Combinatorial Optimization
CSE 301

Lecture 3

Dynamic Programming III

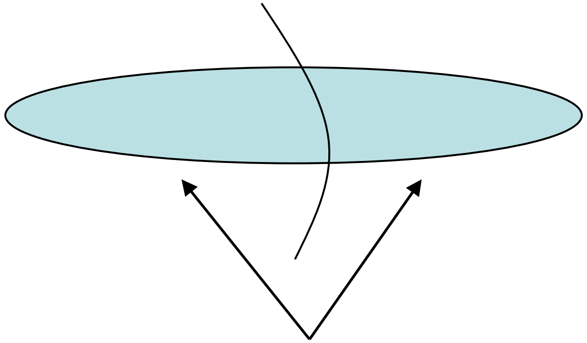
Dynamic Programming

- An algorithm design technique for optimization problems (similar to divide and conquer)
- Applicable when subproblems are not independent
  - Subproblems share subsubproblems
  - A divide and conquer approach would repeatedly solve the common subproblems
  - Dynamic programming solves every subproblem just once and stores the answer in a table

DP - Two key ingredients

- Two key ingredients for an optimization problem to be suitable for a dynamic-programming solution:

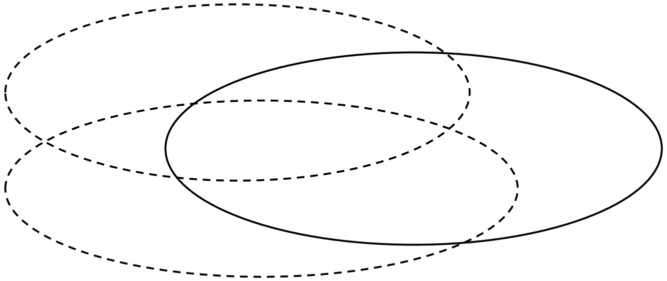
1. optimal substructures



Each substructure is optimal.

(Principle of optimality)

2. overlapping subproblems



Subproblems are dependent.

(otherwise, a divide-and-conquer approach is the choice.)

Elements of Dynamic Programming

- Optimal Substructure
  - An optimal solution to a problem contains within it an optimal solution to subproblems
  - Optimal solution to the entire problem is build in a bottom-up manner from optimal solutions to subproblems
- Overlapping Subproblems
  - If a recursive algorithm revisits the same subproblems over and over ⇒ the problem has overlapping subproblems

Dynamic Programming

- Used for **optimization problems**
  - A set of choices must be made to get an optimal solution
  - Find a solution with the optimal value (minimum or maximum)
  - There may be many solutions that return the optimal value: **an optimal solution**

Typical Steps of DP Algorithm

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

Memoization

- A variation of DP
- Keep the same efficiency as DP
- But in a top-down manner.
- Idea:
    - Each entry in table initially contains a value indicating the entry has yet to be filled in.
    - When a subproblem is first encountered, its solution needs to be solved and then is stored in the corresponding entry of the table.
    - If the subproblem is encountered again in the future, just look up the table to take the value.

Memoized Matrix-Chain

Alg.: MEMOIZED-MATRIX-CHAIN(p)

```
n ← length[p] - 1
for i ← 1 to n
    do for j ← i to n
        do m[i, j] ← ∞
return LOOKUP-CHAIN(p, 1, n)
```

Initialize the m table with large values that indicate whether the values of m[i, j] have been computed

Top-down approach

Memoized Matrix-Chain

Alg.: LOOKUP-CHAIN(p, i, j)

Running time is $O(n^3)$

```
if $m[i, j] < \infty$
    then return $m[i, j]$
if $i = j$
    then $m[i, j] \leftarrow 0$
    else for $k \leftarrow i$ to $j - 1$
            do $q \leftarrow$ LOOKUP-CHAIN(p, i, k) +
                   LOOKUP-CHAIN(p, k+1, j) + $p_{i-1}p_kp_j$
               if $q < m[i, j]$
                   then $m[i, j] \leftarrow q$
return $m[i, j]$
```

$$m[i, j] = \min_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\}$$

Dynamic Progamming vs. Memoization

- Advantages of dynamic programming vs. memoized algorithms
  - No overhead for recursion, less overhead for maintaining the table
  - The regular pattern of table accesses may be used to reduce time or space requirements
- Advantages of memoized algorithms vs. dynamic programming
  - Some subproblems do not need to be solved
  - Easier to think and to implement

Elements of Dynamic Programming

- Optimal Substructure
  - An optimal solution to a problem contains within it an optimal solution to subproblems
  - Optimal solution to the entire problem is build in a bottom-up manner from optimal solutions to subproblems
- Overlapping Subproblems
  - If a recursive algorithm revisits the same subproblems over and over ⇒ the problem has overlapping subproblems

Optimal Substructure - Examples

- Matrix multiplication
  - Optimal parenthesization of $A_i \cdot A_{i+1} \cdots A_j$ that splits the product between A_k and A_{k+1} contains:

    an optimal solution to the problem of parenthesizing $A_{i..k}$ and $A_{k+1..j}$

Parameters of Optimal Substructure

- How many subproblems are used in an optimal solution for the original problem
  - – Matrix multiplication: Two subproblems (subproducts $A_{i..k}$, $A_{k+1..j}$)
- How many choices we have in determining which subproblems to use in an optimal solution
  - – Matrix multiplication: $j - i$ choices for k (splitting the product)

Parameters of Optimal Substructure

- Intuitively, the running time of a dynamic programming algorithm depends on two factors:
  - Number of subproblems overall
  - How many choices we look at for each subproblem
- Matrix multiplication:
  - $\Theta(n^2)$ subproblems ($1 \le i \le j \le n$)
  - At most $n-1$ choices

$\Theta(n^3)$ overall

Summary

- DP two important properties
- Four steps of DP.
- Differences among divide-and-conquer algorithms, DP algorithms, and Memoized algorithm.
- Writing DP programs and analyze their running time and space requirement.

Further Reading

- TSP – Travelling Salesman Problem (Sahni)
- OBST – Optimal Binary Search Tree (Cormen)
- Optimal Polygon Triangulation (Sahni)