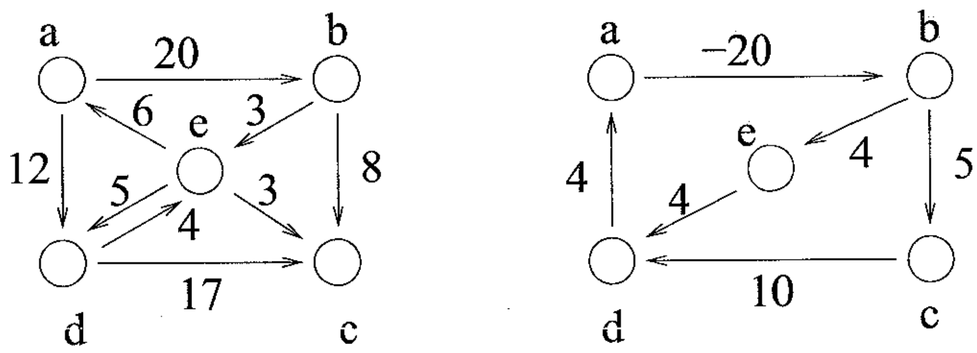
Floyd-Warshall Algorithm

Outline of this Lecture

- Recall the all-pairs shortest path problem.
- Recall the previous solution.
- The Floyd-Warshall Algorithm.

Recall the All-Pairs Shortest Paths Problem

Given a weighted digraph $G = (V, E)$ with a weight function $w : E \to \mathbf{R}$, where R is the set of real numbers, determine the length of the shortest path (i.e., distance) between all pairs of vertices in G. Here we assume that there are no cycle with zero or negative cost.



without negative cost cycle　　with negative cost cycle

Solution Covered in the Previous Lecture

Solution The Previous solution, based on a natural decomposition of the problem. Running time $O(n^4)$.

Our task: develop another dynamic programming algorithm, the Floyd-Warshall algorithm, with time complexity $O(n^3)$.
It also illustrate that there could be more than one way of developing a dynamic programming algorithm.

Solution : the Input and Output Format

Like the previous dynamic programming algorithm, we assume that the graph is represented by an $n \times n$ matrix with the weights of the edges:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E. \end{cases}$$

Output Format: an $n \times n$ matrix $F^{(n)} = [f_{ij}^{(n)}]$ where $f_{ij}^{(n)}$ is the distance from vertex i to j.

Step 1: The Floyd-Warshall Decomposition

Definition: The vertices $v_2, v_3, ..., v_{l-1}$ are called the *intermediate vertices* of the path $p = \langle v_1, v_2, ..., v_l \rangle$.

- Let $f_{ij}^{(k)}$ be the length of the shortest path from i to j such that any intermediate vertices on the path (if any) are chosen from the set $\{1, 2, \ldots, k\}$. $f_{ij}^{(0)}$ is defined to be w_{ij}, i.e., no intermediate vertex.
  Let $F^{(k)}$ be the $n \times n$ matrix $[f_{ij}^{(k)}]$.

- $f_{ij}^{(n)}$ is the distance from i to j. So our aim is to compute $F^{(n)}$.

- Subproblems: compute $F^{(k)}$ for $k = 0, 1, \cdots, n$.

Question: What is the easiest subproblem?

Step 2: Structure of shortest paths

Observation 1:

A shortest path does not contain the same vertex twice.

Proof: A path that contains the same vertex twice, contains a cycle. Removing the cycle gives a shorter path.

Observation 2: For a shortest path from i to j such that any intermediate vertices on the path are chosen from the set $\{1, 2, \ldots, k\}$, there are two possibilities:

1. k is not a vertex on the path,
The shortest such path has length $f_{ij}^{(k-1)}$.
2. k is a vertex on the path.
The shortest such path has length $f_{ik}^{(k-1)} + f_{kj}^{(k-1)}$.

Step 2: Structure of shortest paths

Consider a shortest path from i to j containing the vertex k. It consists of a subpath from i to k and a subpath from k to j.
Each of them only contains intermediate vertices in $\{1, ..., k-1\}$, and must be as short as possible, namely $f_{ik}^{(k-1)}$ and $f_{kj}^{(k-1)}$.
Hence the path has length $f_{ik}^{(k-1)} + f_{kj}^{(k-1)}$.

Combining the two cases we get

$$f_{ij}^{(k)} = \min\{f_{ij}^{(k-1)}, f_{ik}^{(k-1)} + f_{kj}^{(k-1)}\}.$$

Step 3: the Bottom-up Computation

- Bottom: $F^{(0)} = [w_{ij}]$, the weight matrix.

- Compute $F^{(k)}$ from $F^{(k-1)}$ using

$$f_{ij}^{(k)} = \min\left(f_{ij}^{(k-1)}, f_{ik}^{(k-1)} + f_{kj}^{(k-1)}\right)$$

for $k = 1, ..., n$.

Programming

The Floyd-Warshall Algorithm

Let W be the weight (distance) matrix

```
Floyd-Warshall(W, n)
{ for i = 1 to n do
     for j = 1 to n do
     { F[i, j] = W[i, j];

     }
  for k = 1 to n do
     for i = 1 to n do
        for j = 1 to n do
           if (F[i, k] + F[k, j] < F[i, j])
           { F[i, j] = F[i, k] + F[k, j];

           }
  return     the matrix F;
}
 // W and F can be implemented as
 // 2-dimensional arrays or
 // vectors of vectors in C++
```