Greedy

CPSCP

Riccardo Maso, June 25, 2026

Ca' Foscari
University
of Venice

RAVEN – Research on Algorithms Venice

Greedy approach - overview

- Iteratively selects the locally optimal choice.
- Yields highly efficient, lightweight solutions.

Greedy approach - overview

- Iteratively selects the locally optimal choice.
- Yields highly efficient, lightweight solutions.
- When does it work?
  - **Optimal Substructure:** The global best solution is built directly from optimal subproblems.
  - **Greedy Property:** Immediate best choices guarantee the overall best outcome, zero backtracking required.

Coin change

- Let $C = \{c_1, c_2, \ldots, c_n\}$ be a set of n sorted coin denominations, where the lowest denomination is 1, with infinite supply.

- Let V a target amount.

Coin change

- Let $C = \{c_1, c_2, \ldots, c_n\}$ be a set of n sorted coin denominations, where the lowest denomination is 1, with infinite supply.

- Let V a target amount.

$$\text{Minimize } \sum_{i=1}^{n} x_i$$

$$\text{Subject to: } \sum_{i=1}^{n} x_i c_i = V$$

Where x_i is the amount of coin used with value c_i

Greedy approach

- ***Locally optimal choice:***

Greedy approach

- ***Locally optimal choice:*** select the largest coin denomination that does not exceed the remaining amount.

  1. Sort the coin denominations in descending order $(c_1 > c_2 > \ldots > c_n)$,

  2. Compute how many of the largest available coins I can use (V/c_i),

  3. Compute the remaining amount $(V \,\%\, c_i)$ and repeat for the next coin.

Greedy approach

- ***Locally optimal choice:*** select the largest coin denomination that does not exceed the remaining amount.

    1. Sort the coin denominations in descending order $(c_1 > c_2 > \ldots > c_n)$,

    2. Compute how many of the largest available coins I can use (V/c_i),

    3. Compute the remaining amount $(V \,\%\, c_i)$ and repeat for the next coin.

- Does it always work?

Example 1

- $C = \{25,10,5,1\}$, $V = 67$

Example 1

- $C = \{25,10,5,1\}$, $V = 67$
  - Largest valid coin $c_1 = 25$, $x_1 = 2$, $\tilde{V} = 67 - 2 * 25 = 17$,
  - Largest valid coin $c_2 = 10$, $x_2 = 1$, $\tilde{V} = 17 - 1 * 10 = 7$,
  - Largest valid coin $c_3 = 5$, $x_3 = 1$, $\tilde{V} = 7 - 1 * 5 = 2$,
  - Largest valid coin $c_4 = 1$, $x_4 = 2$, $\tilde{V} = 2 - 2 * 1 = 0$,
- Solution $X = \{2,1,1,2\}$.

Example 2

- $C = \{40,15,7,1\}$, $V = 67$

Example 2

- $C = \{40,15,7,1\}$, $V = 67$
  - Largest valid coin $c_1 = 40$, $x_1 = 1$, $\tilde{V} = 67 - 1 * 40 = 27$,
  - Largest valid coin $c_2 = 15$, $x_2 = 1$, $\tilde{V} = 27 - 1 * 15 = 12$,
  - Largest valid coin $c_3 = 7$, $x_3 = 1$, $\tilde{V} = 12 - 1 * 7 = 5$,
  - Largest valid coin $c_4 = 1$, $x_4 = 5$, $\tilde{V} = 5 - 5 * 1 = 0$,
- Solution $X = \{1,1,1,5\}$.

Example 3

- Does a greedy approach always work?

Example 3

- Does a greedy approach always work? No
- $C = \{4,3,1\}, V = 6$

Example 3

- Does a greedy approach always work? No
- $C = \{4,3,1\}$, $V = 6$
  - Greedy gives $X = \{1,0,2\}$
  - Optimal solution $X = \{0,2,0\}$

Example 3

- Does a greedy approach work? Depends!
- $C = \{4,3,1\}$, $V = 6$
  - Greedy gives $X = \{1,0,2\}$
  - Optimal solution $X = \{0,2,0\}$
- An optimal solution can always be found using ***Dynamic Programming***.
  - What is that? -> In the next meetings :)

Extra fact

- **Canonical Systems:** Coin systems where the greedy approach always produces the optimal solution.

- **Real-World Application:** Almost all modern currency systems are designed to be canonical, ensuring that cashiers (and machines) can compute the optimal change efficiently by using a simple greedy strategy.

When Greedy is the Right Choice

- Greedy algorithms are not always bound by restrictive constraints.
- In some cases, the greedy approach is the most efficient and correct method available. Some examples are:
  - Dijkstra’s Algorithm
  - Kruskal’s Algorithm

Scheduling problem

- Let $T = \{p_1, p_2, \ldots, p_n\}$ be a set of jobs with processing time p_i,
- Let $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ be a *permutation* of jobs representing the execution order, where $\pi(i)$ is the job scheduled at the i-th position,
- $C_{\pi(k)} = \Sigma_{j=1}^{k} p_{\pi(j)}$ is the completion time for the job at the position k,
- **Minimise the total completion time:** $\min_{\pi} \sum_{i=1}^{n} C_i$

Solution

- **Greedy approach:** *Shortest job first (SFJ).*
  - Sort the jobs in non-decreasing order of p_i
  - Schedule the jobs in this order.

Example

- $T = \{8,3,1,5\}$

Example

- $T = \{8,3,1,5\}$
- Sort them $p_3(1) \leq p_2(3) \leq p_4(5) \leq p_1(8)$
  - $C_3 = 1$
  - $C_2 = 1 + 3 = 4$
  - $C_4 = 4 + 5 = 9$
  - $C_2 = 9 + 8 = 17$
- $\sum C_i = 31$

Summary

- **Pros:** highly efficient and lightweight. They make a locally optimal choice at each step, never triggering a TLE (Time Limit Exceeded).

- **Cons:** *not always* correct. A local optimum does not always guarantee a global optimum. Could easily trigger a WA (WrongAnswer).

- **Tips:**

- Greedy solutions usually rely on sorting static data or using a Priority Queue for dynamic data,
- Check input size; if constraints are too big, usually Complete Search and Dynamic Programming are too slow.

Exercises

Air Conditioned Minions - Kattis

- You manage a laboratory with N minions.
- Each minion i has a strict temperature preference $[A_i, B_i]$.
- A minion is comfortable if and only if its room temperature T satisfies:

$$A_i \leq T \leq B_i$$

Air Conditioned Minions - Kattis

- You manage a laboratory with N minions.
- Each minion i has a strict temperature preference $[A_i, B_i]$.
- A minion is comfortable if and only if its room temperature T satisfies:

$$A_i \leq T \leq B_i$$

- You can set up multiple rooms, but each room can only be set to one constant temperature.
- Multiple minions can share a room if the temperature satisfies all of their windows.

Air Conditioned Minions - Kattis

- You manage a laboratory with N minions.
- Each minion i has a strict temperature preference $[A_i, B_i]$.
- A minion is comfortable if and only if its room temperature T satisfies:

$$A_i \leq T \leq B_i$$

- You can set up multiple rooms, but each room can only be set to one constant temperature.
- Multiple minions can share a room if the temperature satisfies all of their windows.

➡ **Goal:** *Find the minimum number of rooms to house all N minions.*

Air Conditioned Minions - Kattis

- **Intuitions:**
  - To minimise rooms, maximise how many minions fit into each room.

Air Conditioned Minions - Kattis

- **Intuitions:**
  - To minimise rooms, maximise how many minions fit into each room.
  - When looking at the most restrictive remaining minion, always place the room's temperature at its absolute upper limit to leave maximum flexibility for subsequent minions.

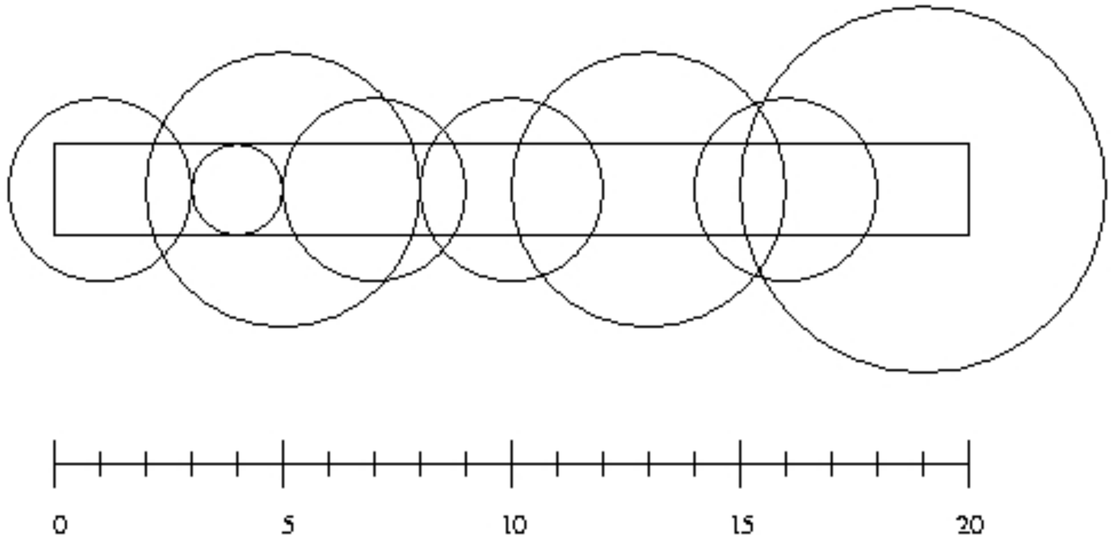
Air Conditioned Minions - Kattis

Algorithm 1: Air Conditioned Minions (Greedy Sweep)

```
1 sort Minions in non-decreasing order of B_i;
2 rooms ← 0,    i ← 0,    n ← len(Minions);
3 while i < n do
4 |   T ← Minions[i].B ;
5 |   rooms ←rooms+1;
6 |   while i < n and Minions[i].A ≤ T do
7 |   |   i ← i + 1;
8 return rooms;
```

Grass - Kattis

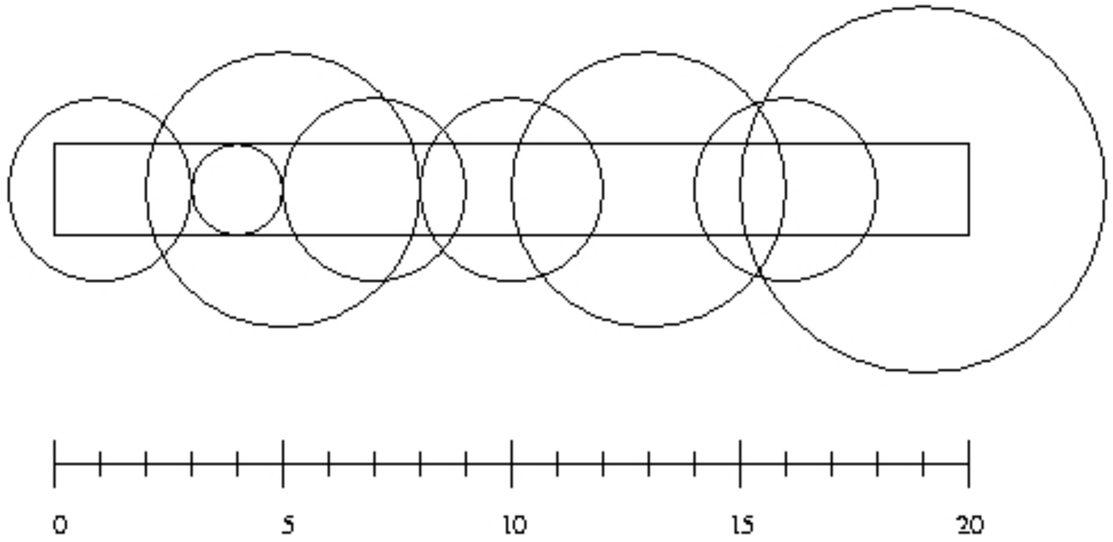
- You have a rectangular strip of grass of length l and width w,
- There are n sprinklers installed along the horizontal centerline of the strip,
- Each sprinkler i is positioned at a distance x_i from the left end and has a watering radius r_i,



Grass - Kattis

- You have a rectangular strip of grass of length l and width w,
- There are n sprinklers installed along the horizontal centerline of the strip,
- Each sprinkler i is positioned at a distance x_i from the left end and has a watering radius r_i,

➡**Goal**: Find the minimum number of sprinklers needed to completely cover the interval $[0,l]$.
If it's impossible, output -1.

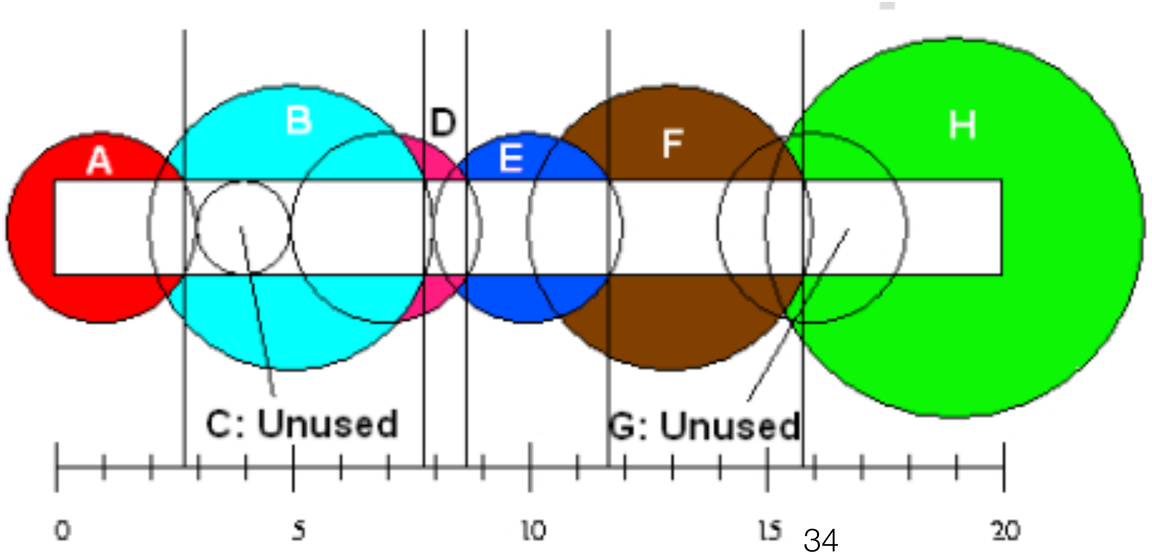


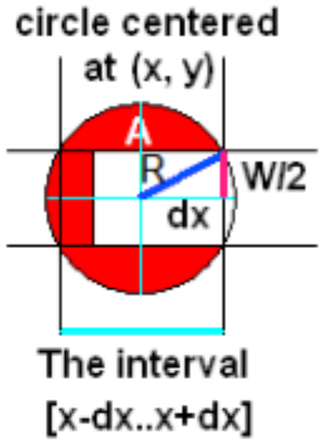
Grass - Kattis

- **Geometry reduction:**

Grass - Kattis

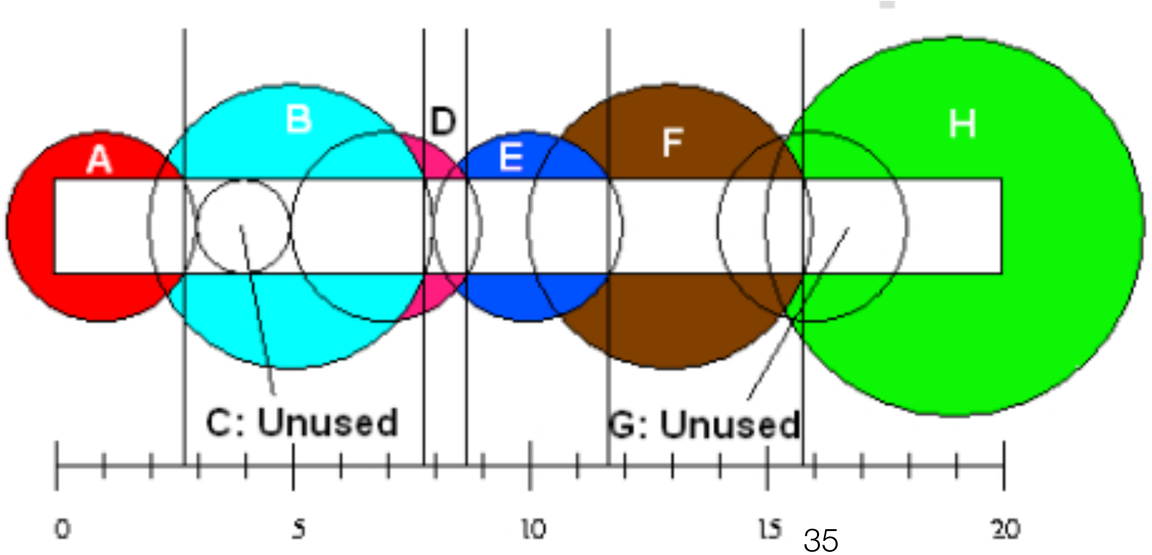
- **Geometry reduction:** Each circle reduces to a horizontal interval where it fully spans the strip of grass width.
- Useful interval is $[x_i - \Delta x_i, x_i + \Delta x_i]$ where $\Delta x_I = \sqrt{r_i^2 - (w/2)^2}$.
- We can remove some sprinklers, which ones?

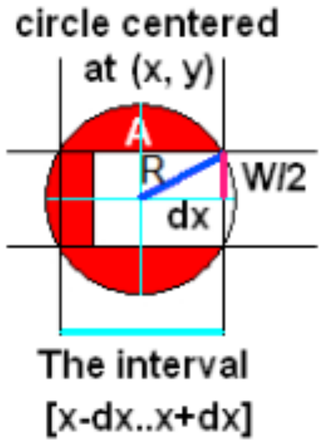




Grass - Kattis

- **Geometry reduction:** Each circle reduces to a horizontal interval where it fully spans the strip of grass width.
- Useful interval is $[x_i - \Delta x_i, x_i + \Delta x_i]$ where $\Delta x_I = \sqrt{r_i^2 - (w/2)^2}$.
- We can remove some sprinklers, which ones? $2r \leq w$.





Grass - Kattis

Algorithm 1: Watering Grass

$Intervals \leftarrow \left\{ [x - \Delta x, x + \Delta x] \;\middle|\; \Delta x = \sqrt{r^2 - (W/2)^2} \geq 0 \right\}$;
sort $Intervals$ **by** $left\_endpoint$;
$target \leftarrow 0, \quad count \leftarrow 0, \quad i \leftarrow 0$;
while $target < L$ **do**
    $max\_r \leftarrow -\infty$;
    while $i < len(Intervals)$ ***and*** $Intervals[i].left \leq target$ **do**
        $max\_r \leftarrow \max(max\_r, Intervals[i].right)$;
        $i \leftarrow i + 1$;
    if $max\_r \leq target$ **then**
        return -1 // *Gap detected; coverage impossible*;
    $target \leftarrow max\_r$;
    $count \leftarrow count + 1$;
return $count$;

Extra

https://open.kattis.com/problems/ballotboxes