17: Dynamic Programming

CS1101S: Programming Methodology

Martin Henz

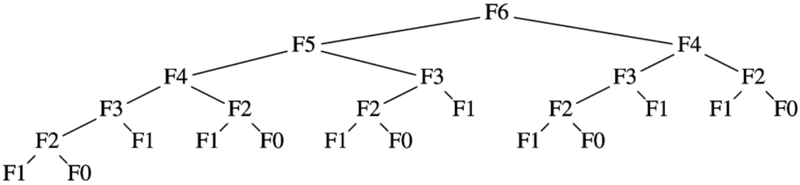
October 19, 2012

1. Fibonacci Numbers
2. Dropping Eggs Puzzle
3. Optimal Binary Search Tree

Inefficient Algorithm

```
function fib(n) {
  if (n <= 1) {
    return 1;
  } else {
    return fib(n - 1) + fib(n - 2);
} }
```

Trace of Recursion



Memoization

```
var fibs = [];
function fib(n) {
  if (fibs[n]!==undefined) {
     return fibs[n];
  } else if (n <= 1) {
     return 1;
  } else {
     var new_fib = fib(n - 1) + fib(n - 2);
     fibs[n] = new_fib;
     return new_fib;
} }
```

A Simple Loop for Fibonacci Numbers

```
function fib(n) {
  if (n <= 1) {
    return 1;
  } else {
    var last = 1, nextToLast = 1; answer = 1;
    var i = 2;
    while (i <= n) {
      answer = last + nextToLast;
      nextToLast = last;
      last = answer;
      i = i + 1;
    }
    return answer;
} }
```

1. Fibonacci Numbers
2. Dropping Eggs Puzzle
3. Optimal Binary Search Tree

Egg Dropping Puzzle

Given

n eggs, building with k floors

Wanted

Smallest number of egg dropping experiments required to find out in all cases, which floors an egg can be safely dropped from

Assumptions

- An egg that survives a fall can be used again.
- A broken egg must be discarded.
- The effect of a fall is the same for all eggs.
- If an egg breaks when dropped, then it would break if dropped from a higher floor.
- If an egg survives a fall then it would survive a shorter fall.
- A first-floor drop may break eggs, and eggs may survive a drop from the highest floor.

Special Case: One Egg

Number of eggs $= 1$, number of floors $= 21$

We need at most 21 experiments

Special Case: Two Eggs

Animated scenario

click here

Observations

Sub-tasks

At each point in time, we have a number of eggs n available and a number of floors k to check

Contiguous floors to check

The height of the floors does not matter. At each point in time we need to check a certain number of contiguous floors, say from 10 to 14.

Height does not matter

Checking 10 to 14 is the same as checking 20 to 24.

A simple algorithm

```
function eggDrop(n, k) {
    if (k =< 1 || n === 1) {
        return k;
    } else {
        var min = large_constant;
        var x = 1;
        var res = undefined;
        while (x <= k) {
            res = max(eggDrop(n-1, x-1),
                      eggDrop(n, k-x));
            if (res < min) min = res;
            x = x + 1;
        }
        return min + 1;
} }
```

Solution Idea

Observation

We compute eggDrop(i,j) over and over again.

Remember results in a table

Allocate a 2-D table eggFloor that remembers the results; after computing s = eggDrop(i,j), remember s in a table.

```
eggDrop[i][j] = s;
```

1. Fibonacci Numbers
2. Dropping Eggs Puzzle
3. Optimal Binary Search Tree

Optimal Binary Search Tree

Given

- a set of words $\{w_1, \ldots, w_n\}$
- probabilities of each word's occurrence $\{p_1, \ldots, p_n\}$

Wanted

Binary tree that includes all words and has the lowest expected cost:

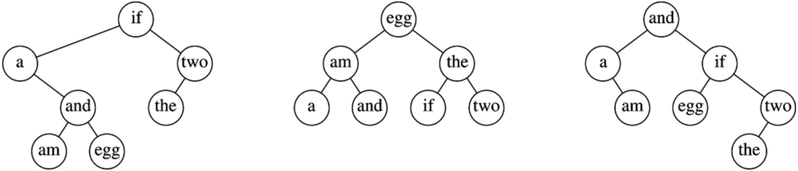
$$\textit{expected cost} = \sum_{i=1}^{n} d_i p_i$$

where d_i is the depth of word i in the tree

Sample Input

Word	Probability
a	0.22
am	0.18
and	0.20
egg	0.05
if	0.25
the	0.02
two	0.08

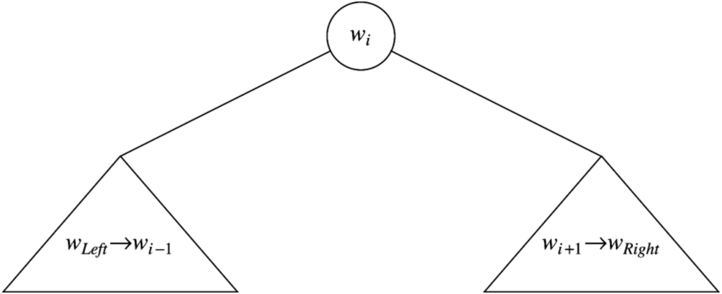
Three Possible Binary Search Trees



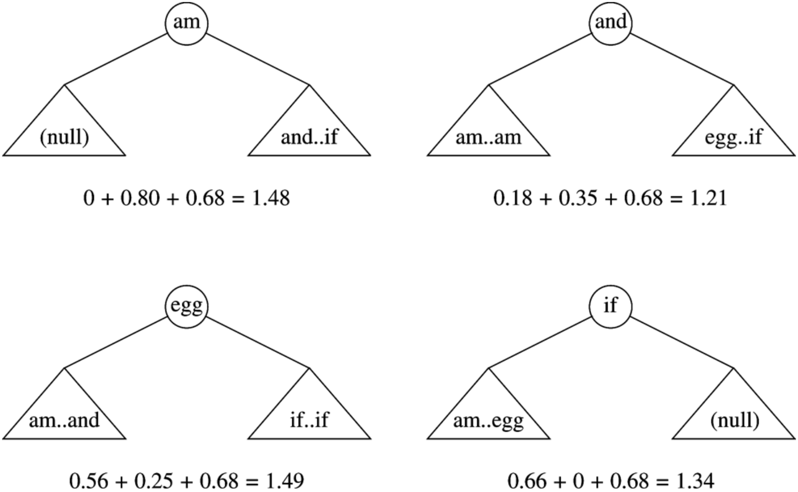
Comparison of the Three Trees

Input		Tree #1		Tree #2		Tree #3	
Word w_i	Probability p_i	Access Cost Once	Access Cost Sequence	Access Cost Once	Access Cost Sequence	Access Cost Once	Access Cost Sequence
a	0.22	2	0.44	3	0.66	2	0.44
am	0.18	4	0.72	2	0.36	3	0.54
and	0.20	3	0.60	3	0.60	1	0.20
egg	0.05	4	0.20	1	0.05	3	0.15
if	0.25	1	0.25	3	0.75	2	0.50
the	0.02	3	0.06	2	0.04	4	0.08
two	0.08	2	0.16	3	0.24	3	0.24
Totals	1.00		2.43		2.70		2.15

Structure of Optimal Binary Search Tree



Example


am
(null)
and..if
0 + 0.80 + 0.68 = 1.48
and
am..am
egg..if
0.18 + 0.35 + 0.68 = 1.21
egg
am..and
if..if
0.56 + 0.25 + 0.68 = 1.49
if
am..egg
(null)
0.66 + 0 + 0.68 = 1.34

Idea

Proceed in order of growing tree size

For each range of words, compute optimal tree

Memoization

For each range, store optimal tree for later retrieval

Computation of Optimal Binary Search Tree

	Left=1		Left=2		Left=3		Left=4		Left=5		Left=6		Left=7	
Iteration=1	a..a		am..am		and..and		egg..egg		if..if		the..the		two..two	
	.22	a	.18	am	.20	and	.05	egg	.25	if	.02	the	.08	two
Iteration=2	a..am		am..and		and..egg		egg..if		if..the		the..two			
	.58	a	.56	and	.30	and	.35	if	.29	if	.12	two		
Iteration=3	a..and		am..egg		and..if		egg..the		if..two					
	1.02	am	.66	and	.80	if	.39	if	.47	if				
Iteration=4	a..egg		am..if		and..the		egg..two							
	1.17	am	1.21	and	.84	if	.57	if						
Iteration=5	a..if		am..the		and..two									
	1.83	and	1.27	and	1.02	if								
Iteration=6	a..the		am..two											
	1.89	and	1.53	and										
Iteration=7	a..two													
	2.15	and												

Run Time

For each cell of table
Consider all possible roots

Overall runtime

$$O(N^3)$$