

Optimization Problems

In the algorithms studied so far, correctness tended to be easier than efficiency. In optimization problems, we are interested in finding a *thing* that maximizes or minimizes some function.

In designing algorithms for optimization problem - we must prove that the algorithm in fact gives the best possible solution.

Greedy Algorithms

Greedy algorithms (which make the best local decision at each step) occasionally produce a global optimum - but you need a proof!

Example: You have six hours to complete as many tasks as possible, all of which are equally important.

Task A - 2 hours	Task D - 3.5 hours
Task B - 4 hours	Task E - 2 hours
Task C - 1/2 hour	Task F - 1 hour

How many can you get done?

Dynamic Programming

Dynamic Programming is a technique for computing recurrence relations efficiently by storing partial results.

Computing Fibonacci Numbers

$F(n) = F(n-1) + F(n-2)$; $F(0) = 0, F(1) = 1$

Recursive Solution:

How slow is slow?

$F(n+1) / F(n) \approx 1.618$

So, $F(n) \approx 1.618^n$, and since our recursion tree has 0 and 1 as leaves, we have approximately 1.618^n calls.

Thus, the recursive algorithm is $O(1.618^n)$

What about Dynamic Programming?

We can calculate $F(n)$ in linear time by storing small values.

```

F[0] = 0
F[1] = 1
for i = 2 to n
    F[i] = F[i-1] + F[i-2]
return F[n]
    
```

Moral: We can sometimes trade space for time.

Greedy vs. Dynamic Programming

Problem: Little Red Riding hood was assembling a picnic basket to bring to her grandmother's house. Unfortunately, she only had a small picnic basket capable of holding 5 pounds. She had available:

- 3.5 pounds of apples (her grandmother's favorite!)
- 2 pounds of grapes (liked 50% as much as apples)
- 3 pounds of plums (liked 90% as much as apples)
- 1 pound of onions (liked 10% as much as apples)

How much of each should Red bring?

Three Components of Dynamic Programming

1. Formulate the answer as a recurrence relation or recursive algorithm.
2. Show that the number of different instances of your recurrence is bounded by a polynomial.
3. Specify an order of evaluation for the recurrence so you always have what you need.

Multiplying a Sequence of Matrices

Suppose we want to multiply a long sequence of matrices

$$A \times B \times C \times D \times \dots$$

$$\begin{bmatrix} 2 & 3 \\ 3 & 4 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 3 \\ 4 & 4 & 5 \end{bmatrix} \times \begin{bmatrix} 5 & 6 & 5 \\ 7 & 6 & 7 \\ 6 & 8 & 9 \end{bmatrix} \times \dots$$

Multiplying an X x Y matrix by a Y x Z matrix takes X x Y x Z multiplications (using a common algorithm).

Example

Suppose we are multiplying 4 matrices:

A is 30x1, B is 1x40, C is 40 x10, and D is 10x25

There are three possible parenthesizations:

- ((A B) C) D $\Rightarrow 30 \times 1 \times 40 + 30 \times 40 \times 10 + 30 \times 10 \times 25 = 20,700$
- (A B) (C D) $\Rightarrow 30 \times 1 \times 40 + 40 \times 10 \times 25 + 30 \times 40 \times 25 = 41,200$
- A (B C) D $\Rightarrow 1 \times 40 \times 10 + 1 \times 10 \times 25 + 30 \times 1 \times 25 = 1,400$

The order makes a big difference in real computation. How do we find the best order?

Optimal Matrix Multiplication

Let $M(i, j)$ be the *minimum* number of multiplications necessary to compute: $\prod_{k=i}^j A_k$

$$M(i, j) = \text{Min}_{i \leq k \leq j-1} [M(i, k) + M(k+1, j) + (d_{i-1} d_k d_j)]$$

$$M(i, i) = 0$$

Matrix Multiplication with DP

		End					
		A	B	C	D	E	F
A	0	1200	700	1400	925	???	
B			0	400	650	775	925
C				0	10000	3250	9250
D					0	1250	2750
E						0	3750
F							0

- A x B x C x D x E x F = (A) (BCDEF)
- (AB) (CDEF)
- (ABC) (DEF)
- (ABCD) (EF)
- (ABCDE) (F)

Finding the Solution

```

int MatrixOrder()
for all i, j M[i, j] = 0;
for j = 1 to n
  for i = j-1 to 1
    M(i, j) = Min1 ≤ k ≤ j-1[M(i, k) + M(k+1, j) + (di-1dkdj)];
    factor[i, j]=k;
return M[1, n];

```

Printing the Solution

```

Procedure ShowOrder(i, j)
if (i=j) write ("Ai");
else
  k = factor [ i, j ] ;
  write " (" ;
  ShowOrder(i, k) ;
  write " x " ;
  ShowOrder (k+1, j) ;
  write ")";

```

The Partition Problem

Problem: k people need to divide up n items. Without taking the items out of order, what is the most even way to divide them up?

Example: Three people need to search through a filing cabinet with 9 file-folders in it. They are each going to take a section, and want them to be as evenly sized as possible, but don't want to break up items within a folder.

100 - 100 - 100 - 100 - 100 - 100 - 100 - 100 - 100

100 - 200 - 300 - 400 - 500 - 600 - 700 - 800 - 900