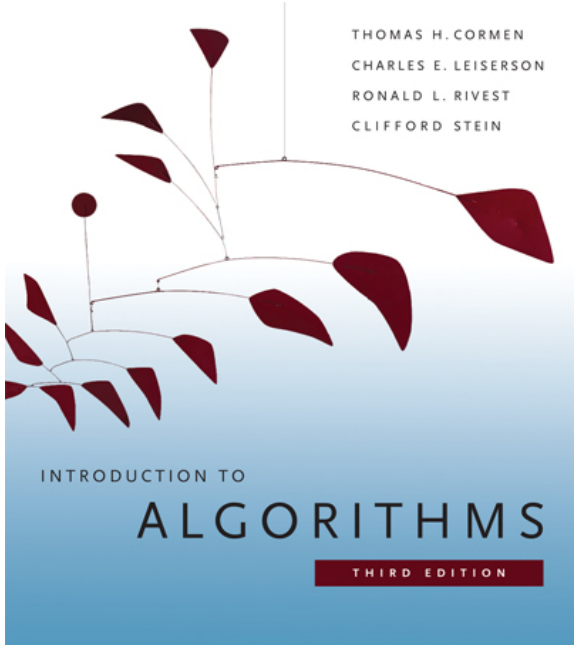
6.006- *Introduction to Algorithms*



Lecture 18

Prof. Constantinos Daskalakis

CLRS 15

Menu

- New technique: Dynamic Programming
  - Computing Fibonacci numbers – Warmup
  - “Definition” of DP
  - Crazy Eights Puzzle

Fibonacci Numbers

- Fibonacci sequence:
  - $F_0=0$, $F_1=1$
  - $F_n=F_{n-1}+F_{n-2}$
  - So $F_0=0$, $F_1=1$, $F_2=1$, $F_3=2$, $F_4=3$, $F_5=5$, $F_6=8$, $F_7=13$,…
  - Interesting fact: $F_n/F_{n-1} \rightarrow \varphi$ (the golden ratio)
  - This is why if something looks beautiful in nature, chances are that it involves two consecutive Fibonacci numbers…

Clockwise Spirals: 34 Counter-clockwise Spirals: 55

34 and 55 are consecutive numbers in Fibonacci sequence…

Fibonacci Numbers

- Fibonacci sequence:
  - $F_0=0$, $F_1=1$
  - $F_n=F_{n-1}+F_{n-2}$
  - So $F_0=0$, $F_1=1$, $F_2=1$, $F_3=2$, $F_4=3$, $F_5=5$, $F_6=8$, $F_7=13$,…
  - Interesting fact: $F_n/F_{n-1} \rightarrow \varphi$ (the golden ratio)
- How fast does F_n grow ?
  - $F_n=F_{n-1}+F_{n-2} \geq 2\,F_{n-2} \quad \Rightarrow F_n = 2^{\Omega(n)}$
- How quickly can we compute F_n?
  (time measured in arithmetic operations)

$F_n = F_{n-1} + F_{n-2}$

- Algorithm I: recursion

```
naive_fibo(n):
    if n=0: return 0
    else if n=1: return 1
    else:
      return naive_fibo(n-1) + naive_fibo(n-2)
```

- Time ? $T(n)=T(n-1)+T(n-2) = O(F_n)$
- Better algorithm ?

$F_n = F_{n-1} + F_{n-2}$

- Algorithm II: memoization

```
memo = { }
fibo(i):
    if i in memo: return memo[i]
    else if i=0: return 0
    else if i=1: return 1
    else:
        f = fibo(i-1) + fibo(i-2)
    memo[i]=f
    return f
return fibo(n)
```

- Time? O(n)

- in the whole recursive execution, I will only go beyond this point, n times

(since every time I do this, I fill in another slot in memo[])

- hence, all other calls to fibo( ) act as reading an entry of an array

Dynamic Programming Definition

- DP ≈ Recursion + Memoization
- DP works when:
  - the solution can be produced by combining solutions of subproblems; $F_n=F_{n-1}+F_{n-2}$
  - the solution of each subproblem can be produced by combining solutions of sub-subproblems, etc;

moreover…. $F_{n-1}=F_{n-2}+F_{n-3}$ $F_{n-2}=F_{n-3}+F_{n-4}$

  - the total number of subproblems arising recursively is polynomial.

$F_1, F_2,\ldots, F_n$

Dynamic Programming Definition

- DP ≈ Recursion + Memoization
- DP works when:

Optimal substructure

The solution to a problem can be obtained by solutions to subproblems.

$F_n = F_{n-1} + F_{n-2}$

moreover….

$F_{n-1} = F_{n-2} + F_{n-3}$ $F_{n-2} = F_{n-3} + F_{n-4}$

Overlapping Subproblems

A recursive solution contains a "small" number of distinct subproblems (repeated many times)

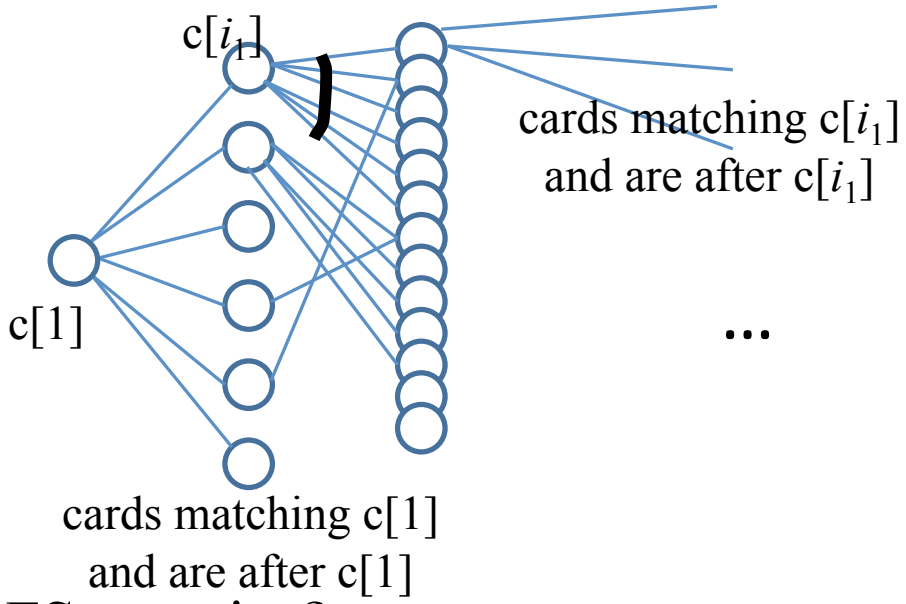
$F_1, F_2, \ldots, F_n$

Crazy 8s

- **Input:** a sequence of cards c[0]…c[n-1].
- E.g., 7♣ 7♥ K♣ K♠ 8♥
- **Goal:** find the longest "trick subsequence" c[i_1]…c[i_k], where $i_1 < i_2 < \ldots < i_k$.
- For it to be a trick subsequence, it must be that:
  ∀ j, c[i_j] and c[i_{j+1}] "match" i.e.
  - they either have the same rank,
  - or the same suit
  - or one of them is an 8
  - in this case, we write: c[i_j] ~ c[i_{j+1}]
- E.g., 7♣ K♣ K♠ 8♥ is the longest such subsequence in the above example

Crazy 8s via graph search

- Longest trick starting at c[1]?
- **Idea:** BFS was good for shortest paths in unweighted graphs. Let's try it for finding a longest path in the graph of matching cards.
- Do BFS starting at c[1], compute BFS tree, and look at deepest level.



- Worst case BFS tree size?
- e.g. 7♣ 10♥ 7♣ 2♣ 5♣ 7♣ 2♣ 5♣10♥ 7♣ 2♣ 5♣ 7♣ 2♣ 5♣…
- size $\geq 2^n$

DP Approach

- Identify subproblem:
- Let trick(i) be the length of the longest trick subsequence that starts at card c[i]
- **Question:** How can I relate value of trick(i) to the values of trick(i+1),…,trick(n)?
- Recursive formula:

$$\text{trick}(i) = 1 + \max_{j>i,\ \text{c}[j] \text{ matches c}[i]} \text{trick}(j)$$

- Maximum trick length:

$$\max_i \text{trick}(i)$$

Implementations

Recursive

- memo = { }
- trick(i):
  - if i in memo: return memo[i]
  - else if i=n-1: return 1
  - else
    - f := 1+max$_{j>i,\ c[j] \text{ matches } c[i]}$ trick(j)
    - memo[i] := f
    - return f
- call trick(0), trick(1),…,trick(n-1)
- return maximum value in memo

Implementations (cont.)

Iterative

memo = { }

for i=n-1 downto 0

  memo[i]= 1+max$_{j>i,\ c[j] \text{ matches } c[i]}$ memo[j]

return maximum value in memo

Runtime: $O(n^2)$

Dynamic Programming

- DP ≈ Recursion + Memoization
- DP works when:

Optimal substructure

An solution to a problem can be obtained by solutions to subproblems.

$\text{trick}(i) = 1 + \max_{j>i,\ c[j] \text{ matches } c[i]} \text{trick}(j)$

moreover….

Overlapping Subproblems

A recursive solution contains a "small" number of distinct subproblems (repeated many times)

$\text{trick}(0), \text{trick}(1), \ldots, \text{trick}(n-1)$

Menu

- New technique: Dynamic Programming
  - Computing Fibonacci numbers – Warmup
  - "Definition" of DP
  - Crazy Eights Puzzle
  - Next Time: **all-pairs shortest paths**

All-pairs shortest paths

- **Input:** Directed graph $G = (V, E)$, where $|V| = n$, with edge-weight function $w : E \to \mathsf{R}$.
- **Output**: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

Assumption: No negative-weight cycles

Dynamic Programming Approach

- Consider the $n \times n$ matrix $A = (a_{ij})$, where:
  - $a_{ij} = w(i,j)$, if $(i, j) \in E$, 0, if i=j, and $+\infty$, otherwise.
- and define:
  - $d_{ij}^{(m)} =$ weight of a shortest path from i to j that uses <u>at most</u> m edges
- Want: $d_{ij}^{(n-1)}$

Claim: We have

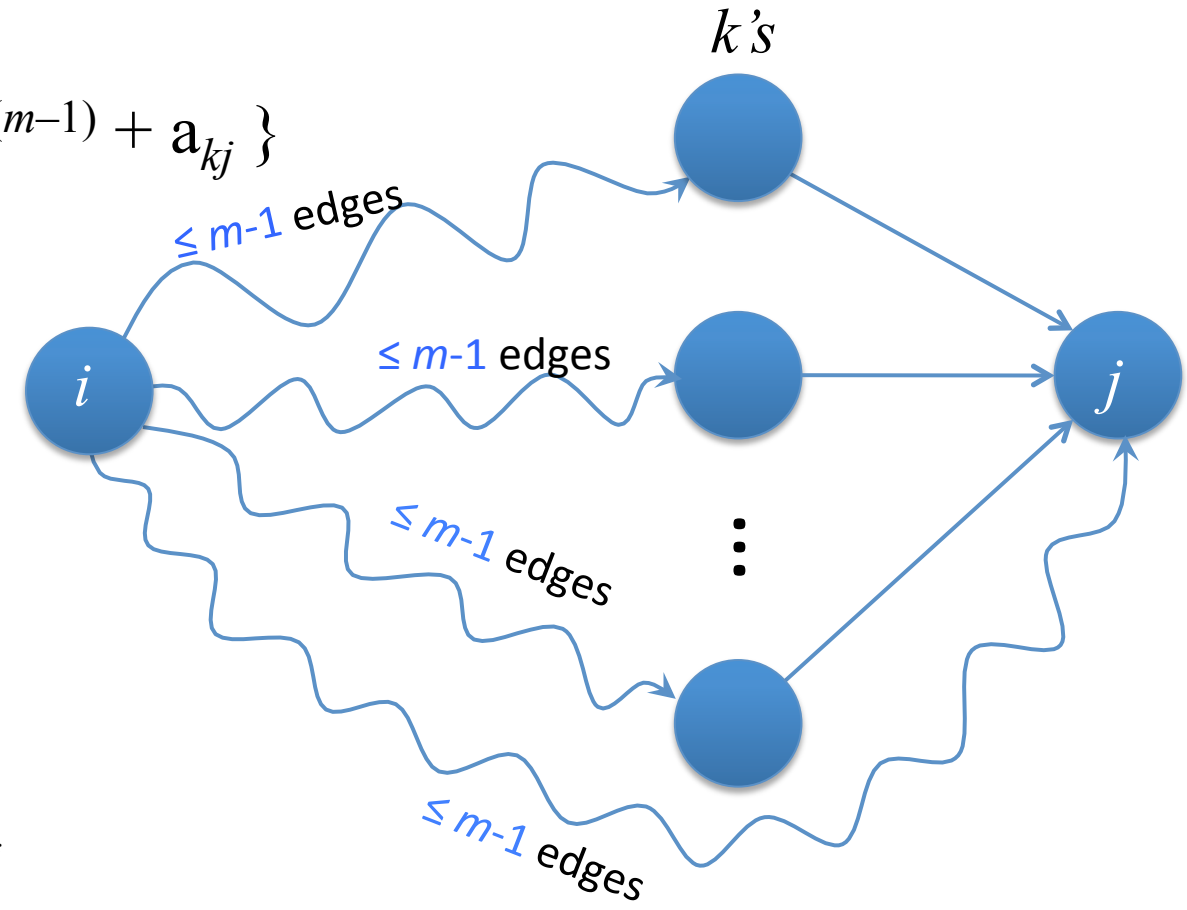
$d_{ij}^{(0)} = 0$, if $i = j$, and $+\infty$, if $i \neq j$;

and for $m = 1, 2, \ldots, n-1$,

$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$.

Proof of Claim

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$$




```
for k ← 1 to n
    if d_ij > d_ik + a_kj
        d_ij ← d_ik + a_kj
```

"Relaxation" (recall Bellman-Ford lecture)

Dynamic Programming Approach

- Consider the $n \times n$ matrix $A = (a_{ij})$, where:
  - $a_{ij} = w(i,j)$, if $(i, j) \in E$, 0, if i=j, and $+\infty$, otherwise.
- and define:
  - $d_{ij}^{(m)} =$ weight of a shortest path from i to j that uses <u>at most</u> m edges
- Want: $d_{ij}^{(n-1)}$

Claim: We have

$d_{ij}^{(0)} = 0$, if $i = j$, and $+\infty$, if $i \neq j$;

and for $m = 1, 2, \ldots, n-1$,

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}.$$

Time to compute $d_{ij}^{(n-1)}$? $O(n^4)$ - similar to n runs of Bellman-Ford

Something less extravagant? Next Lecture

Inventor of Fibonacci Sequence?

- Is it Fibonacci?
- where Fibonacci: Italian Mathematician (1170-1250)
- A: No. Fibonacci just introduced it to Europe.
- Sequence was known to Indian Mathematicians since the 6th century.
- So is it some Indian mathematician?
- That's more of a philosophical question.
- Same as question: Who invented the prime numbers some Greek, Egyptian or Babylonian?
- After all, these numbers play a role in natural systems that existed before humans…