Lecture 18: Pseudopolynomial

Dynamic Programming Steps (SRT BOT)

1. **Subproblem** definition subproblem \( x \in X \)
   - Describe the meaning of a subproblem *in words*, in terms of parameters
   - Often subsets of input: prefixes, suffixes, contiguous substrings of a sequence
   - Often multiply possible subsets across multiple inputs
   - Often record partial state: add subproblems by incrementing some auxiliary variables
   - Often smaller integers than a given integer (*today’s focus*)

2. **Relate** subproblem solutions recursively \( x(i) = f(x(j), \ldots) \) for one or more \( j < i \)
   - Identify a question about a subproblem solution that, if you knew the answer to, reduces the subproblem to smaller subproblem(s)
   - Locally brute-force all possible answers to the question

3. **Topological order** to argue relation is acyclic and subproblems form a DAG

4. **Base** cases
   - State solutions for all (reachable) independent subproblems where relation breaks down

5. **Original problem**
   - Show how to compute solution to original problem from solutions to subproblem(s)
   - Possibly use parent pointers to recover actual solution, not just objective function

6. **Time** analysis
   - \( \sum_{x \in X} \text{work}(x) \), or if \( \text{work}(x) = O(W) \) for all \( x \in X \), then \( |X| \cdot O(W) \)
   - \( \text{work}(x) \) measures **nonrecursive** work in relation; treat recursions as taking \( O(1) \) time
Rod Cutting

• Given a rod of length $L$ and value $v(\ell)$ of rod of length $\ell$ for all $\ell \in \{1, 2, \ldots, L\}$
• Goal: Cut the rod to maximize the value of cut rod pieces
• Example: $L = 7$, $v = [0, 1, 10, 13, 18, 20, 31, 32]$

• Maybe greedily take most valuable per unit length?
• Nope! $\arg \max_\ell v[\ell] / \ell = 6$, and partitioning $[6, 1]$ yields 32 which is not optimal!
• Maximization problem on value of partition

1. Subproblems
   • $x(\ell)$: maximum value obtainable by cutting rod of length $\ell$
   • For $\ell \in \{0, 1, \ldots, L\}$

2. Relate
   • First piece has some length $p$ (Guess!)
   • $x(\ell) = \max\{v(p) + x(\ell - p) \mid p \in \{1, \ldots, \ell\}\}$
   • (draw dependency graph)

3. Topological order
   • Increasing $\ell$: Subproblems $x(\ell)$ depend only on strictly smaller $\ell$, so acyclic

4. Base
   • $x(0) = 0$ (length-zero rod has no value!)

5. Original problem
   • Maximum value obtainable by cutting rod of length $L$ is $x(L)$
   • Store choices to reconstruct cuts
   • If current rod length $\ell$ and optimal choice is $\ell'$, remainder is piece $p = \ell - \ell'$
   • (maximum-weight path in subproblem DAG!)

6. Time
   • # subproblems: $L + 1$
   • work per subproblem: $O(\ell) = O(L)$
   • $O(L^2)$ running time
Is This Polynomial Time?

- **(Strongly) polynomial time** means that the running time is bounded above by a constant-degree polynomial in the input size measured in words.

- In Rod Cutting, input size is \( L + 1 \) words (one integer \( L \) and \( L \) integers in \( v \))

- \( O(L^2) \) is a constant-degree polynomial in \( L + 1 \), so YES: (strongly) polynomial time

```python
# recursive
def cut_rod(l, v):
    if l < 1: return 0  # base case
    if l not in x:      # check memo
        for piece in range(1, l + 1):  # try piece
            x_ = v[piece] + cut_rod(l - piece, v)  # recurrence
            if (l not in x) or (x[l] < x_):  # update memo
                x[l] = x_
    return x[l]

# iterative
def cut_rod(L, v):
    x = [0] * (L + 1)  # base case
    for l in range(1, L + 1):  # topological order
        for piece in range(1, l + 1):  # try piece
            x_ = v[piece] + x[l - piece]  # recurrence
            if x[l] < x_:  # update memo
                x[l] = x_
    return x[L]

# iterative with parent pointers
def cut_rod_pieces(L, v):
    x = [0] * (L + 1)  # base case
    parent = [None] * (L + 1)  # parent pointers
    for l in range(1, L + 1):  # topological order
        for piece in range(1, l + 1):  # try piece
            x_ = v[piece] + x[l - piece]  # recurrence
            if x[l] < x_:  # update memo
                x[l] = x_
        parent[l] = l - piece  # update parent
    l, pieces = L, []
    while parent[l] is not None:  # walk back through parents
        piece = l - parent[l]
        pieces.append(piece)
        l = parent[l]
    return pieces
```
Subset Sum

- Input: Sequence of \( n \) positive integers \( A = \{a_0, a_1, \ldots, a_n\} \)
- Output: Is there a subset of \( A \) that sums exactly to \( T \)? (i.e., \( \exists A' \subseteq A \) s.t. \( \sum_{a \in A'} a = T \))
- Example: \( A = (1, 3, 4, 12, 19, 21, 22), T = 47 \) allows \( A' = \{3, 4, 19, 21\} \)
- Optimization problem? Decision problem! Answer is YES or NO, TRUE or FALSE
- In example, answer is YES. However, answer is NO for some \( T \), e.g., \( 2, 6, 9, 10, 11, \ldots \)

1. Subproblems
   - \( x(i, t) = \text{does any subset of } A[i:] \text{ sum to } t? \)
   - For \( i \in \{0, 1, \ldots, n\}, t \in \{0, 1, \ldots, T\} \)

2. Relate
   - Idea: Is first item \( a_i \) in a valid subset \( A' \)? (Guess!)
   - If yes, then try to sum to \( t \) \( a_i \) 0 using remaining items
   - If no, then try to sum to \( t \) using remaining items
   - \( x(i, t) = \text{OR} \left\{ \begin{array}{ll} x(i + 1, t) & A[i] \text{ if } t \geq A[i] \\ x(i + 1, t) & \text{always} \end{array} \right. \)

3. Topological order
   - Subproblems \( x(i, t) \) only depend on strictly larger \( i \), so acyclic
   - Solve in order of decreasing \( i \)

4. Base
   - \( x(i, 0) = \text{YES} \) for \( i \in \{0, \ldots, n\} \) (space packed exactly!)
   - \( x(0, t) = \text{NO} \) for \( j \in \{1, \ldots, T\} \) (no more items available to pack)

5. Original problem
   - Original problem given by \( x(0, T) \)
   - Example: \( A = (3, 4, 3, 1), T = 6 \) solution: \( A' = (3, 3) \)
   - Bottom up: Solve all subproblems (Example has 35)
Lecture 18: Pseudopolynomial

- Top down: Solve only **reachable** subproblems (Example, only $14!$)

6. **Time**

- # subproblems: $O(nT)$, $O(1)$ work per subproblem, $O(nT)$ time
Is This Polynomial?

- Input size is \( n + 1 \): one integer \( T \) and \( n \) integers in \( A \)

- Is \( O(nT) \) bounded above by a polynomial in \( n + 1 \)? NO, not necessarily

- On \( w \)-bit word RAM, \( T \leq 2^w \) and \( w = \lg(n + 1) \), but we don’t have an upper bound on \( w \)

- E.g., \( w = n \) is not unreasonable, but then running time is \( O(2^n) \), which is exponential

Pseudopolynomial

- Algorithm has pseudopolynomial time: running time is bounded above by a constant-degree polynomial in input size and input integers

- Such algorithms are polynomial in the case that integers are polynomially bounded in input size, i.e., \( n^{O(1)} \) (same case that Radix Sort runs in \( O(n) \) time)

- Counting sort \( O(n + u) \), radix sort \( O(n \log_n u) \), direct-access array build \( O(n + u) \), and Fibonacci \( O(n) \) are all pseudopolynomial algorithms we’ve seen already

- Radix sort is actually weakly polynomial (a notion in between strongly polynomial and pseudopolynomial): bounded above by a constant-degree polynomial in the input size measured in bits, i.e., in the logarithm of the input integers

- Contrast with Rod Cutting, which was polynomial
  
  - Had pseudopolynomial dependence on \( L \)
  
  - But luckily had \( L \) input integers too
  
  - If only given subset of sellable rod lengths (Knapsack Problem, which generalizes Rod Cutting and Subset Sum — see recitation), then algorithm would have been only pseudopolynomial

Complexity

- Is Subset Sum solvable in polynomial time when integers are not polynomially bounded?

- No if \( P \neq \text{NP} \). What does that mean? Next lecture!
Main Features of Dynamic Programs

- Review of examples from lecture

- Subproblems:
  - Prefix/suffixes: Bowling, LCS, LIS, Floyd–Warshall, Rod Cutting (coincidentally, really Integer subproblems), Subset Sum
  - Substrings: Alternating Coin Game, Arithmetic Parenthesization
  - Multiple sequences: LCS
  - Integers: Fibonacci, Rod Cutting, Subset Sum
    * Pseudopolynomial: Fibonacci, Subset Sum
  - Vertices: DAG shortest paths, Bellman–Ford, Floyd–Warshall

- Subproblem constraints/expansion:
  - Nonexpansive constraint: LIS (include first item)
  - $2 \times$ expansion: Alternating Coin Game (who goes first?), Arithmetic Parenthesization (min/max)
  - $\Theta(1) \times$ expansion: Piano Fingering (first finger assignment)
  - $\Theta(n) \times$ expansion: Bellman–Ford (# edges)

- Relation:
  - Branching = # dependant subproblems in each subproblem
  - $\Theta(1)$ branching: Fibonacci, Bowling, LCS, Alternating Coin Game, Floyd–Warshall, Subset Sum
  - $\Theta$ (degree) branching (source of $|E|$ in running time): DAG shortest paths, Bellman–Ford
  - $\Theta(n)$ branching: LIS, Arithmetic Parenthesization, Rod Cutting
  - Combine multiple solutions (not path in subproblem DAG): Fibonacci, Floyd–Warshall, Arithmetic Parenthesization

- Original problem:
  - Combine multiple subproblems: DAG shortest paths, Bellman–Ford, Floyd–Warshall, LIS, Piano Fingering