Algorithm

Yangjun Ahn

yangjunahn@sungshin.ac.kr

https://sites.google.com/sungshin.ac.kr/mhail

Dynamic Programming

Guess

Recursion

Memoization

Dynamic Programming

- **General and powerful design technique of algorithm**
- **Do brute force, but be careful.**
- **Do finding the shortest paths in some DAG.**
- **Solve subproblems and do recycling.**
- **Time complexity?**

  time = (# sub problems) x [(time) / (subproblem)]

Today

- Fibonacci
- Shortest Path
- Guess
- DAG

Fibonacci numbers

- **Sequence**
  - $F_1 = F_2 = 1$
  - $F_n = F_{n-1} + F_{n-2}$
- **Goal: Compute F_n**

Fibonacci numbers

- **Naive recursive algorithm:**

```
fib(n):
        if n<= 2: f = 1
        else f = fib(n-1) + fib(n-2)
        return f
```

- **Time complexity**

$$T(n) = T(n-1) + T(n-2) + \Theta(1) \geq \Theta(2^{n/2})$$

Fibonacci numbers

- **Memoized DP Alg.:**

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

Fibonacci numbers

- **Look at the memorized DP's efficiency.**

  Draw a tree

  or

  fib(k) only recurses the first time it's called for all k

  - memoized calls cost Θ(1)
  - # nonmemoized calls is n
  - nonrecursive work per call: Θ(1)

  ▶ time = Θ(n)

Note. Memoized DP here is not the best alg. for computing Fib!

Fibonacci numbers

- **Bottom-up DP Alg.:**

```
fib = {}
for k in range(1, n+1):
        if k<= 2: f = 1
        else f = fib(k-1) + fib(k-2)
        fib[k] = f
return fib[n]
```

- **exactly same computation as the memorized version.**
- **topological sort of subproblems dependency DAG.**
- **can often save space.**

Shortest Path

- **$\delta(s, v)$ for all v**
- **$\delta(s, v) = \min\limits_{(u, v) \in E} (\underbrace{\delta(s, v)}_{\text{recursive call}} + w(u, v))$**
- **Is this good algorithm?**
- **Subproblem dependencies should be acyclic.**
- **Time?**
  - $\Theta(V^3)$
  - but actually? $\Theta(VE)$ ▶ Bellman-Ford