Chapter 5, Dynamic Programming:

- Introduction (Wednesday)
- Principles of DP, including sample problems (**Today**)
- DP algorithms, solutions to sample problems (next week Monday)
- *Review for Test 2 (next week Wednesday)*
- *Test 2 (next week Thursday, in lab)*
- *No class (next week Friday)*
- Optimal BSTs (week-after Monday)

Today:

- Memoization HW problems
- Finishing of coin-changing problem
- Introduction of three problems
    - 0-1 Knapsack
    - Longest common subsequence
    - Matrix multiplication
- Elements of dynamic programming
    - Optimization problems
    - Optimal substructure
    - Dynamic programming algorithms
- Solution to the knapsack problem (time permitting)

Ex 6.5. Explain why this function can't use memoization:

```
idgen = -1

def make_unique_id(name) :
    # global allows us to modify idgen inside this function
    global idgen
    idgen += 1
    return name + str(idgen)
```

Ex 6.6. Explain why this function can't use memoization:

```
def pick_at_random(seq) :
    return seq[random.randint(0, len(seq)-1)]
```

Ex. 6.7. Explain why this function can't use memoization:

```
f = open('data', 'r')

def next_n_lines(n) :
    lines = ''
    for i in range(n) :
        lines += f.readline()
    return lines
```

Dynamic programming. An algorithmic technique, often applied to optimization problems, in which the inefficiency of overlapping subproblems is avoided by memoization.

Memoization. An algorithmic technique where results of functions or subproblems are stored for later retrieval.

Optimization. A category of algorithmic problem in which one needs to construct an object to minimize a cost or maximize a value.

Overlapping subproblems. When a recursion tree for a formula contains multiple instances of the same subproblem

Recursive characterization. A formula that relates problems to subproblems of the same kind.

Coin-changing

General problem: Given an amount and a currency system, what is the best way (fewest number of coins) to make change for that amount in that currency system.

Example problem instance: What is the best way to make change for 14 units using coins of values `[1,3,4,6]`?

Example subproblem instance: [If we use one 6-unit coin, then] what is the best way to make change for [the remaining] 8 units using [only the remaining coins] of values `[1,3,4]`

Formal notation: Let $C[i][j]$ be the fewest number of coins needed to make change for amount i using coin denominations 0 through j.

Let $C[i][j]$ stand for the fewest number of coins needed to make change for amount i *using only coins 0 through* j.

$$C[i][j] = \begin{cases} 0 & \text{if } i = 0 \\ i & \text{if } j = 0 \\ \min_{0 \le k \le \frac{i}{D[j]}} \{k + C[i - k \cdot D[j]][j-1]\} & \text{otherwise} \end{cases}$$

0-1 Knapsack.

Given a capacity c and the value and weight of n items in arrays V and W, find a subset of the n items whose total weight is less than or equal to the capacity and whose total value is maximal.

V	20	15	90	100
W	1	2	4	5
	0	1	2	3

$c = 7$

set	weight	value	
$\{2,3\}$	9	190	*exceeds capacity*
$\{1,3\}$	7	115	*not optimal*
$\{0,1,2\}$	7	125	*optimal*

Longest common subsequence.

Given two sequences, find the longest subsequence that they have in common.

D A T A S T R U C T U R E S
A L G O R I T M S

A A A A A B
A B A A A A

not

A A A A A B
A B A A A A

A A A A A B A A A A
A B A A A A

not

A A A A A B A A A A
A B A A A A

Matrix multiplication.

$$\begin{pmatrix} 2 & 8 \\ 5 & 7 \end{pmatrix} \begin{pmatrix} 3 & 6 \\ 1 & 4 \end{pmatrix} = \begin{pmatrix} 2\cdot 3 + 8 \cdot 1 & 2 \cdot 6 + 8 \cdot 4 \\ 5 \cdot 3 + 7 \cdot 1 & 5 \cdot 6 + 7 \cdot 4 \end{pmatrix} = \begin{pmatrix} 14 & 44 \\ 22 & 58 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 3 & 12 \\ 2 & 7 & 11 \end{pmatrix} \begin{pmatrix} 4 & 10 \\ 8 & 6 \\ 9 & 5 \end{pmatrix} = \begin{pmatrix} 1 \cdot 4 + 3 \cdot 8 + 12 \cdot 9 & 1 \cdot 10 + 3 \cdot 6 + 12 \cdot 5 \\ 2 \cdot 4 + 7 \cdot 8 + 11 \cdot 9 & 2 \cdot 10 + 7 \cdot 6 + 11 \cdot 5 \end{pmatrix} = \begin{pmatrix} 136 & 88 \\ 163 & 117 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 5 \\ 6 & 8 & 9 \end{pmatrix} \begin{pmatrix} 3 \\ 7 \\ 4 \end{pmatrix} = \begin{pmatrix} 1 \cdot 3 + 2 \cdot 7 + 5 \cdot 4 \\ 6 \cdot 3 + 8 \cdot 7 + 9 \cdot 4 \end{pmatrix} = \begin{pmatrix} 37 \\ 110 \end{pmatrix}$$

Matrix multiplication.

> *Given $n+1$ dimensions of of n matrices to be multiplied, find the optimal order in which to multiply the matrices, that is, find the parenthesization of the matrices that will minimize the number of scalar multiplications.*

Assume the following matrices and dimensions: $A, 3 \times 5$; $B, 5 \times 10$; $C, 10 \times 2$, $D, 2 \times 3$; $E, 3 \times 4$.

$$
\begin{aligned}
(A \times B) \times (C \times (D \times E)) \qquad & 3 \cdot 5 \cdot 10 + 2 \cdot 3 \cdot 4 + 10 \cdot 2 \cdot 4 + 3 \cdot 10 \cdot 4 &=& \quad 374 \\
(A \times (B \times C)) \times (D \times E) \qquad & 5 \cdot 10 \cdot 2 + 2 \cdot 3 \cdot 4 + 3 \cdot 5 \cdot 2 + 3 \cdot 2 \cdot 4 &=& \quad 178 \\
A \times (B \times (C \times (D \times E))) \qquad & 2 \cdot 3 \cdot 4 + 10 \cdot 2 \cdot 4 + 5 \cdot 10 \cdot 4 + 3 \cdot 5 \cdot 4 &=& \quad 364
\end{aligned}
$$

Problem	*Thing to find*	*Optimization*	*Constraint*
Coin-changing	A bag of coins.	Minimize the number of coins.	The coins' values sum to the given amount.
Knapsack	A set of objects	Maximize the sum of the objects' values.	The sum of the objects' weights doesn't exceed the given capacity.
Longest common subsequence	A subsequence in each of two given sequences.	Maximize the length of the subsequences.	The subsequences have the same content.
Matrix multiplication	A way to parenthesize the the matrices being multiplied.	Minimize the number of scalar multiplications required.	The parenthesization is complete and mathematically coherent.
Optimal BST	A BST for a given set of keys	Minimize the expected length of a search.	The tree satisfies the criteria for a BST.

Progression of dynamic-programming problems:

1. Problem statement . . . *recognizing optimal substructure*
2. Recursive characterization . . . *recognizing overlapping subproblems*
3. Dynamic programming algorithm

    Make a table for subproblems
    Initialize base cases in the table
    For all other subproblems / cells in the table
        For each option in the decision for that subproblem
            Lookup subsubproblem results and compare
        Record best choice for that subproblem
    Return minimum cost or maximum value for top-level problem

0-1 Knapsack.

Given a capacity c and the value and weight of n items in arrays V and W, find a subset of the n items whose total weight is less than or equal to the capacity and whose total value is maximal.

V	20	15	90	100
W	1	2	4	5
	0	1	2	3

$c = 7$

set	weight	value	
$\{2, 3\}$	9	190	*exceeds capacity*
$\{1, 3\}$	7	115	*not optimal*
$\{0, 1, 2\}$	7	125	*optimal*

Knapsack

Let $B[i][j]$ be the value of the best way to fill remaining knapsack capacity i using only items 0 through j. Then $B[c][n-1]$ is the value-solution to the entire problem, that is,

$$B[c][n-1] = \max_{K} \sum_{j=0}^{n-1} K[j]V[j]$$

In the general case we have the choice between

$$\underbrace{V[j]}_{\substack{\text{value of}\\ \text{the } j\text{th}\\ \text{item}}} \quad \underbrace{+B[\underbrace{i-W[j]}_{\substack{\text{remaining}\\ \text{capacity}\\ \text{after}\\ \text{taking the}\\ j\text{th item}}}][j-1]}_{\substack{\text{The best way to}\\ \text{fill the remaining}\\ \text{capacity with the}\\ \text{remaining items}}} \qquad \text{versus} \qquad \underbrace{B[i][j-1]}_{\substack{\text{The best way to}\\ \text{fill the unchanged}\\ \text{capacity with the}\\ \text{remaing items}}}$$

Knapsack

$$B[i][j] = \begin{cases} 0 & \text{if } j = 0 \text{ and } W[0] > i \quad \text{(0th doesn't fit)} \\ V[0] & \text{if } j = 0 \text{ and } W[0] \leq i \quad \text{(0th fits)} \\ B[i][j-1] & \text{if } W[j] > i \quad \text{(jth doesn't fit)} \\ \max \left\{ \begin{array}{l} V[j] + B[i - W[j]][j-1], \\ B[i][j-1] \end{array} \right\} & \text{otherwise} \quad \text{(j fits)} \end{cases}$$

Coming up:

Catch up on projects. . .
Do **Traditional RB** *project (due Wed, Nov 6)*
(Recommended: Do **LL RB** *project for your own practice)*

Due **Fri, Nov 8** *(end of day)*
Read Section 6.3
Do Exercises 6.(16, 19, 23, 33)
Take quiz (DP principles)

Due **Mon, Nov 11** *(class time)*
Read Section 6.4
Take quiz (DP algorithms)