Algorithms

Lesson #4: Dynamic Programming

(Based on slides by Prof. Dana Shapira)

Fibonacci Sequence

- `F(0)=0`
- `F(1)=1`
- `F(n)=F(n-1)+F(n-2)`

- Write a recursive algorithm!
- What is its running time?

Binomial Coefficients

$$\frac{n!}{k!(n-k)!} = \binom{n}{k}$$

- Recursive equation:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

- Write a recursive algorithm!
- What is its running time?

Dynamic Programming Approach to Optimization Problems

1. Characterize structure of an optimal solution.
2. Recursively define value of an optimal solution.
3. Compute value of an optimal solution in bottom-up fashion.
4. Construct an optimal solution from computed information.

World Series Odds

- Two teams A and B play to see who is the first to win n games. In world series games $n=4$.
- Team A wins any particular game with probability q (same for all the games).
- What is the probability of A winning the tournament?

World Series Odds

`P(i,j)` - The probability that A will win the tournament, *given that it is missing i games to win and B is missing j games to win.*

```
P(0,j)=1      j>0
P(i,0)=0      i>0
P(i,j)=q*P(i-1,j)+(1-q)*P(i,j-1)   i>0 and j>0
```

Recursive Solution

```
P(i,j){
  if i==0 return 1
  elseif j==0 return 0
  else return q*P(i-1,j)+(1-q)*P(i,j-1)
}
```

- What is its running time?

Dynamic Programming

```
P(n,m){
   for(i=0; i≤n; i++)
         T[0,i] = 1
   for(j=1; j≤m; j++)
         T[j,0] = 0
   for(i=1; i≤n; i++)
      for(j=1; j≤m; j++)
         T[i,j]= q*T[i-1,j]+ (1-q)* T[i,j-1];
return T[n,m]
}
```

	0	1	2	3	4
0		1	1	1	1
1	0	1/2	3/4	7/8	15/16
2	0	1/4	1/2		
3	0	1/8			
4	0	1/16			

$q=1/2$

- What is its running time?

Matrix Chain Multiplication

-Problem:

Given n matrices $M_1, M_2, \ldots, M_n$, compute the product $M_1 M_2 M_3 \ldots M_n$, where M_i has dimension $d_{i-1} \times d_i$, for $i = 1, \ldots, n$.

-objective

compute $M_1, M_2, \ldots, M_n$ with the minimum number of scalar multiplications.

-Given matrices A with dimension $p \times q$ and B with dimension $q \times r$, multiplication AB takes pqr scalar multiplications

Matrix Chain Multiplication

-**Problem:** Parenthesize the product $M_1 M_2 \ldots M_n$ in a way to minimize the number of scalar multiplications.

-**Example:** M_1 --- 2 x 5

M_2 --- 5 x 3

M_3 --- 3 x 4

$M_1(M_2 M_3)$ --- 60 + 40 = 100 multiplications

$(M_1 M_2)M_3$ --- 30 + 24 = 54 multiplications

Matrix Chain Multiplication

- Let m(i,j) be the number of multiplications performed using optimal parenthesis of $M_i M_{i+1} \ldots M_{j-1} M_j$.

$$m(i,j) = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \{m(i,k) + m(k+1,j) + d_{i-1} d_k d_j\} & i < j \end{cases}$$

- Let S(i,j) be the optimal split point for $M_i \ldots M_j$ (meaning S(i,j)=k means we split $(M_i \ldots M_k)(M_{k+1} \ldots M_j)$)

- S(i,j) is needed to reconstruct the solution

Matrix Chain Multiplication

```
MUL(i,j){
   if (i==j) return 0;
   else{
        x←∞
        for k=i to j-1
              y←MUL(i,k)+MUL(k+1,j)+d_{i-1}d_k d_j
              if y<x {
                          x←y
                          S(i,j)←k
              }
        }
}
```

- What is its running time?

Recursive Running Time

$$T(n) = \sum_{k=1}^{n-1} \left(T(k) + T(n-k) + 1 \right)$$

$$T(n) = 2\sum_{k=1}^{n-1} T(k) + (n-1)$$

$$T(n-1) = 2\sum_{k=1}^{n-2} T(k) + (n-2)$$

$$T(n) - T(n-1) = 2T(n-1) + 1$$

$$T(n) = 3T(n-1) + 1 > 3T(n-1) > 3^2 T(n-2) > \cdots > 3^k T(n-k)$$

Dynamic Programming

-Example: M_1 --- 2 x 5

M_2 --- 5 x 3

M_3 --- 3 x 4

m	1	2	3
1	0	30	54
2		0	60
3			0

$$m(i,j) = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j}\{m(i,k) + m(k+1,j) + d_{i-1}d_k d_j\} & i < j \end{cases}$$

S	1	2	3
1		1	2
2			2
3			

Matrix Chain Multiplication

```
MUL(n){
   for (i=1;i≤n;i++) m[i,i]=0;
   for (diff=1;diff≤n-1;diff++){
       for (i=1;i≤n-diff;i++){
               j←i+diff
               x←∞
               for (k=i;k≤j-1;k++)
                       y←m[i,k]+m[k+1,j]+d_{i-1}d_kd_j
                       if y<x {
                               x←y
                               S(i,j)←k
                       }
               m[i,j]=x;
   }
}
```

The Integer Knapsack Problem - again

- Problem –
  Given $G_1, G_2, \dots, G_n$, each G_i with integer weight w_i and value v_i.
  Maximize the profit out of the goods you can put in the knapsack with capacity C.
- $f_i \in \{0,1\}$ (can't cut goods in the middle)
- Maximize $\sum_{i=1}^{n} f_i v_i$
- Subject to $\sum_{i=1}^{n} f_i w_i \leq C$
- How can we solve this using DP?

- Let $T[i,B]$ be the maximum profit achievable with goods $G_1,\ldots,G_i$, under weight limit B

- We are interested in $T[n,C]$

- Recursive formula for $T[i,B]$:

$$T[i,B] = \max(T[i-1,B],\ v_i + T[i-1,B-w_i])$$

- Boundary constraints…

$C = 15$

v	4	8	7	1	2	10
w	9	6	3	4	5	2

weight limit / up to item	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	4	4	4	4	4	4	4
2	0	0	0	0	0	0	**8**	8	8	**8**	8	8	8	8	8	12
3	0	0	0	7	7	7	8	8	8	**15**	15	15	15	15	15	15
4	0	0	0	7	7	7	8	8	8	15	15	15	15	16	16	16
5	0	0	0	7	7	7	8	8	9	15	15	15	15	16	17	17
6	0	0	10	10	10	17	17	17	18	18	19	25	25	25	25	26

What is the running time?

Running time = Θ(Size of the table)

Size of the table = n*C

Is this polynomial time in the size of the input?

NO! Because the size of C in the input is $O(\log C)$!

"Pseudopolynomial"

The table just gives us the profit. How do we reconstruct the actual packing…?