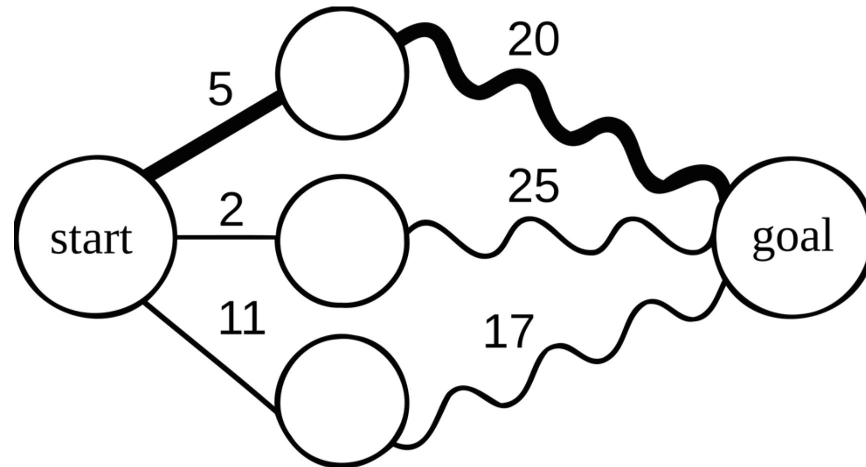
Lecture 18

Dynamic Programming II



Announcements

- Homework 5 Reflection due this Sunday night (4/12)
- Homework 6 due this Sunday night (4/12)
- Group Meetings continue

Dynamic Programming Recipe

- Step 1: Devise simple recursive algorithm for the value of the optimal solution
- Step 2: Write recurrence relation for optimal value
- Step 3: Design bottom-up iterative algorithm
- Step 4: Recover optimal solution

Step 4: Recover the Solution

Idea:

- Save best choice for each subproblem
- Track back from end

Step 4: Recover the Solution

- Save best choice for each subproblem

$\mathrm{WeIS}(n)$:

Initialize list M of size n to hold optimal values

Initialize list B of size n to hold choices

$M[0] \leftarrow 0$

for $j = 1$ to n **do**

$M[j] = \max(v_j + M[p_j], M[j-1])$

if $v_j + M[p_j] > M[j-1]$ **do**

$B[j] = 1$

else **do**

$B[j] = 0$

return M

Step 4: Recover the Solution

- Track back from end

Given lists M, B:

$A \leftarrow \{\}$

$j \leftarrow n$

while $j > 0$ **do**

  if $B[j] == 1$ **do**

    $A = A \leftarrow \{j\}$

    $j \leftarrow p_j$

  else do

    $j \leftarrow j - 1$

Review

Recursive algorithm → find recurrence → write iterative algorithm

Three ways of expressing value of optimal solutions of subproblems

- $\text{ComputeValue}(\text{j})$: recursive algorithm
- $\text{OPT}(j)$: used in recurrence; matches recursive algorithm
- $M[j]$: array to hold optimal values for each distinct subproblem; filled in during the iterative algorithm

New Problem: Rod Cutting

Input:

- Steel rod of length n, which can be cut into integer lengths and sold for price p_i for a piece of length i

Goal:

- Subdivide to maximize total value

New Problem: Rod Cutting

Input:

- Steel rod of length n, which can be cut into integer lengths and sold for price p_i for a piece of length i

Goal:

- Subdivide to maximize total value

Idea:

- Choose length i of first piece, then consider the problem on the remaining rod

Step 1: Recursive Algorithm

```
CutRod(j):
    if j = 0 then return 0
    v = 0
    for i = 1 to j do
        v = max(v, p[i] + CutRod(j − i))
    return v
```

Step 1: Recursive Algorithm

```
CutRod(j):
    if j = 0 then return 0
    v = 0
    for i = 1 to j do
        v = max(v, p[i] + CutRod(j − i))
    return v
```

Running time for $\mathrm{CutRod}(n)$: $\Theta(2^n)$

Step 2: Recurrence

$$\mathrm{OPT}(j) = \max_{1 \le i \le j} \{p_i + \mathrm{OPT}(j - i)\} \text{ for j} \ge 1$$

$$\mathrm{OPT}(0) = 0$$

CutRod(j):

if $j = 0$ then **return** 0

$v = 0$

for $i = 1$ to j **do**

$v = \max(v, p[i] + \mathrm{CutRod}(j - i))$

return v

Step 2: Recurrence

$$\text{OPT}(j) = \max_{1 \le i \le j}\{p_i + \text{OPT}(j-i)\} \text{ for j} \ge 1$$

$$\text{OPT}(0) = 0$$

- The recurrence provides all info needed to design an iterative algorithm
- Fill M so RHS side values are computed before LHS
- Range of j values determines the size of M

$\text{CutRod}(\text{j})$:

> **if** $j = 0$ then **return** 0
> $v = 0$
> **for** $i = 1$ to j **do**
> $v = \max(v, p[i] + \text{CutRod}(j-i))$
> **return** v

Step 3: Iterative Algorithm

```
CutRodItr(n):
    Initialize list M[0, ..., n]
    Set M[0] = 0
    for j = 1 to n do
        v = 0
        for i = 1 to j do
            v = max(v, p[i] + M[j − i])
        Set M[j] = v
    return M
```

Step 3: Iterative Algorithm

```
CutRodItr(n):
    Initialize list M[0, ..., n]
    Set M[0] = 0
    for j = 1 to n do
        v = 0
        for i = 1 to j do
            v = max(v, p[i] + M[j − i])
        Set M[j] = v
    return M
```

Running time for $\mathrm{CutRodItr}(n)$: $\Theta(n^2)$

New Problem: Subset Sum

Input:

- Items $1, 2, \ldots, n$
- Weights w_i for all items (integers)
- Capacity W

Goal:

- Select subset S whose total weight is as large as possible without exceeding W

Step 1: Recursive Algorithm

Note: if i belongs to the optimal solution, then the remaining capacity decreases by w_i

Step 1: Recursive Algorithm

Note: if i belongs to the optimal solution, then the remaining capacity decreases by w_i

$\text{SubsetSum}(j, w)$:

if $j = 0$ then **return** 0

$v = \text{SubsetSum}(j - 1, w)$

if $w_j \leq w$ **do**

$v = \max\left(v, w_j + \text{SubsetSum}(j - 1, w - w_j)\right)$

return v

Step 2: Recurrence

Let $\mathrm{OPT}(j, w)$ be the maximum weight of any subset of items $1, 2, \ldots, j$ that does not exceed w

- $$\mathrm{OPT}(j, w) = \begin{cases} \mathrm{OPT}(j-1, w), \text{if } w_j > w \\ \max\left(\mathrm{OPT}(j-1, w), w_j + \mathrm{OPT}\left(j-1, w-w_j\right)\right), \text{if } w_j \leq w) \end{cases}$$

- $\mathrm{OPT}(0, w) = 0$ for all $w = 0,1, \ldots, W$

Step 2: Recurrence

Let $\mathrm{OPT}(j, w)$ be the maximum weight of any subset of items $1, 2, \ldots, j$ that does not exceed w

- $\mathrm{OPT}(j,w) = \begin{cases} \mathrm{OPT}(j-1,w), \text{if } w_j > w \\ \max\left(\mathrm{OPT}(j-1,w), w_j + \mathrm{OPT}(j-1, w-w_j)\right), \text{if } w_j \leq w) \end{cases}$

j isn't feasible

- $\mathrm{OPT}(0,w) = 0$ for all $w = 0,1,\ldots,W$

Step 2: Recurrence

Let $\mathrm{OPT}(j, w)$ be the maximum weight of any subset of items $1, 2, \ldots, j$ that does not exceed w

- $$\mathrm{OPT}(j, w) = \begin{cases} \mathrm{OPT}(j-1, w), \text{if } w_j > w \\ \max\left(\mathrm{OPT}(j-1, w), w_j + \mathrm{OPT}\big(j-1, w-w_j\big)\right), \text{if } w_j \leq w) \end{cases}$$

- $\mathrm{OPT}(0, w) = 0$ for all $w = 0,1, \ldots, W$

j isn't feasible
j is feasible but not optimal

Step 2: Recurrence

Let $\text{OPT}(j, w)$ be the maximum weight of any subset of items $1, 2, \dots, j$ that does not exceed w

- $$\text{OPT}(j, w) = \begin{cases} \text{OPT}(j-1, w), \text{if } w_j > w \\ \max\left(\text{OPT}(j-1, w), w_j + \text{OPT}\left(j-1, w-w_j\right)\right), \text{if } w_j \le w) \end{cases}$$

- $\text{OPT}(0, w) = 0$ for all $w = 0,1, \dots, W$

j isn't feasible
j is feasible but not optimal
j is in optimal solution

Step 3: Iterative Algorithm

SubsetSumItr(n, W):

Initialize list $M[0{:}n, 0{:}W]$

Set $M[0, w] = 0$ for $w = 0, \dots, W$

for $j = 1$ to n **do**

for $W = 1$ to W **do**

if $w_j > w$ **then** $M[j, w] = M[j-1, w]$

else $M[j, w] = \max\left(M[j-1, w], w_j + M\left[j-1, w - w_j\right]\right)$

return $M[n, W]$

Step 3: Iterative Algorithm

SubsetSumItr(n, W):

  Initialize list $M[0{:}n, 0{:}W]$

  Set $M[0, w] = 0$ for $w = 0, \dots, W$

  for $j = 1$ to n **do**

    for $W = 1$ to W **do**

    if $w_j > w$ **then** $M[j, w] = M[j-1, w]$

    else $M[j, w] = \max\left(M[j-1, w], w_j + M[j-1, w-w_j]\right)$

  return $M[n, W]$

Running time for SubsetSumItr(n, W): $\Theta(nW)$

Step 3: Iterative Algorithm

SubsetSumItr(n, W):

  Initialize list $M[0{:}\,n, 0{:}\,W]$

  Set $M[0, w] = 0$ for $w = 0, \dots, W$

  for $j = 1$ to n **do**

    for $W = 1$ to W **do**

    if $w_j > w$ **then** $M[j, w] = M[j-1, w]$

    else $M[j, w] = \max(M[j-1, w], w_j + M[j-1, w-w_j])$

  return $M[n, W]$

Pseudo-polynomial running time
(running time depends on a numeric value)

Running time for SubsetSumItr(n, W): $\Theta(nW)$

Knapsack Problem

Same as Subset Sum, but now item have value in addition to weight

Input:

- Items $1,2,\ldots,n$
- Weights w_i for all items (integers)
- Values v_i for all items (integers)
- Capacity W

Goal:

- Select subset S whose total value is as large as possible without total weight exceeding W

Next Time

- Network Flow algorithms