15.053 Tuesday, April 30

- Dynamic Programming
  - Recursion
  - Principle of Optimality

Handouts: Lecture Notes

Dynamic Programming

- Transforms a complex optimization problem into a sequence of simpler ones.
- Usually begins at the end and works backwards
- Can handle a wide range of problems
- Relies on recursion, and on the principle of optimality
- Developed by Richard Bellman

Recursion example

- There are 11 people in a room. How many ways can one select exactly 6 of them?
- Let f(n,k) denote the number of subgroups of size k out of n people. We want f(11,6)

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)

The number of subgroups containing "1" is f(10,5).

The number of subgroups not containing "1" is f(10,6).

$$f(n,k) = f(n-1,k-1) + f(n-1, k)$$
$$f(n,n)=f(n,0)=1$$

1								
1	1							
1	2	1						
1	3	3	1					
1	4	6	4	1				
1	5	10	10	5	1			
1	6	15	20	15	6	1		
1	7	21	35	35	21	7	1	
1	8	28	56	70	56	28	8	1

Dynamic Programming Example

- Suppose that there are 30 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 3 matches. Assuming that I go first, how can I be sure of winning the game?
- (Discuss with partner).

DP Example.

- I win if there are 1, 2, or 3 matches
- Backing up one step, I lose if there are 4 matches.
- Backing up another step, I win if there are 5, 6, or 7 matches.
- Backing up another step, I lose if there are 8 matches.
- Conclusion. I lose if there are 4K matches. I win otherwise.

Determining the strategy using DP

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

At each state/stage you can make one of three decisions: take 1, 2 or 3 matches.

- **f(1) = f(2) = f(3) = 1 *(boundary conditions)***

The recursion:

- **f(n) = 1 if f(n-1) = 0 or f(n-2) = 0 or f(n-3) = 0; f(n) = 0 otherwise.**
- **Equivalently, f(n) = 1 – min (f(n-1), f(n-2), f(n-3)).**

Running the Recursion

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

Similar, but more difficult 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. (why?)**

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)).**

Running the Recursion

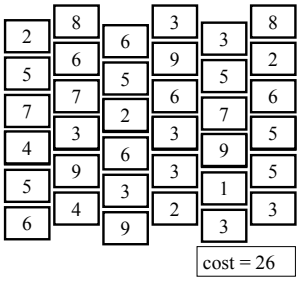
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Who sees the pattern determining the losing hands?

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

Example from BH&M page 453



A path consists of a sequence of squares from left to right such that each square is adjacent to the previous one.

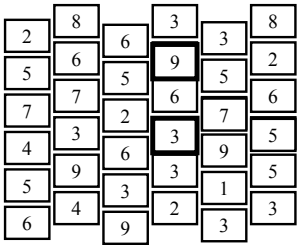
The cost of the path is the sum of the values on the path.

What is the minimum cost path?

Solving the shortest path problem using recursion

- For each square j, let f(j) be the shortest path starting at j and ending at the right hand side.
- If j is a rightmost node, then f(j) is the cost of j.
- j is the ***state/stage***
- f(j) is the ***optimal value function***.

Example from BH&M page 453



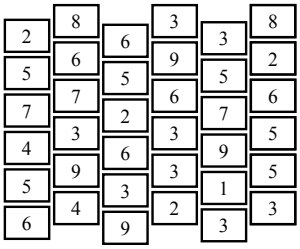
Consider the node with cost 3. Call it node j. (It is the state/stage)

The shortest path starting at that node has a cost of 15.

So f(j) = 15.

What is the cost of the node labeled 9?

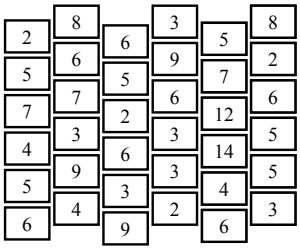
The easy computations



f(j) is easy to compute for the right most nodes.

For other nodes, there are two possible ***decisions***: go up and right, and go down and right.

Taking it to the second stage

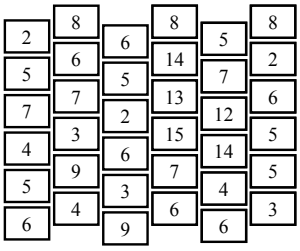


Compute f(j) for the nodes in the second column from the right.

Let c(j) = cost of rectangle j.

Let $R_1(j)$ be the node to the upper right of j.
Let $R_2(j)$ be the node to the lower right of j
Then f(j) = c(j) + min [f(R_1(j)), f(R_2(j))]. Why?

Taking it to the third stage



Compute f(j) for the nodes in the third column from the right.

To do with partner: figure out f(j) for the remaining rectangles.

Dynamic Programming in General

- Break up a complex decision problem into a sequence of smaller decision subproblems.
- Stages: one solves decision problems one "stage" at a time. Stages often can be thought of as "time" in most instances.
  - Not every DP has stages
  - The previous shortest path problem has 6 stages
  - The match problem does not have stages.

Dynamic Programming in General

- States: The smaller decision subproblems are often expressed in a very compact manner. The description of the smaller subproblems is often referred to as the state.
  - match problem: "state" is the number of matches left
- At each state-stage, there are one or more decisions. The DP recursion determines the best decision.
  - match problem: how many matches to remove
  - shortest path example: go right and up or else go down and right

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. (As formulated by Richard Bellman.)
- Whatever node j is selected, the remaining path from j to the end is the shortest path starting at j.

Optimal Capacity Expansion: What is the least cost way of building plants?

Year	Cum. Demand	Cost per plant in $millions
2002	1	54
2003	2	56
2004	4	58
2005	6	57
2006	7	55
2007	8	52

Cost of $15 million in any year in which a plant is built. At most 3 plants a year can be built

Forwards recursion

- Start with stage 1 and determine values for larger stages using recursion
- In the match examples, determine f(n) from f(k) for k < n.

Backwards Recursion

- Boundary conditions: optimum values for states at the last stage
- Determine optimum values for other stages using backwards recursion

Determining state and stage

- Stage: year Y
- State: number of plants
- f(j, Y) = optimum cost of capacity expansion starting with j plants at the end of year Y.
- We want to compute f(0, 2002).
- What is f(j, 2007) for different j?

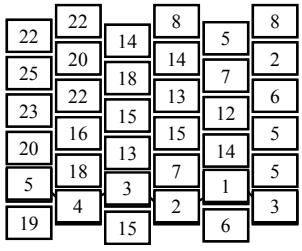
Determining state and stage

- Work with your partner to determine f(j, 2006) for each j. You may use the values for f(j, 2007).
- If you complete this, compute f(j, 2005) for each j.
- Then compute f(j, 2004) for each j.

Summary for Dynamic Programming

- Recursion
- Principle of optimality
- states and stages and decisions
- useful in a wide range of situations
  - games
  - shortest paths
  - capacity expansion
  - more will be presented in next lecture.

Taking it to the final stages



Compute f(j) for the nodes in the remaining columns.

One keeps track of the "next node" on the optimal path to the right.