EECS 336: Lecture 5: Introduction to Algorithms

Dynamic Programming (cont) Bellman-Ford

Reading: 6.4-6.8

"guide to dynamic programming" (Canvas)

Discussion: Peer grading

Last Time:

- Dynamic Programming (a framework)
- Integer Knapsack

Today:

- Sequence Alignment.
- Shortest Paths.

Sequence Alignment

"align sequences to optimize quality of alignment"

input:

- $\mathbf{a} = a_1, \ldots, a_n$ sequence of n symbols.
- $\mathbf{b} = b_1, \ldots, b_m$ sequence of m symbols.
- α_{ij} = cost of aligning a_i and b_j
- δ = gap cost.

output: alignment with minimum total cost.

example:

- $\mathbf{a}$ = "cab";
- $\mathbf{b}$ = "car";
- $\alpha = 0$ for match, 1 for mismatch
- $\delta = 0$

OPT = ...

Framework

I. identify subproblem in English

$\mathrm{OPT}(i, j)$ = "minimal number of symbols to delete to align $a_i, \ldots, a_n$ and $b_j, \ldots, b_m$"

II. specify subproblem recurrence (argue correctness)

$$\begin{aligned}\mathrm{OPT}(i,j) = \min\{&\alpha_{ij} + \mathrm{OPT}(i+1, j+1),\\ &\delta + \mathrm{OPT}(i, j+1),\\ &\delta + \mathrm{OPT}(i+1, j)\}\end{aligned}$$

III. solve the original problem from subproblems

Optimal Sequence Alignment = $\mathrm{OPT}(1, 1)$

IV. identify base case

$\mathrm{OPT}(i, m+1) = \delta(n - i)$,

$\mathrm{OPT}(n+1, j) = \delta(m - j)$.

V. write iterative DP.

VI. runtime analysis.

$O(nm)$ + initialization = $O(nm)$

VII. implement in your favorite language (Python!)

Shortest Paths with Negative Weights

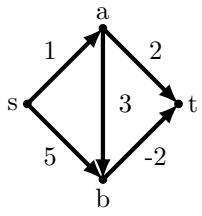
"e.g., currency exchange: nodes are currencies, path weights are exchange rates, minimize produce of path weights."

Note: to minimize product of path weights, can minimize sum of logs of path weights.

Example: $r_1 r_2 = 2^{\log_2 r_1} 2^{\log_2 r_2} = 2^{\log_2 r_1 + \log_2 r_2}$

Note: if $r \leq 1$ then $\log r$ is negative.

Example:



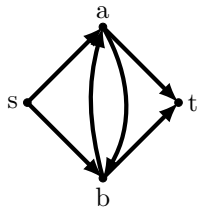
Try Dynamic Programming

OPT(v)

= shortest path from v to t.

= $\min_{u \in N(v)} [\underbrace{c(v,u)}_{weight} + \text{OPT}(u)]$.

Example:



Subproblems have cyclic dependencies!

Imposing measure of progress

"parameterize subproblems to keep track of progress"

Lemma: if G has no negative cycles, then minimum cost path is **simple** (i.e., does not repeat nodes); therefore, it has at most $n-1$ edges.

Proof: (contradiction)

- let P be the min-cost path with fewest number of edges.
- suppose (for the contraction) that P is not simple.

  $\Rightarrow P$ repeats as vertex v.
- no negative cycle $\Rightarrow$ path from v to v non-negative.

  $\Rightarrow$ can "splice out" cycle and not increase length.

  $\Rightarrow$ new path has fewer edges than p.

Idea: if simple path goes $s \rightsquigarrow v \to u \rightsquigarrow t$ then u-t path has one fewer edge than v-t path.

Part I: identify subproblem in english

OPT(v, k)

= "length of shortest path from v to t with at most k edges."

Part II: write recurrence

OPT(v, k)

= $\min_{u \in N(v)} [c(v,u) + \text{OPT}(u, k-1)]$

Correctness: lemma + induction.

Part III: solve original problem

- minimum cost path = OPT($s, n-1$).

Part IV: base case

- for all k: OPT(t, k) = 0
- for all $v \neq t$: OPT($v, 0$) = ∞.

Part V: iterative DP

Algorithm: Bellman-Ford

1. base case:

   for all k: $\text{OPT}[t, k] = 0$

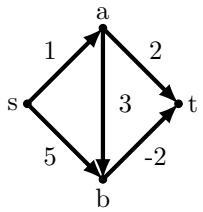
   for all $v \neq t$: $\text{OPT}[v, 0] = \infty$.

2. for $k = 1 \ldots n-1$: for all v:

   $\text{OPT}[v, k] = \min_{u \in N(v)} c(v, u) + \text{OPT}[u, k-1]$.

3. return $\text{OPT}[s, n-1]$.

Example:



	0	1	2	3
s	∞	∞	3	2
a	∞	2	1	1
b	∞	-2	-2	-2
t	0	0	0	0

Part VI: Runtime

$T(n, m) = \overbrace{\text{"size of table"}}^{n^2} \times \overbrace{\text{"cost per entry"}}^{n}$

(better accounting: $T(n, m) = O(n^2 + nm) = O(nm)$)
