DYNAMIC PROGRAMMING

1) Longest path in a DAG

Subproblem $L(v)$ = length of longest path ending in v

2) Longest Increasing Subsequence

Subproblem: $L(i)$ = length of longest increasing subsequence ending in a_i

3) Edit Distance

Subproblem: $E(x[1:i], y[1:j])$ = edit distance between prefixes

4) Knapsack (capacity W, items with weights $w_1, \cdots w_n$ & values $v_1, \cdots v_n$)

4a) Knapsack with Replacement

Subproblem:

$K(C)$ = max total value with capacity C $\quad C = 0, 1, \cdots, W$

4b) Knapsack w/o replacement

Subproblem

$K(C, k)$ = Optimum with total weight $\leq C$ while only using items in $\{1, \cdots, k\}$ $\quad k = 0, 1, 2, \cdots$

5) Single Source Shortest Path

Subproblems

$\text{dist}(v, k)$ = length of shortest path $s \rightsquigarrow v$ using $\leq k$ edges

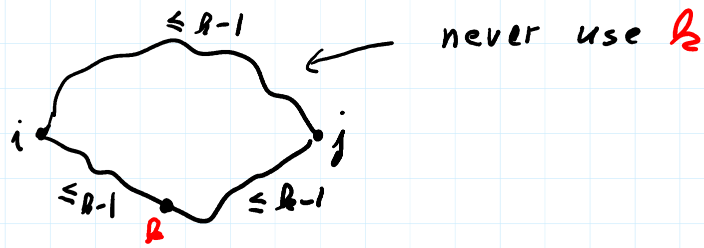
Gives modified version of Bellman Ford $\quad O(n|E|)$ runtime

6) All Pairs Shortest path (Floyd Warshall Algorithm)

Subproblem

$V = \{1, 2, \cdots n\}$

$dist(i, j; k)$ uses only vertices in $\{1, 2, \cdots, k\}$ as intermediate vertices

$dist(i,j,k) = $ 

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

$O(n^3)$ Runtime

7) Travelling Salesman Problem (TSP)

Given: n cities, distances d_{ij} $\quad i \neq j$

Goal: Find path of minimal length,

starting at 1, ending at 1, visiting every city once

Subproblem:

For $S = \{1, \cdots k\}$ let

$C(S) =$ Length of shortest path in S, starting at 1, ending at 1, and visiting every $i \in S$

$C(S, j) =$ Length of shortest path from 1 to $j \in S$ visiting every $i \in S$ once

Note that $\boxed{C(S) = \min_{1 \neq j \in S} C(S, j) + d_{j1} \quad \text{if } |S| \geq 2}$

Recursion

$S' = \{1,2,\cdots k-1\} \to S = \{1,\cdots k\}$, $j \in S'$

<u>Case 1</u>: $j = k$

$C(S,k) =$  $= \min\limits_{1 \neq i \in S'} \left[C(S',i) + d_{ik} \right]$

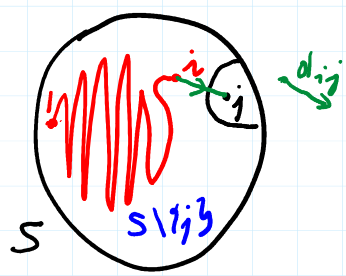
<u>Case 2</u>: $j \neq k$

$C(S,j) =$  $S = S' \cup \{k\}$

<u>What if we know all $|S| < k$ with $1 \in S$</u>

$C(S,j)$ for $|S| = k$ $\quad j \in S,\ j \neq 1$

Then $|S \setminus \{j\}| < k$



<u>Recurrence</u>:

$\forall\ 1 \neq j \in S$

$$C(S,j) = \min_{\substack{i \in S\setminus\{j\} \\ i \neq 1}} C(S\setminus\{j\}, i) + d_{ij} \qquad (R)$$

<u>Initialization</u>:

$$C(\{1,j\}, j) = d_{1j} \qquad \text{For all } j \neq 1$$

<u>Note</u>: $C(S,1)$ is not defined

For convenience, we set

$C(\{1\},1) = 0$ and $C(S,1) = \infty$ if $|S| > 1$,

and for $|S| \geq 2$, $j \neq 1$ modify (R) to

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

This turns out to be equivalent.

Indeed, if $|S|=2$, $1\in S$, $j\neq 1$ then $S\setminus\{j\}=\{1\}$ and (*) gives

$$C(S,j)=C(\{1\},1)+d_{1j}.$$

On the other hand, if $|S|\geq 3$, (*) gives

$$C(S,j)=\min_{i\in S\setminus\{j\}} \underbrace{C(S\setminus\{j\},i)}_{=\infty \text{ if } i=1}+d_{ij} = \min_{\substack{i\in S\setminus\{j\}\\ i\neq 1}} C(S\setminus\{j\},i)+d_{ij}$$

which is the required recurrence relation (R)

Algorithm:

$C(\{1\},1)=0$
For $k=2,\cdots,n$
For all $S\subseteq\{1,\cdots n\}$, $1\in S$, $\lvert S\rvert=k$
$C(S,1)=\infty$
For all $j\in S\setminus\{1\}$
$C(S,j)=\min_{i\in S, i\neq j} C(S\setminus\{j\},i)+d_{ij}$
Output $\min_{j\in\{1,\cdots,n\}} C(\{1,\cdots,n\},j)+d_{j1}$

$\binom{n-1}{k-1}$ cases

$k-1$

$k-2$

Run Time:

$O\left(\sum_k \binom{n}{k} k\cdot k\right)=O(n^2 2^n)$

8) Independent Sets

Def: Given $G=(V,E)$, an independent set is a set $I\subseteq V$ s.th. no pair of vertices $\{x,y\}\subseteq I$ is an edge in E

Example 1: V set of radio stations

$\{i,j\}\in E \Leftrightarrow i,j$ are close enough to lead to interference

I = set of stations that can use the same frequency band

Task: Given $G=(V,E)$ find

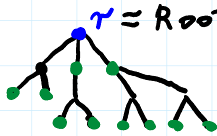
$$\mathrm{Ind}(G) = \max_{I \subseteq V} \{ |I| : I \text{ is an independent set} \}$$

Hard in General!

Maximal Independent Set for Trees

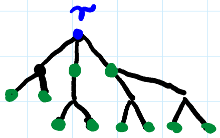
Define Subproblem:

r = Root

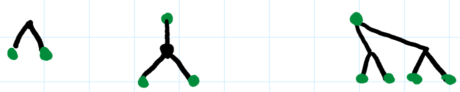
Optimal Solution: 

Case 1: $r \notin I$

r



$|I| = 2 + 3 + 5$



Independent Sets in subtrees under children of r

Subproblem:

T_v = subtree under v

$I(v)$ = size of largest independent set I in T_v

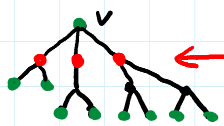
Recursion

Case 1: $v \notin I$, I optimal

$$\Rightarrow |I| = \sum_{w \in C(v)} I(w)$$

$C(v)$ = children of v

Case 2: $v \in I$, I optimal



can't be in I

$$\Rightarrow |I| = 1 + \sum_{w \in G(v)} I(w) \qquad G(v) = \text{grandchildren of } v$$

General Case:

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

Base Case: v is a leaf

$I(v) = 1$

Calculation Order: Leaves to root

Algorithm

	Run time
Input: Tree on $V = \{1, \cdots, n\}$ $\pi(v) = $ parent of v, $\pi(r) = r$	
Topologically sort V (s.th. $v < \pi(v)$ $\forall v \neq r$)	$O(n)$
For $v \in V$ if $\pi(v) \neq v$ Append v to $C(\pi(v))$	$O(n)$
For $v \in V$: $G(v) = \bigcup_{u \in C(v)} C(u)$	$O\left(\sum_v \lvert G(v) \rvert\right)$
For $v = 1, \cdots n$ $I(v) = \max\left\{ \sum_{w \in C(v)} I(w),\ 1 + \sum_{w \in G(v)} I(w) \right\}$	$O\left(\sum_v \{\lvert C(v) \rvert + 1 + \lvert G(v) \rvert\}\right)$

Claim: The run time is $O(n)$

Pf: $\sum_v |C(v)| = \sum_v \#\text{of incoming edges to } v = |E| = n-1$

$$\sum_v |G(v)| = \sum_{v,u} \mathbb{1}_{u \text{ is grandchild of } v}$$

$$\sum_v |G(v)| = \sum_{v,u} \mathbb{1}_{u \text{ is grandchild of } v}$$

$$= \sum_{v,u} \mathbb{1}_{v \text{ is grandparent of } v}$$

$$= \sum_u \underbrace{\sum_v \mathbb{1}_{v \text{ is grandparent of } u}}_{\leq 1}$$

since each u has at most one grandparent v

$$\leq \sum_u 1 = n$$

9) Chain Matrix Multiplication

Multiplying $\overset{n \times \ell}{A} \times \overset{\ell \times m}{B}$

$(A \times B)_{ij} = \sum_{\ell=1}^{\ell} A_{i\ell} B_{\ell j} \rightarrow \ell$ Multipl.

Total time For $A \times B$ $\quad n \cdot \ell \cdot m$

Q: How to calculate

$$\underset{50 \times 20}{A} \times \underset{20 \times 1}{B} \times \underset{1 \times 10}{C} \times \underset{10 \times 100}{D}$$

$$\underbrace{\underset{50 \times 20}{A} \times (\underbrace{\overset{20 \times 1}{B} \times \overset{1 \times 10}{C}}_{20 \times 10})}_{50 \times 10} \times \underset{10 \times 100}{D}$$

Cost:

$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$

$= 200 + 10{,}000 + 50{,}000$

$\boxed{60{,}200}$

$$\underset{50 \times 20}{A} \times \underbrace{(\underbrace{B \times C}_{20 \times 10}) \times \underset{10 \times 100}{D}}_{20 \times 100}$$

$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 +$

$50 \cdot 20 \cdot 00$

$= 200 + 20{,}000 + 100{,}000$

$\boxed{120{,}000}$

$$\underbrace{(\overset{50 \times 20}{A} \times \overset{20 \times 1}{B})}_{50 \times 1} \times \underbrace{(\overset{1 \times 10}{C} \times \overset{10 \times 100}{D})}_{1 \times 100}$$

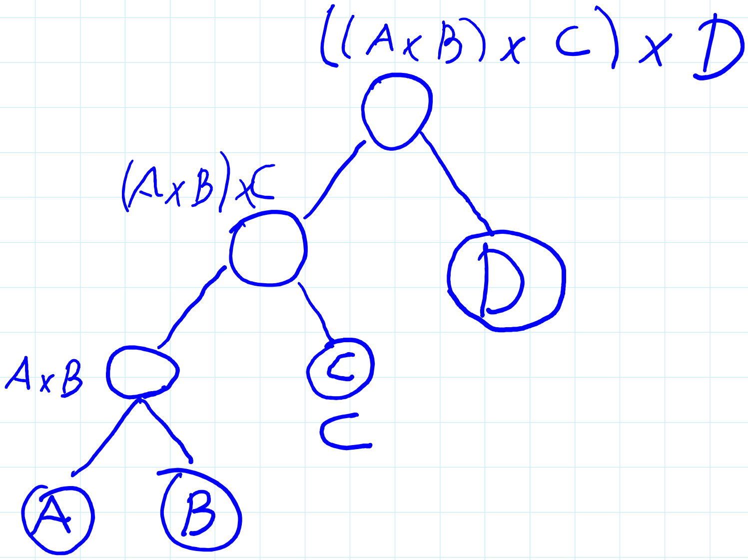
$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 +$

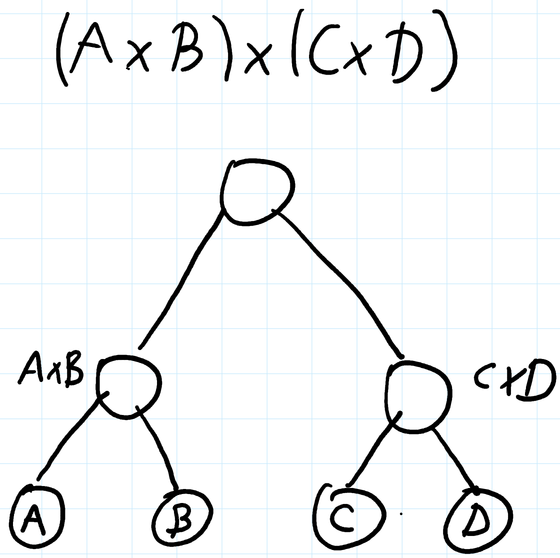
$50 \cdot 100 =$

$\boxed{7000}$

In general, which parenthezation is Fastest?

Binary tree representation:

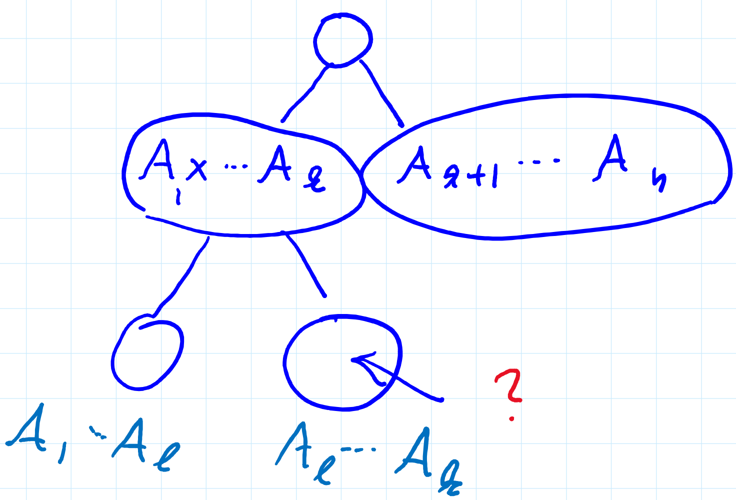




General Case

$A_1 \times A_2 \times \cdots \times A_n$

A_1 is $m_0 \times m_1$, A_2 is $m_1 \times m_2 \cdots$



Subproblem:

$A_i \times A_{i+1} \times \cdots \times A_j$

$C(i,j)$ run time for optimal way to put parentheses

Recurrence:

$$\underbrace{A_i \times \cdots \times A_k}\times\underbrace{A_{k+1}\times\cdots\times A_j}$$

$$C(i,k)+C(k+1,j)+m_{i-1}m_k m_j$$

If optimal

$$C(i,j)=\min_{i\le k\le j}\left(C(i,k)+C(k+1)+m_{i-1}m_k m_j\right)$$

Order of Calculations

$$s=|i-j|=0,1,2,\cdots$$

Algorithm

For $i=1,\cdots n$ $\quad C(i,i)=0$

For $s=1,\cdots n-1$

For $i=1,\cdots,n-s$

$j=i+s$

$$C(i,j)=\min_{i\le k\le j}\left(C(i,k)+C(k+1)+m_{i-1}m_k m_j\right)$$

Return $C(1,n)$

Running Time:

$$O\left(\sum_{i<j}|i-j|\right)=O(n^3)$$