Dynamic Programming

CSE 180
Algorithmic Thinking

Introduction

Dynamic Programming is a clever algorithm design technique:

- Invented by Richard Bellman in the 1950s.
- Used to solve *optimization* problems.
- *Programming* here means *Planning*.
- Useful when solution results from a sequence of decisions.
- Avoids computing the same thing twice:
  - Keeps a table of results as subproblems are solved.
  - Each subproblem is solved (once), then recorded in table.
  - Final state of the table will be (or contain) solution.
- Also avoids considering suboptimal solutions.

Comparison

Dynamic Programming vs. Divide & Conquer

- Dynamic Programming is bottom up, Divide & Conquer is top down.
- In both, problems are divided into subproblems, but in Dynamic Programming, subproblems may be overlapping.
- *Bottom up* means starting at smallest or simplest subproblem,
- *Then* combining subproblem solutions of increasing size until solution of the original problem is reached.

Example: Fibonacci Numbers

Recall definition of Fibonacci numbers:
$f(0) = 0, f(1) = 1, f(n) = f(n-1) + f(n-2)$

Computing the n^{th} Fibonacci number recursively (top down):

$$\begin{tikzcd}
 & & & f(n) \arrow[dll] \arrow[drr] & & & \\
 & f(n-1) \arrow[dl] \arrow[dr] & & + & & f(n-2) \arrow[dl] \arrow[dr] & \\
f(n-2) & + & f(n-3) & & f(n-3) & + & f(n-4)
\end{tikzcd}$$

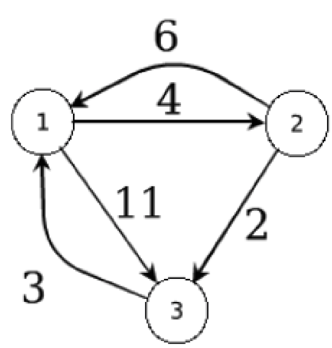
Computing the n^{th} Fibonacci number iteratively (bottom up):

- $f(0) = 0, f(1) = 1, f(2) = 0 + 1 = 1$
- $f(3) = 1 + 1 = 2, f(4) = 1 + 2 = 3, f(5) = 2 + 3 = 5, \ldots$
- $f(n-2) = f(n-3) + f(n-4), f(n-1) = f(n-2) + f(n-3)$
- $f(n) = f(n-1) + f(n-2)$

Floyd's Algorithm

- Robert W. Floyd is the man.
- Given a directed graph with edge weights, find the lengths of the shortest paths between every pair of vertices.
- Compare with Dijkstra's shortest path algorithm that finds the lengths of the shortest paths between one particular vertex and all other vertices.
- Build a matrix at each iteration.
- At iteration k, the matrix $D^{(k)}$ contains minimal paths on intermediate vertices $1 \ldots k$.

Example: Floyd's Algorithm



Start with a weighted directed graph:

$D^{(0)}$	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

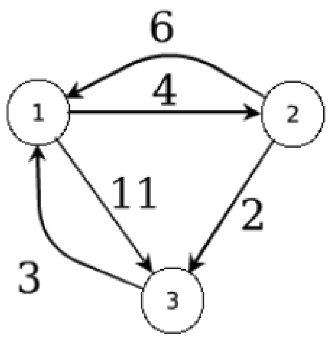
Create matrix $D^{(0)}$ where $D^{(0)}_{ij} = w_{ij}$

Create other matrices $D^{(k)}$ where $k \geq 1, (k = 1, 2, 3, \ldots)$

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

Example: Floyd's Algorithm

The formula: $D_{ij}^{(k)} = \min\left(D_{ij}^{(k-1)},\ D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right)$



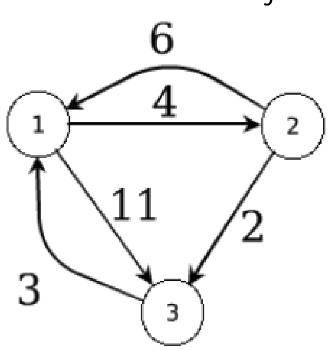
$D^{(0)}$	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

$D^{(1)}$	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

$$D_{11}^{(1)} = \min\left(D_{11}^{(0)},\ D_{11}^{(0)} + D_{11}^{(0)}\right) = \min(0, 0+0) = 0$$
$$D_{12}^{(1)} = \min\left(D_{12}^{(0)},\ D_{11}^{(0)} + D_{12}^{(0)}\right) = \min(4, 0+4) = 4$$
$$D_{13}^{(1)} = \min\left(D_{13}^{(0)},\ D_{11}^{(0)} + D_{13}^{(0)}\right) = \min(11, 0+11) = 11$$
$$\ldots$$
$$D_{32}^{(1)} = \min\left(D_{32}^{(0)},\ D_{31}^{(0)} + D_{12}^{(0)}\right) = \min(\infty, 3+4) = 7$$

Example: Floyd's Algorithm

The formula: $D_{ij}^{(k)} = \min\left(D_{ij}^{(k-1)},\ D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right)$



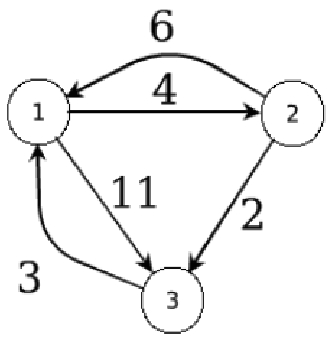
$D^{(1)}$	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

$D^{(2)}$	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

$$D_{13}^{(2)} = \min\left(D_{13}^{(1)},\ D_{12}^{(1)} + D_{23}^{(1)}\right) = \min(11, 4+2) = 6$$

Example: Floyd's Algorithm

The formula: $D_{ij}^{(k)} = \min\left(D_{ij}^{(k-1)},\ D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right)$



$D^{(2)}$	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

$D^{(3)}$	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

$D_{21}^{(3)} = \min\left(D_{21}^{(2)},\ D_{23}^{(2)} + D_{31}^{(2)}\right) = \min(6, 2+3) = 5$

```
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            D[i,j] = min(D[i,j], D[i,k] + D[k,j])
```