Set Cover

**Input:**
- Universe $U = \{1, 2, 3, \ldots, n\}$
- Collection of subsets $S_1, S_2, S_3, \ldots, S_m \subseteq U$ (s.t. $S_1 \cup S_2 \cup \ldots \cup S_m = U$)

**Output:**
- Minimal subcollection that covers $U$
  - Minimal Size $J \subseteq [m]$
  - s.t. $\bigcup_{j \in J} S_j = U$

**For Example:**
- $S_1 = \{1, 3, 5\}$
- $S_2 = \{2, 3, 4\}$
- $S_3 = \{1, 2, 3, 4\}$

**Optimal Solution:**
- $J = \{1, 3, 5\}$ or $J = \{2, 3, 7\}$
- $S_1 \cup S_3 = \{1, 2, 3, 4\}$
- $S_2 \cup S_3 = \{1, 2, 3, 4\}$

**Greedy Strategy:** Pick at any step the set that covers the most new points.

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

1. $J \leftarrow \emptyset$.

2. While $S_J \neq U$:
   - Pick $i \in J$ with largest $|S_i \setminus S_J|$ (covers the most new points)
   - Add $i \in J$.

3. Output $J$.

Is it correct? No.

Counterexample:

Greedy will pick all 6 sets
optimal solution: 5 sets (just the "petals"
"Greedy solution is not too bad":

If optimal solution uses \( k \) 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:** \( n_1 \leq n_0 - \frac{n_0}{k} \).

Since optimal solution uses \( k \) sets

\( \Rightarrow \) if a set in it that covers at least \( \frac{n}{k} \) points

Greedy picks largest set which is of size \( \geq \frac{n}{k} \). \( \Box \)
Claim 2: \( n_{t+1} \leq n_t - \frac{n_t}{k} \).

**Proof:**

Optimal solution covers these \( n_t \) pts.

\[ \Rightarrow \] one of its sets covers \( \geq \frac{n_t}{k} \) 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 \( t \geq 0 \)

\[ n_{t+1} \leq n_t \cdot (1 - \frac{1}{k}) \leq \ldots \leq n_0 \cdot (1 - \frac{1}{k})^{t+1} \leq n \cdot \left( e^{-\frac{t+1}{k}} \right) = n \cdot e^{-\frac{t+1}{k}} \]

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

\[ \text{find } \min_t: \ n \cdot e^{-\left(\frac{t+1}{k}\right)} < 1 \]

\[ \Leftrightarrow \]

\[ n < e^{\left(\frac{t+1}{k}\right)} \]

\[ \Leftrightarrow \]

\[ \ln(n) < \left(\frac{t+1}{k}\right) \]

\[ \Leftrightarrow \]

\[ k \cdot \ln(n) < t+1 \]

\[ \Leftrightarrow \]

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

\[ \Rightarrow \] greedy picks at most \( \left\lfloor k \cdot \ln(n) \right\rfloor + 1 \) 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.

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 \( F_i \).

\[
\begin{align*}
F_0 &= 0, \quad F_1 = 1 \\
\text{For } i = 2, \ldots, n \\
F_i &= F_{i-1} + F_{i-2}.
\end{align*}
\]

\[
F_n = F_{n-1} + F_{n-2}
\]

```
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).
We can view each subproblem as a node and we have directed edges \( i \rightarrow j \) if subproblem \( j \) solution depends directly on subprob. \( i \)’s solution.

The DAG for \( \text{Fib}(n) \):
Problem:

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

Given $s, t \in V$.

Goal: Find 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:
  $$\text{dist}[v] = \min_{u: (u,v) \in E} (\text{dist}(u) + l(u,v))$$

- Write edge cases:
  $$\text{dist}[s] = 0$$
  $$\text{dist}[v] = \infty \quad \text{if } v \text{ is a source}.$$
- Analyze runtime & Memory.

There are $n$ subproblems.

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

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

Memo: $O(n)$. 

Longest Shortest Path in a DAG
Next Time:
- Longest Increasing Subsequence.
- Edit Distance: Aligning DNA sequences, spell checker, plagiarism finding.
- Knapsack
- Traveling Salesman
- All Pairs Shortest Paths.
- Viterbi?

1 2 2 4 7 ≤ 6