Chapter 5, Dynamic Programming:

- Introduction and sample problems (last week Wednesday)
- Principles of DP (last week Friday)
- DP algorithms, solutions to sample problems (Monday)
- Introduce optimal BSTs / review for test 2 (**Today**)
- **Test 2**, *not* covering DP (Friday)
- Finish up optimal BSTs (next week Monday)

Today:

- DP odds and ends
- Review for Test 2
- Introduce the optimal BST problem

Which of the following phrases uses the word *programming* in the same sense (or, at least, most nearly the same sense) as the phrase *dynamic programming* uses the word.

- Parallel programming
- Linear programming
- eXtreme Programming
- Pair programming

Coming up:

Catch up on projects...

(See Schoology for practice problems for Test 2)

Due **Mon, Nov 14** *(end of day) (changed from Nov 9)*
Do Project 6.1.b as a practice problem
Take quiz (on Section 6.4)

Due **Mon, Nov 14** *(end of day)*
Read Section 6.5
(No quiz on Section 6.5)

Due **Wed, Nov 16** *(end of day)*
Read Sections 7.(1 & 2)
Take quiz

Purpose and content

Test 2 assesses your problem-solving and programming ability, especially to see how well you have learned the implementation lessons from the projects that accompany

The test consists in one programming problem from each of the following categories (three problems total):

- ADTs (Chapters 1–3, including array forests, heaps, bit vectors, and linear-time sorting)
- Graphs
- BSTs

Make sure you understand bounded linear search, binary search, iterators, using array indices as keys, breadth- and depth-first traversal, MST and SSSP concepts, BST structure and search, and rules for balanced BST schemes.

When grading the test I will use JUnit tests as an aide to understanding your code, but your score will **not** be based on the number of JUnit tests your code passes. Rather, your submission will be scored and partial credit assessed based on conceptual pieces I find when reading your code. You may include comments, which I will read. But since running your code against test cases will be part of the grading process, submitting code that doesn't compile is unlikely to be strategic.

Unlike projects *you will not have the JUnit tests I use for grading.* I will give you one or two simple JUnit tests per problem, but these will only be to clarify the problem, analogous to a clarification like "For example, if your method is given x, it should return y." **Whether your code passes the one given JUnit test is not a good indicator of whether your solution is correct.** Of course you may write your own JUnit tests, though time constraints may make that difficult.

Optimal binary search trees

Why this problem?

- It connects dynamic programming with the quest for a better map.
- Its hardness is in the right places (building the table—hard; reconstructing solution—trivial).
- It is a representative of a bigger concept: What if we had more information—how would that change the problem.

Game plan:

- Understand the problem itself
- Understand the recursive characterization
- Understand the table-building algorithm

The **optimal binary search tree** problem:

- Assume we know all the keys $k_0, k_1, \ldots k_{n-1}$ ahead of time.
- Assume further that we know the probabilities $p_0, p_1, \ldots p_{n-1}$ of each key's lookup.
- Assume even further that we know the "miss probabilities" $q_0, q_1, \ldots q_n$ where q_i is the probability that an *extraneous key* falling between k_{i-1} and k_i will be looked up.
- We want to build a tree to minimize the *expected cost* of a look up, which is the *total weighted depth* of the tree:

$$\sum_{i=0}^{n-1} p_i \; depth(k_i) + \sum_{i=0}^{n} q_i \; depth(m_i)$$

  where $depth(x)$ is the number of nodes to be inspected on the route from the root to node x, k_i stands for the node containing key k_i [notational abuse], and m_i is the dummy node between keys k_{i-1} and and k_i.
- Note that the rules of probability require $\sum_{i=0}^{n-1} p_i + \sum_{i=0}^{n} q_i = 1$

i	84	eat	24	ham	10	fox	7	rain	4
not	83	will	21	there	9	on	7	see	4
them	61	sam	19	train	9	tree	6	try	4
a	59	with	19	anywhere	8	say	5	boat	3
like	44	am	16	house	8	so	5	that	3
in	40	could	14	mouse	8	be	4	are	2
do	36	here	11	or	8	goat	4	good	2
you	34	the	11	box	7	let	4	thank	2
would	26	eggs	10	car	7	may	4	they	2
and	24	green	10	dark	7	me	4	if	1

Key or miss event	combined frequency
{ }	0
a	59
{ am and anywhere are be boat box car could dark }	92
do	36
{ eat eggs fox goat good green ham here house }	86
i	84
{ if let }	5
in	40
{ }	0
like	44
{ may me mouse }	16
not	83
{ on or rain same say see so thank that the }	65
then	61
{ there they train tree try will with would }	99
you	34
{ }	0

	0	1	2	3	4	5	6	7	
k_i	a	do	i	in	like	not	then	you	
p_i	.073	.045	.104	.05	.055	.103	.076	.042	
q_i	.001	.113	.107	.006	.001	.02	.081	.122	.001