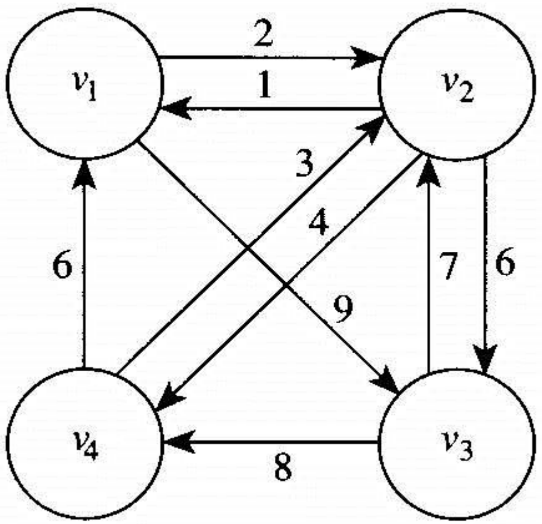
3.6 Traveling Salesperson Problem

- **Def. Tour (Hamiltonian circuit):**
  - **A path from a vertex to itself that passes through each of the other vertices exactly once.**
- **Def. Optimal tour (in a weighted digraph)**
  - **A tour of minimum length**
- **Def. TSP**
  - **Find an optimal tour starting at v_1**
- **Brute-force method**
  - **Consider all possible tours**
  - **(n-1)(n-2) … 1 = (n-1)!**

Figure 3.16 The optimal tour is $[v_1, v_3, v_4, v_2, v_1]$.



Principle of optimality applies?

$$\underbrace{v_1 \to v_k}_{\text{may not be shortest}} \underbrace{\to \cdots\cdots\cdots \to}_{\text{shortest}} v_1 \quad \text{optimal}$$

Figure 3.16 The optimal tour is $[v_1, v_3, v_4, v_2, v_1]$.

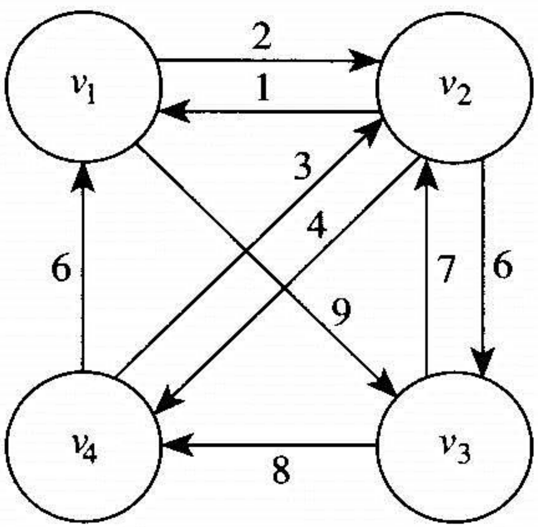


Figure 3.17 The adjacency matrix representation W of the graph in Figure 3.16.

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

Solution process (1/2)

- **W: weight(adjacency) matrix**

  $A \subseteq V$, $V =$ set of all the vertices

  $D[v_i][A]$: **length of a shortest path from v_i to v_1 passing through each vertex in A exactly once.**

  Want : $D[v_1][V-\{v_1\}]$

  - **Example 3.10 – See Fig. 3.16**

    $D[v_2][\{v_3\}] = length[v_2, v_3, v_1] = \infty$

    $D[v_2][\{v_3, v_4\}]$
    $= \min(length[v_2, v_3, v_4, v_1],\ length[v_2, v_4, v_3, v_1])$
    $= \min(20, \infty) = 20$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 2 & 9 & \infty \\ 1 & 0 & 6 & 4 \\ \infty & 7 & 0 & 8 \\ 6 & 3 & \infty & 0 \end{bmatrix} \end{array}$$

$$\boxed{D[v_1][V-\{v_1\}] = \min_{2 \le j \le n} (W[1][j] + D[v_j][V-\{v_1, v_j\}])}$$

Solution process (2/2)

- **In general** $v_i \notin A,\ i \neq 1$

$$D[v_i][A] = \min_{v_j \in A} (W[i][j] + D[v_j][A - \{v_j\}]) \text{ if } A \neq \emptyset$$
$$D[v_i][\emptyset] = W[i][1]$$

- **Computing order**

$|A| = 0,\ |A| = 1,\ \cdots$

Example 3.11 (1/2)

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 9 & \infty \\ 2 & 1 & 0 & 6 & 4 \\ 3 & \infty & 7 & 0 & 8 \\ 4 & 6 & 3 & \infty & 0 \end{array}$$

- **Determine an optimal tour for the graph in Fig. 3.17**
- **First consider the empty set:** $D[v_2][\emptyset]=1,\ D[v_3][\emptyset]=\infty,\ D[v_4][\emptyset]=6$
- **Next consider all sets containing one element:**

$$D[v_3][\{v_2\}] = \min_{v_j \in \{v_2\}} (W[3][j] + D[v_j][\{v_2\}-\{v_j\}])$$
$$= W[3][2] + D[v_2][\emptyset] = 7+1=8$$

- **Similarly,**

$$D[v_4][\{v_2\}] = 3+1=4$$
$$D[v_2][\{v_3\}] = 6+\infty=\infty$$
$$D[v_4][\{v_3\}] = \infty+\infty=\infty$$
$$D[v_2][\{v_4\}] = 4+6=10$$
$$D[v_3][\{v_4\}] = 8+6=14$$

Example 3.11 (2/2)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 2 & 9 & \infty \\ 1 & 0 & 6 & 4 \\ \infty & 7 & 0 & 8 \\ 6 & 3 & \infty & 0 \end{bmatrix} \end{array}$$

- **Next consider all sets containing two elements:**

$$D[v_4][\{v_2, v_3\}] = \min_{v_j \in \{v_2, v_3\}} (W[4][j] + D[v_j][\{v_2, v_3\} - \{v_j\}])$$

$$= \min(W[4][2] + D[v_2][\{v_3\}], W[4][3] + D[v_3][\{v_2\}]$$

$$= \min(3 + \infty, \infty + 8) = \infty$$

- **Similarly,** $D[v_3][\{v_2, v_4\}] = \min(7 + 10, 8 + 4) = 12$

  $D[v_2][\{v_3, v_4\}] = \min(6 + 14, 4 + \infty) = 20$

- **Finally, compute the length of an optimal tour:**

$$D[v_1][\{v_2, v_3, v_4\}] = \min_{v_j \in \{v_2, v_3, v_4\}} (W[1][j] + D[v_j][\{v_2, v_3, v_4\} - \{v_j\}])$$

$$= \min(W[1][2] + D[v_2][\{v_3, v_4\}], W[1][3] + D[v_3][\{v_2, v_4\}],$$

$$W[1][4] + D[v_4][\{v_2, v_3\}])$$

$$= \min(2 + 20, 9 + 12, \infty + \infty) = 21$$

Algorithm 3.11 DP Algorithm for TSP (1/3)

- **<u>Problem</u>: Determine an optimal tour in a weighted, directed graph. The weights are nonnegative numbers.**
- **<u>Inputs</u>: a weighted, directed graph, and *n*, the number of vertices in the graph. The graph is represented by a 2D array *W*, which has both its rows and columns indexed from 1 to *n*, where *W*[*i*][*j*] is the weight on the edge from *i*-th vertex to the *j*-th vertex.**
- **<u>Outputs</u>: a variable *minlength*, whose value is the length of an optimal tour, and a 2D array *P* from which an optimal tour can be constructed. *P* has its rows indexed from 1 to *n* and its columns indexed by all subsets of *V* – {v_1}. *P*[*i*][*A*] is the index of the first vertex after v_i on a shortest path from v_i to v_1 that passes through all the vertices in *A* exactly once.**

Algorithm 3.11 DP Algorithm for TSP (2/3)

```
void travel ( int n, const number W[ ],
                    index P[ ][ ], number& minlength)
{
   index i, j, k;
   number D[1..n][subset of V - {v_1}];
   for (i = 2; i  <= n; i++)
      D[i][Ø ] = W[i][1];
```

$$D[v_i][A] = \min_{v_j \in A} (\mathrm{W}[i][j] + D[v_j][A - \{v_j\}]) \text{ if } A \neq \emptyset$$

$$D[v_i][\emptyset] = W[i][1]$$

Algorithm 3.11 DP Algorithm for TSP (3/3)

$$D[v_i][A] = \min_{v_j \in A}(W[i][j] + D[v_j][A - \{v_j\}]) \text{ if } A \neq \emptyset$$

for (k = 1; k <= n - 2; k++)

for (all subsets A $\subseteq$ V – {v_1} containing k vertices)

for (i such that i $\neq$ 1 and v_i is not in A) {

D[i][A] = $\underset{j:v_j \in A}{\text{minimum}}$ (W[i][j] + D[j][A – {v_j}]);

P[i][A] = value of j that gave the minimum;

}

D[1][V – {v_1}] = $\underset{2 \leq j \leq n}{\text{minimum}}$(W[1][j] + D[j][V – {$v_1$, v_j}]);

P[1][V – {v_1}] = value of j that gave the minimum;

minlength = D[1][V – {v_1}];

}

T(n) of Alg. 3.11 (1/2)

- **Theorem 3.1**

$$\sum_{k=1}^{n} k\binom{n}{k} = \sum_{k=1}^{n} n\binom{n-1}{k-1} = \sum_{k=0}^{n-1} n\binom{n-1}{k} = n\,2^{n-1}$$

- **Basic operation: instruction executed for each value of v_j**
- **Input size: $n = |V|$**
  - **Suppose $|A| = k$**

    $\#$ of subsets A of $V-\{v_1\} = \binom{n-1}{k}$

    for each *A*, we must consider [n – (k + 1)] vertices

    k choices for $v_j \in A$

$$D[v_i][A] = \min_{v_j \in A}\underbrace{(W[i][j] + D[v_j][A-\{v_j\}])}_{k}$$

$$\uparrow$$

$$[n-(k+1)]\binom{n-1}{k}$$

T(n) of Alg. 3.11 (2/2)

$$T(n) = \sum_{k=1}^{n-2} (n-1-k)\binom{n-1}{k}\ k$$

$$(n-1-k)\binom{n-1}{k} = \frac{(n-1-k)(n-1)!}{(n-1-k)!k!}$$

$$= \frac{(n-1)(n-2)!}{(n-2-k)!k!} = (n-1)\binom{n-2}{k}$$

$$= \sum_{k=1}^{n-2} (n-1)\binom{n-2}{k}\ k$$

(by Th. 3.1 n → n-2)

$$= (n-1)(n-2)2^{n-3} \quad \in \Theta(n^2 2^n)$$

Space complexity

$\#$ **of subsets A of** $V - \{v_1\} = 2^{n-1}$

The first index of D & P ranges between 1 and n

$$M(n) = 2 \times n \cdot 2^{n-1} \quad \in \Theta(n\, 2^n)$$

- **Printing the optimal tour**

$$P[1][\{v_2, v_3, v_4\}] = 3$$

$$P[3][\{v_2, v_4\}] \quad = 4$$

$$P[4][\{v_2\}] \quad = 2$$

$$\boxed{v_1 \rightarrow v_3 \rightarrow v_4 \rightarrow v_2 \rightarrow v_1}$$

3.7 Sequence Alignment

- **DNA sequence (a section of DNA)**

A C G T C A G

. . . . . . | | | | | | | . . . . . .

T G C A G T C

purines : A(adenine), G(guanine)

pyrimidines : C(cytosine), T(thymine)

base pair : (A, T), (G, C)

Sequence Alignment

Ex 3.13 DNA sequences

x[0..9] = A A C A G T T A C C

y[0..7] = T A A G G T C A

Alignment 1

- A A C A G T T A C C \\\\ **- : gap**

T A A - G G T - - C A

5 matched, 2 mismatched, 4 gaps

Alignment 2

A A C A G T T A C C

T A - A G G T - C A

5 matched, 3 mismatched, 2 gaps

Sequence Alignment

Cost of an alignment

the sum of all the penalties in an alignment

Assume

penalty for a mismatch = 1 and penalty for a gap = 2

Cost of alignment 1 = 2 * 1 + 4 * 2 = 10

Cost of alignment 2 = 3 * 1 + 2 * 2 = 7

Goal : optimal alignment with the min cost

Principle of Optimality Applies?

Ex 3.14 Suppose the following is an optimal alignment of x[0..9] and y[0..7]

A A C A G T T A C C

T A - A G G T - C A

Then the following must be an optimal alignment of x[1..9] and y[1..7]

A C A G T T A C C

A - A G G T - C A

Dynamic Programming

Let x[0..m-1] and y[0..n-1] be two sequences, penalty for a mismatch = 1 and penalty for a gap = 2

O(i, j) : the cost of the optimal alignment of x[i..m-1] & y[j..n-1]

Want : O(0, 0)

terminal condition

$O(m, j) = 2(n-j),\ j < n$ // insert n-j gaps

$O(i, n) = 2(m-i),\ i < m$ // insert m-i gaps

Dynamic Programming

Three cases

1. x[0] is aligned with y[0]

if x[0] = y[0] no penalty, penalty of 1 otherwise

2. x[0] is aligned with a gap

gap penalty of 2

3. y[0] is aligned with a gap

gap penalty of 2

O(0, 0) = min (O(1, 1)+penalty, O(1, 0)+2, O(0, 1)+2)

O(i, j) = min (O(i+1, j+1)+penalty, O(i+1, j)+2, O(i, j+1)+2)

Dynamic Programming

i \ *j*	0 T	1 A	2 A	3 G	4 G	5 T	6 C	7 A	8 –
0 A									
1 A									
2 C									
3 A									
4 G									
5 T									
6 T									
7 A									
8 C									
9 C									
10 –									

Diagonal 3

Diagonal 2

Diagonal 1

Figure 3.19 ● The array used to find the optimal alignment.

Algorithm 3.12 Divide & Conquer Algorithm

- **Problem**: Determine an optimal alignment of two DNA sequences.
- **Inputs**: DNA sequences x of length m and y of length n.
- **Outputs**: The cost of an optimal alignment.

```
void opt (int i, int j)
 {
     if (i == m) opt = 2(n-j) ;
     else if (j==n) opt = 2(m-i);
          else {
               if  (x[i] == y[j]) penalty = 0; else penalty = 1;
               opt= min(opt(i+1,j+1) )+penalty, opt(i+1, j)+2, opt(i, j+1)+2)
          }
 }    // note : exponential time complexity
```

Dynamic Programming Approach

Compute m+1 by n+1 array (see Fig. 3.19)

Example 3.13 : m = 10, n =8

Diagonal 1

$O(10, 8) = 2(10-10) = 0$

Diagonal 2

$O(9, 8) = 2(10-9) = 2$

$O(10, 7) = 2(8-7) = 2$

Diagonal 3

$O(8, 8) = 2(10-8) = 4$

$$O(9, 7) = \min(O(9+1, 7+1)+\text{penalty}, O(9+1, 7)+2, O(9, 7+1)+2)$$
$$= \min(0+1, 2+2, 2+2) = 1$$

$O(10, 6) = 2(8-6) = 4$

Dynamic Programming Approach

	j	0	1	2	3	4	5	6	7	8
i		T	A	A	G	G	T	C	A	–
0	A	**7**	8	10	12	13	15	16	18	20
1	A	6	**6**	8	10	11	13	14	16	18
2	C	6	5	**6**	8	9	11	12	14	16
3	A	7	5	**4**	6	7	9	11	12	14
4	G	9	7	5	**4**	5	7	9	10	12
5	T	8	8	6	4	**4**	5	7	8	10
6	T	9	8	7	5	3	**3**	5	6	8
7	A	11	9	7	6	4	2	**3**	4	6
8	C	13	11	9	7	5	3	**1**	3	4
9	C	14	12	10	8	6	4	2	**1**	2
10	–	16	14	12	10	8	6	4	2	**0**

Figure 3.20 ● The completed array used to find the optimal alignment.

Find the optimal alignment

Find the path from the upper-left corner to the lower-right corner (See Fig. 3.20)

1. Choose O[0][0] = 7

2. Find the second array item in the path

(a) Check O[0][1]

O(0, 1) + 2 = 8 + 2 = 10 ≠ 7 // 2 : gap penalty

(b) Check O[1][0]

O(1, 0) + 2 = 6 + 2 = 8 ≠ 7 // 2 : gap penalty

(c) Check O[1][1]

O(1, 1) + 1 = 6 + 1 = 7 // 1 : mismatch penalty x[0] ≠ y[0]

The second array item in the path is O[1][1] and Continue

Find the optimal alignment 2

$O(i, j) = \min(O(i+1, j+1)+\text{penalty}, O(i+1, j)+2, O(i, j+1)+2)$

case 1	**case 2**	**case 3**

$P(i, j)$ = case that gave the minimum

Ex :

	aligned
$P(0, 0) = 1$	**x[0] and y[0]**
$P(1, 1) = 1$	**x[1] and y[1]**
$P(2, 2) = 2$	**x[2] and gap**
$P(3, 2) = 1$	**x[3] and y[2]**

Find the optimal alignment 2

Ex : X[0..3] = AACA and Y[0..2] = TAA

O(i,j)

	0	1	2	3
0	3	4	6	8
1	2	2	4	6
2	3	1	2	4
3	4	2	0	2
4	6	4	2	0

P(i,j)

	0	1	2	
0	1	2	2	
1	1	1	2	
2	3	1	2	
3	3	3	1	