Dynamic Programming

Part I

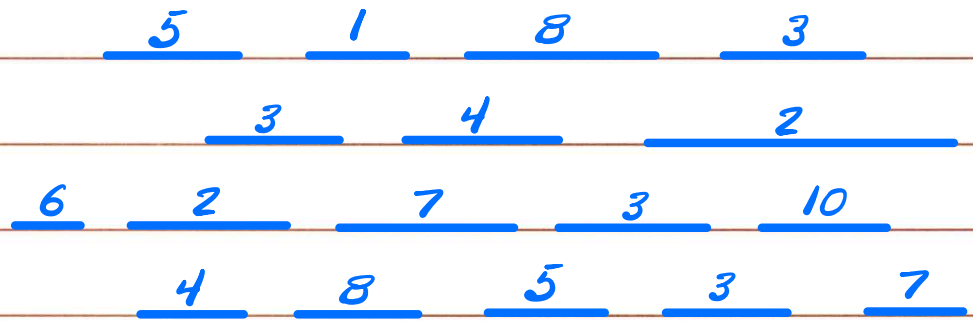
General Approach to Solving Optimization Problems Using Dynamic Programming

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

Weighted Interval Scheduling Problem

Input: A set of requests $\{1..n\}$, where the i^{th} request starts at $s(i)$, ends at $f(i)$, and has weight $w(i)$

Output: A subset of requests $S \subseteq \{1...n\}$ of mutually compatible intervals so as to maximize $\sum_{i \in S} w(i)$



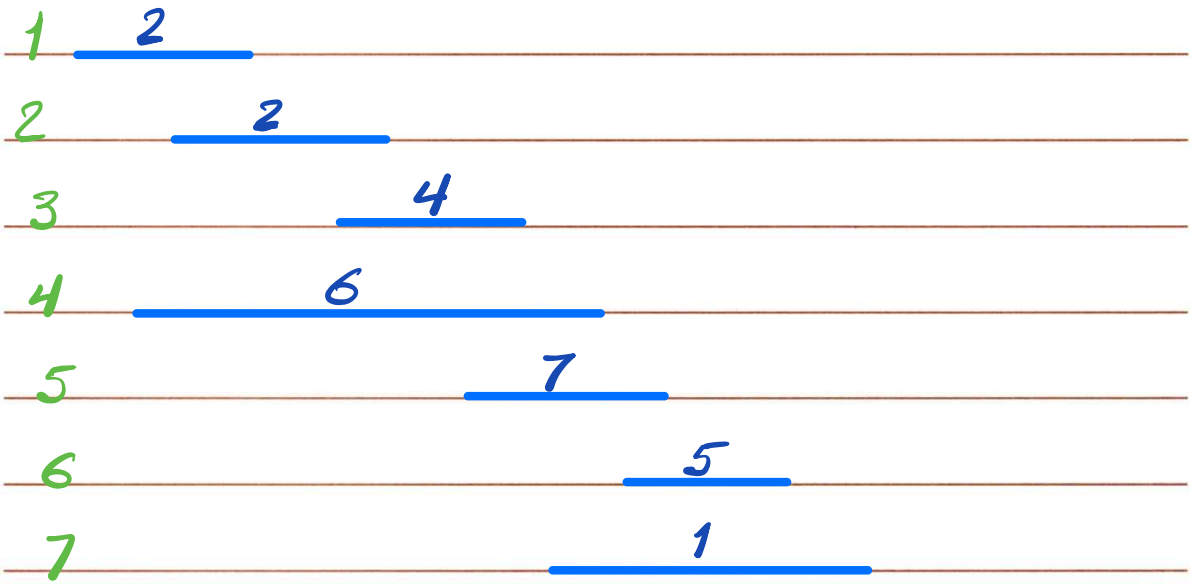
Case 1 - If it is,
value of the optimal solution =

Case 2 - If it isn't,
value of the optimal solution =

To find compatible subproblems quickly, sort requests in order of non-decreasing finish time

$$f_1 \leq f_2 \leq \cdots \leq f_n$$

And then define $P(j)$ for an interval j to be the largest index $i < j$ such that intervals i and j are disjoint.



Def. Let O_j denote the optimal solution to the problem consisting of requests $\{1 \dots j\}$, and let $OPT(j)$ denote the value of O_j

We can now use this notation to describe the two cases:

Solution

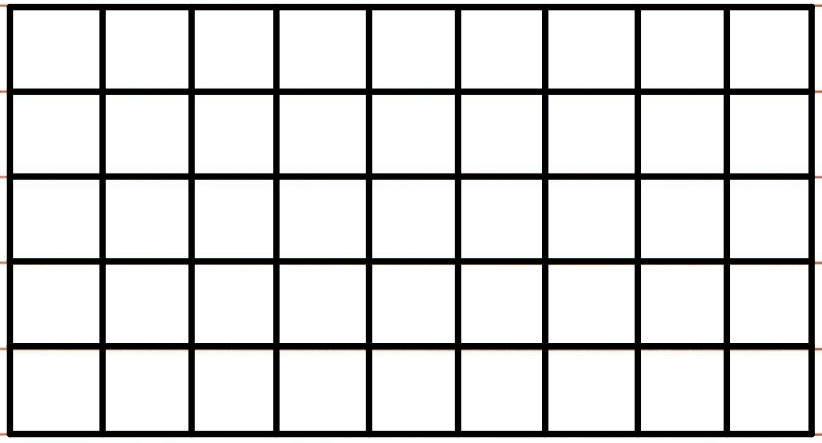
```
Compute_Opt(j)
    if j=0 then
        Return 0
    else


    endif
```

Complexity Analysis

Top-down pass: Compute an opt. solution

Iterative Solution

Usually the values of the opt. solutions to unique subproblems are stored in 1, 2, 3, ... dimensional arrays. The order in which these values are computed depends on the recurrence formula.



Video Game Problem

Coin Problem

Austrian coin denominations (Pre-Euro)

1

5

10

20

25

Question: How can we make change for n schillings using the minimum no. of coins?

Bottom up pass

0/1 Knapsack

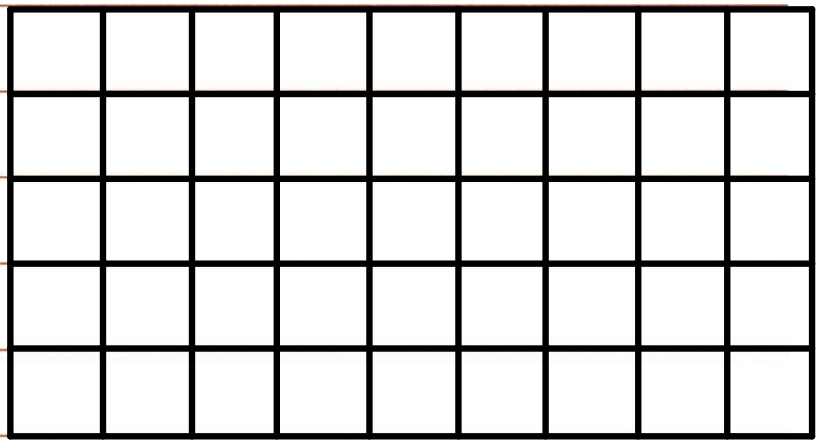
&

Subset Sum

Subset Sum Problem

- Given a single resource available for W units of time, and
- A set of requests $\{1 \ldots n\}$ that can be scheduled at any time between 0 to W.
- And where request i takes w_i time to process,
- The objective is to schedule jobs such that we maximize the resource's utilization

Bottom-up pass



Complexity Analysis

Polynomial Time

An algorithm runs in polynomial time if its running time is a polynomial in the length of the input

2. Graduate students get a lot of free food at various events. Suppose you have a schedule of the next n days marked with those days when you get a free dinner, and those days on which you must acquire dinner on your own. On any given day you can buy dinner at the cafeteria for $3. Alternatively, you can purchase one week's groceries for $10, which will provide dinner for each day that week (that day and the six that follow). However, because you don't have a fridge, the groceries will go bad after seven days (including the day of purchase) and any leftovers must be discarded. Due to your very busy schedule, these are your only two options for dinner each night. Your goal is to eat dinner every night while minimizing the money you spend on food.

3. You are in Downtown of a city and all the streets are one-way streets. You can only go east (right) on the east-west (left-right) streets, and you can only go south (down) on the north-south (up-down) streets. This is called a Manhattan walk.

a) In Figure A below, how many unique ways are there to go from the intersection marked S (coordinate (0,0)) to the intersection marked E (coordinate (n,m))?
Formulate the solution to this problem as a dynamic programming problem. Please make sure that you include all the boundary conditions and clearly define your notations you use.

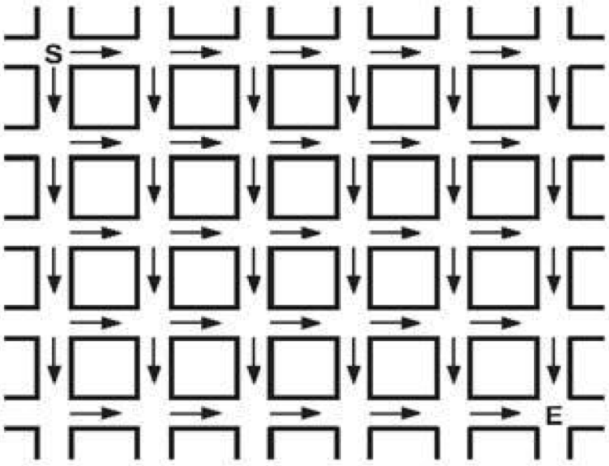


Figure A.

b) Repeat this process with Figure B; be wary of dead ends.

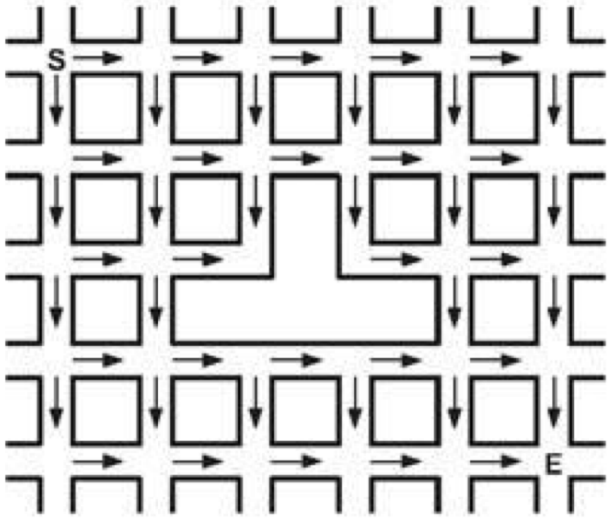


Figure B.

1. You are to compute the total number of ways to make a change for a given amount m. Assume that we have an unlimited supply of coins and all denominations are sorted in ascending order:$1 = d_1 < d_2 < \ldots < d_n$. Formulate the solution to this problem as a dynamic programming problem.