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 ℓ for all $\ell \in \{1, 2, \ldots, L\}$
- Goal: Cut the rod to maximize the value of cut rod pieces
- Example: $L = 7$, $v \underset{\ell =}{=} [\underset{0}{0}, \underset{1}{1}, \underset{2}{10}, \underset{3}{13}, \underset{4}{18}, \underset{5}{20}, \underset{6}{31}, \underset{7}{32}]$
- 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!
- Solution: $v[2] + v[2] + v[3] = 10 + 10 + 13 = 33$
- Maximization problem on value of partition

1. **Subproblems**
   - $\boxed{x(\ell)\text{: maximum value obtainable by cutting rod of length } \ell}$
   - 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 ℓ: Subproblems $x(\ell)$ depend only on strictly smaller ℓ, 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 ℓ and optimal choice is ℓ', 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
x = {}
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]
```

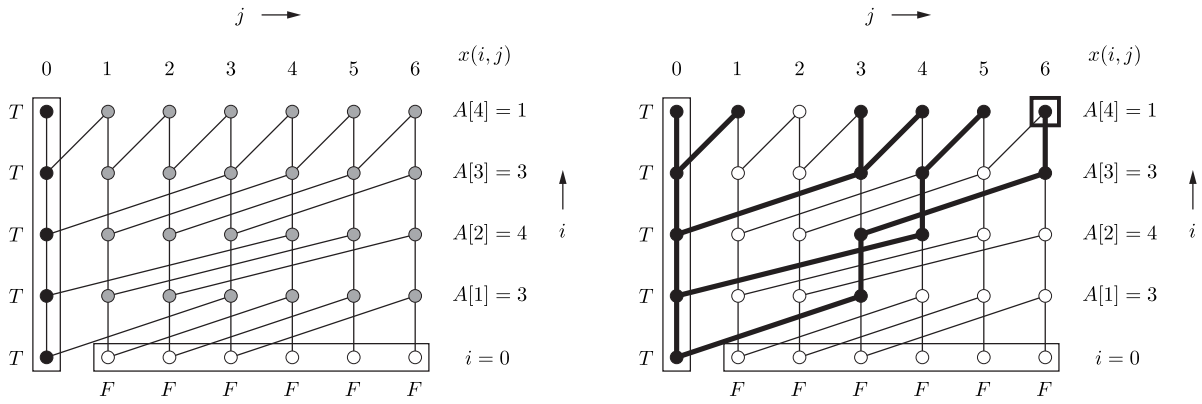
```python
# iterative
def cut_rod(L, v):
    x = [0] * (L + 1)                                 # base case
    for l in range(L + 1):                            # topological order
        for piece in range(1, l + 1):                 # try piece
            x_ = v[piece] + x[l - piece]             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

```python
# 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
```

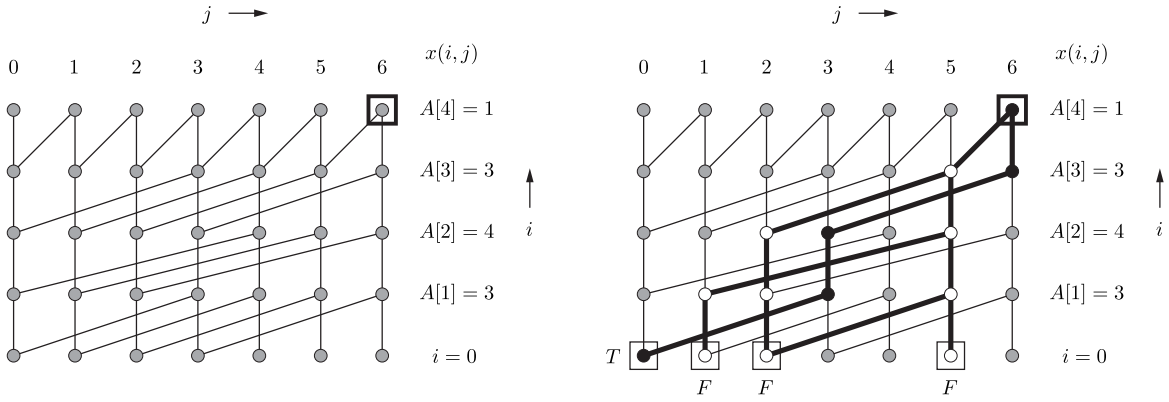
Subset Sum

- 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**
   - $\boxed{x(i,t) = \text{does any subset of } A[i:] \text{ sum to } t?}$
   - 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 $t - a_i \geq 0$ using remaining items
   - If no, then try to sum to t using remaining items
   - $x(i,t) = \text{OR} \begin{cases} x(i+1, t - A[i]) & \text{if } t \geq A[i] \\ x(i+1, t) & \text{always} \end{cases}$
3. **Topological order**
   - Subproblems $x(i,t)$ only depend on strictly larger i, so acyclic
   - Solve in order of decreasing i
4. **Base**
   - $x(i, 0) =$ YES for $i \in \{0, \ldots, n\}$ (space packed exactly!)
   - $x(0, t) =$ 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)



- 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 $w \geq \lg(n + 1)$, but we don't have an upper bound on w
- E.g., $w = n$ is not unreasonable, but then running time is $O(n2^n)$, 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 $\geq L$ 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 $\neq$ 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)
  - $\mathbf{2}\times$ **expansion:** Alternating Coin Game (who goes first?), Arithmetic Parenthesization (min/max)
  - $\boldsymbol{\Theta(1)}\times$ **expansion:** Piano Fingering (first finger assignment)
  - $\boldsymbol{\Theta(n)}\times$ **expansion:** Bellman–Ford (# edges)
- **Relation:**
  - **Branching** = # dependant subproblems in each subproblem
  - $\boldsymbol{\Theta(1)}$ **branching:** Fibonacci, Bowling, LCS, Alternating Coin Game, Floyd–Warshall, Subset Sum
  - $\boldsymbol{\Theta(\text{degree})}$ **branching** (source of $|E|$ in running time): DAG shortest paths, Bellman–Ford
  - $\boldsymbol{\Theta(n)}$ **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

6.006	Lecture 18	Apr. 28, 2020

TODAY: Dynamic Programming IV (of 4)
- integer subproblems
- pseudopolynomial time
- examples → rod cutting
  → subset sum

Recall: SRTBOT paradigm for recursive alg. design & (with memoization) for DP alg. design
- Subproblem definition
  - for sequence S, try → prefixes S[:i] $\Theta(n)$
    → suffixes S[i:] $\Theta(n)$
    → substrings S[i:j] $\Theta(n^2)$
  - product for multiple
  - for nonneg. integer K, try integers in [0, K]
  - add subproblems & constraints to "remember state"
- Relate subproblem solutions recursively
  - identify question about subproblem solution that, if you knew answer, reduces to "smaller" subprob(s)
  - locally brute-force all answers to that question
  - can think of correctly guessing answer, then loop
- Topological order on subproblems to guarantee acyclic relation
  → subproblem/call DAG (a) → (b) ≡ b needs a
- Base cases of relation
- Original problem: solve via subproblem(s)
- Time analysis $\leq \sum_{\text{subprob.}}$ nonrec. work in relation
  ≤ # subproblems · nonrecursive work in relation + work to solve original

Rod cutting: given rod of length L (or hardwood board etc.) & value $v(\ell)$ of rod of length ℓ for all $\ell \in \{1, 2, \ldots, L\}$, what's max-value partition of length-L rod?

sale price

↳ cuts into rods whose lengths sum to L

- example: $L = 7$,

ℓ:	1	2	3	4	5	6	7
$v(\ell)$:	1	10	13	18	20	31	32

  - $2+2+3 \to 10+10+13 = 33$ value — optimal
  - $6+1 \to 31+1 = 32$ value — suboptimal
    - ↳ max $v(\ell)/\ell$ "bang for buck"

Subproblems for nonneg. integer input K: integers $\in [0, K]$ (recall Fibonacci)

- Subproblems: $X(\ell)$ = max. value of partition of rod of length ℓ, for $0 \le \ell \le L$
  (in this case, corresponds to prefix of v ~ coincidence)
- Relate: $X(\ell) = \max\{v(p) + X(\ell - p) \mid 1 \le p \le \ell\}$
  - guessing "length of a piece we sell"
- Topo. order: increasing ℓ i.e. for $\ell = 0, 1, 2, \ldots, L$
- Base case: $X(0) = 0$
- Original: $X(L)$
- Time: $\Theta(L)$ subprobs. · $O(L)$ nonrec. work in relation $= O(L^2)$

(Strongly) polynomial time ≡ time $\le O(1)$-degree polynomial in input size (in words)

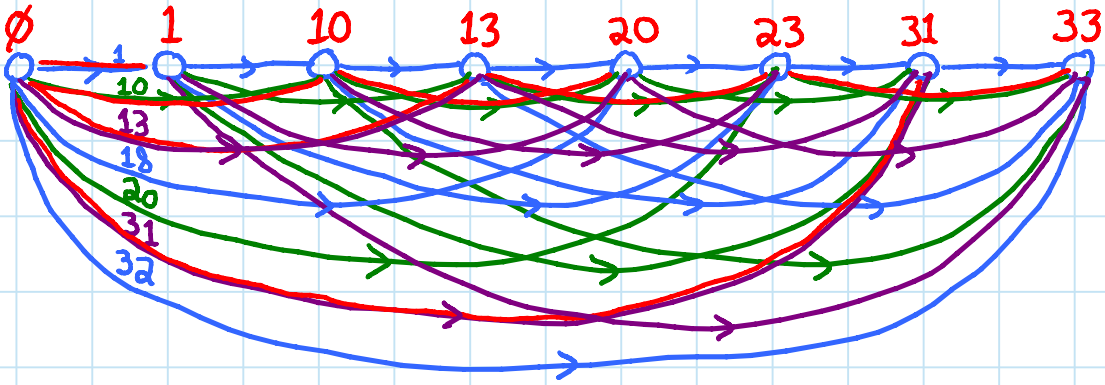
↳ here, $L+1$ ⇒ quadratic time ☺

Subproblem DAG: $L = 7$.

ℓ:	1	2	3	4	5	6	7
$v(\ell)$:	1	10	13	18	20	31	32

ℓ: $\emptyset$ 1 2 3 4 5 6 7

$v(1) = 1$
$v(2) = 10$
$v(3) = 13$
$v(4) = 18$
$v(5) = 20$
$v(6) = 31$
$v(7) = 32$

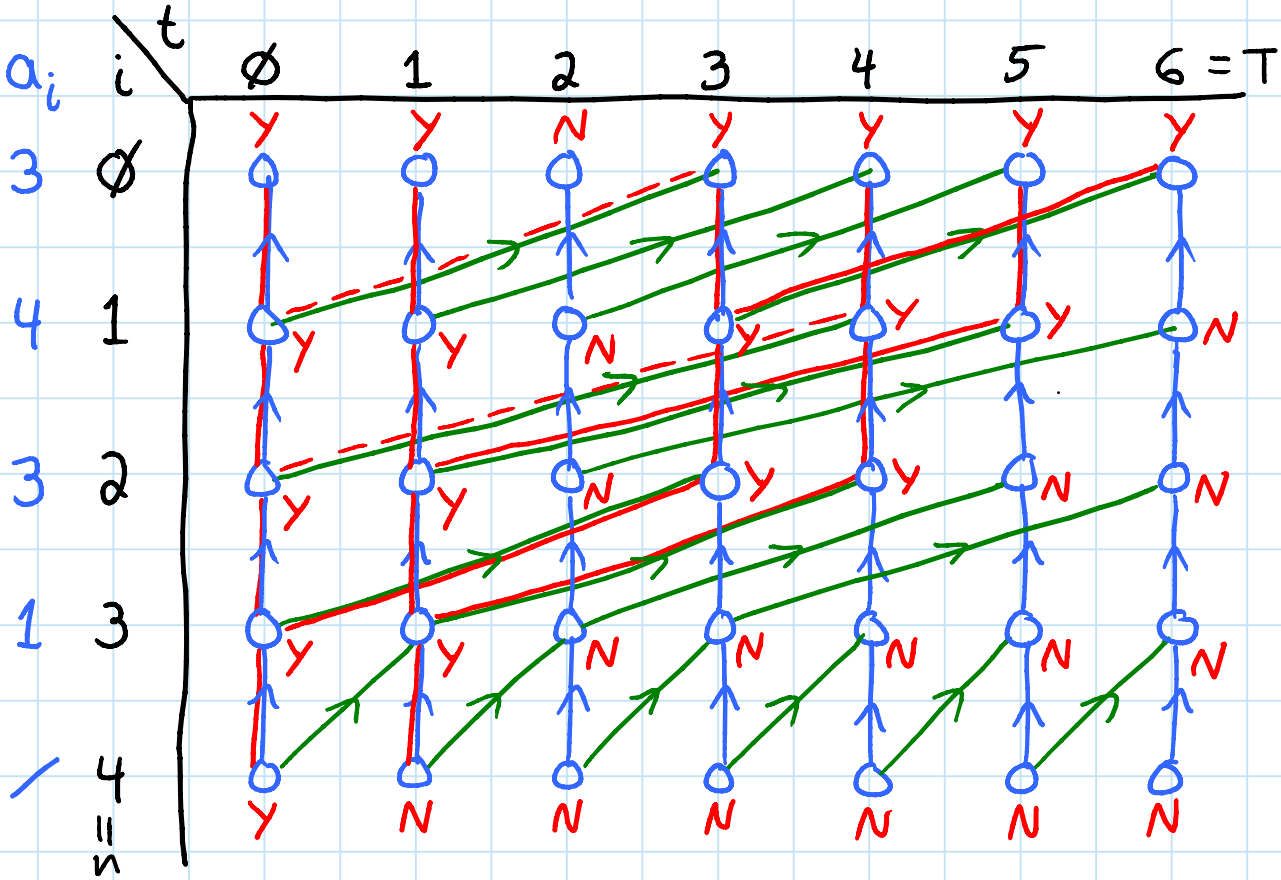


- equivalent to longest path in DAG from $\ell = \emptyset$ vertex to $\ell = L$ vertex

Subset sum: given n positive integers $A = \{a_0, a_1, \ldots, a_{n-1}\}$ & target sum T, does any subset $S \subseteq A$ sum to T?

- example: $A = \{2, 5, 7, 8, 9\}$
  - $T = 21 \Rightarrow$ YES: $S = \{5, 7, 9\}$ Blackjack?
  - $T = 25 \Rightarrow$ NO
- decision problem: YES/NO answer
  (cf. most DPs we've seen, solving optimization probs.)
- Subproblems: $X(i, t)$ = does any subset of $X[i:]$ sum to t?
- Relate: $X(i, t) = \text{OR}\{X(i+1, t),$ ← $a_i \notin S$
  any() in Python ↵ $X(i+1, t-a_i)$ if $a_i \le t\}$ ← $a_i \in S$
  - guessing "is $a_i \in S$?"
- Topo. order: decreasing i (calls always increase i; t sometimes doesn't change)
- Base cases: $X(n, t) = \begin{cases} \text{YES} & \text{if } t = \emptyset \\ \text{NO} & \text{otherwise} \end{cases}$
- Original: $X(\emptyset, T)$
  - can also solve e.g. "how close to T can I come?"
  - can recover subset S via parent pointers
- Time: $\Theta(n \cdot T)$ subprobs $\cdot$ $O(1)$ nonrec. work in relation
  $= O(n \cdot T)$

Subproblem DAG: $A = \{3, 4, 3, 1\}$ multiset! $T = 6$



Is $O(nT)$ polynomial? NO

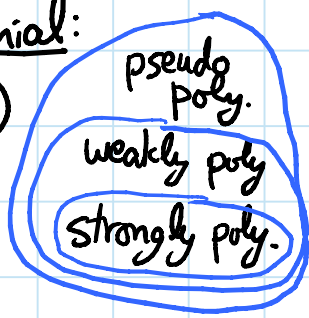
- input size $= n + 1$ (words)
- T can be 2^w, potentially huge w.r.t. n
  - only know $w \geq \lg n$ ~ can be much larger
  - e.g. if $w = n$, T can be exponential in n

Pseudopolynomial: $O(1)$-degree polynomial in input size & input integers

$\Rightarrow$ polynomial if integers are polynomially bounded ($\hookrightarrow \leq n^{O(1)}$)

(same case when Radix Sort runs in $O(n)$ time)

- counting & radix sort, direct-access array, Fibonacci
  $O(n+u)$ $O(n \log_n u)$ $O(n+u)$ build ($\hookrightarrow$ hashing is poly.) $O(n)$, input = $\{n\}$, size = 1 (word) ($\hookrightarrow \exists$ poly. alg.)
  are all pseudopoly.
- radix sort is actually weakly polynomial:
  $O(1)$-degree polynomial in log(input integers)
  i.e. input size in bits



Cf. rod cutting: which was polynomial

- had pseudopoly. dependence on L
- but luckily had $\geq L$ input integers too
- if only given subset of sellable lengths
  (KNAPSACK PROBLEM - also generalizes Subset Sum)
  then algorithm would only be pseudopolynomial

Does Subset Sum have a polynomial-time alg.?
NO, assuming $P \neq NP$ ~ next lecture!

Main features of DPs: review of examples from lecture

- subproblems
  - prefix/suffixes: Bowling, LCS, LIS, Floyd-Warshall, (Rod Cutting), Subset Sum
  - substrings: Alternating Coin Game, Parenthesization
  - multiple sequences: LCS
  - integers: Fibonacci, Rod Cutting, Subset Sum
    - pseudopolynomial: Fibonacci, Subset Sum
  - vertices: DAG SPs, Bellman-Ford, Floyd-Warshall (↗ on vertex set)
- subproblem constraints/expansion
  - nonexpansive constraint: LIS (include first item)
  - 2x: Alternating Coin Game (who goes first?), Parenthesization (min/max)
  - $\Theta(1)$x: Piano Fingering (first finger assignment)
  - $\Theta(n)$x: Bellman-Ford (# edges)
- relation (↗ # dependant subproblems)
  - $\Theta(1)$ branching: Fibonacci, Bowling, LCS, Coin Game, Floyd-Warshall, Subset Sum
  - Θ(degree) branching: DAG SPs, Bellman-Ford → $|E|$ in run time
  - $\Theta(n)$ branching: LIS, Parenthesization, Rod Cutting
  - combine multiple solutions (not path in DAG): Fibonacci, Floyd-Warshall, Parenthesization
- original
  - combine multiple subproblems: DAG SPs, Bellman-Ford, Floyd-Warshall, LIS, Piano Fingering