Discussion Week 7

Dynamic Programming

Review: Subset Sum

Problem Statement

- A single resource
- Requests {1,...,n} each take time w_i to process
- Can schedule jobs at any time between 0 to W

Objective: To schedule jobs such that we maximize the machine’s utilization

Review: Subset Sum

Recurrence Relation:

If $w < w_i$ then $OPT(i,w) = OPT(i-1, w)$

Else $OPT(i,w) = \max(OPT(i-1, w), w_i + OPT(i-1, w-w_i))$

Review: Subset Sum

```
subset_sum(n,W)
    Array M[0,w] = 0 for each w= 0 to W
    For i=1 to n
        For w=0 to W
            If w < w_i then M[i,w] = M[i-1,w]; S
            Else M[i,w] = max(M[i-1,w], w_i + M[i-1, w-w_i])
    Return M[n,w]
```

Subset Sum: Top Down Pass

Reconstruct the solution with top down pass

Let S[i,w] represent whether to include request i

```
subset_sum(n,W)
    Array M[0,w] = 0 for each w= 0 to W
    For i=1 to n
        For w=0 to W
            If w < w_i then M[i,w] = M[i-1,w]; **S[i,w] = 0**
            Else
                If M[i-1,w] >= w_i + M[i-1, w-w_i]: M[i,w] = M[i-1,w]; **S[i,w] = 0**
                Else: M[i,w] = w_i + M[i-1, w-w_i]; **S[i,w] = 1**
    Return M[n,w]
```

Subset Sum: Top Down Pass

Reconstruct solution:

```
i=n; w = W;
While (i > 0)
    If (S[i,w] == 1)
        { output “Include request i”; w = w-w_i; i = i-1 }
    Else i = i-1
```

Knapsack Problem

Given n items

Item i has weight $w_i > 0$ and value $v_i >= 0$

You have a weight limit of W on the total weight, and want to select a set S of weight at most W maximizing the total value from your set.

Knapsack Problem

How to modify subset_sum algorithm?

```
subset_sum(n,W)
    Array M[0,w] = 0 for each w= 0 to W
    For i=1 to n
        For w=0 to W
            If w < w_i then M[i,w] = M[i-1,w]; S
            Else M[i,w] = max(M[i-1,w], w_i + M[i-1, w-w_i])
    Return M[n,w]
```

Knapsack Problem

```
knapsack(n,W)
    Array M[0,w] = 0 for each w= 0 to W
    For i=1 to n
        For w=0 to W
            If w < w_i then M[i,w] = M[i-1,w]; S
            Else M[i,w] = max(M[i-1,w], v_i + M[i-1, w-w_i])
    Return M[n,w]
```

Coin Problem

Austrian Schillings: Have coins worth 1, 5, 10, 20, 25

Find the minimum number of coins to pay i shillings

Coin Problem

Recurrence Relation:

OPT(0) = 0

$$\mathrm{OPT}(i) = \min \begin{pmatrix} \mathrm{OPT}(i-1) + 1, \\ \mathrm{OPT}(i-5) + 1, \\ \mathrm{OPT}(i-10) + 1, \\ \mathrm{OPT}(i-20) + 1, \\ \mathrm{OPT}(i-25) + 1 \end{pmatrix}$$

Problem 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:$1 = d1 < d2 < … < dn$. Formulate the solution to this problem as a dynamic programming problem.

Problem 1 Solution

We will define COUNT(n, m) as the total number of ways to make change for amount m using coin denominations 1 to n.

The recurrence formula will be: COUNT(n,m) = COUNT(n-1, m) + COUNT(n, m-d_n)

Note: COUNT(n-1, m) is the number of ways to make change for amount m without using coin denomination n. And COUNT(n, m- d_n) represents the number of ways to make change for amount m using at least one coin of denomination n

Problem 1 Solution

Initialization:

COUNT $(i,0) = 1$ for $0 < i \leq n$ since there is a way to pay the amount 0 (by paying 0 of all coin types)

COUNT$(0,j) = 0$ for $0 < j \leq m$ since there is no way to pay a non-zero amount without any coins

Problem 1 Solution

Bottom up pass:

```
For i = 1 to n
    For j = 1 to m
        COUNT(n,m) = COUNT(n-1, m)
        If ( m - dn ≥ 0 ) COUNT(n,m) = COUNT(n,m) + COUNT(n, m- dn)
    Endfor
Endfor
```

The total count will be at COUNT(n,m)

This will take O(mn) which is pseudopolynomial since m is the numerical value of an input term.

Problem 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.

Problem 2 Solution

Solution:

We will define OPT(i) as the minimum cost of dinner for days 1 to i.

The recurrence formula will be:

$$
OPT(i) = \begin{cases} OPT(i-1) \text{ if there is free food on day } i \\ \text{Otherwise, } \min(\, OPT(i-1) + 3 \,,\, OPT(i-7) + 10 \,) \end{cases}
$$

Problem 2 Solution

Initialization:

We need to initialize OPT(0..6).

These are trivial problems to solve.

Bottom up pass:

```
For i = 7 to n
    If Free(i) then OPT(i) = OPT(i-1)
    Else OPT(i) = Min( OPT(i-1) + 3 , OPT(i-7) + 10 )
Endfor
```

Problem 2 Solution

The minimum cost of dinner will be at OPT(n).

This will take O(n) which is polynomial if we assume that the input consists of an array of size n called Free() where Free(i) is true when there is free food on day i, and false otherwise.

Problem 2 Solution

To be able to determine the dinner schedule, we will go top down:

```
i = n
While i > 0
    If Free(i) then
        Mark up the calendar with “free food” on day i
        i = i - 1
    Else if OPT(i-1) + 3 < OPT(i-7) + 10 then
        Mark up the calendar with “cafeteria” on day i
        i = i - 1
    Else
        Mark up the calendar with “go grocery shopping” on day i-6
        Mark up the calendar with “eat at home” on days i-5 to i
        I = i-7
```

The top down pass will also take O(n) time. So the whole solution runs in O(n) time.

Problem 3a

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.

In Figure A, 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.

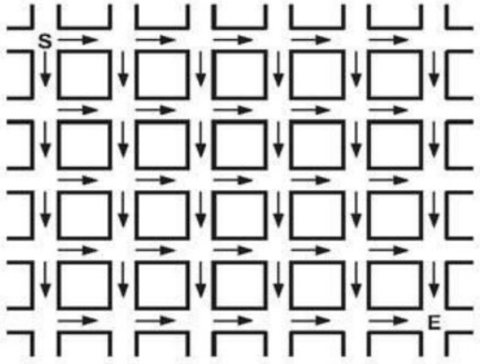

Figure A.

Problem 3a Solution

Let’s say E is at coordinates (0,0) and s is at coordinates (n,m). We will define COUNT(n, m) as the total number of ways to go from coordinates (n,m) to (0,0).

The recurrence formula will be: COUNT(i,j) = COUNT(i, j-1) + COUNT(i-1, j)

Initialization:

COUNT (i,0) = 1 for $0 < i \leq n$ since there is only one way to go (horizontally) from (i,0) to (0,0)

COUNT (0,j) = 1 for $0 < j \leq n$ since there is only one way to go (vertically) from (0,j) to (0,0)

Problem 3a Solution

Bottom up pass:

```
For i = 1 to n
    For j = 1 to m
        COUNT(i,j) = COUNT(i, j-1) + COUNT(i-1, j)
    Endfor
Endfor
```

The total count will be at COUNT(n,m)

Problem 3a Solution

This will take O(mn) which is pseudo-polynomial if we assume that the input only consists of the coordinates n and m. If the input consisted of the 2D array of all intersections, then the run time will be polynomial.

Problem 3b

In Figure B, how many unique ways are there to go from the intersection marked S (coordinate (0,0)) to the intersection marked E (coordinate (n,m))?

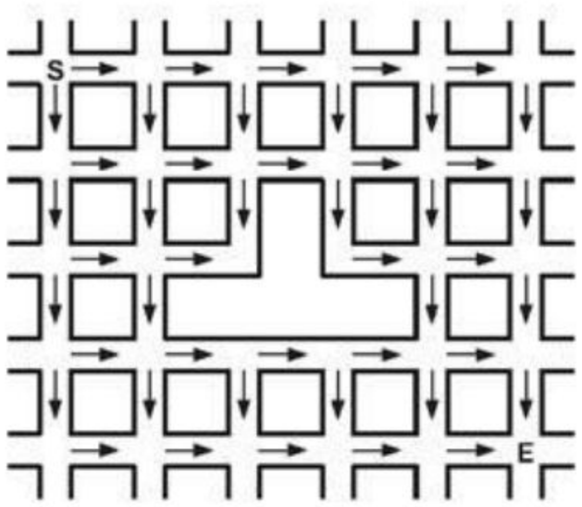

Figure B.

Problem 3b Solution

We can use the same approach and recurrence formula as in part a, except that we need to apply special recurrence formulae to the intersections that are affected namely (2, 2) and (3, 2). So, the implementation will be modified like this:

Bottom up pass:

```
For i = 1 to n
    For j = 1 to m
        If (n=2 and m=2) then COUNT(i,j) = COUNT(i-1, j)
        Else if (n=3 and m=2) then COUNT(i,j) = 0
        Else COUNT(i,j) = COUNT(i, j-1) + COUNT(i-1, j)
Return COUNT(n,m)
```