Lecture Overview

- Fibonacci Warmup
- Memoization and subproblems
- Shortest Paths
- Crazy Eights
- Guessing Viewpoint

Readings

CLRS 15

Dynamic Programming (DP)

Big idea: hard yet simple

- Powerful algorithmic design technique
- Large class of seemingly exponential problems have a polynomial solution ("only")
  via DP
- Particularly for optimization problems (min / max) (e.g., shortest paths)

* DP ≈ “controlled brute force”
* DP ≈ recursion + re-use

Fibonacci Numbers

\[ F_1 = F_2 = 1; \quad F_n = F_{n-1} + F_{n-2} \]

Naive Algorithm

follow recursive definition

\[
\begin{align*}
\text{fib}(n): \\
& \text{if } n \leq 2: \text{ return 1} \\
& \text{else return fib}(n - 1) + \text{fib}(n - 2) \\
\implies & T(n) = T(n - 1) + T(n - 2) + O(1) \approx \phi^n \\
& \geq 2T(n - 2) + O(1) \geq 2^{n/2} \\
& \text{EXPONENTIAL - BAD!}
\end{align*}
\]
Simple Idea

memoize

\[
\text{memo} = \{ \}
\]
\[
\text{fib}(n):
\]
\[
\begin{array}{ll}
\text{if } n \text{ in memo: return memo}[n] \\
\text{else: if } n \leq 2 : f = 1 \\
\text{else: } f = \text{fib}(n - 1) + \underbrace{\text{fib}(n - 2)}_{\text{free}} \\
\end{array}
\]
\[
\text{memo}[n] = f \\
\text{return } f
\]
\[
T(n) = T(n - 1) + O(1) = O(n)
\]

* DP $\approx$ recursion + memoization

- remember (memoize) previously solved “subproblems” that make up problem
  - in Fibonacci, subproblems are $F_0, F_1, \cdots, F_n$
- if subproblem already solved, re-use solution

* $\implies$ time = $\frac{\# \text{ of subproblems}}{\text{time/subproblem}}$

- in fib: $\frac{\# \text{ of subproblems}}{}$ is $O(n)$ and time/subproblem is $O(1)$ - giving us a total time of $O(n)$.

\[\text{Figure 1: Naive Fibonacci Algorithm}\]
Shortest Paths

• Recursive formulation:
  \[ \delta(s, t) = \min \{ w(s, v) + \delta(v, t) \mid (s, v) \in E \} \]

• does this work with memoization?
  no, cycles \( \Rightarrow \) infinite loops (see Figure 2).

\[ \text{Figure 2: Shortest Paths} \]

• in some sense necessary for neg-weight cycles

• works for directed acyclic graphs in \( O(V + E) \)
  (recursion effectively DFS/topological sort)

• trick for shortest paths: removing cyclic dependency.
  
  \(- \quad \delta_k(s, t) = \text{shortest path using } \leq k \text{ edges} \)
  
  \[ \begin{align*}
  &= \min \{ \delta_{k-1}(s, t) \} \cup \{ w(s, v) + \delta_{k-1}(v, t) \mid (s, v) \in E \} \\
  \ldots \text{except } \delta_k(t, t) &= \phi, \quad \delta_\phi(s, t) = \infty \text{ if } s \neq t \\
  \delta(s, t) &= \delta_{n-1}(s, t) \text{ assuming no negative cycles} \\
  \end{align*} \]

  \[ \implies \text{time} = \underbrace{O(n^3)}_{\text{for } s, t, k \ldots} \cdot \underbrace{O(n^2)}_{\text{for really } O(n^2)} \cdot \underbrace{O(n)}_{\text{really } \deg V} \]

  \[ = O(V \cdot \sum_{V} \deg(V)) = O(VE) \]

* Subproblem dependency should be acyclic.
Crazy Eights Puzzle

- given a sequence of cards \( c[\phi], c[1], \ldots, c[n-1] \)
e.g., \( 7\heartsuit, 6\heartsuit, 7\diamondsuit, 3\diamondsuit, 8\clubsuit, J\spadesuit \)
- find longest left-to-right “trick” (subsequence)

\[
c[i_1], c[i_2], \ldots c[i_k] \quad (i_1 < i_2 < \cdots i_k)
\]
where \( c[i_j] \& c[i_{j+1}] \) “match” for all \( j \)
have some suit or rank or one has rank 8

- recursive formulation:

\[
\text{trick}(i) = \text{length of best trick starting at } c[i]
= \ 1 + \max(\text{trick}(j) \text{ for } j \text{ in range}(i+1,n) \text{ if match } (c[i], c[j]))
\]
best \( = \max(\text{trick}(i) \text{ for } i \text{ in range}(n)) \)

- memoize: \( \text{trick}(i) \) depends only on \( \text{trick}(>i) \)

\[
\Longrightarrow \text{time} = \frac{\# \text{subproblems}}{O(n)} \cdot \frac{\text{time/subproblem}}{O(n)}
= O(n^2) \quad \text{(to find actual trick, trace through max’s)}
\]

“Guessing” Viewpoint

- what is the first card in best trick? guess!
i.e., try all possibilities & take best result
  - only \( O(n) \) choices

- what is next card in best trick from \( i \)? guess!
  - if you pretend you knew, solution becomes easy (using other subproblems)
  - actually pay factor of \( O(n) \) to try all

- * use only small \( \frac{\# \text{choices/guesses per subproblem}}{\text{poly}(n)\sim O(1)} \)