BITS, PILANI – K. K. BIRLA GOA CAMPUS

Design & Analysis of Algorithms

(CS F364)

Lecture No. 5

Coin Change Problem

Suppose we want to compute the minimum number of coins with values

d[1], d[2], …,d[n] where each d[i]>0

& where coin of denomination i has value d[i]

Let c[i][j] be minimum number of coins required to pay an amount of j units 0<=j<=N using only coins of denomination s 1 to i, 1<=i<=n

C[n][N] is the solution to the problem

Coin Change Problem

In calculating c[i][j], notice that:

- Suppose **we do not use** the coin with value d[i] in the solution of the (i,j)-problem,

  then **c[i][j] = c[i-1][j]**
- Suppose **we use** the coin with value d[i] in the solution of the (i,j)-problem,

  then **c[i][j] = 1 + c[i][j-d[i]]**

Since we want to minimize the number of coins, we choose whichever is the better alternative

Coin Change Problem – Recurrence

Therefore

c[i][j] = min{c[i-1][j], 1 + c[i][j-d[i]]}

&

c[i][0] = 0 for every i

Alternative 1

Recursive algorithm

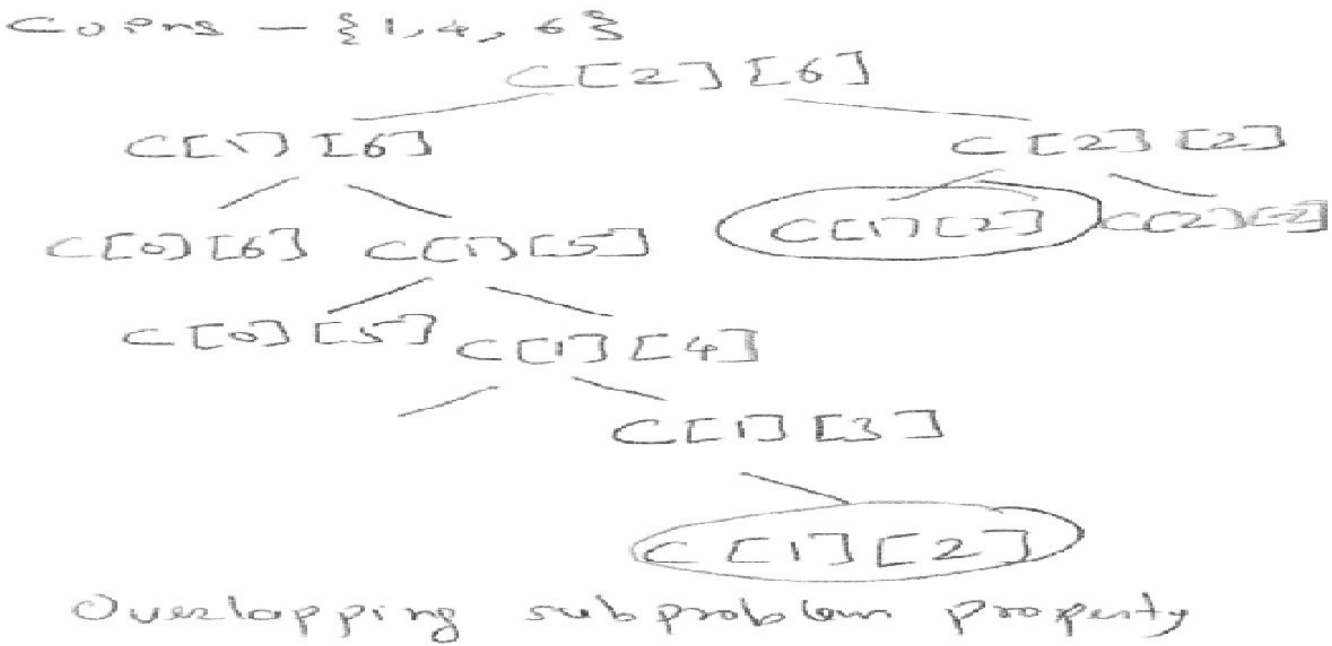
When a recursive algorithm revisits the same problem over and over again, we say that the optimization problem has **overlapping subproblems.**

How to observe/prove that problem has overlapping subproblems.

Answer – **Draw Computation tree and observe**

Overlapping Subproblems

Computation Tree



Dynamic-programming algorithms typically take advantage of overlapping subproblems by solving each subproblem once and then storing the solution in a table where it can be looked up when needed, using constant time per lookup.

Coin Change Problem

- **Example**

We have to pay 8 units with coins worth 1,4 & 6 units

For example c[2][6] is obtained in this case as the smaller of c[1][6] and 1+ c[2][6-d[2]] = 1+c[2][2]

The other entries of the table are obtained similarly

	0	1	2	3	4	5	6	7	8
d[1]=1	0	1	2	3	4	5	6	7	8
d[2]=4	0	1	2	3	1	2	3	4	2
d[3]=6	0	1	2	3	1	2	1	2	2

The table gives the solution to our problem for all the instances involving a payment of 8 units or less

Analysis

Time Complexity

We have to compute **$n(N+1)$** entries

Each entry takes **constant time** to compute

Running time – **$O(nN)$**

Question

- How can you modify the algorithm to actually compute the change (i.e., the multiplicities of the coins)?
- Modify the algorithm to handle exceptional cases.

Optimal Substructure Property - ReCap

- **Why does the solution work?**

Optimal Substructure Property/ Principle of Optimality

- ***The optimal solution to the original problem incorporates optimal solutions to the subproblems.***
- ***In an optimal sequence of decisions or choices, each subsequence must also be optimal***

This is a hallmark of problems amenable to dynamic programming.

- Not all problems have this property.

Optimal Substructure Property

- In our example though we are interested only in ***c[n][N],*** we took it granted that all the other entries in the table must also represent optimal choices.
- If ***c[i][j]*** is the optimal way of making change for ***j*** units using coins of denominations 1 to i, then ***c[i-1][j]*** & ***c[i][j-d[i]]*** must also give the optimal solutions to the instances they represent

Optimal Substructure Property

How to prove **Optimal Substructure Property?**

Generally by Cut-Paste Argument or By Contradiction

Note

Optimal Substructure Property looks obvious

But it does not apply to every problem.

Exercise:

Give an problem which does not exhibit Optimal Substructure Property.

Dynamic Programming Algorithm

The **dynamic-programming** algorithm can be broken into a sequence of **four steps.**

1. Characterize the structure of an optimal solution.

   Optimal Substructure Property

2. **Recursively define** the value of an optimal solution.
3. Compute the value of an optimal solution in a bottom-up fashion.

   Overlapping subproblems

4. Construct an optimal solution from computed information.

(not always necessary)

Matrix Chain Multiplication

- **Recalling Matrix Multiplication**

If **A** is **p x q** matrix and **B** is **q x r** matrix

then the product **C = AB** is a **p x r** matrix given by

$$c[i,j] = \sum_{k=1}^{q} a[i,k]b[k,j]$$

where $1 \leq i \leq p$ and $1 \leq j \leq r$

OR

Dot product of **ith row of A** with **jth column of B**

Properties

Properties of Matrix Multiplication

If A is p x q matrix and B is q x r matrix then the product C = AB is a p x r matrix

- If AB is defined, BA may not be defined except for square matrices
- Even if BA is defined, it is possible that AB ≠ BA i.e. **Matrix Multiplication is not commutative**
- Each element of the product requires *q* multiplications,
- And there are *pr* elements in the product
- Therefore, Multiplying an $p \times q$ and a $q \times r$ matrix requires ***pqr* multiplications**

Properties

- **Matrix multiplication is associative**

  i.e., $A_1(A_2A_3) = (A_1A_2\,)A_3$

  So parenthesization does not change result
- It may appear that the amount of work done won't change if you change the parenthesization of the expression
- **But that is not the case!**

Example

- Let us use the following example:
  - Let A be a 2x10 matrix
  - Let B be a 10x50 matrix
  - Let C be a 50x20 matrix
- Consider computing **A(BC):**

Total multiplications = 10000 + 400 = **10400**

- Consider computing **(AB)C**:

Total multiplications = 1000 + 2000 = **3000**

Substantial difference in the cost for computing

Matrix Chain Multiplication

- Thus, our **goal** today is:
- Given a **chain of matrices** to multiply,
  determine the **fewest number of multiplications**
  necessary to compute the product.
- Let $\mathbf{d_i x d_{i+1}}$ denote the dimensions of matrix A_i.
- Let $\mathbf{A = A_0\ A_1\ \ldots\ A_{n-1}}$
- Let $\mathbf{N_{i,j}}$ denote the minimal number of multiplications necessary to find the product: $A_i\ A_{i+1}\ \ldots\ A_j$.
- To determine the minimal number of multiplications necessary $\mathbf{N_{0,n-1}}$ to find A,
- That is, determine how to parenthisize the multiplications

Matrix Chain Multiplication

1st Approach –

Brute Force

- Given the matrices $A1,A2,A3,A4$ Assume the dimensions of $A1=d0\times d1$ etc
- **Five possible parenthesizations** of these arrays, along with the number of multiplications:

$(A1A2)(A3A4):d0d1d2+d2d3d4+d0d2d4$

$((A1A2)A3)A4:d0d1d2+d0d2d3+d0d3d4$

$(A1(A2A3))A4:d1d2d3+d0d1d3+d0d3d4$

$A1((A2A3)A4):d1d2d3+d1d3d4+d0d1d4$

$A1(A2(A3A4)):d2d3d4+d1d2d4+d0d1d4$

Matrix Chain Multiplication

Questions?

- **How many possible parenthesization?**
- **At least lower bound?**

  The number of parenthesizations is atleast $\mathbf{\Omega(2^n)}$

  Exercise: Prove

The **exact number** is given by the **recurrence relation**

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

Because, the original product can be split into **two parts**

In **(n-1)** places.

Each split is to be parenthesized **optimally**

Matrix Chain Multiplication

Solution to the recurrence is the famous **Catalan Numbers**

$T(n) = \Omega(4^n/3^{n/2})$

Question : Any better approach?

Yes

Dynamic Programming

Matrix Chain Multiplication

Step1:

Optimal Substructure Property

If a particular parenthesization of the whole product is optimal,

then any sub-parenthesization in that product is optimal as well.

What does it mean?

- ***If*** **(A (B ((CD) (EF)))) is optimal**
- ***Then*** **(B ((CD) (EF))) is optimal as well**

How to Prove?

Matrix Chain Multiplication

- **Cut - Paste Argument**
  - **Because if it wasn't,**

  and say (((BC) (DE)) F) was better,

  then it would also follow that

  (A (((BC) (DE)) F)) was better than

  (A (B ((CD) (EF)))),

  - **contradicting** its **optimality!**