15.053 Thursday, May 2

- Dynamic Programming
  - Review
  - More examples

Handouts: Lecture Notes

Match game example

- Suppose that there are 50 matches on a table, and the person who picks up the last match wins. At each alternating turn, my opponent or I can pick up 1, 2 or 6 matches. Assuming that I go first, how can I be sure of winning the game?

Determining the strategy using DP

- n = number of matches left (n is the ***state/stage***)
- g(n) = 1 if you can force a win at n matches.
  g(n) = 0 otherwise ***g(n) = optimal value function.***

At each state/stage you can make one of three decisions: take 1, 2 or 6 matches.

- g(1) = g(2) = g(6) = 1 ***(boundary conditions)***
- g(3) = 0; g(4) = g(5) = 1.

The recursion:

- g(n) = 1 if g(n-1) = 0 or g(n-2) = 0 or g(n-6) = 0;
  g(n) = 0 otherwise.
- Equivalently, g(n) = 1 – min (g(n-1), g(n-2), g(n-6))

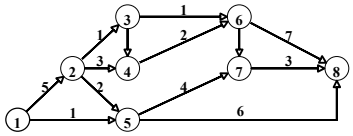
The same table

1	2	3	4	5	6	7	
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	
22	23	24	25	26	27	28	
29	30	31	32	33	34	35	
36	37	38	39	40	41	42	
43	44	45	46	47	48	49	50

Principle of Optimality

- Any optimal policy has the property that whatever the current state and decision, the remaining decisions must constitute an optimal policy with regard to the state resulting from the current decision.
- Whatever node j is selected, the remaining path from j to the end is the shortest path starting at j.

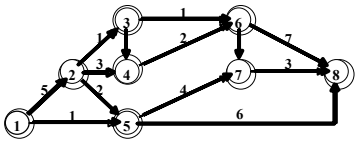
Finding shortest paths in graphs with no directed circuits.



If a network has no directed cycles, then the nodes can be labeled so that for each arc (i,j), i < j.

Such a node labeling is called a ***topological order***.

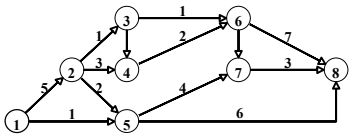
Finding a topological order



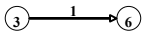
Find a node with no incoming arc. Label it node 1.

For i = 2 to n, find a node with no incoming arc from an unlabeled node. Label it node i.

More on topological orders



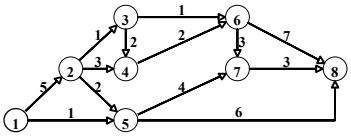
Topological orders are important for dynamic programming, and other calculations, such as the way that Excel calculates values in spreadsheets.



Interpretation: the calculation in cell 6 relies on the value in cell 3.

A topological order gives the order in which cell values can be calculated.

The shortest path problem on acyclic graphs



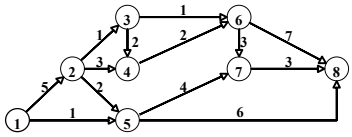
Let d(j) denote the shortest length of a path from node 1 to node j.

Let c_{ij} = length of arc (i,j)
What is d(1)?

Find d(j) using a recursion.

d(j) is the shortest length of a path from node 1 to node j.

Let c_{ij} = length of arc (i,j)



What is d(j) computed in terms of d(1), … d(j-1)?

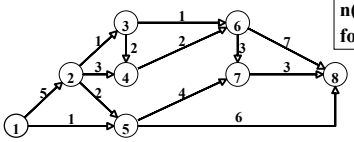
Compute f(2), …, f(8)

Example: d(4) = min { 3 + d(2), 2 + d(3) }

DPs for enumeration

Let n(j) be the number of different paths from node 1 to node 8

n(1) = 1
n(j) = $\sum_{(i,j)\in A}$ n(i) for j > 1.



n(1) = 1, n(2) = 1, n(3) = 1	
n(4) = n(2) + n(3) = 2;	n(5) = n(1) + n(2) = 2
n(6) = n(3) + n(4) = 3;	n(7) = n(5) + n(6) = 5
n(8) = n(5) + n(6) + n(7) = 10	

Finding optimal paragraph layouts

Tex optimally decomposes paragraphs by selecting the breakpoints for each line optimally. It has a subroutine that computes the ugliness F(i,j) of a line that begins at word i and ends at word j-1. How can we use F(i,j) as part of a dynamic program whose solution will solve the paragraph problem.

Tex optimally decomposes paragraphs by selecting the breakpoints for each line optimally. It has a subroutine that computes the ugliness F(i,j) of a line that begins at word i and ends at word j-1. How can we use F(i,j) as part of a dynamic program whose solution will solve the paragraph problem.

Solving the paragraph problem

- Recall: F(i,j) is the ugliness of a line beginning at word i and ending at work j-1.
- Let g(j) be the "ugliness" of the best paragraph layout including words 1, 2, …, j-1.
- Let n be the number of words in the paragraph.
- Initial condition: g(0) = 0.
- What is the ugliness of the best paragraph layout. Is it g(n)?
- What is the correct recursion?

Capital Budgeting, again

Investment budget = $14,000

Investment	1	2	3	4	5	6
Cash Required (1000s)	$5	$7	$4	$3	$4	$6
NPV added (1000s)	$16	$22	$12	$8	$11	$19

Solving the Stockco Problem

- At stage j compute for each budget from 0 to $14,000 compute the best NPV while limited to stocks 1 to j.
- Let f(j,k) be the best NPV using a budget of exactly k and limited to stocks 1, 2, …, j only.
- f(1,k) is easy to calculate
- The optimum solution to our original problem is f(6, 14)

Capital Budgeting: stage 1

Consider stock 1: cost $5, NPV: $16

Budget used up

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Best NPV so far

0	-	-	-	-	16	-	-	-	-	-	-	-	-	-

f(0,0) = 0, f(0,k) is undefined for k > 0

f(1,k) = max (f(0,k), f(0, k-5) + $16).

Capital Budgeting: stage 2

Consider stock 2: cost $7, NPV: $22

Budget used up

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Best NPV so far

0	-	-	-	-	16	-	-	-	-	-	-	-	-	-
0	-	-	-	-	16	-	22	-	-	-	-	38	-	-

f(2,k) = max (f(1,k), f(1, k-7) + $22).

Capital Budgeting: stage 3

Consider stock 3: cost $4, NPV: $12

Budget used up

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Best NPV so far

0	-	-	-	-	16	-	-	-	-	-	-	-	-	-
0	-	-	-	-	16	-	22	-	-	-	-	38	-	-
0	-	-	-	12	16	-	22	-	28	-	-	38	-	-

f(3,k) = max (f(2,k), f(2, k-4) + $12).

Capital Budgeting: stage 4

Consider stock 4: cost $3, NPV: $8

Budget used up

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Best NPV so far

0	-	-	-	-	16	-	-	-	-	-	-	-	-	-
0	-	-	-	-	16	-	22	-	-	-	-	38	-	-
0	-	-	-	12	16	-	22	-	28	-	-	38	-	-
0	-	-	-	12	16	-	22	24	28	30	-	38	-	-

Capital Budgeting: stage 5

Consider stock 5: cost $4, NPV: $11

Budget used up

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Best NPV so far

0	-	-	-	-	16	-	-	-	-	-	-	-	-	-
0	-	-	-	-	16	-	22	-	-	-	-	38	-	-
0	-	-	-	12	16	-	22	-	28	-	-	38	-	-
0	-	-	-	12	16	-	22	24	28	30	-	38	-	-
0	-	-	-	12	16	-	22	24	28	30	33	38	39	41

Capital Budgeting: stage 6

Consider stock 6: cost $6, NPV: $19

Budget used up

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Best NPV so far

0	-	-	-	-	16	-	-	-	-	-	-	-	-	-
0	-	-	-	-	16	-	22	-	-	-	-	38	-	-
0	-	-	-	12	16	-	22	-	28	-	-	38	-	-
0	-	-	-	12	16	-	22	24	28	30	-	38	-	-
0	-	-	-	12	16	-	22	24	28	30	33	38	39	41
0	-	-	-	12	16	-	22	24	28	31	35	38	41	43

Capital budgeting in general

$$\text{Max} \quad \sum_{j=1..n} c_j x_j$$

$$\text{s.t} \quad \sum_{j=1..n} a_j x_j \le b$$

$$x \text{ binary}$$

Let f(k, v) =

$$\text{Max} \quad \sum_{j=1..k} c_j x_j$$

$$\text{s.t} \quad \sum_{j=1..k} a_j x_j = v$$

$$x \text{ binary}$$

The recursion

- f(0,0) = 0; f(0,k) is undefined for k > 0
- f(k, v) = min (f(k-1, v), f(k-1, v-a_k) + c_k)
  either item k is included, or it is not

The optimum solution to the original problem is
max { f(n, v) : 0 ≤ v ≤ b }.

Note: we solve the capital budgeting problem for all right hand sides less than b.

The Knapsack Problem

Is there a feasible solution to

31 x + 35 y + 37 z = 422

x, y, z are non-negative integers.

let f(1, j) = 1 if there is a solution to
31 x = j,
x is a non-negative integer

let f(2, j) = 1 if there is a solution to
31 x + 35 y = j,
x and y are non-negative integer

The Knapsack Problem

Is there a feasible solution to

31 x + 35 y + 37 z = 422

x, y, z are non-negative integers.

let f(3 ,j) = 1 if there is a solution to
31 x + 35 y + 37 z = j,
x, y, and z are non-negative integer

Note: there are three stages to this problem.

The recursion for the knapsack problem

Is there a feasible solution to

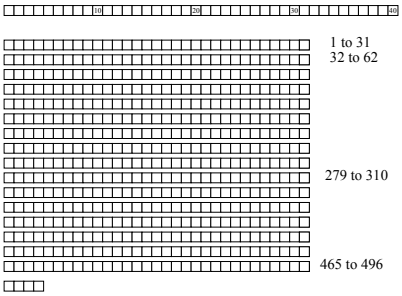
31 x + 35 y + 37 z = 422

x, y, z are non-negative integers.

f(0,0) = 1, f(0,k) = 0 for k > 0

f(1,k) = 1 if f(0,k) = 1
or f(0, k-31) = 1
or f(1, k-31) = 1

f(1,k) = max { f(0,k), f(0, k-31), f(1, k-31) }



The recursion for the knapsack problem

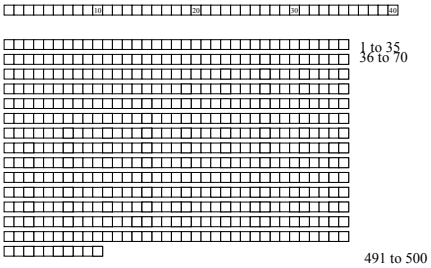
Is there a feasible solution to

31 x + 35 y + 37 z = 422

x, y, z are non-negative integers.

f(2,k) = 1 if f(1,k) = 1
or f(1, k-35) = 1
or f(2, k-35) = 1

f(2,k) = max { f(1,k), f(1, k-35), f(2, k-35) }



The recursion for the knapsack problem

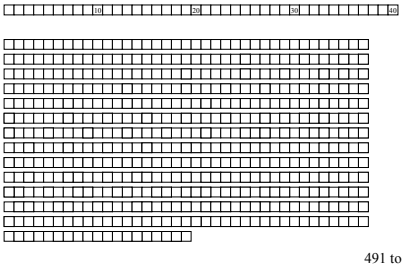
Is there a feasible solution to

31 x + 35 y + 37 z = 422

x, y, z are non-negative integers.

f(3,k) = 1 if f(2,k) = 1
or f(2, k-37) = 1
or f(3, k-37) = 1

f(2,k) = max { f(2,k), f(2, k-37), f(3, k-37) }



Dynamic Programming Review

- **Recursion**
- **Principle of optimality**
- **states and stages and decisions**
- **useful in a wide range of situations**
  - **shortest paths**
  - **capacity expansion**
  - **knapsack**
  - **many, many more**