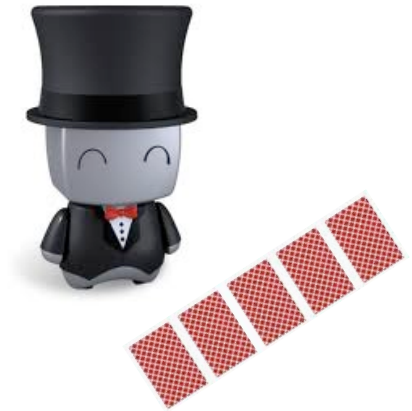


Robot Magician



You must program a robot magician.

The robot has a standard deck of 52 cards shuffled and face-down in front of it. When someone names a card, the robot must flip over cards until it finds the one named. The cards are then reset (exactly) and the next person gets to name a card.

But there's a problem! The robot's RAM is faulty, so every time a new card is named, it forgets every card it's seen before.

Write an algorithm that will maximize the robot's chance of *always* finding the correct card in 26 tries or fewer.

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?

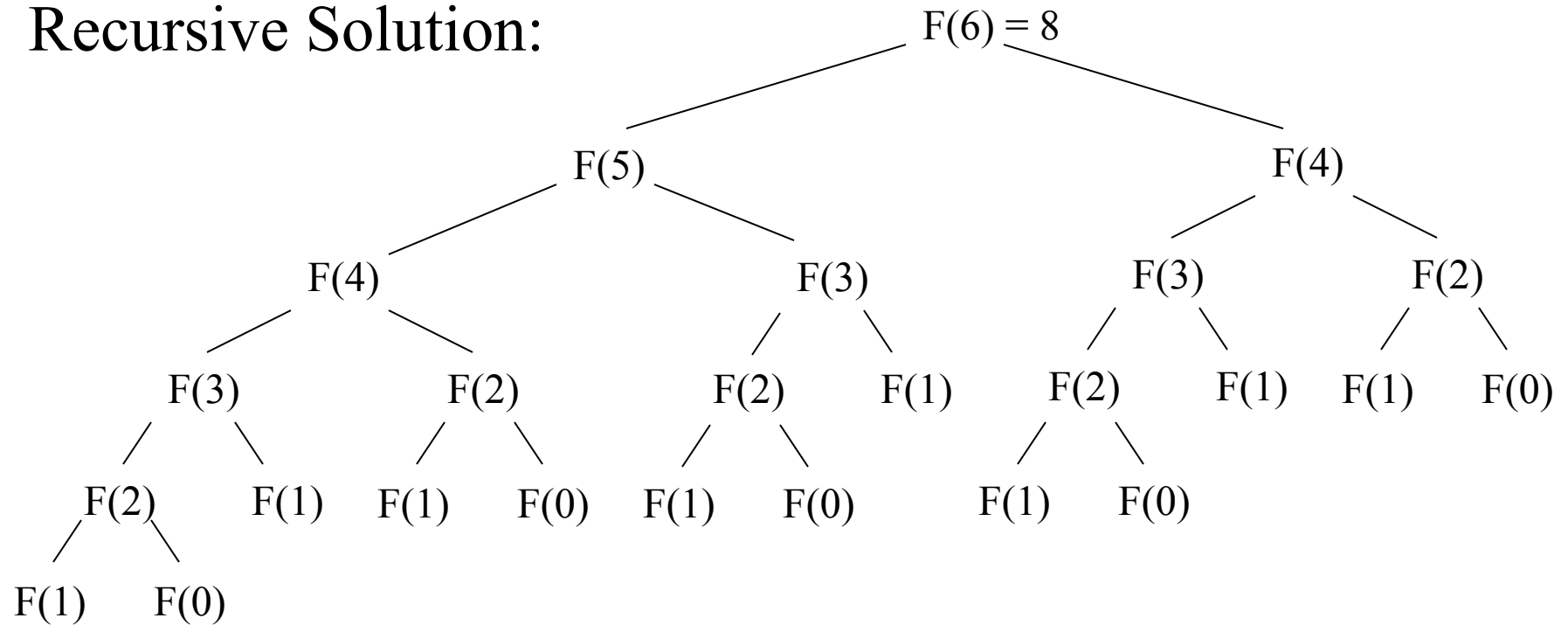
Dynamic Programming

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

Computing Fibonacci Numbers

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

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 $X \times Y$ matrix by a $Y \times Z$ matrix takes $X \times Y \times Z$ multiplications (using a common algorithm).

Example

Suppose we are multiplying 4 matrices:

A is **30×1**, B is **1×40**, C is **40 ×10**, and D is **10×25**

There are three possible parenthesizations:

$$((A B) C) D \Rightarrow 30 \times 1 \times 40 + 30 \times 40 \times 10 + 30 \times 10 \times 25 = \mathbf{20,700}$$

$$(A B) (C D) \Rightarrow 30 \times 1 \times 40 + 40 \times 10 \times 25 + 30 \times 40 \times 25 = \mathbf{41,200}$$

$$A ((B C) D) \Rightarrow 1 \times 40 \times 10 + 1 \times 10 \times 25 + 30 \times 1 \times 25 = \mathbf{1,400}$$

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

$$A = 30 \times 1$$

$$B = 1 \times 40$$

$$C = 40 \times 10$$

$$D = 10 \times 25$$

$$E = 25 \times 5$$

$$F = 5 \times 30$$

	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

$$A \times B \times C \times D \times E \times F = (A) (BCDEF)$$

$$(AB) (CDEF)$$

$$(ABC) (DEF)$$

$$(ABCD) (EF)$$

$$(ABCDE) (F)$$

Finding the Solution

```
int MatrixOrder()
```

```
forall  $i, j$   $M[i, j] = 0$ ;
```

```
for  $j = 1$  to  $n$ 
```

```
    for  $i = j-1$  to 1
```

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

```
        factor[ $i, j$ ]= $k$ ;
```

```
return  $M[1, n]$ ;
```

Printing the Solution

Procedure ShowOrder(i, j)

if ($i=j$) write (“ A_i ”);

else

$k = \text{factor} [i, j]$;

 write “ (” ;

 ShowOrder(i, k) ;

 write “ \times ” ;

 ShowOrder ($k+1, j$) ;

 write “) ” ;

Problem

Give a $O(n^2)$ algorithm to find the longest monotonically increasing sequence in a sequence of n numbers.

6 2 9 8 3 1 7 4 5

First, assume that the sequence has to be consecutive.
Then assume you can take any sub-set.

The Principle of Optimality

To use dynamic programming, the problem must observe the *principle of optimality*, that whatever the initial state is, remaining decisions must be optimal with regard the state following from the first decision.

Combinatorial problems may have this property but may use too much memory/time to be efficient.

In other words: *The details of our past solutions won't affect our current solution.*

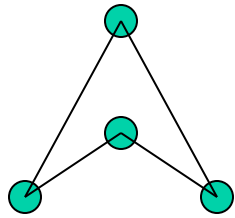
Example: The Traveling Salesman Problem

What recurrence relation will return the optimal solution to the Traveling Salesman Problem?

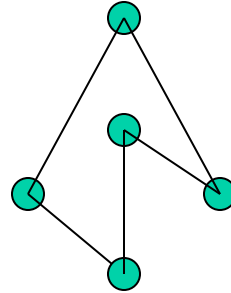
If $T(i)$ is the optimal tour on the first i points, will this help us in solving larger instances of the problem?

Can't we set $T(i+1)$ to be $T(i)$ with the additional point inserted in the position that will result in the shortest path?

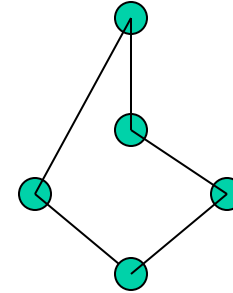
No!



T(4)



T(5)



Shortest Tour

A Correct Algorithm

Let $C(i, j)$ be the edge cost of moving from i to j . Further, Let $T(i ; j_1, j_2, \dots, j_k)$ be the optimal tour that goes from city 1 to i in minimal time, passing through each of the k cities exactly once in any order.

The cost of the optimal tour is thus $T(1 ; 2, 3, \dots, n)$.

$$T(i ; j_1, j_2, \dots, j_k) = \min_{1 \leq m \leq k} C(i, j_m) + T(j_m ; j_1, j_2, \dots, j_{m-1}, j_{m+1}, \dots, j_k)$$

When *can't* you use Dynamic Programming?

Dynamic programming computes recurrences efficiently by storing partial results. Thus speed ups occur when there are limited partial results are frequently reused.

Since there are $n!$ permutations of an n -element set, we cannot use dynamic programming to store the best solution for each subpermutation. There are 2^n subsets of an n -element set - we cannot use dynamic programming to store the best solution for each.

So, when *can* you use it?

There are only $n(n-1)/2$ contiguous substrings of a string, each described by a starting and ending point, so we can use it for string problems.

There are only $n(n-1)/2$ possible subtrees of a binary search tree, each described by a maximum and minimum key, so we can use it for optimizing binary search trees.

Summary

Dynamic programming works best on objects that are linearly ordered and cannot be rearranged - characters in a string, files in a filing cabinet, points around the boundary of a polygon, the left-to-right order of leaves in a search tree.

Whenever your objects are ordered in a left-to-right way, you should *smell* dynamic programming!

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