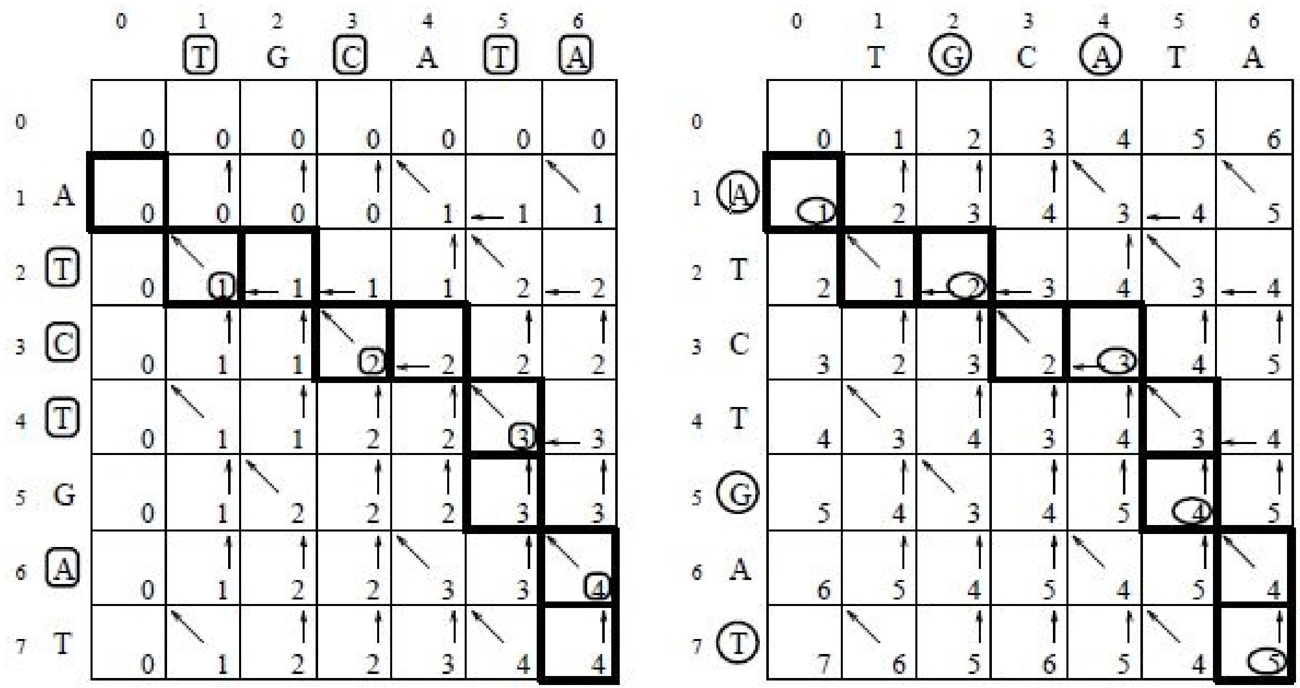
- A subsequence of a string v is simply an (ordered) sequence of characters (not necessarily consecutive) from v.
- Common subsequence of strings $v = v_1 \ldots v_n$ and $w = w_1 \ldots w_m$ as a sequence of positions in v, $1 \le i_1 < i_2 < \cdots < i_k \le n$
- A sequence of positions in w, $1 \le j_1 < j_2 < \cdots < j_k \le m$

Edit Distance btw V and W

- Under the assumption that only insertions and deletions are allowed—is

  $d(v , w) = n + m - 2s(v, w)$
- $s(v, w)$ is the longest common subsequence.
- Corresponds to the minimum number of insertions and deletions to transform v to w.

What did we do?



Computing similarity s(V,W)=4
V and W have a subsequence TCTA in common

Computing distance d(V,W)=5
V can be transformed into W by deleting A,G,T and inserting G,A

Alignment:

```
A T - C - T G A T
- T G C A T - A -
```

A shortest sequence of two insertions and three deletions transformed v to w.

Recursive LCS Formulation

- s (i,0) = s (0,j) = 0 for all 1 <= i <= n and 1<= j<= m. One can see that s(i,j) satisfies the following recurrence:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, & \text{if } v_i = w_j \end{cases}$$

The first term corresponds to the case when vi is not present in the LCS of the i-prefix of v and j-prefix of w (this is a deletion of vi); the second term corresponds to the case when wj is not present in this LCS (this is an insertion of wj); and the third term corresponds to the case when both vi and wj are present in the LCS (vi matches wj).

LCS ALGORITHM

LCS(**v**, **w**)

for $i \leftarrow 0$ **to** n

  $s_{i,0} \leftarrow 0$

for $j \leftarrow 1$ **to** m

  $s_{0,j} \leftarrow 0$

for $i \leftarrow 1$ **to** n

  for $j \leftarrow 1$ **to** m

$$s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, & \textbf{if } v_i = w_j \end{cases}$$

$$b_{i,j} \leftarrow \begin{cases} \text{“}\uparrow\text{”} & \textbf{if } s_{i,j} = s_{i-1,j} \\ \text{“}\leftarrow\text{”} & \textbf{if } s_{i,j} = s_{i,j-1} \\ \text{“}\nwarrow\text{”}, & \textbf{if } s_{i,j} = s_{i-1,j-1} + 1 \end{cases}$$

return $(s_{n,m}, \mathrm{b})$

PRINT LCS

```
PRINTLCS(b, v, i, j)
    if i = 0 or j = 0
        return
    if b_{i,j} = "↖"
        PRINTLCS(b, v, i − 1, j − 1)
        print v_i
    else
        if b_{i,j} = "↑"
            PRINTLCS(b, v, i − 1, j)
        else
            PRINTLCS(b, v, i, j − 1)
```