Opening Questions

1. In the Optimal Binary Search Tree problem we are not just trying to balance the tree, instead we are trying to minimize what?
2. What is different about the 0-1 knapsack problem as compared to the other problems we solved with dynamic programming?

Optimal Binary Search Tree

The Optimal Binary Search Tree problem is a special case of a BST where the data is static (no inserts or deletes) and we know the probability of each key in the data being searched for. We want to minimize total expected search time for the BST. Recall expectation

$$E(X) = \sum_{s \in S} X(s)p(s)$$

1. Apply the expectation formula to the Optimal Binary Search Tree problem with n keys and $C(i)$ is how deep item i is. $P(i)$ is probability of search for item i.
2. The brute force approach would be to find all possible BSTs, find the expected search time of each and pick the minimum one. How many BSTs are there?
3. Step 1: Generically define the structure of the optimal solution to the Optimal Binary Search Tree problem. The optimal binary search tree with n keys and $C(i)$ is how deep item i is and $P(i)$ is probability of search for item i is:
4. Step 2: Recursively define the optimal solution. Assume $A(i, j)$ is the optimal answer for keys i to j. Make sure you include the base case.
5. Use proof by contradiction to show that Optimal Binary Search Tree problem has optimal substructure, i.e. the optimal answer to problem must contain optimal answers to sub-problems.
6. Step 3: Compute solution using a table bottom up for the Optimal Binary Search Tree problem. Use your answer to question 4 above. Note the overlapping sub-problems as you go.
7. Step 4: Construct Optimal solution

A	B	C	D	E	F
.2	.16	.08	.22	.21	.13

Optimal Binary Search Tree: http://www.cse.yorku.ca/~aaw/Gubarenko/BSTAnimation.html