Dynamic Programming II

Solving a problem by identifying a collection of subproblems and solving them smallest to largest

Last Time:
1. Fibonacci
2. Shortest paths in DAGs
3. Longest increasing subsequences
4. Edit distance

This time: More problems

DP design steps

1. Identifying subproblems ($F_1, \ldots, F_n$)
2. Compute recurrence ($F_n = F_{n-1} + F_{n-2}$)
3. Design algorithm (bottom-up Fib)

Knapsack

Input: total weight W
n items with weights $w_1, \ldots, w_n$
and values $v_1, \ldots, v_n$ (all integers)

Output: Most valuable combination of items w/ total weight $\leq W$

Two variants: with repetition v. without repetition

$W = 10$

Item	Weight	Value
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

With replacement
(item 1) + 2·(item 4) = \$48

Without replacement
(item 1) + (item 3) = \$46

Knapsack with replacement / repetition

1. Identify subproblems

best knapsack of weight W

$i_1 \; i_2 \; \cdots \; i_{k-1} \; i_k = i$

best knapsack with weight $\leq W - w_i$ | w_i

$K(C)$ = max value achievable w/ total weight $\leq C$

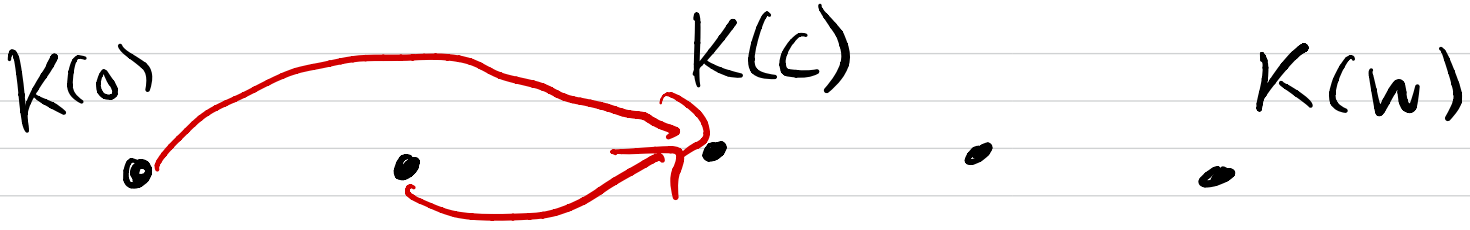
2. Recurrence?

$$K(C) = \max_{i \,:\, w_i \leq C} \{ v_i + K(C - w_i) \}$$

3. Algorithm

```
K(0) = 0
for C = 1 to W
    K(C) = max_{i : w_i ≤ C} { v_i + K(C - w_i) }
return K(W)
```

Runtime: $O(nW)$


K(0)
K(c)
K(w)

Knapsack without repetition

1. Identify subproblems $K(C)$ = max value achievable w/ weight $\leq C$???

$\underbrace{i_1\ i_2\ \cdots\ i_{k-1}}_{\text{best knapsack w/ weight} \leq C - w_i \text{ not including } i}\ \underbrace{i_k = i}_{w_i}$ ✓ doesn't work!

$K(C, j)$ = max value achievable w/ total weight $\leq C$ only using items $1, \ldots, j$

2. Recurrance?

Opt $K(C, j) =$ [Yes 1, No 2, ..., Yes $j-1$] j

Opt $K(C - w_j, j-1),\ K(C, j-1)$

$$K(C, j) = \max\{v_j + K(C - w_j, j-1),\ K(C, j-1)\}$$

Base case: $K(0, j) = 0$, $K(C, 0) = 0$

3. Algorithm

Initialize all $K(0,j)$ and $K(c,0)$ to 0

for $j = 1$ to n

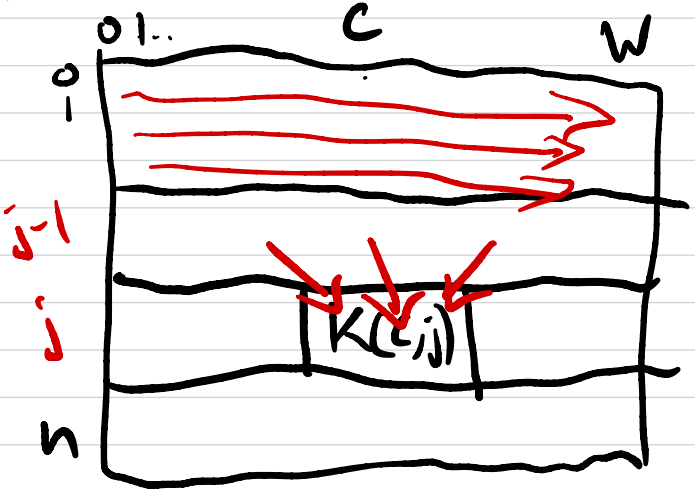
  for $c = 1$ to W

    if $w_j > c$: $K(c,j) = K(c,j-1)$

    else $K(c,j) = \max\{ v_j + K(c-w_j, j-1),$
    $K(c,j-1)$

return $K(W,n)$

K:



Runtime: # subproblems = $O(nW)$

time per subproblem = $O(1)$

total: $O(nW)$

Space: ~~$O(nW)$~~

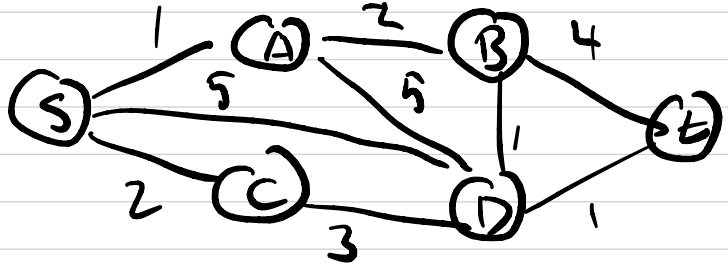
$O(W)$

Shortest path

(positive or negative)

Input: directed $G=(V,E)$, lengths for each edge $\ell(u,v)$
$s,t \in V$, integer k

Output: Length of shortest $s \to t$ path only using k edges



Shortest $s \to t$ path: length 5

shortest $s \to t$ path w/ $k=2$: length 6

1. Subproblems? $dist(v,i)$ = length of shortest $s \to v$ path using $\leq i$ edges

2. Recurrence? Shortest $s \to v$ path w/ $\leq i$ edges

$s \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \cdots \longrightarrow v_{i-1} \longrightarrow v_i = v$

shortest $s \to v_{i-1}$ path of length $i-1$

$$dist(v,i) = \min_{u:(u,v)\in E} \{ dist(u,i-1) + \ell(u,v),\ dist(v,i-1) \}$$

3. Algorithm

$n = |V|, m = |E|$

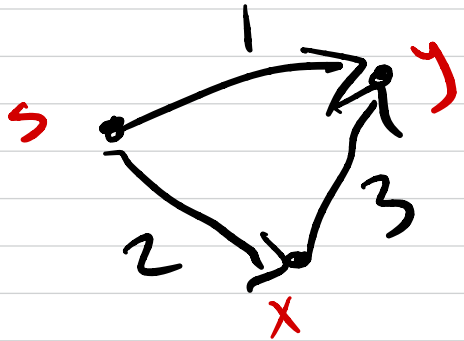
Initialize all $dist(v, 0) = \infty$
$dist(s, 0) = 0$

for $i = 1, \dots, k$
  for each $v \in V$
    $dist(v, i) = \min_{u:(u,v)\in E} \{ dist(u, i-1) + \ell(u, v),\ dist(v, i-1) \}$

return $dist(t, k)$

Runtime:
$O(k \cdot n + km)$

Shortest path:
set $k = n - 1$



$dist(s, 0) = 0$
$dist(x, 1) = 2$
$dist(y, 1) = 1$
$dist(y, 2) =$

$dist(s, 1) + \ell(s, y) = 1$
$dist(x, 1) + \ell(x, y) = 5$

All pairs shortest paths (APSP)

Input: $G = (V, E)$, lengths $\ell(u,v)$ for each edge

Output: $\forall u, v \in V$, $dist(u,v)$ = length of shortest $u \to v$ path

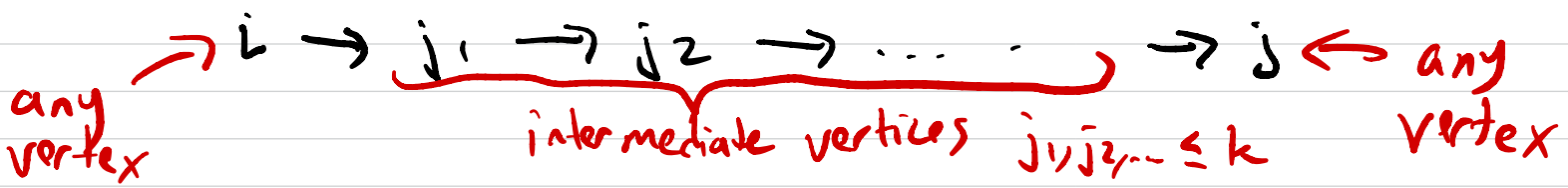
Idea 1: use Bellman-Ford for each $s \in V$

$$O(n) \cdot \underbrace{O(n(n+m))}_{\approx O(nm)} \approx O(n^2 m)$$

Idea 2: use DP

we'll prove $O(n^3)$

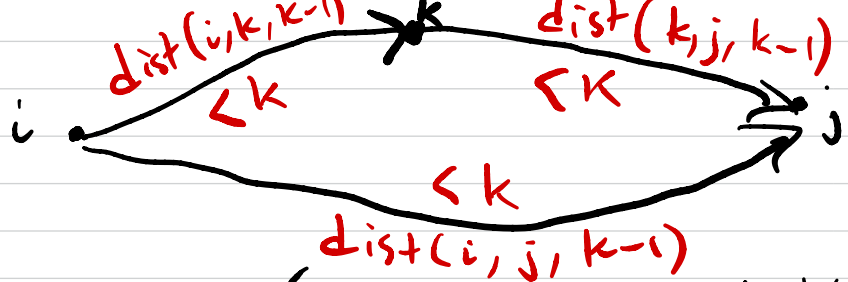
1. Subproblems? Suppose $V = \{1, \dots, n\}$



$dist(i,j,k)$ = length of shortest $i \to j$ path with intermediate vertices are in $1, \dots, k$

$dist(i,j,0) = \ell(i,j)$

2. Recurrence for $dist(i,j,k)$



$$dist(i,j,k) = \min\{ dist(i,j,k-1),\ dist(i,k,k-1) + dist(k,j,k-1) \}$$

Algorithm

For all $(i,j) \in E$ dist$(i,j,0) = \ell(i,j)$ (∞ if not edge)

for $k = 1 \ldots n$
  for $i = 1$ to n
    for $j = 1$ to n

$$\text{dist}(i,j,k) = \min\{\text{dist}(i,k,k-1) + \text{dist}(k,j,k-1), \text{dist}(i,j,k-1)\}$$

Runtime: $O(n^3)$ time

Fun fact: runtime of best APSP alg: $O\left(\frac{n^3}{e^{\sqrt{\log n}}}\right)$