Dynamic programming

• A general algorithmic paradigm
• The go-to solution for problems with natural subproblems.
• Typically, running time polynomial but not linear (e.g Bellman-Ford is $O(EV)$).
• Most commonly used method for tractable combinatorial problems.
• Useful in interviews.
Dynamic programming blueprint

- Find good subproblems
- Come up with a recursive solution (recurrence relation)
- Use memorization to avoid running time blowup.
Warmup: Fibonacci numbers

F(0) = 1
F(1) = 1
F(n) = F(n-1) + F(n-2) for n >= 2

Compute F(n)

Naïve solution

```c
long F(long n) {
    if ((n==0)||(n==1)) return 1;
    return F(n-1) + F(n-2);
}
```
Running time?

Naïve solution

```c
long F(long n) {
    if ((n==0)||(n==1)) return 1;
    return F(n-1)+F(n-2);
}
```

\[ T(n) = T(n-1)+T(n-2)+1 \]

\[ T(n) \sim F(n) \sim \phi^n, \quad \phi = \frac{1+\sqrt{5}}{2} \approx 1.62 \]

Can we do better?
Replace recursion with memorization!

```cpp
long F(long n) {
    if ((n==0)||(n==1)) return 1;
    return F(n-1)+F(n-2);
}

long Fmem(long n) {
    long [] ans = new long[n+1];
    ans[0]=1; ans[1]=1;
    for (int i=2; i<=n; i++)
        ans[i]=ans[i-1]+ans[i-2];
    return ans[n];
}
```
Replace recursion with memorization!

```java
long Fmem(long n) {
    long [] ans = new long[n+1];
    ans[0]=1; ans[1]=1;
    for (int i=2; i<=n; i++)
        ans[i]=ans[i-1]+ans[i-2];
    return ans[n]; }
```

Running time?
O(n)
Can do even better?
Yes, can only do O(log n) multiplications.
**Goal.** Paint a row of \( n \) houses red, green, or blue so that

No two adjacent houses have the same color.

Minimize total cost, where \( \text{cost}(i, \text{color}) \) is cost to paint \( i \) given color.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>22</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Cost to paint house \( i \) the given color
Naïve solution

Try coloring the last house on the block in all three possible colors.
$C(n) =$ the min cost of coloring the block so that the last house gets color $C$.

$$B(n) = \text{cost}(n, B) + \min(G(n - 1), R(n - 1))$$

Reduction to a smaller subproblem

Subproblem = paining the first $i$ houses, with the last house having a prescribed color.

Can solve those and memorize the answer.
**Subproblems.**

\[ R[i] = \min \text{ cost to paint houses } 1, \ldots, i \text{ with } i \text{ red.} \]
\[ G[i] = \min \text{ cost to paint houses } 1, \ldots, i \text{ with } i \text{ green.} \]
\[ B[i] = \min \text{ cost to paint houses } 1, \ldots, i \text{ with } i \text{ blue.} \]

Optimal cost = \( \min \{ R[n], G[n], B[n] \} \).

**Recurrence equations:**

\[ R[i+1] = \text{cost}(i+1, \text{red}) + \min \{ B[i], G[i] \} \]
\[ G[i+1] = \text{cost}(i+1, \text{green}) + \min \{ R[i], B[i] \} \]
\[ B[i+1] = \text{cost}(i+1, \text{blue}) + \min \{ R[i], G[i] \} \]

\[ R[0] = G[0] = B[0] = 0 \]

Running time. \( O(n) \).
## House Coloring Problem

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cost to paint house i the given color.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>22</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
**House Coloring Problem**

![Image of houses colored in different colors]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>22</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>16</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Cost to paint house $i$ the given color

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>21</td>
<td>29</td>
<td>46</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>15</td>
<td>18</td>
<td>35</td>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>B</td>
<td>16</td>
<td>13</td>
<td>13</td>
<td>20</td>
<td>26</td>
<td>36</td>
</tr>
</tbody>
</table>
## House Coloring Problem

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>21</td>
<td>29</td>
<td>46</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>15</td>
<td>18</td>
<td>35</td>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>B</td>
<td>16</td>
<td>13</td>
<td>13</td>
<td>20</td>
<td>26</td>
<td>36</td>
</tr>
</tbody>
</table>

Cost to paint house i the given color.
Aside: Can formulate as a shortest path problem.
**Aside:** Can formulate as a shortest path problem

**Goal:** shortest path from S to T

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8</td>
<td>9</td>
<td>22</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>T</td>
<td>16</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

![Diagram of network with nodes and weights]
ASIDE: CAN FORMULATE AS A SHORTEST PATH PROBLEM

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>21</td>
<td>29</td>
<td>46</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>15</td>
<td>18</td>
<td>35</td>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>B</td>
<td>16</td>
<td>13</td>
<td>13</td>
<td>20</td>
<td>26</td>
<td>36</td>
</tr>
</tbody>
</table>

![Diagram showing a network of houses with distances labeled]
**Aside: Can formulate as a shortest path problem**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>21</td>
<td>29</td>
<td>46</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>15</td>
<td>18</td>
<td>35</td>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>B</td>
<td>16</td>
<td>13</td>
<td>13</td>
<td>20</td>
<td>26</td>
<td>36</td>
</tr>
</tbody>
</table>

![Graph showing shortest path problem](image-url)
Dynamic programming blueprint

- Find good subproblems
- Come up with a recursive solution
- Use memorization to avoid running time blowup.
- Key question: What are the subproblems?
Problem. Given $n$ coin denominations $\{c_1, c_2, \ldots, c_n\}$ and a target value $V$, find the fewest coins needed to make change for $V$ (or report impossible).

Recall. Greedy cashier’s algorithm is optimal for U.S. coin denominations, but not for arbitrary coin denominations.

Ex. $\{1, 10, 21, 34, 70, 100, 350, 1295, 1500\}$. Optimal. $140\text{¢} = 70 + 70$. 
**Coin Changing**

**Def.** \( \text{OPT}(v) = \min \text{ number of coins to make change for } v. \)

**Goal.** \( \text{OPT}(V). \)

**Multiway choice.** To compute \( \text{OPT}(v) \),
Select a coin of denomination \( c_i \) for some \( i \).
Select fewest coins to make change for \( v - c_i. \)

**Recurrence:**

\[
\text{OPT}(v) = \begin{cases} 
\infty & \text{if } v < 0 \\
0 & \text{if } v = 0 \\
\max_{1 \leq i \leq n} \{ 1 + \text{OPT}(v - c_i) \} & \text{otherwise}
\end{cases}
\]

**Running time.** \( \mathcal{O}(n V) \).
Once again, can be formulated as shortest path on an appropriately chosen graph.
Once again, can be formulated as shortest path on an appropriately chosen graph.

All edge weights = 1

Smallest number of coins to produce $V = \text{shortest path from 0 to } V.$
Once again, can be formulated as shortest path on an appropriately chosen graph.

All edge weights = 1

Smallest number of coins to produce $V = \text{shortest path from 0 to } V$. Is time $\sim n^V$ “efficient”? 
Back to graph problems

• Bellman-Ford solves the shortest path problem.
• Calculates $D(s, v)$: the distance from a given source $s$ to all vertices $v$.
• Solution idea: proceed in rounds; in each round “relax” edges from each vertex in sequence.
• What is produced after $k$ such relaxations?
Relax order: A B C D E F G H
Vertex distances after 3 rounds?
Round 1

Relax order: A B C D E F G H
D->C; D->H
Round 2

Relax order: A B C D E F G H

C→B; C→G; C→H
Round 2

Relax order: A B C D E F G H
C->B; C->G; C->H; D hasn’t changed
Round 2

Relax order: A B C D E F G H
C->B; C->G; C->H; G->F; G->H
Relax order: A B C D E F G H
C->B; C->G; C->H; G->F; G->H
Round 3

Relax order: A B C D E F G H
B->A; B->F; B->G
Relax order: A B C D E F G H
B->A; B->F; B->G
Relax order: A B C D E F G H
B->A; B->F; B->G; C, D haven’t changed
Round 3

Relax order: A B C D E F G H
B->A; B->F; B->G; F->E; F->A
Relax order: A B C D E F G H
B→A; B→F; B→G; F→E; F→A
Round 3

Relax order: A B C D E F G H
B->A; B->F; B->G; F->E; F->A; G->F; G->H
Round 3

Relax order: A B C D E F G H
B->A; B->F; B->G; F->E; F->A; G->F; G->H
Calculates $D(s, v)$: the distance from a given source $s$ to all vertices $v$.

Solution idea: proceed in rounds; in each round “relax” edges from each vertex in sequence. Assume no negative cycles.

What is produced after $k$ such relaxations?

$BF(s, v, k) \leq$ the length of the shortest path from $s$ to $v$ with at most $k$ hops [why not $=$?]

$D(s, v) = BF(s, v, V)$
Bellman-Ford as DP

- Subproblem: paths with at most $k$ hops.
- Relationship:

  $$D(s, v, 0) = \begin{cases} 
  0 & \text{if } s = v \\
  \infty & \text{otherwise}
  \end{cases}$$

  $$D(s, v, k + 1) = \min_{u \to v}(D(s, u, k) + c(u \to v))$$

- Note: standard implementation does not compute $D(s, v, k)$, but something potentially smaller.
All-pairs shortest path

• Given a digraph $G$ with no negative cycles, want to compute a table $D(u, v)$ of all shortest distances.
• $D(u, v, k) =$?
• The hard part is coming up with the “right” subproblems!
• Number of steps: possible, but less efficient.
Floyd–Warshall algorithm

- \( D(u, v, k) = \) the shortest path where the only intermediate vertices allowed are vertices \( \{1, ..., k\} \) (assume the vertices are \( \{1, ..., n\} \))
- \( D(u, v) = D(u, v, n) \)
- \( D(u, v, 0) = c(u \rightarrow v) \)
- \( D(u, v, k + 1) = \min(D(u, v, k), D(u, k + 1, k) + D(k + 1, v, k)) \)
- Running time?
- \( O(V^3) \) (compared to \( O(VE) \) for BF).
Subproblem by # of hops

- \( L(u, v, k) = \) the shortest path with at most \( k \) hops
- \( L(u, v) = L(u, v, n) \) (in fact, \( L(u, v, n - 1) \))
- \( L(u, v, 1) = c(u \rightarrow v) \)
- \( L(u, v, k + 1) \)
  \[ = \min_w (D(u, w, k) + c(w \rightarrow v)) \]

- Running time?
- \( O(VE) \) per update, \( V \) updates, \( (V^2E) \) total.
  (compared to \( O(VE) \) for BF: same as running it \( V \) times).
Better update?

- $L(u, v, k) =$ the shortest path with at most $k$ hops
- $L(u, v) = L(u, v, n)$
- $L(u, v, 1) = c(u \rightarrow v)$
- $L(u, v, 2k) = \min_{w} (D(u, w, k) + D(w, v, k))$
- Cost per update: $O(V^3)$
- Number of updates: $O(\log V)$
- Total cost $O(V^3 \log V)$ (compared to $O(V^3)$ for Floyd–Warshall).
Dynamic programming: summary

• Find good subproblems
• Come up with a recursive solution (recurrence relation)
• Use memorization to avoid running time blowup.
• Most important: the right subproblems
• Secondary: better update and storage strategies