Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

CSC373S: Algorithm Design, Analysis & Complexity

Lecture 11

Monday January 30, 2017
based on notes by Denis Pankratov

Dynamic Programming (*continue*)

DP Subparadigms

1. **Input:** $X_1, \dots, X_n$
   Subproblems: $X_1, \dots, X_i$
   Example: Weighted interval scheduling, longest increment subsequence

2. **Input:**
   $$X_1, \dots, X_m$$
   $$Y_1, \dots, Y_n$$
   Subproblems:
   $$\begin{Bmatrix} X_1, \dots, X_i \\ Y_1, \dots, Y_j \end{Bmatrix}$$
   Number of subproblems: $\mathcal{O}(mn)$
   Example: Largest common subsequence (substring)

3. **Input:** $X_1, \dots, X_n$
   Subproblems: $X_i, \dots, X_j$ (*contiguous*)
   Number of subproblems: $\mathcal{O}(n^2)$
   Example: Chain matrix multiplication

4. **Input:** $X_1, \dots, X_n$, N-positive integer
   Subproblems: $(X_i, \dots, X_i, K)$, where $1 \leq K \leq N$
   Number of subproblems: $\mathcal{O}(nN)$ (*not polynomial in the size of input*)
   Example: O/I Knapsack

5. **Input:** rooted tree
   Subproblems: subtrees rooted at various vertices
   Number of subproblems: number of vertices
   Example: Maximum independent set in trees

Coin Change Problem

Input:

D - array of n denominations (positive integers)
N - positive integer

Output: X - array of n non-negative integers, such that:

$$\sum_{i=1}^{n} X[i]D[i] = N$$
$$\sum_{i=1}^{r} X[i] \text{ is as small as possible}$$

Semantic Array: $C[K] = \min \#$ of coins needed to make change for $K, 0 \leq K \leq N$

Computational Array:

$$C[K] = 1 + \min\{C[K - D[i]] | 1 \leq i \leq n, D[i] \leq K\}$$
$$C[0] = 0$$

(*Note: The level of detail below will be expected in the midterm*)
Equivalence: Optimal solution for value K, ($K \geq 1$), has to contain at least one coin of some denomination, say $D[i]$, *and* optimal solution for the reduced problem, value $K - D[i]$.

This is true because of the standard *cut-&-paste* argument (formally, it is argued by contradiction, and it suffices to refer to this argument unless explicitly told to expand on it).

Pseudocode for the value of $opt(\sum_{i=1}^{n} X[i])$:

```
def CoinChange(D, n):
    # 0-based indexing
    init array C of size N
    C[0] = 0

    # runtime: O(nN)
    for K = 1 to N:
        C[k] = float('inf')

        # runtime: O(n)
        for j = 1 to n:
            # runtime: O(1)
            if D[j] <= K and C[K - D[j]] + 1 < C[K]:
                C[K] = C[K - D[j]] + 1

    return C[N]
```

Runtime: $\mathcal{O}(nN)$

Definition: The edit distance between two strings is the minimum # of character insertions, deletions, replacements that are needed to transform one string into another.

Example:

S N O W Y
S U N N Y

S - N O W Y
S U N N - Y

1. insert u
2. replace o w/ n
3. delete w

Edit Distance: 3

Claim: transforming 1^{st} string into 2^{nd} takes the same # of operations as transforming the 2^{nd} into the 1^{st}

Input: the strings

$$X[1..m]$$

$$Y[1..n]$$

Output: edit distance between X & Y

Without loss of generality (WLOG):

1. transform X into Y
2. consider transformation operations performed from *left* to *right*

Semantic Array: $C[i,j] =$ edit distance between $X[1..i]$ and $Y[1..j], 0 \leq i \leq m, 0 \leq j \leq n$

Computational Array:

$$C[i,j] = \min \begin{cases} C[i-1,j-1], \text{if } X[i] = Y[j] \\ C[i-1,j-1] + 1 \leftrightarrow \text{turning character } X[i] \text{ into } Y[j] \\ C[i-1,j] + 1 \leftrightarrow \text{deleting character } X[i] \\ C[i,j-1] + 1 \leftrightarrow \text{inserting character } Y[j] \text{ after } X[i] \end{cases}$$

Exercise: finish proof of correctness, write pseudocode & analyze runtime