Two Dynamic Programming Examples

"Try All Values of k"

"Fill In the Table One Diagonal at a Time"

Example 1 Matrix-chain multiply.
Example from Figure 15.5, page 376 of the text.

Input 30, 35, 15, 5, 10, 20, 25
p_0 p_1 p_2 p_3 p_4 p_5 p_6

Matrix A_1 is p_0 by p_1; A_2 is p_1 by p_2; A_3 is p_2 by p_3;
A_4 is p_3 by p_4; A_5 is p_4 by p_5; A_6 is p_5 by p_6

Reminder: when we multiply a p-by-q matrix and a q-by-r matrix,

- Cost is p*q*r. (That's the number of multiplications. There are also p*(q-1)*r adds.)
- The result is a p-by-r matrix.

Recursive solution
To figure out the cost to multiply input range i to j: pick some k between i and j, and then use the optimal way to multiply i to k and the optimal way to multiply k to j:

$$cost(i,j) = cost(i,k)+cost(k,j) + p_i*p_k*p_j$$

Try all values of k in the range $i+1 \le k \le j-1$. Pick the cheapest one. If you write this as a recursive routine, the runtime is exponential because subproblems get recomputed many times. The dynamic programming solution is to use a table to store the solutions to subproblems.

The table used in dynamic programming
Table[i,j]=lowest cost for multiplying the matrices represented by $p_i...p_j$. That's A_{i+1} ... A_j. Only use the table entries with j>i. Fill in the table one diagonal at a time.

First diagonal j=i+1
Table[0,1] = 0. There is no multiply cost. We just have matrix A_1.

Second diagonal j=i+2
Table[0,2] = 15750. This is A_1A_2. The cost is $p_0*p_1*p_2$=15750.
The other entries along this diagonal are computed analogously.

	0	1	2	3	4	5	6
0		0	15,750				
1			0	2,625			
2				0	750		
3					0	1000	
4						0	5000
5							0
6							

Third diagonal j=i+3

Table[0,3]. This is $A_1A_2A_3$. Two ways to choose the top-level parentheses:

k=1 $(A_1)(A_2A_3)$ cost Table[0,1]+Table[1,3]+$p_0*p_1*p_3$ = 0+2625"5250=7875

k=2 $(A_1A_2)(A_3)$ cost Table[0,2]+Table[2,3]+$p_0*p_2*p_3$ = 15750+0+2250=18000

The winner is k=1, with cost 7875.

	0	1	2	3	4	5	6
0		0	15,750	7875			
1			0	2,625	4375		
2				0	750	2500	
3					0	1000	3500
4						0	5000
5							0
6							

Fourth diagonal j=i+4

Table[0,4]. We have $A_1A_2A_3A_4$. Three ways to choose the top-level parentheses:

k=1 $(A_1)(A_2A_3A_4)$ cost Table[0,1]+Table[1,4]+$p_0*p_1*p_4$ = 0+4375+10,500=14875

k=2 $(A_1A_2)(A_3A_4)$ cost Table[0,2]+Table[2,4]+$p_0*p_2*p_4$ = 15,750+750+4500=21000

k=3 $(A_1A_2A_3)(A_4)$ cost Table[0,3]+Table[3,4]+$p_0*p_3*p_4$ = 7875+0+1500=9375

The winner is k=3, with a cost of 9375

	0	1	2	3	4	5	6
0		0	15,750	7875	9375		
1			0	2,625	4375	7125	
2				0	750	2500	5375
3					0	1000	3500
4						0	5000
5							0
6							

Practice filling in the remaining entries of the table, for j=i+5 (two entries) and j=i+6 (one entry). You can check your answers using Figure 15.5, page 376. The final answer is 15,125, in the top right corner of the table. This final answer is obtained using k=3. Note that Figure 15.5 shows two tables: the table on the right records the value 3 in the top right corner. This second table is needed in order to keep track of the parenthesization that yields the optimal cost. The table I am showing here only records the optimal cost, and doesn't record "how to get there".

In summary, the top right corner of our table says that the best way to multiply these six matrices has a cost of 15,125. The 3 in the top right corner of the second table says the optimal parenthesization at the top level is $(A_1A_2A_3)(A_4A_5A_6)$. In order to figure out how to parenthesize the two subproblems, refer to the numbers recorded at [0,3] and [3,6].

Runtime for this algorithm is $O(n^3)$. Justification: there are $\Theta(n^2)$ table entries to fill in, and each entry takes $O(n)$ time to fill in. (As you see above, the entries along the biggest diagonal are fast to fill in, and as we get toward the corners it takes longer, because there are more possible values of k to try out.)

Example 2 CYK parsing algorithm.
This algorithm as quite similar to matrix-chain multiply: fill in a table along the diagonals.

The grammar is in Chomsky normal form, which means that the right side of a production consists of exactly two non-terminal symbols, or one terminal symbol. Any context-free grammar can be converted to Chomsky normal form (without changing the language accepted by the grammar).

Here is a small example grammar:

```
S -> N INTR   // Start symbol can be replaced by Noun INTRansivite-verb
INTR -> TRANS N // INTRansitive-verb can be TRANSitive-verb Noun
INTR -> INTR PP // INTRansitive-verb PrepositionalPhrase
N -> N PP
PP -> PREP N   // PrepositionaPhrase can be PREPosition Noun
N -> they        N -> cans        N -> fish
PREP -> in       PREP -> by        PREP -> with
INTR -> sleep    INTR -> fish
TRANS -> study   TRANS -> visit
```

This sentence is the input to the parser:

They study fish in cans

1 2 3 4 5

Table[i,j]=the set of nonterminals that can generate input items i to j
The answer is in the top right of the table: if that entry contains S (the start symbol) then the sentence is accepted by the grammar.
Only use the table entries with j>=i. Fill in the table one diagonal at a time.

First diagonal j=i
Table[1,1] This is the word "they". There is a rule "N -> they", so write "N" into this entry.
Table[3,3] This is the word "fish"; there are rules "N->fish" and "INTR->fish"

	1	2	3	4	5
1	N				
2		TRANS			
3			N, INTR		
4				PREP	
5					N

Second diagonal j=i+1
Table[1,2]. This is "They study". Look for a rule that has Table[1,1] and Table [2,2] on the right side: look for N TRANS on the right side. There is no such rule.
Table[2,3]. This is "study fish". Look for a rule that has Table[2,2] and Table[3,3] on the right side: look for TRANS followed by N or INTR. There is a rule "INTR->TRANS N".
The other entries along this diagonal are computed analogously.

	1	2	3	4	5
1	N	-			
2		TRANS	INTR		
3			N, INTR	-	
4				PREP	PP
5					N

Third diagonal j=i+2

Table[1,3]. This is "They study fish". Try k=1 and k=2 to see if either works.

k=1. Look for a rule that has Table[1,1] and Table [2,3] on the right side: look for N INTR on the right side. There is the rule "S->N INTR".

k=2. Look for a rule that has Table[1,2] and Table[3,3] on the right side: not possible because Table[1,2] is empty.

	1	2	3	4	5
1	N	-	S		
2		TRANS	INTR	-	
3			N, INTR	-	INTR,N
4				PREP	PP
5					N

The final table is

	1	2	3	4	5
1	N	-	S	-	S
2		TRANS	INTR	-	INTR
3			N, INTR	-	INTR,N
4				PREP	PP
5					N

The entry at the top right is computed as follows:

Table[1,5]. This is "They study fish in cans." Try k=1,2,3,4 and see if any of them work.

k=1. Look for Table[1,1] and Table [2,5] on the right side: N INTR. Yes S->N,INTR

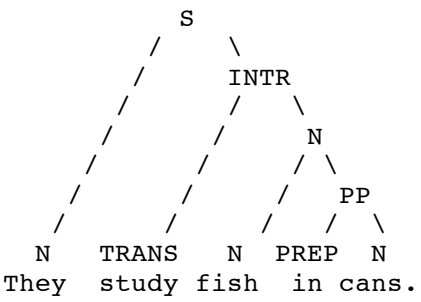
k=2. Look for Table[1,2] and Table [3,5] on the right side: empty Table[1,2]. Fail.

k=3. Look for Table[1,3] and Table [4,5] on the right side: S PP. Fail.

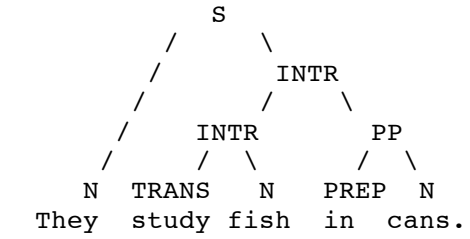
k=4. Look for Table[1,4] and Table [5,5] on the right side: empty Table[1,4]. fail.

The top right entry contains S, so we know that this sentence is accepted by the grammar. To produce a parse tree, use a separate table to keep track of which production rules were used. (Analogous to the second table for matrix-chain multiply, shown in the right half of Figure 15.5.)

Two possible parses:



The fish are in cans; they are studying the fish.



They are in cans, and studying fish there. Similar to "They study fish in laboratories."