Lecture 17: Dyn. Prog. III

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
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

---

Recall: DAG Shortest Paths [L15]

- **S**ubproblems: $\delta(s, v)$ for all $v \in V$
- **R**elation: $\delta(s, v) = \min\{\delta(s, u) + w(u, v) \mid u \in \text{Adj}^-(v)\} \cup \{\infty\}$
- **T**opo. order: Topological order of G

Single-Source Shortest Paths Revisited

1. **Subproblems**
   - Expand subproblems to add information to make acyclic!
     (an example we've already seen of subproblem expansion)
   - $\boxed{\delta_k(s, v) = \text{weight of shortest path from } s \text{ to } v \text{ using } \textit{at most } k \text{ edges}}$
   - For $v \in V$ and $0 \leq k \leq |V|$
2. **Relate**
   - Guess last edge (u, v) on shortest path from s to v
   - $\delta_k(s, v) = \min\{\delta_{k-1}(s, u) + w(u, v) \mid (u, v) \in E\} \cup \{\delta_{k-1}(s, v)\}$
3. **Topological order**
   - Increasing k: subproblems depend on subproblems only with strictly smaller k
4. **Base**
   - $\delta_0(s, s) = 0$ and $\delta_0(s, v) = \infty$ for $v \neq s$ (no edges)
   - (draw subproblem graph)
5. **Original problem**
   - If has finite shortest path, then $\delta(s, v) = \delta_{|V|-1}(s, v)$
   - Otherwise some $\delta_{|V|}(s, v) < \delta_{|V|-1}(s, v)$, so path contains a negative-weight cycle
   - Can keep track of parent pointers to subproblem that minimized recurrence
6. **Time**
   - # subproblems: $|V| \times (|V| + 1)$
   - Work for subproblem $\delta_k(s, v)$: $O(\deg_{\text{in}}(v))$

$$\sum_{k=0}^{|V|} \sum_{v \in V} O(\deg_{\text{in}}(v)) = \sum_{k=0}^{|V|} O(|E|) = O(|V| \cdot |E|)$$

This is just **Bellman-Ford**! (computed in a slightly different order)

All-Pairs Shortest Paths: Floyd–Warshall

- Could define subproblems $\delta_k(u, v) =$ minimum weight of path from u to v using at most k edges, as in Bellman–Ford
- Resulting running time is $|V|$ times Bellman–Ford, i.e., $O(|V|^2 \cdot |E|) = O(|V|^4)$
- Know a better algorithm from L14: Johnson achieves $O(|V|^2 \log |V| + |V| \cdot |E|) = O(|V|^3)$
- Can achieve $\Theta(|V|^3)$ running time (matching Johnson for dense graphs) with a simple dynamic program, called **Floyd–Warshall**
- Number vertices so that $V = \{1, 2, \ldots, |V|\}$

1. **Subproblems**
   - $d(u, v, k) =$ minimum weight of a path from u to v that only uses vertices from $\{1, 2, \ldots, k\} \cup \{u, v\}$
   - For $u, v \in V$ and $1 \leq k \leq |V|$
2. **Relate**
   - $x(u, v, k) = \min\{x(u, k, k-1) + x(k, v, k-1), x(u, v, k-1)\}$
   - Only constant branching! No longer guessing previous vertex/edge
3. **Topological order**
   - Increasing k: relation depends only on smaller k
4. **Base**
   - $x(u, u, 0) = 0$
   - $x(u, v, 0) = w(u, v)$ if $(u, v) \in E$
   - $x(u, v, 0) = \infty$ if none of the above
5. **Original problem**
   - $x(u, v, |V|)$ for all $u, v \in V$
6. **Time**
   - $O(|V|^3)$ subproblems
   - Each $O(1)$ work
   - $O(|V|^3)$ in total
   - Constant number of dependencies per subproblem brings the factor of $O(|E|)$ in the running time down to $O(|V|)$.

Arithmetic Parenthesization

- Input: arithmetic expression $a_0 *_1 a_1 *_2 a_2 \cdots *_{n-1} a_{n-1}$
  where each a_i is an integer and each $*_i \in \{+, \times\}$
- Output: Where to place parentheses to maximize the evaluated expression
- Example: $7 + 4 \times 3 + 5 \to ((7) + (4)) \times ((3) + (5)) = 88$
- Allow **negative** integers!
- Example: $7 + (-4) \times 3 + (-5) \to ((7) + ((-4) \times ((3) + (-5)))) = 15$

1. **Subproblems**
   - Sufficient to maximize each subarray? No! $(-3) \times (-3) = 9 > (-2) \times (-2) = 4$
   - $\boxed{x(i, j, \text{opt}) = \text{opt value obtainable by parenthesizing } a_i *_{i+1} \cdots *_{j-1} a_{j-1}}$
   - For $0 \le i < j \le n$ and $\text{opt} \in \{\min, \max\}$
2. **Relate**
   - Guess location of outermost parentheses / last operation evaluated
   - $x(i, j, \text{opt}) = \text{opt}\,\{x(i, k, \text{opt}') *_k x(k, j, \text{opt}'')) \mid i < k < j; \text{opt}', \text{opt}'' \in \{\min, \max\}\}$
3. **Topological order**
   - Increasing $j - i$: subproblem $x(i, j, \text{opt})$ depends only on strictly smaller $j - i$
4. **Base**
   - $x(i, i + 1, \text{opt}) = a_i$, only one number, no operations left!
5. **Original problem**
   - $X(0, n, \max)$
   - Store parent pointers (two!) to find parenthesization (forms binary tree!)
6. **Time**
   - # subproblems: less than $n \cdot n \cdot 2 = O(n^2)$
   - work per subproblem $O(n) \cdot 2 \cdot 2 = O(n)$
   - $O(n^3)$ running time

Piano Fingering

- Given sequence $t_0, t_1, \ldots, t_{n-1}$ of n **single** notes to play with right hand (will generalize to multiple notes and hands later)
- Performer has right-hand fingers $1, 2, \ldots, F$ ($F = 5$ for most humans)
- Given metric $d(t, f, t', f')$ of **difficulty** of transitioning from note t with finger f to note t' with finger f'
  - Typically a sum of penalties for various difficulties, e.g.:
  - $1 < f < f'$ and $t > t'$ is uncomfortable
  - Legato (smooth) play requires $t \neq t'$ (else infinite penalty)
  - Weak-finger rule: prefer to avoid $f' \in \{4, 5\}$
  - $\{f, f'\} = \{3, 4\}$ is annoying
- Goal: Assign fingers to notes to minimize total difficulty
- First attempt:

1. **Subproblems**
   - $x(i) =$ minimum total difficulty for playing notes $t_i, t_{i+1}, \ldots, t_{n-1}$
2. **Relate**
   - Guess first finger: assignment f for t_i
   - $x(i) = \min\{x(i+1) + d(t_i, f, t_{i+1}, ?) \mid 1 \leq f \leq F\}$
   - Not enough information to fill in $?$
   - Need to know which finger at the start of $x(i+1)$
   - But different starting fingers could hurt/help both $x(i+1)$ and $d(t_i, f, t_{i+1}, ?)$
   - Need a table mapping start fingers to optimal solutions for $x(i+1)$
   - I.e., need to expand subproblems with start condition

- Solution:

1. **Subproblems**
   - $x(i, f) =$ minimum total difficulty for playing notes $t_i, t_{i+1}, \ldots, t_{n-1}$ starting with finger f on note t_i
   - For $0 \leq i < n$ and $1 \leq f \leq F$
2. **Relate**
   - Guess next finger: assignment f' for t_{i+1}
   - $x(i, f) = \min\{x(i+1, f') + d(t_i, f, t_{i+1}, f') \mid 1 \leq f' \leq F\}$
3. **Topological order**
   - Decreasing i (any f order)
4. **Base**
   - $x(n-1, f) = 0$ (no transitions)
5. **Original problem**
   - $\min\{x(0, f) \mid 1 \leq f \leq F\}$
6. **Time**
   - $\Theta(n \cdot F)$ subproblems
   - $\Theta(F)$ work per subproblem
   - $\Theta(n \cdot F^2)$
   - No dependence on the number of different notes!

Guitar Fingering

- Up to $S =$ number of strings different ways to play the same note
- Redefine "finger" to be tuple (finger playing note, string playing note)
- Throughout algorithm, F gets replaced by $F \cdot S$
- Running time is thus $\Theta(n \cdot F^2 \cdot S^2)$

Multiple Notes at Once

- Now suppose t_i is a set of notes to play at time i
- Given a bigger transition difficulty function $d(t, f, t', f')$
- Goal: fingering $f_i : t_i \to \{1, 2, \ldots, F\}$ specifying how to finger each note (including which string for guitar) to minimize $\sum_{i=1}^{n-1} d(t_{i-1}, f_{i-1}, t_i, f_i)$
- At most T^F choices for each fingering f_i, where $T = \max_i |t_i|$
  - $T \leq F = 10$ for normal piano (but there are exceptions)
  - $T \leq S$ for guitar
- $\Theta(n \cdot T^F)$ subproblems
- $\Theta(T^F)$ work per subproblem
- $\Theta(n \cdot T^{2F})$ time
- $\Theta(n)$ time for $T, F \leq 10$

Video Game Appliactions

- Guitar Hero / Rock Band
  - $F = 4$ (and 5 different notes)
- Dance Dance Revolution
  - $F = 2$ feet
  - $T = 2$ (at most two notes at once)
  - Exercise: handle sustained notes, using "where each foot is" (on an arrow or in the middle) as added state for suffix subproblems

6.006	Lecture 17	Apr. 23, 2020

TODAY: Dynamic Programming III (of 4)
- subproblem constraints & expansion
- examples
  - Bellman-Ford SSSP (review)
  - Floyd-Warshall APSP
  - arithmetic parenthesization
  - piano/guitar fingering

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
  - 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
  - $\leq$ # subproblems · nonrecursive work in relation + work to solve original

SSSP: example we've already seen of subprob. expansion

- DAG: [L15]
  - Subprobs: $\delta(s,v)$ for $v \in V$
  - Relate: $\delta(s,v) = \min\{\delta(s,u) + w(u,v) \mid u \in \text{Adj}^-(v)\}$
    - guessing "last edge (u,v) on shortest $s \to v$ path?"
    - same as DAG relaxation [L11] in different order
  - Topo. order: topological order of G
- general graphs: above relation has cycles
  - Subprobs: $\delta_k(s,v)$ = weight of shortest $s \to v$ path
    constraint → using $\leq k$ edges
    for $v \in V$ & $0 \leq k \leq |V|$
  - Relate: $\delta_k(s,v) = \min\{\delta_{k-1}(s,u) + w(u,v) \mid u \in \text{Adj}^-(v)\} \cup \{\delta_{k-1}(s,u)\}$ ← $<k$ edges
    - same as Bellman-Ford [L12] in different order
  - Topo. order: increasing k key: acyclic!
  - Base cases: $\delta_0(s,s) = 0$; $\delta_0(s,v) = \infty$ for $v \neq s$
  - Original: $\delta_{|V|-1}(s,v)$ for $v \in V$
    & $\delta_{|V|}(s,v)$ for neg-weight cycle detection
  - Time: $\sum_{k=0}^{|V|} \sum_{v \in V} |\text{Adj}^-(v)| = \Theta(|V| \cdot |E|)$

APSP:

- Subprobs.: $\delta_k(u,v)$ = min. weight of $\leq k$-edge $u \to v$ path
  $\Rightarrow O(|V|^2 \cdot |E|) = O(|V|^4)$ i.e. $|V| \times$ Bellman-Ford
- expensive to guess last edge of $u \to v$ path...

Floyd-Warshall:

- number vertices $V = \{1, 2, \ldots, |V|\}$
- Subprobs.: $d(u,v,k)$ = weight of shortest $u \to v$ path
  constraint → using only vertices $\in \{u,v\} \cup \{1,2,\ldots,k\}$
  for $u,v \in V$ & $0 \leq k \leq |V|$
- Relate: $d(u,v,k) = \min\{d(u,v,k-1),$ ← don't use k
  $d(u,k,k-1) + d(k,v,k-1)\}$ ← use k
  - guessing "is k on the shortest $u \to v$ path?"
- Topo. order: increasing k
  e.g. for $k = 0, 1, \ldots, |V|$:
  for $u \in V$:
  for $v \in V$:
- Base cases: $d(u,v,0) = \begin{cases} 0 & \text{if } u = v \\ w(u,v) & \text{if } (u,v) \in E \\ \infty & \text{otherwise} \end{cases}$
- Original: $d(u,v,|V|)$ for $u,v \in V$ (assuming no neg-weight cycles)
- Time: $\Theta(|V|^3)$ subprobs. $\cdot$ $\Theta(1)$ nonrec. work
  $= \Theta(|V|^3)$ – replaced $|E|$ with $|V|$!
  - matches Johnson's algorithm $O(|V|^2 \lg |V| + |V| \cdot |E|)$ for $|E| = \Theta(|V|^2)$ [but worse otherwise]
  - simple algorithm

Arithmetic Parenthesization: given arithmetic formula

$a_0 *_1 a_1 *_2 a_2 \cdots *_{n-1} a_{n-1}$,

where each $a_i \in \mathbb{Z}$ and $*_i \in \{+, \times\}$,

place parentheses to maximize outcome

- example: $7 + 4 \times 3 + 5 \Rightarrow (7 + 4) \times (3 + 5) = 88$
- idea: guess last operation $*_i$ performed
  i.e. locally brute-force which i this is
  - $\Rightarrow$ split into prefix $a_0 *_1 \cdots a_{i-1}$
    & suffix $a_i *_{i+1} \cdots a_{n-1}$
  - $\Rightarrow$ need substrings for subproblems
- can we recursively maximize prefix & suffix?
  only if $a_i \geq 0$!
  - example: $7 + -4 \times 3 + -5 \Rightarrow 7 + (-4 \times (3 + -5))$, $8 \to -2$, $= 15$
  - sometimes negative is good...
- enough to maximize absolute value?
  NO, e.g., at top level, want actual max
- suffices to compute min & max possible values

- Subprobs.: $X(i, j, \text{opt}) =$ opt value for $a_i *_{i+1} \cdots *_{j-1} a_{j-1}$
  for $0 \leq i < j \leq n$ & $\text{opt} \in \{\min, \max\}$
- Relate: $X(i, j, \text{opt}) = \text{opt}\{X(i, k, \text{opt}') *_k X(k, j, \text{opt}'') \mid i < k < j;\ \text{opt}', \text{opt}'' \in \{\min, \max\}\}$
- Topo. order: increasing $j - i$
- Base cases: $X(i, i+1, \text{opt}) = a_i$
- Original: $X(0, n, \max)$
- Time: $\Theta(n^2)$ subprobs. $\cdot$ $\Theta(n)$ choices $= \Theta(n^3)$

Piano fingering: [Parncutt, Sloboda, Clarke, Raekallio, Desain 1997] [Hart, Bosch, Tsai 2000] [Al Kasimi, Nichols, Raphael 2007] ... etc.

- given sequence $t_0, t_1, \ldots, t_{n-1}$ of n single notes to play with right hand
  (will generalize to multiple notes/hands later)
- fingers $1, 2, \ldots, F = 5$ for most humans
- metric $d(t, f, t', f')$ of difficulty of going from note t with finger f to note t' with finger f'
  e.g. (sum of penalties):
  - $1 < f < f'$ & $t > t' \Rightarrow$ uncomfortable
  - stretch rule: $t \ll t' \Rightarrow$ uncomfortable
  - legato (smooth) $\Rightarrow \infty$ if $t = t'$
  - weak-finger rule: prefer to avoid $f' \in \{4, 5\}$
  - $3 \to 4$ & $4 \to 3$ annoying ~ etc.
- goal: assign fingers to notes to minimize total difficulty ← F^n possible assignments

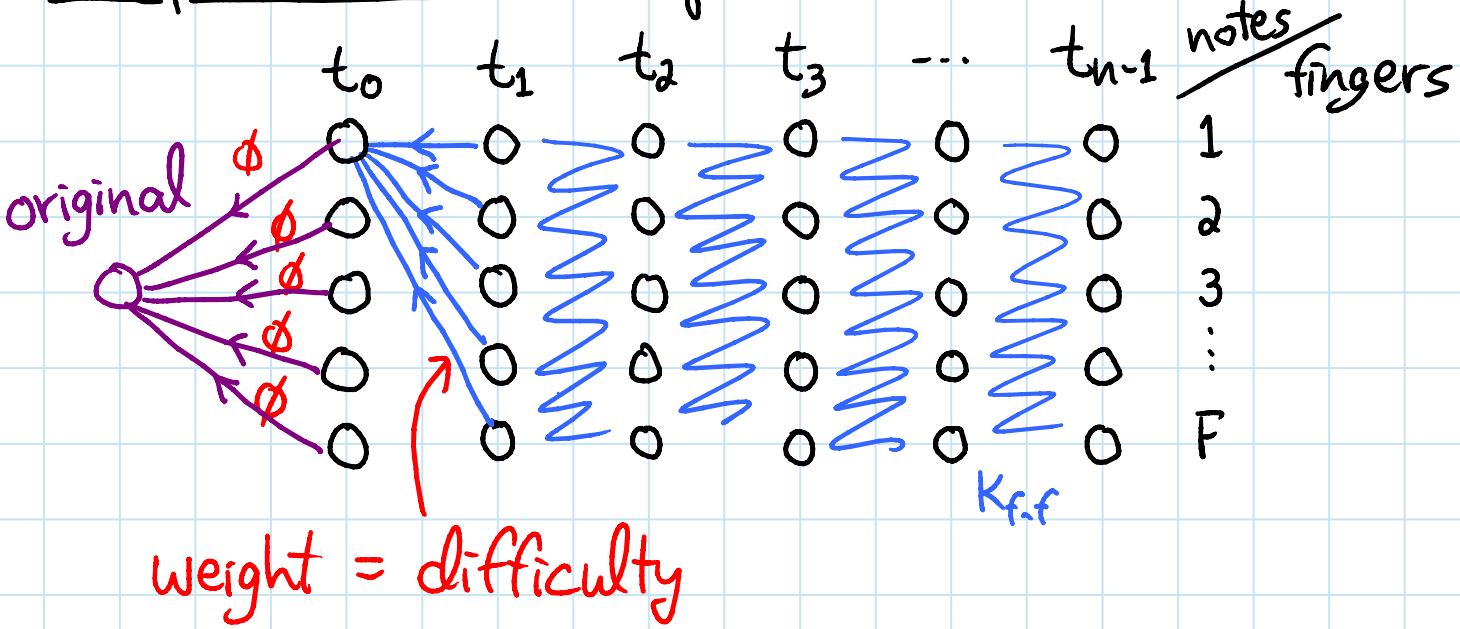
Attempt: Suffixes

- Subproblems: $X(i)$ = min. total difficulty for playing notes $t_i, t_{i+1}, \ldots, t_{n-1}$
- Relate: $X(i) = \min_{1 \le f \le F} [X(i+1) + d(t_i, f, t_{i+1}, ?)]$
  guessing "first finger?" → not enough information! ↑
  - need to know which finger at start of $X(i+1)$
  - but different starting fingers could hurt/help
  - need a table mapping start fingers → solutions
  - i.e. need to expand subprobs. with start condition

Solution:

- Subproblems: $X(i, f)$ = min. total difficulty for playing notes $t_i, t_{i+1}, \ldots, t_{n-1}$ starting with finger f on note t_i ← condition
  for $0 \le i < n$ & $1 \le f \le F$
- Relate: $X(i, f) = \min_{1 \le f' \le F} \left[X(i+1, f') + d(t_i, f, t_{i+1}, f') \right]$
  guessing "next finger?"
- Topo. order: decreasing i (any f order)
  i.e. for $i = n, n-1, \ldots, \emptyset$:
  for $f = 1, 2, \ldots, F$:
- Base cases: $X(n-1, f) = \emptyset$ (no transitions)
- Original: $\min_{1 \le f \le F} X(\emptyset, f)$
  what's first finger? guess / locally brute-force
- Time: $\Theta(nF)$ subprobs. · $\Theta(F)$ nonrec. work in relation
  $+ \Theta(F)$ for original
  $= \Theta(nF^2)$
  no dependence on # notes!

Subproblem DAG: equivalent to SSSP



Guitar fingering: up to S (# strings) ways to play same note!
- redefine "finger" = (finger playing note, string playing note)

$\Rightarrow F \to F \cdot S$
$\Rightarrow \Theta(n F^2 S^2)$ time

Multiple notes at once: (piano/guitar chords)
- input: t_i = set of notes to play at time i
  $d(t, f, t', f')$
- goal: fingering $f_i : t_i \to \{1, 2, \ldots, F\}$ (including string for guitar)
  minimizing $\sum_{i=1}^{n-1} d(t_{i-1}, f_{i-1}, t_i, f_i)$

$\Rightarrow \le F^T$ choices for fingering f — STATE
where $T = \max_i |t_i|$ $\le F = 10$ for normal piano, $\le S$ for guitar

$\Rightarrow \Theta(n F^T)$ subprobs. $\cdot$ $\Theta(F^T)$ nonrec. work
$= \Theta(n F^{2T})$
$= \Theta(n)$ for $T, F \le 10$

Guitar Hero / Rock Band: $F = 4$ (& 5 notes)

Dance Dance Revolution: $F = 2$ feet, $T = 2$ (or 4?)
- 4 notes: ⇦ ⇩ ⇧ ⇨, or 8 if using both pads
- exercise: handle sustained notes
  using state = where each foot is $\le 5^2$ or 9^2
  ↳ on arrow, or in center