CSC373

Week 4

5 steps of Dynamic Programming

- 1. Optimal substructure definition
- 2. Memorization –
  - Define an appropriate array to store intermediate values
- 3. Rewrite the recurrence relation in terms of the array(s) defined in step 2
- 4. Bottom-up approach
- 5. Compute a path to an actual solution(not just value)

Example 1

Given: n requests $\{1, 2, \ldots, n\}$ with s_i (start time) and f_i (finish time)

Goal: Schedule jobs in such a way that you obtain maximum possible value/weight

Solution: Sort the jobs by their finish time and define
$$P(i) = \max i \text{ such that } i < j \text{ and request } i \text{ does not overlap with } j$$

Example 1 (continued)

Let O_n denote an optimal solution and $OPT(n)$ be its value

Step I: (sub-problems)

      Case 1: $n \in O_n$
$$OPT(n) = w_n + OPT\big(P(n)\big)$$
      Case 2: $n \notin O_n$
$$OPT(n) = OPT(n-1)$$

Example 1 (continued)

Step II: Define array for cache

Define $M[j] =$ the optimum value obtained with jobs $\{1, \ldots, j\}$

Step III: (most difficult part)

$$M_j = \begin{cases} w_j + M[P[i]] & if \ j \in O_j \\ M[j-1] & if \ j \notin O_j \end{cases}$$

Example 1 (continued)

Step IV:

    M_Compute_Opt(j):

        If j = 0 then

            return 0

        If M[j] is defined, then

            return M[j]

        Else

            M[j] = max(M_compute_Opt(j-1), w_j+ M_compute_Opt(P[j])

            return M[j]

        EndIf

M[n] is the final optimal value complexity: O(n)

Example 1 (continued)

Bottom-up Approach
> M_Bottom_up_Opt():
>> Define M[0 ... n] = (undefined)
>> M[0] = 0
>> for j = 1 to n
>>> M[j] = max(M[j-1], w_j + M[P[j]])
>> Endfor
> Return M[n]

Complexity $\Theta(n)$

Example 1 (continued)

Step V: Finding an optimal solution with the optimal value

```
M_compute_Path(j, M):
        If j = 0:
                return " "
        Else
                if w_j+M[P[j]] > M[j-1] then
                        output M_compute_math(P[j], M) + j
                Else
                        output M_compute_path(j-1, M)
                Endif
        Endif
```

Example 1 (continued)

M_compute_Path_Iterative(n, M):
        for j = n to 1;
                if M[P[j]] + w_j > M[j-1]
                        output j + " "
                        j = P[j]
                Else
                        j = j – 1
                EndIf
        Endfor

Example 2: Rod Cutting Problem

P[i] = Price for rod of length i

Goal: Cut up the rod into pieces so that the sum of the price of the pieces is maximum

Example 2 (continued)

Step I: Defining optimal substructure

if you cut the rod at location i then
$$OPT(n) = \max_{1 \le i \le n}(P[i] + OPT(n - i))$$

$$OPT(j) = \max_{1 \le i \le j}(P[i] + OPT(j - i))$$

Example 2 (continued)

Step II: Array definition
$M[j]$ = optimal value on a rod of length j

Step III:
$$M[j] = \max_{1 \le i \le j}(P[i] + M[j-1])$$

Example 2 (continued)

Step IV: Bottom-up Approach

Bottom_up_cut_Rod(P, n):
        Define M[0, …, n], S[0, …, n]
        M[0] = 0, S[0] = 0
        for j=1 to n:
                q = -∞
                for i=1 to j:
                        if q < P[i] + M[j-i] then
                                q = p[j] + M[j-i]
                                s[j] = i
                Endfor
        Endfor
        Return M and S

Complexity: 1+2+3+4+…+n = $\Theta(n^2)$

Example 2 (continued)

Proof that algorithm works

Proof by induction:
  n=0 M[0] = 0

Inductive Hypothesis(strong):
  Assume M[j] is the optimal value for $0 \leq j < n$
  (i.e. M[j] = O[j], for $0 \leq j < n$)
Inductive Step:
  Let the first cut for the optimal solution O be at $1 \leq i \leq n$
  O[n] = P[i] +O[n-i] = P[i] +M[n-i](by inductive hypothesis)
  => O[n] = M[n] (Since O is optimal)

Example 2 (continued)

Step V: Find a way of cutting up the rod optimally

Print_Path(P, n):
     (M, S) = Bottom_up_cut_Rod(P, n)
     while n > 0:
          print S[n]
          n = n – S[n]

Example 3

Subset Sum & Knapsack Problem

Given: n jobs $\{1, \ldots, n\}$ each with a value v_i and a weight w_i and a bound W
(Each jobs runs for 1 unit of time)

Goal: Find $S \subseteq \{1, \ldots, n\}$ such that
$\qquad \Sigma_{i \in S} V_i$ is maximum possible subject to $\Sigma_{i \in S} w_i \leq W$
$\qquad$ Subset sum is a special case of knapsack where $v_i = w_i$

Example 3 (continued)

Let O_n be an optimal solution and $OPT(n)$ be its value

$$OPT(n) = w_n + OPT(n-1)$$
wrong!

Example 3 (continued)

To take care of the constraint
$OPT(n, W)$
$$If\ n \notin O_n : OPT(n, W) = OPT(n - 1, W)$$
$$If\ n \in O_n : OPT(n, W) = w_n + OPT(n - 1, W - w_n)$$

Example 3 (continued)

Step I: $OPT(j, W) = \max(OPT(j-1, W), OPT(j-1, W-w_j) + w_j)$

Step II: Define array
M[1 ……n   1 ……w]
M[j, w] = the optimum value on {1, ……,j} jobs with upperbound w

Step III:
      M[j, W] = max(M[j-1][w], m[j-1, w-w_j] + w_j)

Example 3 (continued)

Step IV:

Subset_sum(n, W):
      Define M[0……n 0……W]
      M[0,w] = 0 for w = 0, ……,W
      for j = 1 to n:
            for w=1 to W:
                  if w < w_j then
                      M[j,w] = M[j-1, w]
                else
                    M[j, w] = max(M[j-1, w], M[j-1, w-w_j]+w_j)
      End
      Return M[n,w]

Example 3 (continued)

Complexity $\Theta(nw)$

Polynomial

      Expression involving an actual input value is called pseudo-polynomial

Knapsack – NP-complete (will talk about it later)

              "Approximation Algorithm"

          Functional knapsack: Polynomial

Example 3 (continued)

Step V: Actual Solution

     Run through M[j, w] and figure out if j was in the schedule or not

     Time complexity: $\Theta(n)$

Example 4: Largest Common Subsequence Problem

Given: Two sequences

$\quad$ $X = <x_1, x_2, ..., x_n>$

$\quad$ $Y = <y_1, y_2, ..., y_n>$

Goal: Find out a subsequence that is common to both x and y, and that has the maximum possible length

E.X. $\quad$ $X = <5, 10, 13, 12, 11>$

$\quad\quad$ $Y = <6, 10, 13, 7, 5, 11>$

$<10, 13>$ is a common subsequence of X and Y

$<10, 13, 11>$ is a longest common subsequence of X and Y

Example 4 (continued)

Solution:

Step I:

Case I: $X_m = Y_n$

$OPT(X, Y) = OPT(<x_1, ..., x_{m-1}>, <y_1, ..., y_{n-1}) + x_m$

Case II: $X_m \neq Y_n$

$OPT(X_{1...m}, Y_{1...n}) = \max(OPT(X_{1...m-1}, Y_{1...n}), OPT(X_{1...m}, Y_{1...n-1}))$

Example 4 (continued)

Step II: Array definition

  M[0......m 0 ......n]

  M[i,j]=length of a LCS(longest common subsequence of $X_{1......i}$ $Y_{1......j}$

Step III:

$$M[i,j] = \begin{cases} 0 & if\ i = 0\ or\ j = 0 \\ M[i-1,j-1]+1 & if\ x_i = y_j \\ \max(M[i-1,j], M[i,j-1]) & if\ x_i \neq y_j \end{cases}$$

Example 4 (continued)

Step IV: Bottom-up
       LCS(X[1, …,m] Y[1, …,m])
              Define M[0 ……m 0……n]
              for i=0……m
                    M[i,0]=0
              for j=0……m
                    M[0,j]=0
              for i=1……m:
                    for j=1……n:
                          if X[i]=Y[i] then
                              M[i,j]=M[i, j-1] +1
                        else
                              M[i,j]=max(M[i, j-1], M[i-1, j])
              Endfor
              Return M[m,n]

Complexity $\Theta(mn)$ - Polynomial

Example 4 (continued)

Step V:

```
LCSPath(M, X, Y, i, j)
        if i=0 or j=0 then
                return "   "
        else if X[i] = Y[i] then
                return LCSPath(M, X, Y, i-1, j-1)+X[i]
        else
                if M[i, j-1] > M[i-1, j] then
                        return LCSPath(M, X, Y, I, j-1)
                else
                        return LCSPath(M, X, Y, i-1, j)
        Endif
```

Complexity $\Theta(m + n)$