EECS 336: Lecture 5: Introduction to Algorithms

Dynamic Programming (cont) Bellman-Ford

Reading: 6.4-6.8

"guide to dynamic programming" (Canvas)

Last Time:

- Dynamic Programming (a framework)
- Integer Knapsack

Today:

- Integer Knapsack (cont)
- Shortest Paths.

Recall: Integer Knapsack

intput:

- n objects $N = \{1, ..., n\}$
- s_i = size of object i (integer)
- v_i = value of object i
- C = capacity of knapsack (integer)

output:

- subset $S \subseteq N$ of objects that
  (a) fit in knapsack together

      i.e., $\sum_{i \in S} s_i \leq C$

  (b) maximize total value

      i.e., $\sum_{i \in S} v_i$

Framework

I. identify subproblem in english

   OPT(i, D) = "optimal value of knapsack with capacity D with objects $\{i, ..., n\}$"

II. specify subproblem recurrence

$$\text{OPT}(i, D) = \max(\text{OPT}(i+1, D), \underbrace{v_i + \text{OPT}(i+1, D - s_i)}_{\text{if } s_i \leq D})$$

III. solve the original problem (from subproblems)

   Optimal Integer Knapsack = OPT$(1, C)$

IV. identify base case

   for all D: OPT$(n+1, D) = 0$

V. write iterative DP.

   (see last thurs)

VI. runtime analysis.

   $O(nC)$

VII. (for homework) implement iterative DP.

   (any language most students can read. e.g., Python)

Recall Approach: Find a First Decision

"e.g., either object 1 is in the knapsack or not"

Alternative Approach: Isolate Previous Decisions

Suppose:

- already processed jobs $\{1, ..., i\}$, and
- used capacity D.

Note: previous decisions succinctly summarized by i and D

Part I: subproblem in english

$\text{OPT}(i, D) =$ "value from remaining knapsack if

- alread processed jobs $\{1, ..., i\}$
- used capacity D."

Part II: recurrence

$$\text{OPT}(i, D) = \max(\text{OPT}(i+1, D), \underbrace{v_i + \text{OPT}(i+1, D+s_i)}_{\text{if } D+s_i \leq C})$$

...

Shortest Paths with Negative Weights

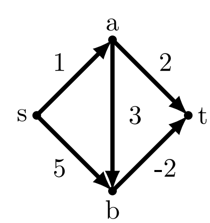
"e.g., currency exchange: nodes are currencies, path weights are exchange rates, minimize produce of path weights."

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:



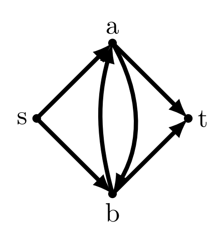
Try Dynamic Programming

$\text{OPT}(v)$

$= \text{shortest path from } v \text{ to } t.$

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

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 $s \rightsquigarrow v \rightarrow u \rightsquigarrow t$ then u-t path has one fewer edge than v-t path.

Part I: identify subproblem in english

$\mathrm{OPT}(v,k)$

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

Part II: write recurrence

$\mathrm{OPT}(v,k)$

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

Correctness: lemma + induction.

Part III: solve original problem

- minimum cost path = $\mathrm{OPT}(s, n-1)$.

Part IV: base case

- for all k: $\mathrm{OPT}(t,k) = 0$
- for all $v \neq t$: $\mathrm{OPT}(v,0) = \infty$.

Part V: iterative DP

Algorithm: Bellman-Ford

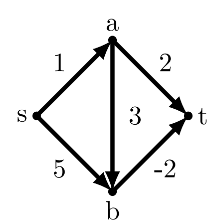
1. base case:

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

   for all $v \neq t$: $\mathrm{OPT}[v,0] = \infty$.
2. for $k = 1 \ldots n-1$: for all v:

   $\mathrm{OPT}[v,k] = \min_{u \in N(v)} \mathrm{OPT}[u,k-1]$.
3. return $\mathrm{OPT}[s,n-1]$.

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