Dynamic Programming Recipe

- Define a set of problems, such that
  - base case - easy to solve
  - final case - matches (closely) the final problem we want to solve.
- Write it as a recursion: Solve bigger problem in terms of the smaller problems. (Should be a DAG on the problem instances!)
- Compute the problems on the DAG in the linearized order!

The Knapsack Problem

Want Knapsack of weight at most 29,

item	weight	value
1	15	43
2	6	18
3	7	21
4	8	23

Value: 43+18+23 = 84.

Knapsack with Repetition

Given: W, $(v_1, w_1), (v_2, w_2), \ldots, (v_n, w_n)$.

Find: highest value multiset of items of weight $\leq W$.

Dynamic Program.

Subproblems?
$K(w) =$ "Best value of knapsack of weight w"

Solve subproblem $K(w)$ ...using smaller subproblems?

Consider solution. $\{i, j, k, \ldots\}$.

Take out one item, say i, weight is $w - w_i$.

Rest should be best knapsack of weight $w - w_i$.
$K(w - w_i)$ and add value of v_i.

$K(w) = \max_i (K(w - w_i) + v_i)$ for $w - w_i \geq 0$.

$K(0) = 0$

Example

Weight 29

item	weight	value
1	15	43
2	6	18
3	7	21
4	8	23

$K(0) = 0$.
Recurrence: $K(w) = \max_i(K(w - w_i) + v_i)$, $K(w - w_i)$ is defined.

$K(1), \ldots, K(5)$ undefined

$K(6) = \max_i(K(6 - w_i) + v_i) = K(0) + v_2 = 0 + 18 = 18$.
$K(7) = 21$, $K(8) = 23$, $K(9), K(10), K(11)$ undefined.
$K(12) = K(12 - 6) + 18 = K(6) + 18 = 36$.
$K(13) = K(6) + 21 = K(7) + 18 = 39$,
$K(14) = K(7) + 21 = 42$,
$K(15) = K(8) + 21 = 44$
$K(16) = K(8) + 23 = 46$. $K(17)$ undefined, $K(18) = K(12) + 18 = 54$, ...

Read off highest valued $K(w)$ for value of solution.

Complexity: Knapsack with Repetition.

$K(w) = \max_i(K(w-w_i)+v_i)$ for $w - w_i \geq 0$.

$K(0) = 0$

W entries, $O(n)$ time per entry.
(Scan over all n items in $\max_i$.)

Total: $O(nW)$ time.

Knapsack without Repetition

Given: Weight: W, Items: $(v_1, w_1), (v_2, w_2), \ldots, (v_n, w_n)$.

Find: highest value *set* of items of weight $\leq W$.

Previous Dynamic Program.

$K(w) =$ "Best value of knapsack of weight w"

$K(w) = \max_i K(w - w_i) + v_i$ for $w - w_i \geq 0$.

$K(0) = 0$

No way to control for using items over and over again!

Knapsack without Repetition

Given: Weight: W, Items: $(v_1, w_1), (v_2, w_2), \ldots, (v_n, w_n)$.

Find: highest value subset of items of weight $\leq W$.

Subproblem?

Idea? Subset! In subset or not! Sequential.

Best knapsack using first i items?

Best depends on how much space is left.

Best knapsack of weight w using first i items.

$K(w, i) =$ "Best weight w Knapsack with subset of first i items."

Either add item or not!

$K(w, i) = \max\{K(w - w_i, i - 1) + v_i, K(w, i - 1)\}$

$K(0, 0) = 0$

Example no Repetition

	item	weight	value
	1	15	43
Weight 30	2	6	18
	3	7	21
	4	8	23

$K(0,0) = 0$.
Recurrence: $K(w,i) = \max(K(w,i-1), K(w-w_i,i-1)+v_i)$.

$K(0,1) = 0$, $K(15,1) = 43$, All other $K(w,1)$ undefined.

$K(0,2) = 0$, $K(6,2) = 18$, $K(15,2) = 43$, $K(21,2) = 61$,
All other $K(w,i)$ undefined.

$K(0,3) = 0$, $K(6,3) = 18$, $K(7,3) = 21$, $K(13,3) = 39$, $K(15,3) = 43$,
$K(22,3) = 64$ ...

....

Read off highest value of $K(w,n)$ for answer.

Complexity: Knapsack without Repetition.

$K(w,i) = \max\{K(w-w_i, i-1)+v_i, K(w,i-1)\}$

$K(0,0) = 0$

Time: nW entries, $O(1)$ time per entry. $O(nW)$ time.

Dag?

With Repetition? Supproblem view: $K(w) = \max_i(K(w - w_i) + v_i)$

(A) $\Theta(W)$ nodes, $O(nW)$ edges.

(B) $\Theta(nW)$ nodes, $O(W)$ edges.

(C) Strongly connected.

(A) W Table entries: $K(w)$

One incoming edge for each of n items.

Without Repetition?

Subproblem View: $K(w,i) = \max(K(w,i-1), K(w-w_i,i-1)+v_i)$.

(A) $\Theta(W)$ nodes, $O(nW)$ edges.

(B) $\Theta(nW)$ nodes, $O(nW)$ edges.

(C) Strongly connected.

(B) nW Table entries: $K(w,i)$

Two incoming edges (with/without item).

Test your knowledge!

Knapsack with weights and sizes(volume?), find best?

Given items: $(v_1, w_1, s_1), \ldots (v_n, w_n, s_n)$, W, S.

Find best subset of items with weight $\leq W$ and size $\leq S$?

Need more information about subproblem solution!

Subproblem?
$K(w, s, i)$
- Best knapsack of weight w , size s, using first i items.

Exercise: how do you fill in table!

That's weak!

Time: $O(nW)$.

Polynomial in the number W.

Not necessarily polynomial in input size!
That is, number of bits. $\log W$.

Size: 2,000,000

item	weight	value
1	2,000,000	1,999,999
2	1,000,001	1,000,001

Takes 2,000,000 steps!
Even exponential search takes fewer steps!

Weakly polynomial. Like adding roman numerals!

Still useful!

For example, when number of items is bigger than 10.

Chain Matrix Multiplication

Matrix Multiplication: $A \times B \times C$.

A is 50×1, B is 1×50,
C is m by n.

What could m and n be?

(A) $m = 50$, $n = 1$

(B) $m = 1$, $n = 50$.

Compute $X = A \times B$ and then compute $X \times C$?

How many multiplications?

How many entries in X? Dimension is 50×50.
2500 entries.
One multiplication per entry of X
2500 multiplications to get X.

$X \times C$ - Output is 50×1 50 multiplications for each ouput.
2500 more multiplications. Total: 5000

Can you do better?

A is 50×1, B is 1×50,
C is 50 by 1.

How about $Y = B \times C$? Then $A \times Y$.

Y is 1×1. Just a dot product!
50 multiplications.

How many multiplications for AY?

50.

Total is 100! ..versus 5000.

Associativity ...matters!

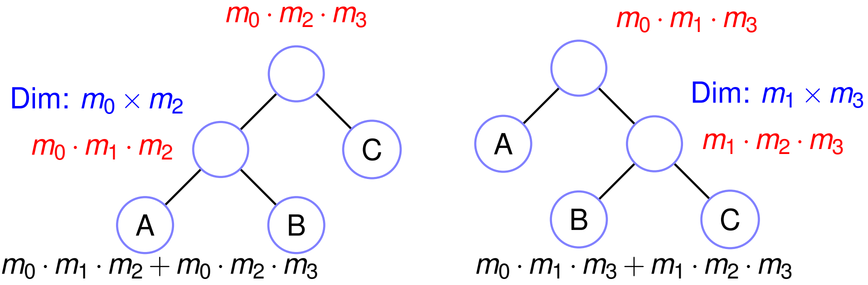
Matrix Chain Multiplication.

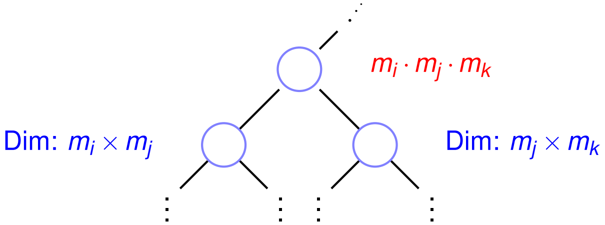
Given $A_1 \times A_2 \times \cdots A_n$, optimal association (order)?

Input: A_1 is $m_0 \times m_1$. A_2 is $m_1 \times m_2$. $\cdots$

Given $m_0, m_1, m_2, \ldots, m_n$, find optimal parenthesization.

Example Parenthesizations: A,B,C: m_0, m_1, m_2, m_3
$(A \times B) \times C$ or $A \times (B \times C)$.





Cost is cost of subtrees plus top product and so on.

subtree on $m_i, \ldots, m_j$ yields m_i by m_j matrix

subtree on $m_j, \ldots, m_k$ yields m_j by m_k matrix

cost of top product: $m_i \times m_j \times m_k$.

Try all trees. Evaluate! 2^{2n} trees. Uh oh!

Dynamic Programming: Subproblems?

$C(i,k)$ - optimal cost for multiplying $A_i, \ldots, A_{k-1}$

Combine: $C(i,k) = \min_{i<j<k}(C(i,j) + C(j,k) + m_i m_j m_k)$.

$C(i,i+1) = 0$. No cost for single matrix.

$O(n^2)$ subproblems. $O(n)$ time per problem. $\rightarrow O(n^3)$ time. Space?