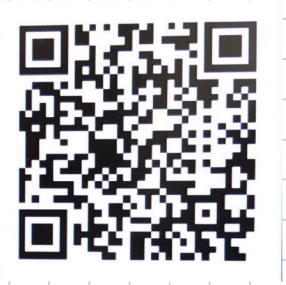
CS 170

Lecture 13

- use iClicker

Sanjam Garg.

join.iclicker.com/RGWR

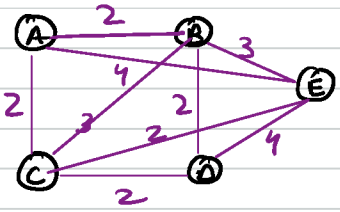


DP Approach

1) Define appropriate subproblems
2) Write a recurrence relation
3) Determine order of computation.
   ↳ induces a DAG structure

Travelling Salesman Problem.

Input: n cities & distances d_{ij} $(i \neq j)$
Goal: Find minimum distance path from city 1 back to city 1 visiting each city exactly once.



Brute-force: (i) $1 \rightarrow 2 \rightarrow 3 \cdots \rightarrow n$
(ii) $1 \rightarrow 3 \rightarrow 2 \cdots \rightarrow n$
$\vdots$
$n! \rightarrow n^n$ (costly)

Start at city 1 but you are allowed to end at any other city while visiting all the cities.

$C(S, j)$ = least cost path ① start at 1 ② visit each node in S (exactly once) ③ ends in node j

S: $\subseteq \{1, \ldots n\}$; j: ends in j

Base: $|S|=1$ $C(\{1\}, 1) = 0$

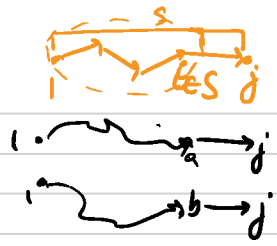
$C(\{2\}, 1) \rightarrow$ never happens.

$|S|=2$ $C(S, j) \rightarrow \infty$ $j = 1$

$\rightarrow ?$ $j \neq 1$

$\forall |S|$ $C(S, 1) = \infty$

$$C(S, j) = \min_{i \in S : i \neq j} \{ C(S \setminus \{j\}, i) + d_{ij} \}$$



$|S| = 2$ $\quad$ $|S| = 3$

$C(\{1\}, 1) = 0$

$C(S, 1) = \infty$

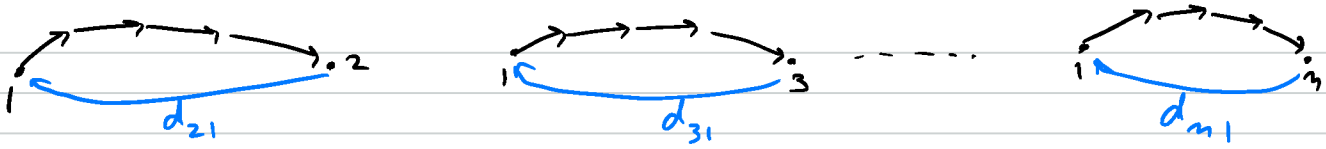
$C(S, j)$

$\{1, 2\}, 2 \quad d_{12}$

$\{1, 3\}, 3 \quad d_{13}$

$\{1, n\}, n \quad d_{1n}$

$S = \{1, 2, 3\}$ $\quad j = 2$ $\quad j = 3$



Solution to TS problem = $\min_j \{ C(\{1 \ldots n\}, j) + d_{j1} \}$

Algorithm

$C(\{1\}, 1) = 0$

$O(n)$ ← for $s = 2$ to n

$O(2^n)$ ← for all subsets $S \subseteq \{1 \dots n\}$ of size s and containing 1:

$C(S, 1) = \infty$
for all $j \in S, j \neq 1$

$O(n)$ ← $C(S, j) = \min_{\substack{i \in S \\ i \neq j}} \{ C(S - \{j\}, i) + d_{ij} \}$

return $\min_j \{ C(\{1 \dots n\}, j) + d_{j1} \}$

$2^n \times n \times n = O(n^2 2^n)$

Independent Set

Definition: Graph $G = (V, E)$, an *independent set* is a set $I \subseteq V$ such that $\forall u, v \in I$ we have that $(u, v) \notin E$.

Given: Graph $G = (V, E)$

Find: $Ind(G) = \max_{I \subseteq V} \{ |I| : I \text{ is an independent set} \}$

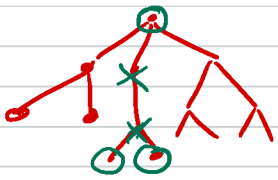
↳ this is hard.

Maximal Independent Set for Tree

Given: Tree $G=(V,E)$

Find: $Ind(G) = \max_{I \subseteq V} \{ |I| : I \text{ is an independent set} \}$

$I(v)$ = Size of the maximal independent set of the sub-tree rooted at v



Base case: if v has no children then $I(v) = 1$

$$I(v) = \max \left\{ \sum_{u \in C(v)} I(u) \;,\; 1 + \sum_{u \in G(v)} I(u) \right\}$$

$C(v)$: children of v; node v; $G(v)$: grandchildren of v

Algorithm

Input: Tree on $V = \{1 \dots n\}$

$\pi(v)$ = parent of v, $\pi(r) = r$

$O(n)$ → Topologically sort V (s.t. $\pi(v) > v \quad \forall v \neq r$)

$O(n)$ → For $v \in V$: if $\pi(v) \neq v$ then add v to $C(\pi(v))$; if $\pi(\pi(v)) \neq \pi(v)$ then add v to $G(\pi(\pi(v)))$

$O(n)$ → For $v \in V$: if $C(v) = \phi$ then $I(v) = 1$

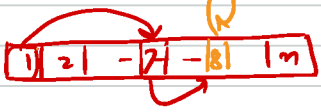
$O(n)$ → For all $v \in V$:

$$I(v) = \max \left\{ \sum_{w \in C(v)} I(w) \;,\; 1 + \sum_{w \in G(v)} I(w) \right\}$$

Output $I(r)$

$$\sum c_1 O(1) \; c_2 O(2) \; \dots \; c_n O(n)$$

$$O(n) \leftarrow \sum g_1 O(1) \; g_2 O(2) \; \dots \; g_n O(n)$$



TS ↓

Coin denomination problem

Input: $x_1 \dots x_n$; V

Goal: Possible to make change of V using given coins?
Find: minimum # of coins needed.

Example: 5, 10; $V = 15$ 5, 10; $V = 12$ 1, 3, 4; $V = 6$

Greedy → 4, 1, 1

optimal → 3, 3

$K(v)$ = min # of coins needed to give change for v
(we set it to ∞ if not possible)

$$K(v) = \min\left\{\min_{i:x_i \le v}\{K(v-x_i)+1\}, \infty\right\}$$

Base case: $K(0) = 0$

Input: $x_1 \dots x_n$; V
$K(0) = 0$ For $C = 1$ to V $K(C) = \min\{\infty, \min_{i:x_i \le C}\{1 + K(C - x_i)\}\}$ Output $K(V)$

Task Scheduling Problem

Input: n jobs with start & end times
$[s_1, t_1]$ .... $[s_n, t_n]$
w_1 w_n

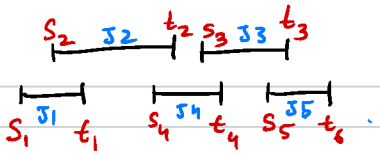
Task: Schedule as many non-overlaping tasks as possible

Claim: ~~Greedy non-overlapping~~ first time finish is ~~optimal~~!

$$\max_T \left\{ \sum_{i \in T} w_i \right\}$$

↳ must be non overlaping

Example:



optimal → J1, J4, J5

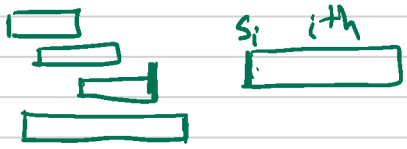
(I) Sort the jobs by their finish time.

$[s_1', t_1']$ ---- -- $[s_n', t_n']$

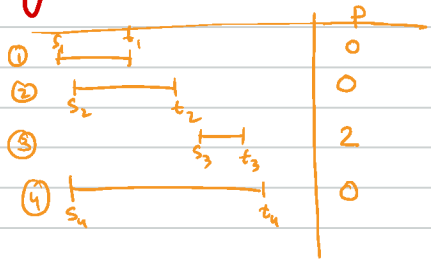
$t_1' \le t_2' \le t_3' \le \dots \le t_n'$

(II) $p(i)$ = the last jobs that can be scheduled if i^{th} job has been scheduled

$= \max \{ j < i \mid F_j < S_i \}$



$OPT(i)$ = optimal weight obtained by scheduling jobs 1 -- i only.



$$OPT(i) = \max \{ OPT(i-1), w_i + OPT(p(i)) \}$$

Base: $OPT(0) = 0$ or $OPT(1) = w_1$

1. Longest path in DAG → explicit DAG
2. Edit distance → grid graph
3. Matrix chain Multiplication → interval subproblems.
4. APSP → DP with "allowed intermediate vertices/states"
5. TSP → Subset subproblems.
6. Knapsack → subproblems based on capacity
   - ↳ with replacement
   - without replacement
   - + coin denomination
7. Independent Set → DP on a tree structure
8. * weight task scheduling → similar to longest paths.