Lecture 11

Today:
- Finish Greedy-Set Cover.
- Dynamic Programming.

Set Cover

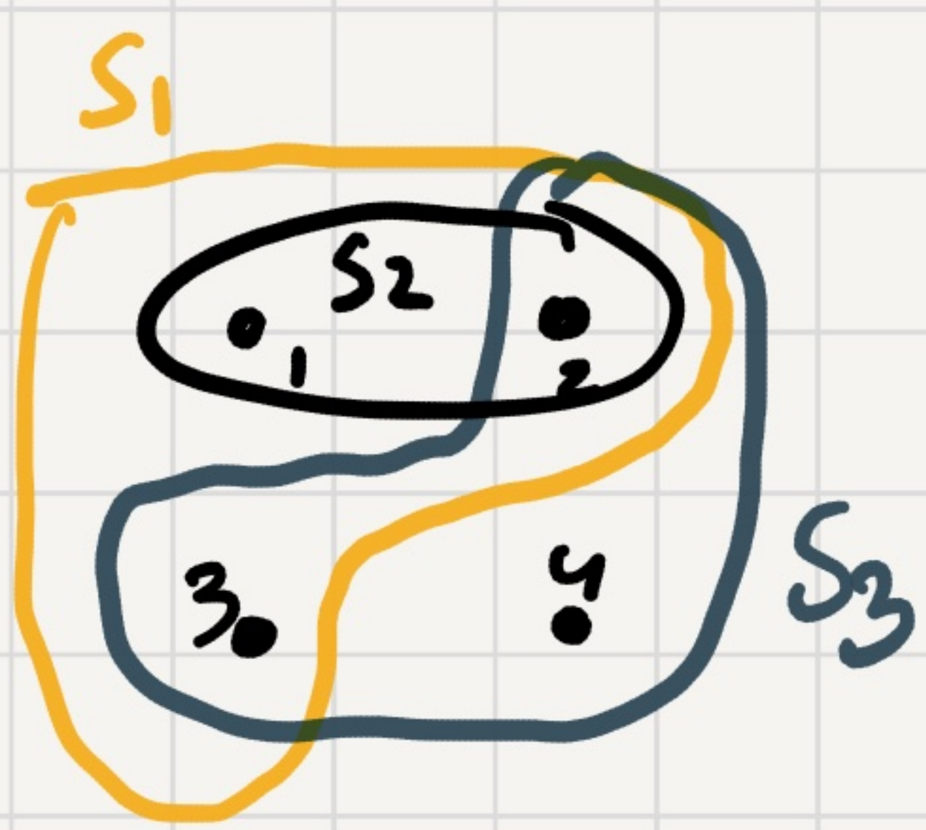
Input: Universe $\mathcal{U} = \{1, 2, 3, \ldots, n\}$

Collection of subsets $S_1, S_2, S_3, \ldots, S_m \subseteq \mathcal{U}$ (s.t. $S_1 \cup S_2 \cup \ldots \cup S_m = \mathcal{U}$.)

Output: Minimal subcollection that covers $\mathcal{U}$.

Minimal Size $J \subseteq [m]$ s.t. $\bigcup_{j \in J} S_j = \mathcal{U}$

For Example:



Optimal Solution:

$J = \{1,3\}$ or $J = \{2,3\}$

$\downarrow$ $\downarrow$

$S_1 \cup S_3 = \{1,2,3,4\}$ $S_2 \cup S_3 = \{1,2,3,4\}$.

Greedy strategy? Pick at any step the set that covers the most new points.

Algorithm:

$S_J \triangleq \bigcup_{j \in J} S_j$

1. $J \leftarrow \emptyset$.
2. While $S_J \neq U$:

   Pick $i \notin J$ with largest $|S_i \setminus S_J|$

   (covers the most new points)

   Add $i \in J$.
3. output J.

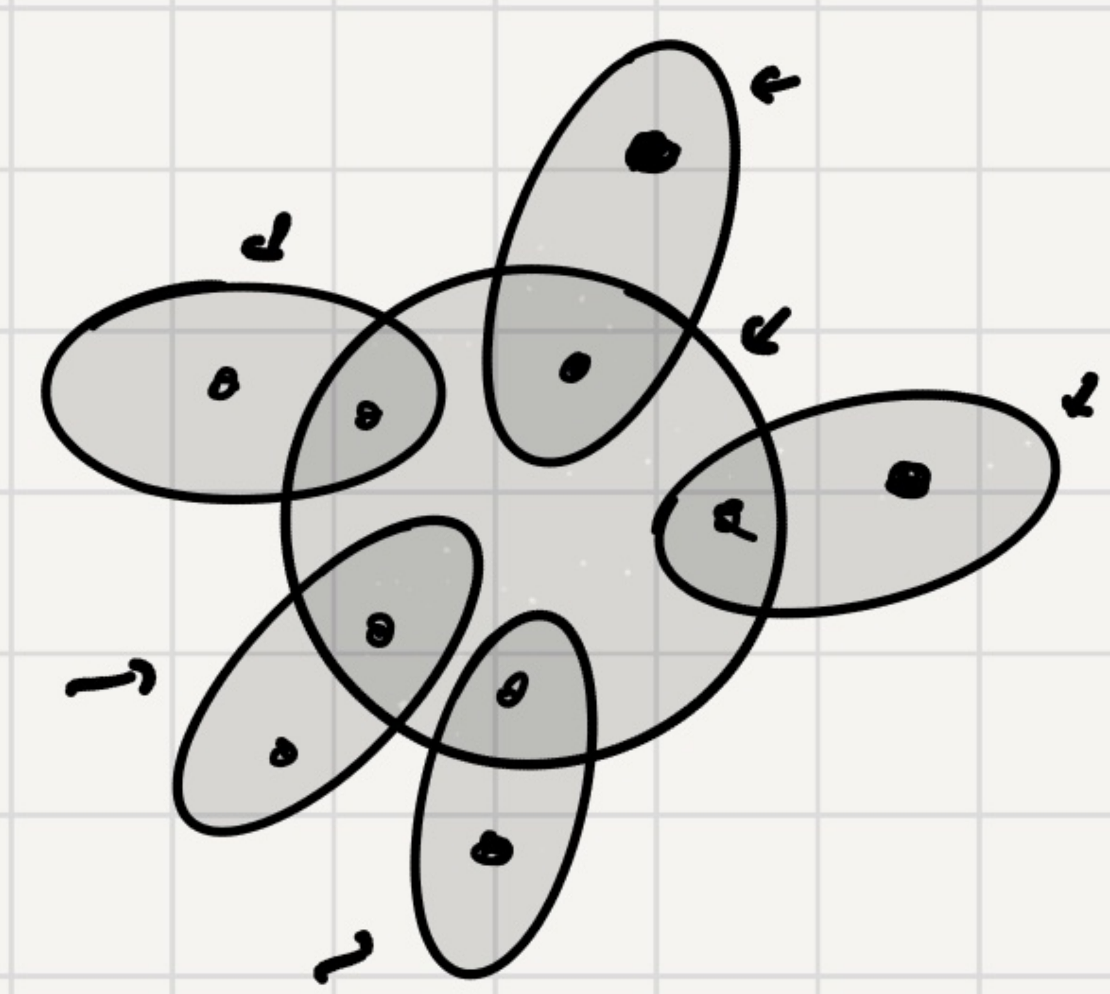
Is it correct? No.

Greedy will pick all 6 sets

optimal solution: 5 sets

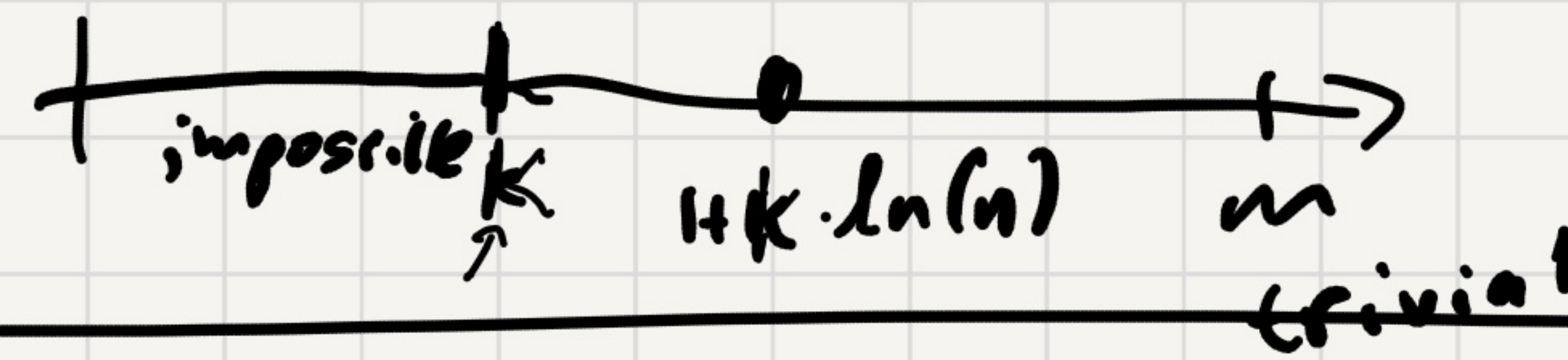
(just the "petals")

Counterexample:



Theorem: "Greedy solution is not too bad":

If optimal solution uses $\underline{\underline{k}}$ sets, then greedy uses at most $k \cdot \ln(n)+1$ sets.

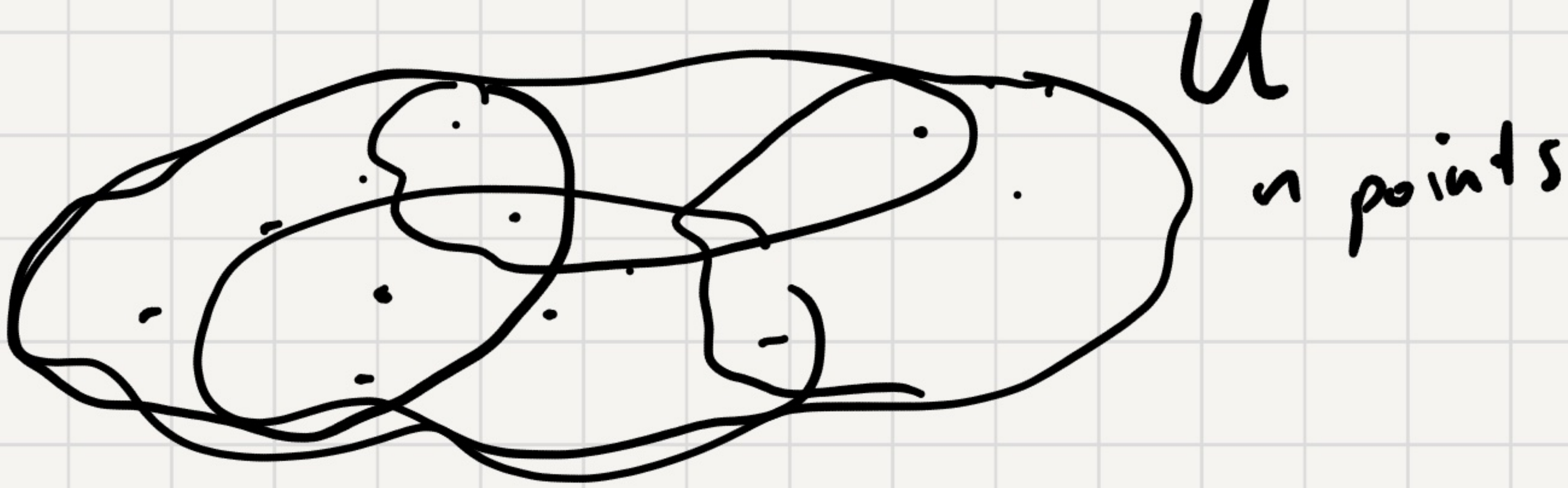


Proof: We keep track of n_t = # of uncovered points after t iterations of the greedy algorithm.

$n_0 = n = |U|$. We'll show that n_t decreases rapidly

$\Rightarrow$ after not too many iterations, $n_t = 0$.

Claim 1: $n_1 \leq n_0 - n_0/k$.



Since optimal solution uses k sets

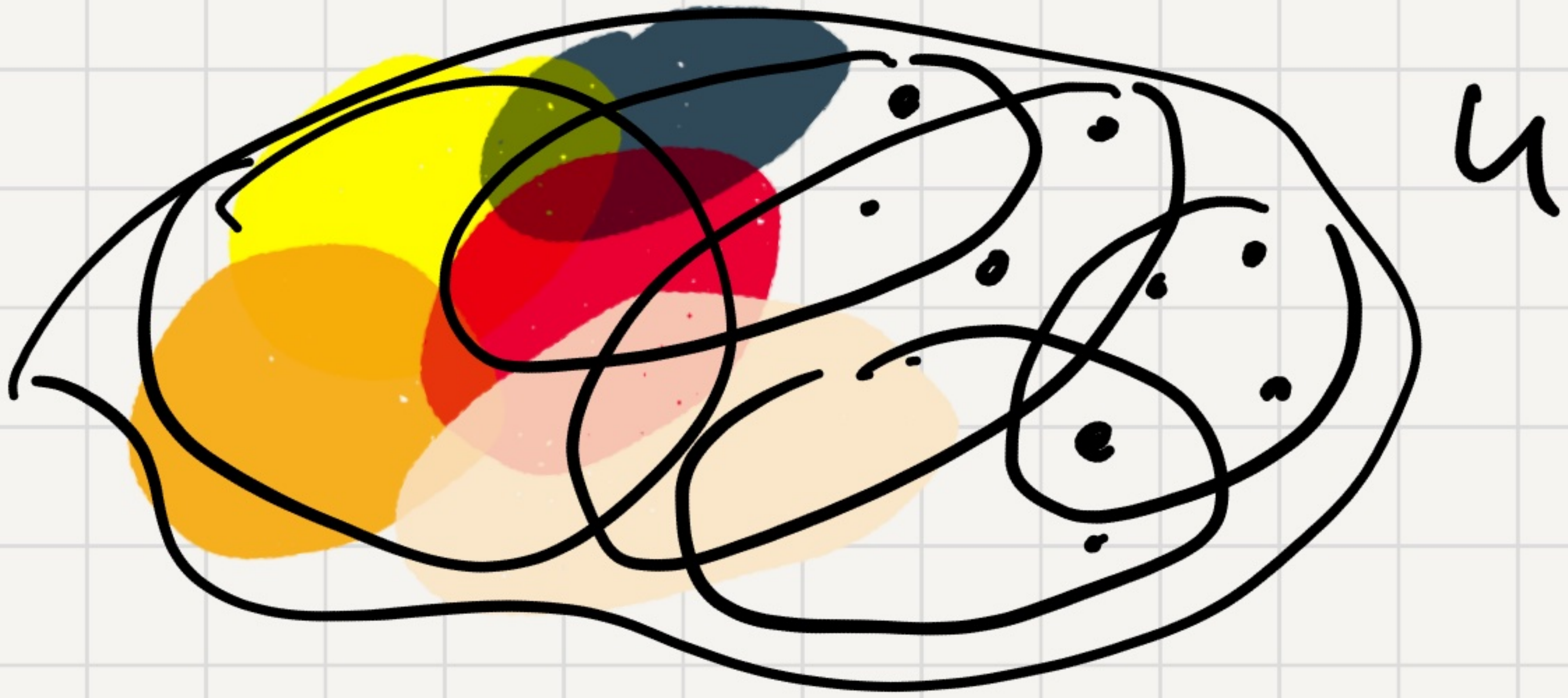
$\Rightarrow \exists$ a set in it that covers at least n/k points.

Greedy picks largest set which is of size $\geq n/k$.

$\square$

Claim 2: $n_{t+1} \le n_t - \frac{n_t}{k}$.

Proof:



Optimal solution covers these n_t pts.

$\Rightarrow$ One of its sets covers $\ge \frac{n_t}{k}$ new pts.

$\Rightarrow$ The set picked by greedy covers at least $\frac{n_t}{k}$ new pts.

Q.E.D. Claim 2.

Back to main proof: We showed that for all $t \ge 0$ $n_{t+1} \le n_t \cdot (1 - 1/k)$.

We get $$n_{t+1} \le n_t (1 - 1/k) \le \ldots \le n_0 \cdot (1 - 1/k)^{t+1} \le n \cdot (e^{-1/k})^{t+1} = n \cdot e^{-(t+1)/k}$$

$\forall x \ge 0$: $1 - x \le e^{-x}$

Sufficient to find minimal t such that $n \cdot e^{-(t+1)/k} < 1$ since then $n_{t+1} < 1$ and greedy covered all pts.

find min t:

$$n \cdot e^{-(t+1)/k} \overset{?}{<} 1$$

$$\Leftrightarrow \quad n \overset{?}{<} e^{(t+1)/k}$$

$$\Leftrightarrow \quad \ln(n) \overset{?}{<} (t+1)/k$$

$$k \cdot \ln(n) \overset{?}{<} t+1$$

$\Longrightarrow$ Picking $t = \lfloor k \cdot \ln(n) \rfloor$ guarantees that $n_{t+1} < 1$ and thus $n_{t+1} = 0$.

$\Rightarrow$ greedy picks at most $\lfloor k \cdot \ln(n) \rfloor + 1$ sets.

Q.E.D. Theorem

New Topic: Dynamic Programming

Main Idea: To solve a big problem find subproblems s.t. the solution to the big problem can be easily derived from the solutions to subproblems.

- Solve all subproblems "from small to large".

Alternative view: Recursion, but using memoization.

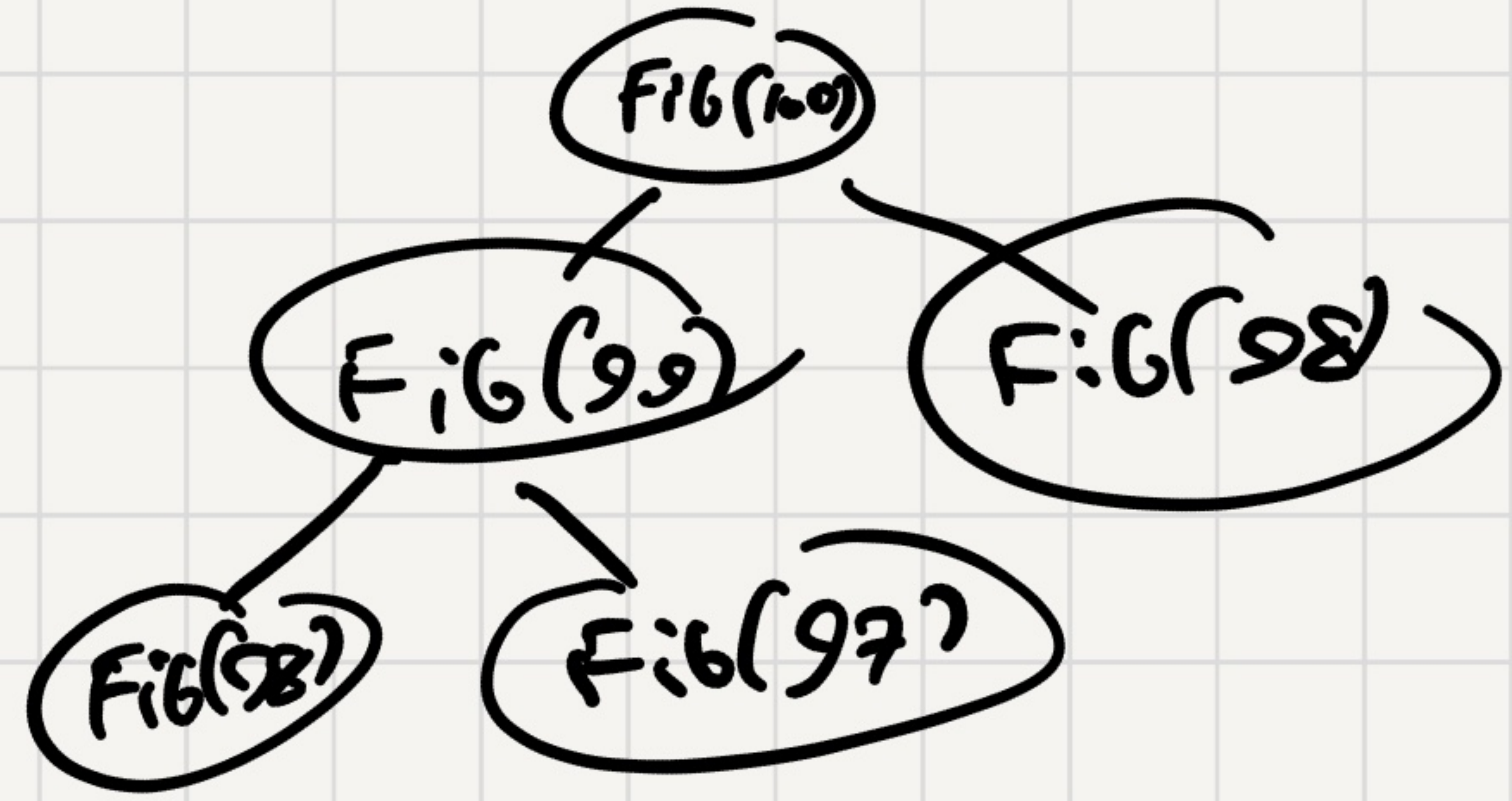
Example: Given n, compute the n'th Fib. number, F_n.

Subproblems: For $i = 2, 3, \ldots, n-1$ compute F_i.

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

$$\begin{cases} F_0 = 0 \quad F_1 = 1 \\ \text{For } i = 2, \ldots, n \\ \quad F_i = F_{i-1} + F_{i-2}. \end{cases}$$

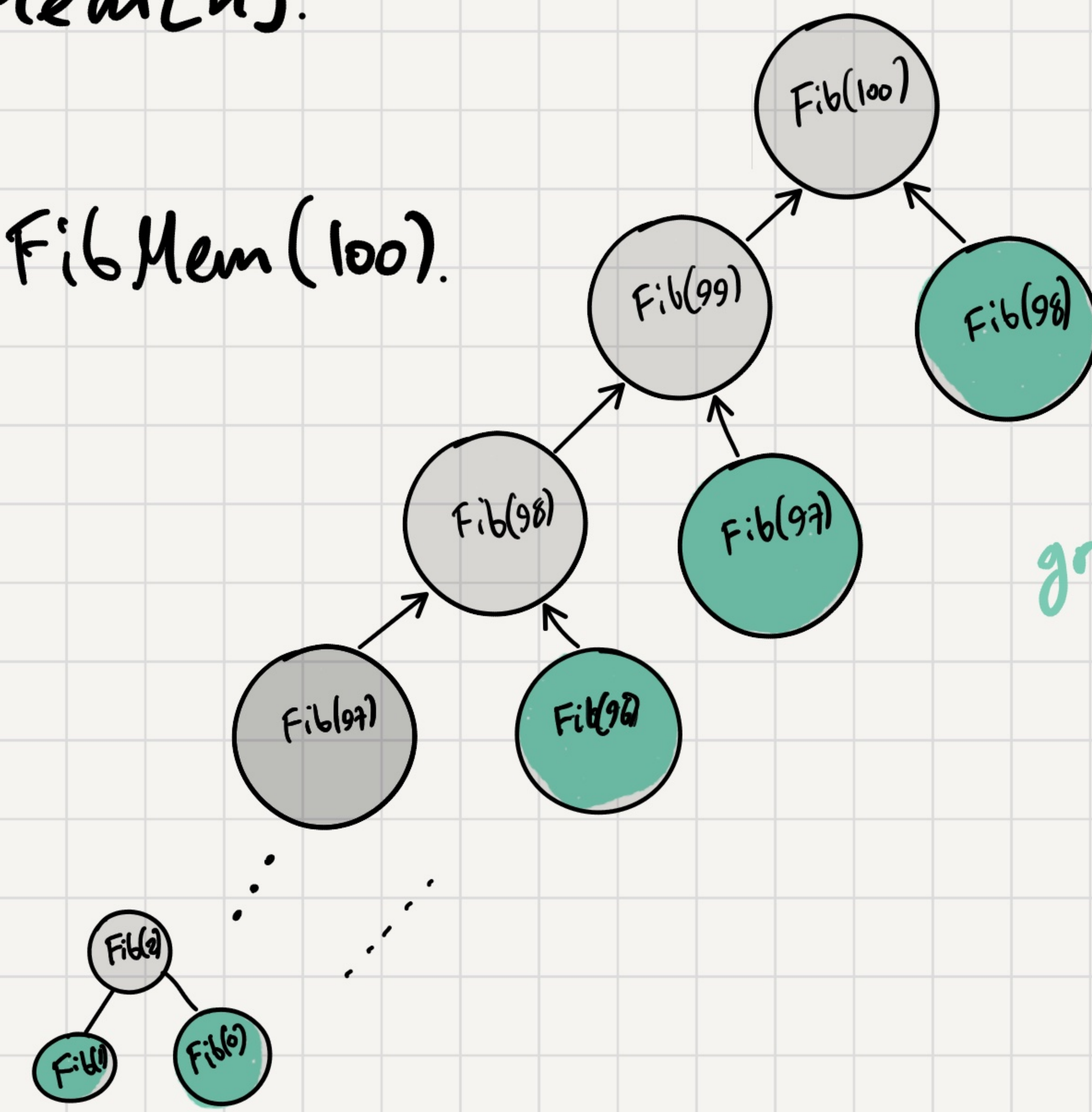
```
def Fib(n):
    if n ≤ 1: return n.
    return Fib(n-1) + Fib(n-2).
```



```
def FibMem(n):
    if n ≤ 1 : return n
    if n in Mem:
        return Mem[n]
    Mem[n] = Fib(n-1) + Fib(n-2).
    return Mem[n].
```

Example:

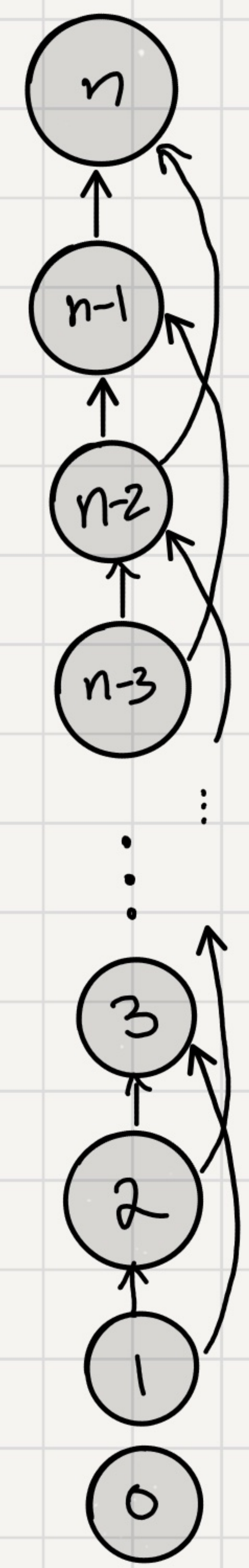
Recursion tree for FibMem(100).



green nodes - from memoization.

We can view each subproblem as a node and we have directed edges $i \to j$ if subproblem j solution depends directly on subprob. i's solution.

The DAG for Fib(n):



Problem 1

~~Shortest~~ Longest Path in a DAG

Recall: Given $G = (V, E)$ with $\ell : E \to \mathbb{Z}$ (we can handle both positive & negative weights)

Given $s, t \in V$.

Goal: Find ~~shortest~~ longest path s to t.

Approach:
- Define a collection of subproblems:
  shortest path from s to v for any $v \in V$.
- Write a recurrence:
  $$dist[v] = \min_{u:(u,v)\in E} \left(dist(u) + \ell(u,v) \right)$$
- Write edge cases $dist[s] = 0$
  $dist[v] = \infty$ if v is a source.
- Analyze runtime & Memory.

Runtime: There are n subproblems.

Mem: $O(n)$.

Each subproblem takes $O(\text{indeg}(v) + 1)$.

Overall: $\sum_{v \in V} c \cdot (1 + \text{indeg}(v)) = c \cdot (|V| + |E|) = O(|V| + |E|)$.

Next Time:

- Longest Increasing Subsequence.
- Edit Distance:
  - → Aligning DNA sequences.
  - → Spell checker
  - → Plagiarism finding.
- Knapsack
- Traveling SalePerson
- All Pairs shortest Paths.
- Viterbi?

1 3 2 4 7 5 6