• The Greedy Method does not always work, even when it is "obvious" that it does. Because of this, in order to use the Greedy Method, we must prove the correctness of the algorithm. Intuitive arguments are very likely incorrect.
IMMEDIATE BENEFIT VS OPPORTUNITY COSTS

IMMEDIATE BENEFIT: How does the choice we’re making now contribute to the objective function

OPPORTUNITY COST: How does the choice we’re making now restrict future choices?

Greedy (usually) Take best immediate benefit and ignore opportunity costs.

Greedy is optimal: Best immediate benefits outweigh opportunity costs
This is a toy version of a problem that comes up in computational economics. We are continuing with our tradition of using the cookie monster as a totem for greedy algorithms, since Cookie personifies greed.

You have $N$ identical cookies to be divided between $M$ cookie monsters.

You know how much the $J$’th cookie each monster gets will make that monster happy. $Happy[I,J]$ is a matrix saying how happy getting the $J$’th cookie makes monster $I$. (Only the number of cookies matters, not which)

For this version, we assume “diminishing returns”: $Happy[I,1] \geq Happy[I,2] \geq \cdots \geq Happy[I,k]$ (We can remove this using DP)

You want to distribute cookies to the monsters to maximize total monster happiness.
5 cookies, 3 monsters
Monster 1: 1st cookie 50, 2nd cookie 30, 3rd cookie 20, 4th +5th : 0
Monster 2: All cookies 20
Monster 3: 1st cookie 30, 2nd cookie 25, 3rd cookie 20, 4th cookie 15, 5th cookie 10

What should we do?
5 cookies, 3 monsters
Monster 1: 1\textsuperscript{st} cookie 50, 2\textsuperscript{nd} cookie 30, 3\textsuperscript{rd} cookie 20, 4\textsuperscript{th} +5\textsuperscript{th} : 0
Monster 2: All cookies 20
Monster 3: 1\textsuperscript{st} cookie 30, 2\textsuperscript{nd} cookie 25, 3\textsuperscript{rd} cookie 20, 4\textsuperscript{th} cookie 15, 5\textsuperscript{th} cookie 10

What should we do? Give 1\textsuperscript{st} cookie to Monster 1.

Then monster 1 would get 30 from 2\textsuperscript{nd} cookie, same as monster 3 from 1\textsuperscript{st} cookie it gets, and bigger than 1\textsuperscript{st} cookie for monster 2
5 cookies, 3 monsters

Monster 1: 1\textsuperscript{st} cookie 50, 2\textsuperscript{nd} cookie 30, 3\textsuperscript{rd} cookie 20, 4\textsuperscript{th} +5\textsuperscript{th}: 0

Monster 2: All cookies 20

Monster 3: 1\textsuperscript{st} cookie 30, 2\textsuperscript{nd} cookie 25, 3\textsuperscript{rd} cookie 20, 4\textsuperscript{th} cookie 15, 5\textsuperscript{th} cookie 10

Give 1\textsuperscript{st} cookie to Monster 1.

Give 1\textsuperscript{st} cookie to Monster 3. Benefits for next: 30, 20, 25

Give 2\textsuperscript{nd} cookie to Monster 1. Benefits for next: 20, 20, 25

Give 2\textsuperscript{nd} cookie to Monster 3. Benefits for next: 20, 20, 20

Give 3\textsuperscript{rd} cookie to Monster 1. Total happiness: 100+ 0+55=155
Give the next cookie to the monster who would enjoy it most. More precisely, if monster I currently has $J_I$ cookies, the amount it would benefit by getting one more is $Happy[I,J_I + 1]$. Give the next cookie to the I that has maximum value for this, breaking ties arbitrarily.
What does it mean that the greedy algorithm solves an optimization problem?

I: problem instance.
GS: greedy solution to I
OS: other (optimal) solution to I
Would be incorrect if Value(OS) > Value (GS)
So we need to show: For every instance I, let GS be the greedy algorithm’s solution to I. Let OS be any other solution for I. Then Value(OS) ≤ Value (GS) (or Cost(GS) ≤ Cost (OS) for minimization)
Tricky part: OS is arbitrary solution, not one that makes sense. We don’t know much about it
We’ll see a number of general methods to prove optimality:
1. Modify the solution, aka Exchange, Transformation: most general
2. Greedy-stays-ahead: more intuitive
3. Greedy achieves the bound: also comes up in approximation, LP, network flow
4. Unique local optimum: dangerously close to a common fallacy

Which one to use is up to you, but only Modify-the-solution applies universally, others can be easier but only work in special cases.
Final goal: there is an optimal solution that contains all of the greedy algorithm’s decisions, in other words, the greedy solution is an optimal solution.

Format 1: Show that there is an optimal solution that contains the first greedy decision. Then use recursion/induction to handle the rest.

Format 2: Show by induction on k that there is an optimal solution containing the first k decisions
MODIFY-THE-SOLUTION (FIRST FORMAT)

- General structure of modify-the-solution:
  1. Let $d$ be the first decision the greedy algorithm makes, and let $g$ be the greedy choice at $d$. (Goal: there is an optimal solution choosing $g$)
  2. Let OS be any optimal solution. Can assume OS does not choose g, since otherwise we’ve achieved our goal.
  3. Show how to modify OS into some solution $OS'$ that chooses $g$, and that is at least as good as OS.
  4. Use 1-3 in an inductive argument. GS = g + GS(smaller problem), OS' = g + some other solution to same smaller problem.
3. Show how to modify OS into some solution $OS'$ that chooses $g$, and that is at least as good as OS.

A: Define $OS'$ from OS, $d$, $g$.

B: Prove $OS'$ is a valid solution. Use OS is valid, definition of $d$, $g$.

C: Prove $OS'$ is also optimal. Use definition of objective function, $d$, $g$, to compare objective on OS to $OS'$. Need to show $OS'$ is at least as good as OS.

If there are multiple cases, do A-C for each one
Let GS be the sequence of N monsters that get each cookie in the greedy solution. Let OS be any way of assigning N cookies to the monsters. We want to show the total monster happiness for GS is at least as high as for OS.

1st greedy move: Look at Happy[I,1] for each I. Pick the I with the maximum value to get cookie.

Modify the solution lemma: Let I be argmax Happy[I,1]. Assume OS is an assignment that doesn’t start by giving I a cookie. Then there is an assignment OS’ that does start by giving I a cookie, with at least the total happiness of OS.
Modify the solution lemma: Let $I$ be $\text{argmax } \text{Happy}[i,1]$. Assume $\text{OS}$ is an assignment that doesn’t start by giving $I$ a cookie. Then there is an assignment $\text{OS}'$ that does start by giving $I$ a cookie, with at least the total happiness of $\text{OS}$.

Case 1: $I$ eventually gets a cookie in $\text{OS}$

Case 2: $I$ never gets a cookie in $\text{OS}$
MTS CASE 1: I GETS A COOKIE LATER

Define OS': OS: 1\textsuperscript{st} move : I' gets cookie
move t later: I gets a cookie
OS': Like OS but:
  1\textsuperscript{st} move: I gets cookie
  move t: Give I' a cookie
OS' is still solution: Same number of cookies distributed to monsters
Compare total happiness: All monsters except I, I' same happiness.
I ends up with same number of cookies, too. So does I'.
So total happiness for OS' is same as for OS.
Define OS’: OS: 1st move: I’ gets cookie
OS’: Like OS but:
    1st move: I gets cookie
OS’ is still solution: Same number of cookies distributed to monsters
Compare total happiness: All monsters except I, I’ same happiness.
    I happiness increases by Happy[I,1], since went from 0 cookies to 1.
    I’ happiness decreases by Happy[I’,J] for some J
What do we know to relate these two?
Define OS': OS: 1st move: I' gets cookie
OS': Like OS but:

1st move: I gets cookie

OS' is still solution: Same number of cookies distributed to monsters

Compare total happiness: All monsters except I, I' same happiness.
I happiness increases by Happy[I,1], since went from 0 cookies to 1.
I' happiness decreases by Happy[I',J] for some J

Happy[I,1] ≥ Happy[I',1] by definition of greedy algorithm
Happy[I',1] ≥ Happy[I',J] by diminishing returns condition

Therefore, increase for I ≥ the decrease for I', so TH(OS') ≥ TH(OS)
In each case:

Define OS’. What do we have to work with? OS, definition of greedy algorithm. Very important to do FIRST. Can’t prove things about OS’ without defining it FIRST.

Show OS’ meets any requirements for a solution (constraints). What do we have to work with? We know OS meets all the requirements, definition of greedy

Compare objective functions for OS and OS’
What increased? What decreased? How do they balance out?
Lemma: For every instance of Cookie Distribution, there is an optimal solution that starts by giving a cookie to the same monster as the greedy algorithm does.
Once we’ve given monster I a cookie, it’s the same type of problem, Except that:

- N-1 cookies to distribute
- Ignore Happy[J, N] for $J \neq I$
- Shift Happy[I,K]. I’th row of Happy now becomes:
  \[ Happy[I, 2], Happy[I, 3], \ldots, Happy[I, N] \]

Call this instance NewHappy, N-1.
There is an optimal solution that always picks the greedy choice.

- Proof by strong induction on \( n \), the number of events.
- Base Case. \( n = 0 \) or \( n = 1 \). The greedy (actually, any) choice works.

- Inductive Hypothesis (Strong.)
- Assume that the greedy algorithm is optimal for any \( k \) events for \( 0 \leq k \leq n - 1 \).
- Goal: Greedy is optimal for any \( n \) events

Proof: Let Events’ be all the events that don’t conflict with E1. Apply the lemma to OS to get \( OS' \).

\[
\begin{align*}
\text{GS} &= E1 + \text{GS(Events')} \\
\text{OS'} &= E1 + \text{Some solution for Events'} \\
|\text{GS}| &= 1 + |\text{GS(Events')}| \geq 1 + |\text{Some solution for Events'}| = |\text{OS'}| \geq |\text{OS}|
\end{align*}
\]

Conclusion: The GS is optimal for every set of events
Unless you do it wrong.

We prove by induction on $N$, number of cookies, that greedy solution is optimal, i.e., $\text{TH}(\text{GS}) \geq \text{TH}(\text{OS})$ for any solution $\text{OS}$

Base case: $N=0$. No cookies, any solution is optimal

Induction step: assume $\text{GS}$ is optimal for any instance with $N-1$ cookies

Let $\text{OS}$ be any solution.

Lemma: There is an $\text{OS}'$ with $\text{OS}' = \text{give monster 1 cookie} + \text{some solution to NewHappy with N-1 cookies}$

$\text{GS}(\text{Happy}, N) = \text{give monster 1 cookie} + \text{GS}(\text{NewHappy}, N-1)$
Induction step: assume GS is optimal for any instance with N-1 cookies
Let OS be any solution.
Lemma: There is an OS’ with OS’= give monster 1 cookie + some solution to NewHappy with N-1 cookies and TH(OS’) ≥ TH(OS)
GS(Happy, N) = give monster 1 cookie + GS(NewHappy, N-1)
OS’ = give monster 1 cookie + OS(NewHappy, N-1)
TH(GS) = Happy[I, 1] + TH(GS(NewHappy, N-1)) ≥ Happy[I, 1] + TH(OS(NewHappy, N-1)) = TH(OS’) ≥ TH(OS)
We’ve shown GS is at least as good as any other solution.
We usually present the greedy algorithm as: Apply first greedy move.
Simplify recursively
Repeat.

The purpose of the induction step is to make sure we defined ``simplify recursively'' correctly. The induction hypothesis means the ``repeat'' step works. The modify-the-solution lemma means the ``apply the first greedy move'' step works.
We need to repeatedly find the $I$ that gives us the maximum value of $Happy[I, J_I + 1]$, where we’ve currently given monster $I$ $J_I$ cookies. When we do, we increment $J_I$.

Most obvious way: Keep all $J_I$ in array, look through all $I$, take max.

Total time: $O(NM)$, because we look through all monsters each of $N$ iterations.

Can we do better using say, data structures?
We need to repeatedly find the I that gives us the maximum value of \( \text{Happy}[I, J_I + 1] \), where we’ve currently given monster I \( J_I \)Cookies

When we do, we increment \( J_I \)

What do we need to do in one step: Set of values, one per monster.
Access: need to find maximum
Update: Replace maximum with new element.
What do we need to do in one step: Set of values, one per monster.
Access: need to find maximum, Replace maximum: DeleteMax
Replace with new element: Insert.
Good match: binary heap.
Need to know what values in heap mean, so also should have fields
I: monster number, J: current position in row.
Create max-heap H of triples (I,J, Value), ordered by Value
Insert (I,1, Happy[I,1]) into H for each I
FOR T=1 to N do:
  (I,J,V) = deletemax.H;
  Give cookie to monster I
  Insert (I,J+1, Happy[I,J+1]) into H
Create max-heap \( H \) of triples \((I, J, \text{Value})\), ordered by Value
Insert \((I, 1, \text{Happy}[I, 1])\) into \( H \) for each \( I \)

FOR \( T = 1 \) to \( N \) do:

\((I, J, V) = \text{deletemax}.H;\)

Give cookie to monster \( I \)
Insert \((I, J+1, \text{Happy}[I, J+1])\) into \( H \)

\( M + 2N \) heap operations. Heap stays \( M \) size, so heap operations \( O(\log M) \). Total time: \( O((N+M) \log M) \)
This is a proof technique that does not work in all cases. The way it works is to logically determine a bound (lower or upper.) Then show that the greedy strategy achieves this bound and therefore is correct.
Total happiness (any solution). \( \leq \) sum of \( N \) largest values for distinct array positions

Greedy achieves the bound:
Claim: The happiness per step for the greedy solution is exactly the \( N \) largest array entries.

Therefore, total happiness in greedy solution \( \geq \) total happiness in any solution
Modify-the-solution is most general to prove greedy algorithms are correct when they are correct.

When the greedy algorithm isn’t correct, we still sometimes want to use it, because it is fast and comes somewhat close. Achieves-the-bound can be generalized to show greedy algorithms “approximate” the optimal solution, even when they aren’t optimal.