Lecture 6.2

- Dr. Anurag Sharma

Dynamic Programming (TSP)

Travelling Salesman Problem

- Suppose a salesperson is planning a sales trip that includes 20 cities. Each city is connected to some of the other cities by a road.
- To minimize travel time, we want to determine a shortest route that starts at the salesperson's home city, visits each of the cities once, and ends up at the home city.
- This problem of determining a shortest route is called the **Traveling Salesperson problem (TSP)**.

TSP

- An instance of this problem can be represented by a weighted graph, in which each vertex represents a city.
- A tour in a directed graph is a path from a vertex to itself that passes through each of the other vertices only once.
- An optimal tour in a weighted, directed graph is such a path of minimum length.
- No one has ever found an algorithm for the Traveling Salesperson problem whose worst-case time complexity is better than exponential. Yet, no one has ever proved that the algorithm is not possible.

Principle of optimality

- "The principle of optimality states that an optimal solution to an instance of a problem always contains optimal solutions to all sub-instances."
- In case of shortest path problem if v_k is a vertex on an optimal path from v_i to v_j, then the subpaths* from v_i to v_k and from v_k to v_j must also be optimal. [*with all same nodes.]

Solve TSP?

- The TSP is to find an optimal tour when at least one tour exists
- One method is to apply the **Brute-force** approach – i.e. start with one city and consider each remaining city in turn, but this will yield a factorial time!
- However, **dynamic programming** can also be applied to this problem.
  - Use DP paradigm and principle of optimality to divide the problem using bottom up approach.

DP for TSP

- If v_k is the first vertex after v_1 on an optimal tour, the subpath of that tour from v_k to v_1 must be a shortest path from v_k to v_1 that passes through each of the other vertices exactly once
- Let

  W = adjacency matrix for a graph

  V = set of all the vertices

  A = a subset of V
- $D[vi]\ [A]$ = length of shortest path from v_i to v_1 passing through each vertex in A exactly once

Cont.

- $V - \{v_1, v_j\}$ contains all vertices except $v1$ and vj and since principle of optimality applies, length of an optimal tour =

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

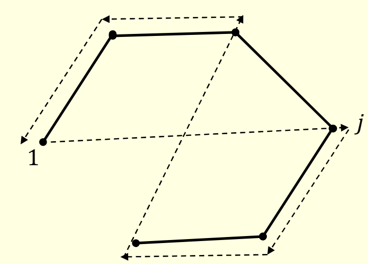
- In general for $i \neq 1$ and v_i not in A, $D[v_i][A]\ =$

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

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

Cont.

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



Example

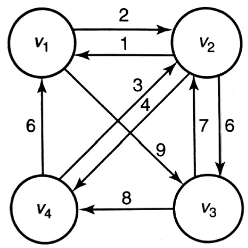


Figure 3.16 ● The optimal tour is $[v_1, v_3, v_4, v_2, v_1]$.

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

Figure 3.17 ● The adjacency matrix representation

Cont.

- $D[v_2][\phi] = 1$
- $D[v_3][\phi] = \infty$
- $D[v_4][\phi] = 6$

- $D[v_3][\{v_2\}] = 7 + 1 = 8$
- $D[v_4][\{v_2\}] = 4$

- $D[v_2][\{v_3\}] = 6 + \infty = \infty\ [v_2 - v_1 \Rightarrow v_2 - v_3 - v_1]$
- $D[v_4][\{v_3\}] = \infty$

- $D[v_2][\{v_4\}] = 4 + 6 = 10$
- $D[v_3][\{v_4\}] = 8 + 6 = 14\ \ [v_3 - v_1 \Rightarrow v_3 - v_4 - v_1]$

Cont.

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

$= \min_{j:\, v_j \in \{v_2, v_3\}} \left((W[4][2] + D[v_2][\{v_2, v_3\} - \{v_2\}), (W[4][3] + D[v_3][\{v_2, v_3\} - \{v_3\}]\right)$

$= \min_{j:\, v_j \in \{v_2, v_3\}} \left((W[4][2] + \underline{D}[v_2][\{v_3\}), (W[4][3] + \underline{D}[v_3][\{v_2\}]\right)$

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

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

$$= \min_{j:\, v_j \in \{v_2, v_3, v_4\}} \begin{pmatrix} W[1][2] + D[v_2][\{v_3, v_4\}, \\ W[1][3] + D[v_3][\{v_2, v_4\}, \\ W[1][4] + D[v_4][\{v_2, v_3\} \end{pmatrix}$$

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

- ~~What is $D[v_3][\{v_2, v_4\}]$ pictorially?~~

Algorithm

```
void travel (int n,
             const number W[][],
             index P[][],
             number& minlength)
{
  index i, j, k;
  number D[1..n][subset of V − {v1}];

  for (i = 2; i <= n; i++)
     D[i][∅] = W[i][1];
  for (k = 1; k <= n − 2; k++)
     for (all subsets A ⊆ V − {v1} containing k vertices)
        for (i such that i ≠ 1 and vi is not in A){
           D[i][A] = minimum (W[i][j] + D[j][A − {vj}]);
                     j:vj ∈ A
           P[i][A] = value of j that gave the minimum;
        }
  D[1][V − {v1}] = minimum (W[1][j] + D[j][V − {v1, vj}]);
                   2 ≤ j ≤ n
  P[1][V − {v1}] = value of j that gave the minimum;
  minlength = D[1][V − {v1}];
}
```

Time complexity?

- $T(n) = ?$
- According to the for loops in the algorithm:
- First loop: $k = 1: \sim n$ i.e., n times
- Second loop: $\binom{n}{k}$ for every k
- Third loop: $\mathrm{n} - k \approx n$
- Roughly, $T(n) = (n)\sum_{i=1}^{n} k\binom{n}{k}$
- $T(n) = (n)n2^n = n^2 2^n$
- Big O order is $O(2^n)$ (better than $n!$)

What is $\sum_{k=0}^{n} k\binom{n}{k}$? (from https://math.stackexchange.com)

Since the binomial coefficients have the $n-k$ symmetry, we can put

$$\sum_{k=0}^{n}(n-k)\binom{n}{n-k}$$

thus

$$S_n = \sum_{k=0}^{n} k\binom{n}{k} = \sum_{k=0}^{n}(n-k)\binom{n}{n-k}$$

But the RHS is

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

Now

$$S_n = n\sum_{k=0}^{n}\binom{n}{k} - \sum_{k=0}^{n} k\binom{n}{k}$$

or

$$S_n = n\sum_{k=0}^{n}\binom{n}{k} - S_n$$

$$S_n = n2^n - S_n$$

$2S_n = n2^n$

$S_n = n2^{n-1}$