Project Proposals

✧ Your grade does not reflect how good a project you are proposing!

✧ It reflects how well you followed the instructions for the proposal!
What’s the point of the Project?

1. Learn and *understand* stuff about algorithms

2. Get a good grade
   - Help those of you who don’t do well in exams to improve your course grade
How to get a good grade:

At the end of the project, you will submit a written report that will be used as the sole basis for evaluating your project. Specifically, your report will be expected:

✦ To show that you understand and can implement standard algorithms.
✦ To show that you can write programs that are understandable, and algorithmically sound.
✦ To provide evidence that you understand how complexity theory shows up in practice.
✦ To demonstrate your initiative, originality, and algorithmic insights.
✦ To show that you can communicate your work clearly and concisely in a well-structured document.
Specific items that may appear in a report include:

- Descriptions of the algorithm(s) that you are working with, in your own words.
- Worked examples of your own devising to show how the algorithms work (i.e., not the ones in the book, the slides, the original papers where the algorithms were introduced, or other resources authored by other people).
- Details of the testing strategy that you have used. This is likely to involve the construction of code to generate large pseudo-random test cases, and code to verify that the results produced by your code are correct.
- Summaries of experimental data (e.g., tables and/or graphs showing the algorithms' behavior over a range of different inputs).
- Reflections on what you have learned as a result of your experience.
## Rubric

### Can understand and implement standard algorithms /10

<table>
<thead>
<tr>
<th>Description</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describes or depicts algorithms using examples</td>
<td></td>
</tr>
<tr>
<td>Discusses implementation issues</td>
<td></td>
</tr>
</tbody>
</table>

### Can write programs that are understandable and are algorithmically sound /30

<table>
<thead>
<tr>
<th>Description</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program fragments are presented</td>
<td></td>
</tr>
<tr>
<td>... are understandable</td>
<td></td>
</tr>
<tr>
<td>... are algorithmically sound</td>
<td></td>
</tr>
<tr>
<td>Sound measurement technique</td>
<td></td>
</tr>
<tr>
<td>Generation of experimental data sets</td>
<td></td>
</tr>
<tr>
<td>Testing for correct results</td>
<td></td>
</tr>
</tbody>
</table>

### Makes connections between implementation and complexity theory /15

<table>
<thead>
<tr>
<th>Description</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compares measured and predicted performance</td>
<td></td>
</tr>
<tr>
<td>Explains discrepancies, if any</td>
<td></td>
</tr>
<tr>
<td>Discusses why an asymptotically inferior algorithm might perform better</td>
<td></td>
</tr>
</tbody>
</table>
Rubric, continued

Demonstrates initiative, originality, and algorithmic insights

<table>
<thead>
<tr>
<th>Initiative</th>
<th>Originality</th>
<th>Algorithmic insights</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Document Communicates clearly

<table>
<thead>
<tr>
<th>Document is concise</th>
<th>Experimental procedure is described</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language is clear and correct</td>
<td>Experimental results presented</td>
</tr>
<tr>
<td>Purpose of project is described</td>
<td>Conclusions Presented</td>
</tr>
</tbody>
</table>
Lecture 13: Dynamic Programming

Andrew P. Black

Department of Computer Science
Portland State University
Dynamic programming:

- Solves problems by breaking them into smaller sub-problems and solving those.
- Which algorithm design technique is this like?
  - Brute force
  - Decrease-and-conquer
  - Divide-and-conquer
  - None that we’ve seen so far
Question:

- Compare Dynamic Programming with Decrease-and-Conquer:
  A. They are the same
  B. They both solve a large problem by first solving a smaller problem
  C. In decrease and conquer, we don’t “memoize” the solutions to the smaller problem
  D. In dynamic programming, we do “memoize” the smaller problems
  E. B, C & D
  F. B & C
  G. B & D
Dynamic Programming

- Dynamic programming differs from decrease-and-conquer because in dynamic programming we remember the answers to the smaller sub-problems.

- Why?
  A. To use more space
  B. In the hope that we might re-use them
  C. Because we know that the sub-problems overlap
Dynamic programming:
Dynamic programming:

- Solves problems by breaking them into smaller sub-problems and solving those.
Dynamic programming:

- Solves problems by breaking them into smaller sub-problems and solving those.
- like: decrease & conquer
Dynamic programming:

- Solves problems by breaking them into smaller sub-problems and solving those.

  - like: decrease & conquer

- Key idea: do not compute the solution to any sub-problem more than once;
Dynamic programming:

- Solves problems by breaking them into smaller sub-problems and solving those.
  - like: decrease & conquer
- Key idea: do not compute the solution to any sub-problem more than once;
  - instead: save computed solutions in a table so that they can be reused.
Dynamic programming:

✦ Solves problems by breaking them into smaller sub-problems and solving those.
  ✦ like: decrease & conquer

✦ Key idea: do not compute the solution to any sub-problem more than once;
  ✦ instead: save computed solutions in a table so that they can be reused.

✦ Consequently: dynamic programming works well when the sub-problems overlap.
Dynamic programming:

- Solves problems by breaking them into smaller sub-problems and solving those.
  - like: decrease & conquer
- Key idea: do not compute the solution to any sub-problem more than once;
  - instead: save computed solutions in a table so that they can be reused.
- Consequently: dynamic programming works well when the sub-problems overlap.
  - unlike: decrease & conquer
Why Dynamic Programming?

✧ If the subproblems are not independent, i.e. subproblems share sub-subproblems,
✧ then a decrease and conquer algorithm repeatedly solves the common sub-subproblems.
✧ Thus: it does more work than necessary
✧ The “memo table” in DP ensures that each sub-problem is solved (at most) once.
For dynamic programming to be applicable:

- At most polynomial-number of subproblems
  - otherwise: still exponential
- Solution to original problem is easy to compute from solutions to subproblems
- Natural ordering on subproblems from “smallest” to “largest”
- An easy-to-compute recurrence that allows solving a larger subproblem from a smaller subproblem
Optimization problems:

- Dynamic programming is typically (but not always) applied to *optimization problems*.
  - In an optimization problem, the goal is to find a solution among many possible candidates that *minimizes* or *maximizes* some particular value.
  - Such solutions are said to be *optimal*. 
Optimization problems:

- Dynamic programming is typically (but not always) applied to *optimization problems*
  - In an optimization problem, the goal is to find a solution among many possible candidates that *minimizes* or *maximizes* some particular value.
- Such solutions are said to be *optimal*.
  - There may be more than one “optimal” solution: true or false?
Example: Fibonacci Numbers

✨ The familiar recursive definition:

fib 0 = 0
fib 1 = 1
fib (n+2) = fib (n+1) + fib n

✨ Grows very rapidly:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
832040 (30th), …
354224848179261915075 (100th), …
Question

What is the order of growth of the Fibonacci function?

A. $O(n)$
B. $O(n^2)$
C. $O(1.61803^{...n})$
D. $O(2^n)$
E. $O(e^n)$
F. $O(n!)$
In fact, the $i^{\text{th}}$ Fibonacci number is the integer closest to
\[
\varphi^i / \sqrt{5}
\]
where:
\[
\varphi = \frac{1 + \sqrt{5}}{2} = 1.61803 \ldots
\]
(the "golden ratio")

Thus, the result of the Fibonacci function grows exponentially.
Complexity of brute-force $fib$:

Let $nfib$ be the number of calls needed to evaluate $fib \ n$, implemented according to the definition.

\[
\begin{align*}
    nfib\ 0 &= 1 \\
    nfib\ 1 &= 1 \\
    nfib\ (n+2) &= 1 + nfib\ (n+1) + nfib\ n
\end{align*}
\]

- Grows even more rapidly than $fib$!
  - Hence $fib$ is at least exponential 😞
- However: many calls to $fib$ have the same argument …
Repeated calls, same argument:
Repeated calls, same argument:

6
Repeated calls, same argument:
Repeated calls, same argument:
Repeated calls, same argument:
Repeated calls, same argument:
Repeated calls, same argument:
Repeated calls, same argument:
Repeated calls, same argument:
Repeated calls, same argument:
Repeated calls, same argument:

```
6
5 4
4 3 2
3 2 1 0
2 1 0
1 0
```
Avoiding repeated calculations:

- We can use a table to avoid doing a calculation more than once:

```java
int tableFib(int n) {
    if (table[n] = -1) {
        table[n] ← tableFib(n-1) + tableFib(n-2);
    }
    return table[n];
}
```

- Table size is fixed, but values can be shared over many calls.
Riding the wave:

- Alternatively, we can look at the way the entries in the table are filled:

```
 0  1  -1  -1  -1  -1  -1  -1  -1  -1  -1  ...
```
Riding the wave:

- Alternatively, we can look at the way the entries in the table are filled:

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>21</td>
<td>34</td>
<td>55</td>
</tr>
</tbody>
</table>
Riding the wave:

- Alternatively, we can look at the way the entries in the table are filled:

| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | ...

This leads to code:

```c
int a ← 0, b ← 1;
for i from 0 to n do {
    int c = a + b;
    a ← b;
    b ← c;
}
return a;
```

No limits on \( n \) now, but values cannot be reused.
Riding the wave:

- Alternatively, we can look at the way the entries in the table are filled:

|   | 0 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | ...
|---|---|---|---|---|---|---|----|----|----|----|----|-----|

This leads to code:

```c
int a ← 0, b ← 1;
for i from 0 to n do {
    int c = a + b;
    a ← b;
    b ← c;
}
return a;
```

Complexity is $O(n)$!

No limits on $n$ now, but values cannot be reused.
Binomial Coefficients

Again, a recursive definition:

\[ C(n, k) = C(n-1, k-1) + C(n-1, k) \]

for \( n > k > 0 \)

\[ C(n, 0) = 1 \]
\[ C(n, n) = 1 \]

What’s the complexity of a naive implementation of \( C \)?
Binomial Coefficients

- Again, a recursive definition:

  \[ C(n, k) = C(n-1, k-1) + C(n-1, k) \]

  for \( n > k > 0 \)

  \[ C(n, 0) = 1 \]

  \[ C(n, n) = 1 \]

- Notice: the calls overlap
Binomial Coefficients

✦ Again, a recursive definition:

✦ $C(n, k) = C(n-1, k-1) + C(n-1, k)$ for $n > k > 0$

$C(n, 0) = 1$

$C(n, n) = 1$

✦ Notice: the calls overlap
Binomial Coefficients

Again, a recursive definition:

\[ C(n, k) = C(n-1, k-1) + C(n-1, k) \]
for \( n > k > 0 \)

\[ C(n, 0) = 1 \]
\[ C(n, n) = 1 \]

Notice: the calls overlap
Binomial Coefficients

- Again, a recursive definition:
  - \( C(n, k) = C(n-1, k-1) + C(n-1, k) \)
  - for \( n > k > 0 \)
  - \( C(n, 0) = 1 \)
  - \( C(n, n) = 1 \)

- Notice: the calls overlap
Binomial Coefficients

✧ Again, a recursive definition:

✧ \[ C(n, k) = C(n-1, k-1) + C(n-1, k) \]
  
  for \( n > k > 0 \)

\[ C(n, 0) = 1 \]

\[ C(n, n) = 1 \]

✧ Notice: the calls overlap
Binomial Coefficients

✧ Again, a recursive definition:

✧ $C(n, k) = C(n-1, k-1) + C(n-1, k)$

for $n > k > 0$

$C(n, 0) = 1$
$C(n, n) = 1$

✧ Notice: the calls overlap
Binomial Coefficients

✧ Again, a recursive definition:

✧ \( C(n, k) = C(n-1, k-1) + C(n-1, k) \) for \( n > k > 0 \)

\( C(n, 0) = 1 \)
\( C(n, n) = 1 \)

✧ Notice: the calls overlap

\[
\begin{align*}
5, 3 & \\
4, 2 & \\
3, 1 & \\
3, 2 & \\
3, 3 & \\
2, 0 & \\
2, 1 & \\
2, 2 & \\
\end{align*}
\]
Binomial Coefficients

Again, a recursive definition:

\[ C(n, k) = \begin{cases} 1, & \text{if } n = 0 \text{ or } k = 0, \\ C(n-1, k-1) + C(n-1, k), & \text{if } n > k > 0. \end{cases} \]

Notice: the calls overlap.
Construct a Table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>k − 1</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n − 1</td>
<td>1</td>
<td>C(n − 1, k − 1)</td>
<td>C(n − 1, k)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C(n, k)</td>
</tr>
</tbody>
</table>
Construct a Table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>(k - 1)</th>
<th>(k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(k)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n - 1)</td>
<td>1</td>
<td></td>
<td>(C(n - 1, k - 1))</td>
<td>(C(n - 1, k))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(n)</td>
<td>1</td>
<td>(C(n - 1, k - 1))</td>
<td>(C(n - 1, k))</td>
<td>(C(n, k))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Construct a Table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>k – 1</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n – 1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>$C(n – 1, k – 1)$</td>
<td>$C(n – 1, k)$</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>$C(n, k)$</td>
<td></td>
</tr>
</tbody>
</table>
Algorithm

\textbf{ALGORITHM} \quad \textit{Binomial}(n, k)

// Computes $C(n, k)$ by the dynamic programming algorithm
// Input: A pair of nonnegative integers $n \geq k \geq 0$
// Output: The value of $C(n, k)$
for $i \leftarrow 0$ to $n$ do
    for $j \leftarrow 0$ to $\min(i, k)$ do
        if $j = 0$ or $j = i$
            $C[i, j] \leftarrow 1$
        else $C[i, j] \leftarrow C[i - 1, j - 1] + C[i - 1, j]$

return $C[n, k]$
Algorithm

for $i \leftarrow 0$ to $n$ do
    for $j \leftarrow 0$ to $\min(i, k)$ do
        if $j = 0$ or $j = i$
            $C[i, j] \leftarrow 1$
        else $C[i, j] \leftarrow C[i - 1, j - 1] + C[i - 1, j]$
    
return $C[n, k]$

Time efficiency?
Algorithm

```
for i ← 0 to n do
    for j ← 0 to min(i, k) do
        if j = 0 or j = i
            C[i, j] ← 1
        else C[i, j] ← C[i - 1, j - 1] + C[i - 1, j]
    return C[n, k]
```

Time efficiency?

A. \( O(n) \)
B. \( O(k) \)
C. \( O(1) \)
D. \( O(nk) \)
E. None of the above
Algorithm

\[
\text{for } i \leftarrow 0 \text{ to } n \text{ do} \\
\quad \text{for } j \leftarrow 0 \text{ to } \min(i, k) \text{ do} \\
\quad \quad \text{if } j = 0 \text{ or } j = i \\
\quad \quad \quad C[i, j] \leftarrow 1 \\
\quad \quad \text{else } C[i, j] \leftarrow C[i - 1, j - 1] + C[i - 1, j] \\
\text{return } C[n, k]
\]

Space efficiency?
Algorithm

\[
\begin{align*}
&\text{for } i \leftarrow 0 \text{ to } n \text{ do} \\
&\quad \text{for } j \leftarrow 0 \text{ to } \min(i, k) \text{ do} \\
&\qquad \text{if } j = 0 \text{ or } j = i \\
&\qquad \quad C[i, j] \leftarrow 1 \\
&\qquad \text{else } C[i, j] \leftarrow C[i - 1, j - 1] + C[i - 1, j] \\
&\text{return } C[n, k]
\end{align*}
\]

Space efficiency?

A. O(n)  
B. O(k)  
C. O(1)  
D. O(nk)  
E. None of the above
Problem
Problem

a) Compute $C(6,3)$ by applying the dynamic programming algorithm and the recurrence

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$

$$C(n, 0) = 1, \quad C(n, n) = 1$$
Problem

a) Compute $C(6,3)$ by applying the dynamic programming algorithm and the recurrence

$$C(n, k) = C(n-1, k-1) + C(n-1, k)$$
$$C(n, 0) = 1, \quad C(n, n) = 1$$

b) Is it also possible to compute $C(n,k)$ by filling the algorithm’s dynamic programming table column by column rather than row by row?
Problem

Which of the following algorithms for computing a binomial coefficient is most efficient?

a. Use the formula

\[ C(n, k) = \frac{n!}{k!(n-k)!}. \]

b. Use the formula

\[ C(n, k) = \frac{n(n-1)\ldots(n-k+1)}{k!}. \]

c. Apply recursively the formula

\[
\begin{align*}
C(n, k) &= C(n-1, k-1) + C(n-1, k) \quad \text{for } n > k > 0, \\
C(n, 0) &= C(n, n) = 1.
\end{align*}
\]

d. Apply the dynamic programming algorithm.
Problem

Which of the following algorithms for computing a binomial coefficient is most efficient?

a. Use the formula

\[ C(n, k) = \frac{n!}{k!(n-k)!}. \]

2n multiplications, 1 division

b. Use the formula

\[ C(n, k) = \frac{n(n-1)\ldots(n-k+1)}{k!}. \]

c. Apply recursively the formula

\[
\begin{align*}
C(n, k) &= C(n-1, k-1) + C(n-1, k) \quad \text{for } n > k > 0, \\
C(n, 0) &= C(n, n) = 1.
\end{align*}
\]

d. Apply the dynamic programming algorithm.
Problem

Which of the following algorithms for computing a binomial coefficient is most efficient?

a. Use the formula

\[ C(n, k) = \frac{n!}{k!(n-k)!} \]

(b) by using elementary combinatorics.

b. Use the formula

\[ C(n, k) = \frac{n(n-1)\ldots(n-k+1)}{k!} \]

(c) by a dynamic programming algorithm.

c. Apply recursively the formula

\[
\begin{align*}
C(n, k) &= C(n-1, k-1) + C(n-1, k) \quad \text{for } n > k > 0, \\
C(n, 0) &= C(n, n) = 1.
\end{align*}
\]

d. Apply the dynamic programming algorithm.

2n multiplications, 1 division

2k multiplications, 1 division
Problem

Which of the following algorithms for computing a binomial coefficient is most efficient?

a. Use the formula

\[ C(n, k) = \frac{n!}{k!(n-k)!}. \]

2n multiplications, 1 division

b. Use the formula

\[ C(n, k) = \frac{n(n-1)...(n-k+1)}{k!}. \]

2k multiplications, 1 division

c. Apply recursively the formula

\[ C(n, k) = C(n-1, k-1) + C(n-1, k) \quad \text{for } n > k > 0, \]
\[ C(n, 0) = C(n, n) = 1. \]

C(n, k) additions

d. Apply the dynamic programming algorithm.
Problem

Which of the following algorithms for computing a binomial coefficient is most efficient?

a. Use the formula
   \[ C(n, k) = \frac{n!}{k!(n-k)!} \]
   \(2n\) multiplications, \(1\) division

b. Use the formula
   \[ C(n, k) = \frac{n(n-1)...(n-k+1)}{k!} \]
   \(2k\) multiplications, \(1\) division

b. Use the formula
   \[ C(n, k) = C(n-1, k-1) + C(n-1, k) \quad \text{for} \quad n > k > 0, \]
   \[ C(n, 0) = C(n, n) = 1. \]

C. Apply recursively the formula

   \[ C(n, k) = \begin{cases} 1 & \text{if } k = 0 \text{ or } k = n \text{,} \\ C(n-1, k-1) + C(n-1, k) & \text{otherwise.} \end{cases} \]

\(C(n, k)\) additions


d. Apply the dynamic programming algorithm.

\(\Theta(nk)\)
Knapsack Problem by DP
Knapsack Problem by DP

- Given \( n \) items of
  
  integer weights:  \( w_1 \quad w_2 \quad \ldots \quad w_n \)
  
  values:  \( v_1 \quad v_2 \quad \ldots \quad v_n \)
Knapsack Problem by DP

- Given $n$ items of integer weights: $w_1, w_2, \ldots, w_n$
- values: $v_1, v_2, \ldots, v_n$
- a knapsack of integer capacity $W$
Knapsack Problem by DP

- Given $n$ items of
  - integer weights: $w_1, w_2, \ldots, w_n$
  - values: $v_1, v_2, \ldots, v_n$

  a knapsack of integer capacity $W$

  find most valuable subset of the items that fit into the knapsack
Knapsack Problem by DP

- Given \( n \) items of integer weights: \( w_1 \ w_2 \ \ldots \ w_n \)
- values: \( v_1 \ v_2 \ \ldots \ v_n \)
- a knapsack of integer capacity \( W \)

find most valuable subset of the items that fit into the knapsack

- How can we set this up as a recursion over smaller subproblems?
Knapsack Problem by DP

- Given $n$ items of integer weights: $w_1, w_2, \ldots, w_n$
  values: $v_1, v_2, \ldots, v_n$

  a knapsack of integer capacity $W$

  find most valuable subset of the items that fit into the knapsack

- How can we set this up as a recursion over smaller subproblems?
Knapsack Problem by DP
Knapsack Problem by DP

Consider problem instance defined by first $i$ items and capacity $j$ ($j \leq W$).
Knapsack Problem by DP

Consider problem instance defined by first $i$ items and capacity $j$ ($j \leq W$).

Let $V[i, j]$ be value of optimal solution of this problem instance. Then
Knapsack Problem by DP

Consider problem instance defined by first $i$ items and capacity $j$ ($j \leq W$).

Let $V[i, j]$ be value of optimal solution of this problem instance. Then

$$V[i, j] = \begin{cases} 
\max (V[i-1, j], v_i + V[i-1, j-w_i]) & \text{if } j \geq w_i \\
V[i-1, j] & \text{if } j < w_i 
\end{cases}$$
Knapsack Problem by DP

Consider problem instance defined by first $i$ items and capacity $j$ ($j \leq W$).

Let $V[i, j]$ be value of optimal solution of this problem instance. Then

$$V[i, j] = \begin{cases} 
\max (V[i-1, j], v_i + V[i-1, j-w_i]) & \text{if } j \geq w_i \\
V[i-1, j] & \text{if } j < w_i 
\end{cases}$$
Knapsack Problem by DP

Consider problem instance defined by first $i$ items and capacity $j$ ($j \leq W$).

Let $V[i, j]$ be value of optimal solution of this problem instance. Then

$$V[i, j] = \begin{cases} 
\max (V[i-1, j], v_i + V[i-1, j-w_i]) & \text{if } j \geq w_i \\
V[i-1, j] & \text{if } j < w_i 
\end{cases}$$

Initial conditions: $V[0, j] = 0$ and $V[i, 0] = 0$
Knapsack Problem by DP (example)

Knapsack of capacity $W = 5$

<table>
<thead>
<tr>
<th>item</th>
<th>weight</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>$12</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$10</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>$20</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>$15</td>
</tr>
</tbody>
</table>

$$V[i, j] = \begin{cases} \max (V[i-1, j], v_i + V[i-1, j-w'_i]) & \text{if } j \geq w'_i \\ V[i-1, j] & \text{if } j < w'_i \end{cases}$$

capacity, $j$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$w_1 = 2, v_1 = 12$
$w_2 = 1, v_2 = 10$
$w_3 = 3, v_3 = 20$
$w_4 = 2, v_4 = 15$
Can we do this “top down”?

- Yes: use a memo function
- Not: a “memory function”

Idea: record previously computed values “just in time”
ALGORITHM  \textit{MFKnapsack}(i, j)

//Implements the memory function method for the knapsack problem
//Input: A nonnegative integer \( i \) indicating the number of the first
//items being considered and a nonnegative integer \( j \) indicating
//the knapsack’s capacity
//Output: The value of an optimal feasible subset of the first \( i \) items
//Note: Uses as global variables input arrays \( Weights[1..n] \), \( Values[1..n] \),
//and table \( V[0..n, 0..W] \) whose entries are initialized with \(-1\)’s except for
//row 0 and column 0 initialized with 0’s

\textbf{if} \( V[i, j] < 0 \)
\phantom{1}\hspace{1cm} \textbf{if} \( j < Weights[i] \)
\phantom{1}\hspace{2cm} \text{value} \leftarrow \textit{MFKnapsack}(i - 1, j) \)
\phantom{1}\hspace{1cm} \textbf{else}
\phantom{1}\hspace{2cm} \text{value} \leftarrow \max(\textit{MFKnapsack}(i - 1, j),
\phantom{1}\hspace{3.5cm} Values[i] + \textit{MFKnapsack}(i - 1, j - Weights[i]))
\phantom{1}\hspace{1cm} V[i, j] \leftarrow \text{value} \)
\textbf{return} \( V[i, j] \)
Summary

✧ Dynamic programming is a good technique to use when:
  ▪ Solutions defined in terms of solutions to smaller problems of the same type.
  ▪ Many overlapping subproblems.

✧ Implementation can use either:
  ▪ top-down, recursive definition with memoization
  ▪ explicit bottom-up tabulation
Problem

a. Apply the bottom-up dynamic programming algorithm to the following instance of the knapsack problem:

<table>
<thead>
<tr>
<th>item</th>
<th>weight</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>$25</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$20</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$15</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$40</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>$50</td>
</tr>
</tbody>
</table>

, capacity $W = 6$.

b. How many different optimal subsets does the instance of part (a) have?

c. In general, how can we use the table generated by the dynamic programming algorithm to tell whether there is more than one optimal subset for the knapsack problem’s instance?
Problem:

The sequence of values in a row of the dynamic programming table for an instance of the knapsack problem is always non-decreasing:

True or False?
Problem:

✧ The sequence of values in a *column* of the dynamic programming table for an instance of the knapsack problem is always non-decreasing:

✧ True or False?
Problem

### Exercises 8.4

1. a. Apply the bottom-up dynamic programming algorithm to the following instance of the knapsack problem:

<table>
<thead>
<tr>
<th>item</th>
<th>weight</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>$25</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$20</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$15</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>$40</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>$50</td>
</tr>
</tbody>
</table>

, capacity $W = 6$.

b. How many different optimal subsets does the instance of part (a) have?

c. In general, how can we use the table generated by the dynamic programming algorithm to tell whether there is more than one optimal subset for the knapsack problem's instance?

2. a. Write a pseudocode of the bottom-up dynamic programming algorithm for the knapsack problem.

b. Write a pseudocode of the algorithm that finds the composition of an optimal subset from the table generated by the bottom-up dynamic programming algorithm for the knapsack problem.

3. For the bottom-up dynamic programming algorithm for the knapsack problem, prove that

a. its time efficiency is in $\Theta(nW)$.

b. its space efficiency is in $\Theta(nW)$.

c. the time needed to find the composition of an optimal subset from a filled dynamic programming table is in $O(n + W)$.

4. a. True or false: A sequence of values in a row of the dynamic programming table for an instance of the knapsack problem is always nondecreasing.

b. True or false: A sequence of values in a column of the dynamic programming table for an instance of the knapsack problem is always nondecreasing?

5. Apply the memory function method to the instance of the knapsack problem given in Problem 1. Indicate the entries of the dynamic programming table that are:

(i) never computed by the memory function method on this instance;

(ii) retrieved without a recomputation.

Apply the memo function method to the above instance of the knapsack problem. Which entries of the dynamic programming table are (i) never computed by the memo function, and (ii) retrieved without recomputation.
In the table below, the cells marked by a minus indicate the ones for which no entry is computed for the instance in question; the only nontrivial entry that is retrieved without recomputation is (2, 1).

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>-</td>
<td>-</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>15</td>
<td>20</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>15</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>65</td>
</tr>
</tbody>
</table>

6. Since some of the cells of a table with $n+1$ rows and $W+1$ columns are filled in constant time, both the time and space efficiencies are in $\Theta(nW)$. 

Note: In fact, we can also stop the algorithm as soon as $j$, the unused capacity of the knapsack, becomes 0.
Warshall’s Algorithm

- Computes the transitive closure of a relation.
  - reachability in a graph is only an example of such a relation …
Warshall’s Algorithm

**Warshall Algorithm 1**
Warshall($M_R: n \times n$ 0-1 matrix)
$W := M_R (W = [w_{ij}])$
for($k=1$ to $n$) {
  for($i=1$ to $n$) {
    for($j=1$ to $n$) {
      $w_{ij} = w_{ij} \lor (w_{ik} \land w_{kj})$
    }
  }
} return $W$

**Warshall Algorithm 2**
Warshall($M_R: n \times n$ 0-1 matrix)
$W := M_R (W = [w_{ij}])$
for($k=1$ to $n$) {
  for($i=1$ to $n$) {
    for($j=1$ to $n$) {
      if($w_{ik}=1$) {
        for($j=1$ to $n$) {
          $w_{ij} = w_{ij} \lor w_{kj}$
        }
      }
    }
  }
} return $W$
Example

\[ M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix} \]

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ M_R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \]

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} \]

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[W_0 = M_R = \text{direct connections between nodes}\]
Example

\[ M_R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \]

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ M_R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \]

\[ W_0 = M_R = \text{direct connections between nodes} \]
Example

\[ M_R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \]

\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_0 = M_R = \text{direct connections between nodes} \]

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]
$W_0 = M_R = \text{direct connections between nodes}$

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
\]

**Warshall Algorithm 1**

Warshall($M_R$: $n \times n$ 0-1 matrix)  
$W := M_R$ ($W = [w_{ij}]$)  
for($k=1$ to $n$) {  
    for($i=1$ to $n$) {  
        for($j=1$ to $n$) {  
            $w_{ij} = w_{ij} \lor (w_{ik} \land w_{kj})$
        }
    }
}
return $W$
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

**Warshall Algorithm 1**

Warshall(\(M_R: n \times n\) 0-1 matrix)
\[
W := M_R (W = [w_{ij}])
\]
for(\(k=1\) to \(n\)) {
  for(\(i=1\) to \(n\)) {
    for(\(j=1\) to \(n\)) {
      \[w_{ij} = w_{ij} \lor (w_{ik} \land w_{kj})\]
    }
  }
return \(W\)
$W_0 = M_R = \text{direct connections between nodes}$

$W_1 = W_0 + \text{connections thru node 1}$

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

$w_{ij} \leftarrow w_{ij} \lor (w_{ik} \land w_{kj})$

**Warshall Algorithm 1**

Warshall($M_R$: $n \times n$ 0-1 matrix)

$W := M_R$ ($W = [w_{ij}]$)

for $k=1$ to $n$ {
  for $i=1$ to $n$ {
    for $j=1$ to $n$ {
      $w_{ij} = w_{ij} \lor (w_{ik} \land w_{kj})$
    }
  }
}

return $W$
$W_0 = M_R = \text{direct connections between nodes}$

$W_1 = W_0 + \text{connections thru node 1}$

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[w_{ij} \leftarrow w_{ij} \lor (w_{ik} \land w_{kj})\]
\( W_0 = M_R = \text{direct connections between nodes} \)

\( W_1 = W_0 + \text{connections thru node 1} \)

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\( w_{ij} \leftarrow w_{ij} \lor (w_{ik} \land w_{kj}) \)
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[ M_R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad W_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i1} \land w_{1j}) \]
$W_0 = M_R = \text{direct connections between nodes}$

$W_1 = W_0 + \text{connections thru node 1}$

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\quad W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[w_{ij} \leftarrow w_{ij} \lor (w_{i1} \land w_{1j})\]
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[ M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix} \quad W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
\end{bmatrix} \]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i1} \land w_{1j}) \]
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[ M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} \quad \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix} \]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i1} \land w_{1j}) \]
$W_0 = M_R = \text{direct connections between nodes}$

$W_1 = W_0 + \text{connections thru node 1}$

$M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}$

$W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
\end{bmatrix}$

$w_{ij} \leftarrow w_{ij} \lor (w_{i1} \land w_{1j})$  \text{add new 1 when } i = 4 \text{ and } j = 3$
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[ M_R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i1} \land w_{1j}) \quad \text{add new 1 when } i = 4 \text{ and } j = 3 \]
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i1} \land w_{1j}) \]
$W_0 = M_R = \text{direct connections between nodes}$

$W_1 = W_0 + \text{connections thru node 1}$

$M_R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$

$W_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[ M_R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad W_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad W_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \]
\[ W_0 = M_R = \text{direct connections between nodes} \]

\[ W_1 = W_0 + \text{connections thru node 1} \]

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
W_2 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i2} \land w_{2j}) \]
\[ W_0 = M_R = \text{direct connections between nodes} \]

\[ W_1 = W_0 + \text{connections thru node 1} \]

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
W_2 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i2} \land w_{2j}) \]
$W_0 = M_R = \text{direct connections between nodes}$

$W_1 = W_0 + \text{connections thru node 1}$

$$M_R = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad W_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad W_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$w_{ij} \leftarrow w_{ij} \lor (w_{i2} \land w_{2j})$
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
W_2 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\omega_{ij} \leftarrow \omega_{ij} \lor (\omega_{i2} \land \omega_{2j})
\]
\[ W_0 = M_R = \text{direct connections between nodes} \]

\[ W_1 = W_0 + \text{connections thru node 1} \]

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
W_2 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i2} \land w_{2j}) \]
$W_0 = M_R = \text{direct connections between nodes}$

$W_1 = W_0 + \text{connections thru node 1}$

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_2 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

$w_{ij} \leftarrow w_{ij} \lor (w_{i2} \land w_{2j})$
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
W_2 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i2} \land w_{2j}) \]
$W_0 = M_R =$ direct connections between nodes

$W_1 = W_0 +$ connections thru node 1

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\quad \quad \quad
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\quad \quad \quad
W_2 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[w_{ij} \leftarrow w_{ij} \lor (w_{i2} \land w_{2j})\]
\( W_0 = M_R = \text{direct connections between nodes} \)

\( W_1 = W_0 + \text{connections thru node 1} \)

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
W_2 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
w_{ij} \leftarrow w_{ij} \lor (w_{i2} \land w_{2j})
\]

\[
W_3 = \begin{bmatrix}
1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}
\]
\[ W_0 = M_R = \text{direct connections between nodes} \]
\[ W_1 = W_0 + \text{connections thru node 1} \]

\[
M_R = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
W_1 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[
W_2 = \begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}
\]

\[ w_{ij} \leftarrow w_{ij} \lor (w_{i2} \land w_{2j}) \]

\[
W_3 = \begin{bmatrix}
1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}
\]

\[
W_4 = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}
\]