CS 491 CAP
Intermediate Dynamic Programming

Victor Gao
University of Illinois at Urbana-Champaign

Oct 27, 2017

Overview

- Linear DP
- Knapsack DP
- DP on a Grid
- Interval DP
- Division/Grouping DP
- Tree DP
- Set DP

Review: How to solve a DP problem?

- General steps to solve a DP problem:
  - Model the subproblems
  - Write down the recursion and the base case(s)
  - Implement the algorithm using memoization
    - No need to do it in the iterative way

Linear DP

- Linear DP Problems are one of the easiest ones to solve.
- Example: Given an array of integers, find the maximum length of a subsequence that is strictly increasing. (LIS)

- General approach: divide the problem into sub problems like "... ending with element i/... of the first i elements" and then derive a recursive relation of $f[i]$ and all the previous terms $f[0], f[1] \ \ldots \ f[i-1]$

LIS

- LIS: Given an array of integers, find the maximum length of a subsequence that is strictly increasing.

- Let v be the array Let f[i] be the maximum length of a strictly increasing sequence ending with element[i]

- Recursion:

```
f[i]= max(1,
          max({f[j] such that
               v[i] > v[j]})
```

illinois.edu

Knapsack Problem

- Given n items, each with a weight w_i and value v_i
- You have a bag of capacity c

- What is the maximum of the total value of the items you could fit in your bag?

Solution

- How do we define the sub problem?
- Let `f[i][j]` be the maximum value obtained by fitting the first *i* items into a bag of capacity *j*
- Recursion:

```
f[i][j] = max(f[i-1][j],
              f[i-1][j-w[i]] + v[i])
```

- What are the base cases?

Variations

- The knapsack problem we just introduced is actually called **0-1 knapsack**

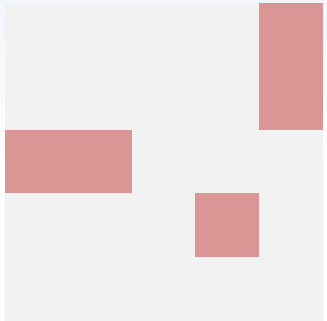
- There is only one copy for each item, so you make decisions of either taking it (1) or not taking it (0)

- Variations may allow multiple (or infinite) copies of a single item

DP on a Grid

- Given an n by m grid, with some of the cells being obstacles.

- You are only allowed to go right or down. What is the number of distinct paths from the top left corner to the bottom right corner?



Solution

- Let `f[i][j]` be the number of paths from the top left corner to cell (i, j)
- Recursion:

```
f[i][j] = f[i-1][j] + f[i][j-1]
```

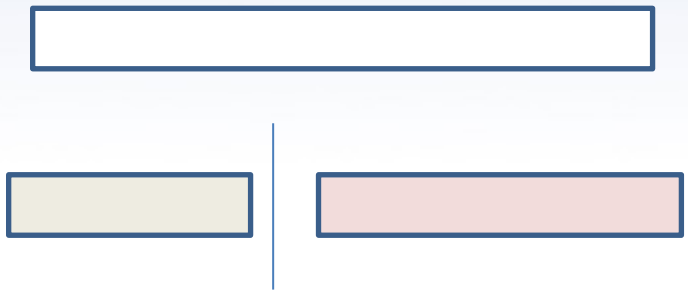
- What are the base cases?

```
f[0][0] = 1
f[i][j] = 0 if cell (i, j) is blocked
```

Interval DP

- Interval DP is a type of DP problems that typically asks for some value over some intervals.

- This gives us an intuitive way to construct the subproblems : bigger intervals are formed by smaller ones!



Interval DP

- Given n piles of stones with v_i stones in each pile.
- Each time you can pick two adjacent piles and combine them together. The cost of this operation is the total number of stones in the two piles.
- You want to know the minimum cost of combining all stones into one pile.

Solution

- Let `f[i][j]` be the cost of combining the stones in interval [i, j] into one pile.

- Recursion:

```
f[i][j] = min(f[i][k] + f[k+1][j] + sum(i, j))
                for all k such that i <= k < j

sum(i, j) is the total number of stones in
    the interval
```

- What are the base cases?

```
f[i][j] = 0 for all i = j
```

Division/Grouping DP

- Division/Grouping DP (I made up the name, don't know if there's a better one) is a type of problems similar to interval DP, but instead of asking for some optimum over an interval, it asks you to *divide* the interval into several parts, so that some other quantity is maximized.

Division/Grouping DP

- Given a string that contains n digits of 0 to 9, divide it into k parts, such that the product of the parts is maximized (treat each part as a number)

- Example: divide "1231" into 3 parts

```
Optimal: 1*2*31 = 62
```

Solution

- Let `f[i][j]` be the maximum product obtained by dividing the interval $[0, i]$ into j parts
- Recursion:

```
f[i][j] = max(f[k][j-1] * str[k+1 ... i])
                for all 0 <= k < i
```

- The base cases are a bit tricky. Hint: some divisions are impossible.

Tree DP

- We have been doing DP on arrays, matrices and intervals. Then… Why not DP on a tree?

- This type of problems asks for some optimal value over a tree structure. Decomposition is actually intuitive and obvious: recurse on the subtrees.

Tree DP

- We want to pick a set of nodes from a tree such that none of the nodes in the set is a parent or child of any other nodes.

- What is the maximum number of nodes in the set?

Solution

- Let `f[i][j]` be the maximum number obtained on the subtree rooted by node *i*. *j* is a Boolean value (0 or 1), indicating whether node i is in the selected set or not.

- Recursion:

```
f[i][0] = sum(max(f[k][0], f[k][1])) + 1
                for all k such that node k is
                    a child of node i
f[i][1] = sum(f[k][0]) + 1
                for all k such that node k is
                    a child of node i
```

Set DP

- We have been using the f array differently for each problem. The index of each dimension could be interpreted as indices of an interval, coordinates, boolean value...

- By using a bitmask as the index of a dimension, we can represent and solve problems that require more complicated prior knowledge.

Representing a Subset

- Let $x = \{x_1, x_2 \ldots x_n\}$ be a set of n elements.
- How can we represent a subset of it?
- For each element in the set, we use one bit to denote whether it is in the subset or not. (Bitmask)

```
for set {1, 2, 3}
000 -> {}  111-> {1, 2, 3}
001 -> {1}  010 -> {2}  100 -> {3}
011 -> {1, 2}  101 -> {1, 3}, 110 -> {2, 3}
```

Traveling Salesman Problem (TSP)

- Given n interconnected cities ($n \leq 20$) and the cost to travel from city i to city j is $v_{i,j}$, what is minimum of the total cost of visiting all cities exactly once? (You don't need to go back to the city you depart initially.)

- Hint: You need to know which cities you have already visited in the past, not just (along with) the city you visited last time.

Summary

- DP is more like a design diagram instead of a specific algorithm.

- The problems we talked about today are the most common ones that you could encounter. However do not limit your thought to them, some problems you meet in the future may require new ways to represent and compute the data.

Questions?