Dynamic Programming

Oliver-Matis Lill

April 10, 2018

Dynamic Programming

- Dynamic Programming (DP) is one of the most common problem solving methodologies
- DP consists of two parts
  1. Division of problem into "smaller" subproblems
  2. Recurrence relation that allows you to compute subproblem based on other "smaller" subproblems
- Generally you solve subproblems from "smaller" to "larger" until you get a solution to the subproblem you want
- Very similar to mathematical induction, difference is: induction is used to prove things while DP is used to compute things
- If the total no. of subproblems is $O(N)$ and the complexity of computing recurrence relation is $O(R)$, then total time complexity is usually $O(NR)$

Fibonacci Numbers Problem

- You have Q queries where each query asks for the modular value of some Fibonacci number f_i where $i \leq 10^6$
- Fibonacci numbers satisfy the recurrence relation $f_i = f_{i-1} + f_{i-2}$ with the base cases $f_0 = 0$ and $f_1 = 1$
- The numbers look like this:

i	0	1	2	3	4	5	6	7	8
f_i	0	1	1	2	3	5	8	13	21

- How would you solve this problem?
- How does this problem relate to Dynamic Programming?

Fibonacci Numbers Solution

- We can construct a DP solution in the following fashion:
  1. Subproblems: f_i = the i-th Fibonacci number, for $i \in [0, \ldots, N]$
  2. Recurrence relation: $f_i = f_{i-2} + f_{i-1}$
- We have $O(N)$ subproblems and calculating recurrence relation takes $O(1)$, therefore time complexity for calculating $f_0, \ldots, f_N$ is $O(N)$
- This problem is simple because recurrence relation is given and division into subproblems is obvious

Knapsack Problem

- You have a knapsack with capacity $W \leq 10^5$
- You have $N \leq 100$ items, each with size s_i and value v_i
- You want to fill the knapsack with items such that their total value is maximal while knapsack capacity is not exceeded
- This is the infamous NP-Complete Knapsack Problem simplified by the low upper-bound on W

Knapsack Idea

- Let's look at the last item (s_N, v_N)
- There are two possibilities
  1. It's in the knapsack. This means that our picks from items $(s_1, v_1), \ldots, (s_{N-1}, v_{N-1})$ must not exceed total size $W - s_N$, and their total value should be maximal
  2. It's not in the knapsack. This means that we must fill the knapsack optimally with items $(s_1, v_1), \ldots, (s_{N-1}, v_{N-1})$
- Can you see how we could use these insights to formulate a knapsack solution?

Knapsack Problem Solution

- The DP construction will be the following:
  1. Division into subproblems:

$$d[i][j] = \begin{array}{l}\text{maximum total value achievable with knapsack} \\ \text{capacity } i \text{ using items } 1, \ldots, j\end{array}$$

     Here $i \in [0, \ldots, W]$ and $j \in [0, \ldots, N]$
  2. Recurrence Relation:

$$d[i][j] = \max\{d[i - s_j][j - 1] + v_j, d[i][j - 1]\}$$

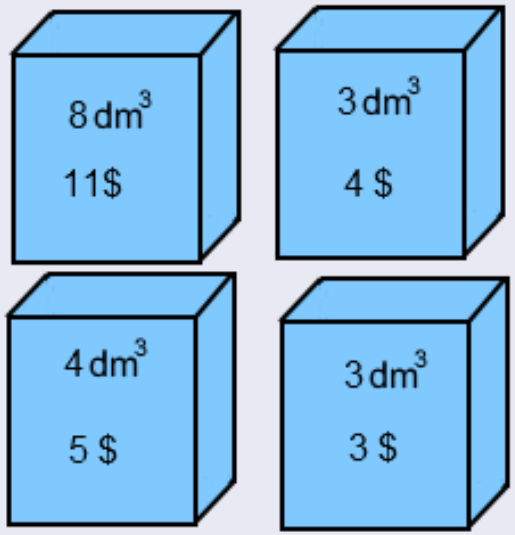
     Note that $d[i - s_j][j - 1] + v_j$ corresponds to picking item (s_j, v_j) while $d[i][j - 1]$ corresponds to not picking it
- $O(WN)$ subproblems, recurrence takes $O(1)$ to compute, therefore the time complexity is $O(WN)$

Knapsack Problem Example

Example

i / j	0	1	2	3	4
10	0	11	11	11	12
9	0	11	11	11	11
8	0	11	11	11	11
7	0	0	4	9	9
6	0	0	4	5	7
5	0	0	4	5	5
4	0	0	4	5	5
3	0	0	4	4	4
2	0	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

d[10][4] =
max(d[10][3], d[7][3] + 3)



Knapsack Problem Question

Exercises

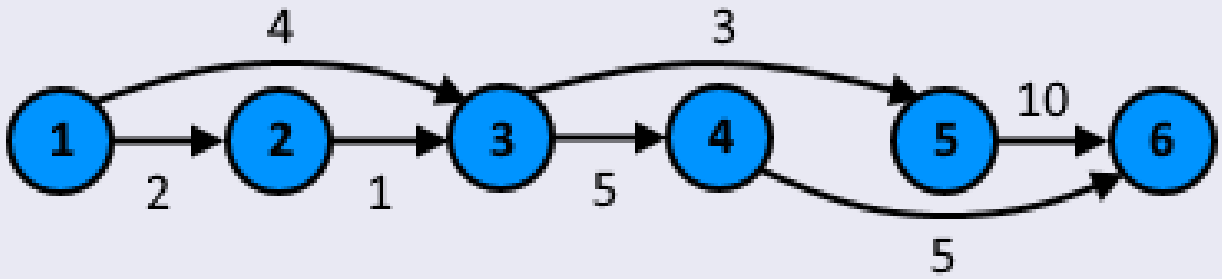
Suppose we changed the limits:

1. Raise the limit on capacity to $W \leq 10^{18}$
2. Lower the limit on values to $v_i \leq 1000$

How would you solve this variation of the problem?

Minimum Path Problem

- You have $N \leq 10^5$ cities, numbered $1, \ldots, N$
- You have $M \leq 10^5$ one-way roads, each road i starts from some city s_i ends in some other city e_i and has length l_i
- For each road the starting city must have lower index than the endpoint, in other words $s_i < e_i$
- Find the minimum distance from city 1 to city N



Minimum Path Solution

- The DP construction would be:
  1. Subproblems:

     $$d_i = \text{minimum path length from city 1 to city } i$$

     For $i \in [1, \ldots, N]$
  2. Recurrence Relation:

     $$d_i = \min_{j \in \text{in}(i)} \{d_{s_j} + l_j\}$$

     where $\text{in}(i)$ is the set of roads entering city i
- The time complexity is $O(\sum_{i=1}^{N}(|\text{in}(i)| + 1)) = O(N + M)$

A Question

- Suppose we lifted the restriction that $s_i < e_i$ (meaning we are now dealing with general graphs)
- Could we still use Dynamic Programming to find the shortest path between two nodes?