CS 341: ALGORITHMS

Lecture 8: greedy algorithms

Slides by Trevor Brown (some material from Doug Stinson)

trevor.brown@uwaterloo.ca
(http://tbrown.pro)
Optimization Problems

**Problem:** Given a problem instance, find a feasible solution that maximizes (or minimizes) a certain objective function.

**Problem Instance:** Input for the specified problem.

**Problem Constraints:** Requirements that must be satisfied by any feasible solution.

**Feasible Solution:** For any problem instance $I$, $\text{feasible}(I)$ is the set of all outputs (i.e., solutions) for the instance $I$ that satisfy the given constraints.

**Objective Function:** A function $f : \text{feasible}(I) \rightarrow \mathbb{R}^+ \cup \{0\}$. We often think of $f$ as being a profit or a cost function.

**Optimal Solution:** A feasible solution $X \in \text{feasible}(I)$ such that the profit $f(X)$ is maximized (or the cost $f(X)$ is minimized).
SOLVING OPTIMIZATION PROBLEMS

• Lots of techniques
• We will study **greedy** approaches first
• Later, dynamic programming
  • Sort of like divide and conquer
    but can **sometimes** be much more efficient than D&C
• Greedy algorithms are usually
  • Very fast, but hard to prove optimality for
  • Structured as follows...
The Greedy Method

partial solutions

Given a problem instance $I$, it should be possible to write a feasible solution $X$ as a tuple $[x_1, x_2, \ldots, x_n]$ for some integer $n$, where $x_i \in \mathcal{X}$ for all $i$. A tuple $[x_1, \ldots, x_i]$ where $i < n$ is a partial solution if no constraints are violated.

Note: it may be the case that a partial solution cannot be extended to a feasible solution.

choice set

For a partial solution $X = [x_1, \ldots, x_i]$ where $i < n$, we define the choice set

$$\text{choice}(X) = \{y \in \mathcal{X} : [x_1, \ldots, x_i, y] \text{ is a partial solution}\}.$$
The Greedy Method (cont.)

local evaluation criterion
For any $y \in X$, $g(y)$ is a local evaluation criterion that measures the cost or profit of including $y$ in a (partial) solution.

extension
Given a partial solution $X = [x_1, \ldots, x_i]$ where $i < n$, choose $y \in \text{choice}(X)$ so that $g(y)$ is as small (or large) as possible. Update $X$ to be the $(i + 1)$-tuple $[x_1, \ldots, x_i, y]$.

greedy algorithm
Starting with the “empty” partial solution, repeatedly extend it until a feasible solution $X$ is constructed. This feasible solution may or may not be optimal.

Local evaluation means we cannot consider future choices when deciding whether to include $y$ in our solution.

We irrevocably decide to include $y$ (or not). We do not reconsider.

We choose the next element to include greedily by taking the $y$ that gives the maximum local improvement.

This may or may not be a good idea...
Greedy algorithms do no **looking ahead** and no **backtracking**.

Greedy algorithms can usually be implemented efficiently. Often they consist of a **preprocessing step** based on the function \( g \), followed by a **single pass** through the data.

In a greedy algorithm, only **one feasible solution** is constructed.

The execution of a greedy algorithm is based on **local criteria** (i.e., the values of the function \( g \)).

**Correctness:** For certain greedy algorithms, it is possible to prove that they always yield optimal solutions. However, these proofs can be tricky and complicated!
Problem: Interval Selection

95% Confidence Interval? Why not 100% confidence?
PROBLEM: INTERVAL SELECTION

- **Input**: a set \( A = \{A_1, \ldots, A_n\} \) of time intervals
  - Each interval \( A_i \) has a start time \( s_i \) and a finish time \( f_i \)
- **Feasible solution**: a subset \( B \) of \( A \) containing **pairwise disjoint** intervals
- **Output**: a feasible solution of **maximum size**
  - i.e., one that maximizes \( |B| \)

Where \( s_i \) and \( f_i \) are positive integers

- **Chosen**
- **Rejected**

Bad solution. Not optimal!
POSSIBLE GREEDY STRATEGIES
FOR INTERVAL SELECTION

1. Sort the intervals in increasing order of starting times. At any stage, choose the earliest starting interval that is disjoint from all previously chosen intervals (i.e., the local evaluation criterion is \( s_i \)).

2. Sort the intervals in increasing order of duration. At any stage, choose the interval of minimum duration that is disjoint from all previously chosen intervals (i.e., the local evaluation criterion is \( f_i - s_i \)).

3. Sort the intervals in increasing order of finishing times. At any stage, choose the earliest finishing interval that is disjoint from all previously chosen intervals (i.e., the local evaluation criterion is \( f_i \)).

Does one of these strategies yield a correct greedy algorithm?
**Strategy 1: Proving Incorrectness**

- **Idea:** Find one input for which the algorithm gives a non-optimal solution or an infeasible solution.

### Strategy 1

Sort the intervals in increasing order of starting times. At any stage, choose the earliest starting interval that is disjoint from all previously chosen intervals (i.e., the local evaluation criterion is $s_i$).

### Consider input:

- $[0, 10), [1, 3), [5, 7)$.

---

<table>
<thead>
<tr>
<th>X-axis</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chosen</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

*x-axis*
HOW ABOUT STRATEGY 2?

<table>
<thead>
<tr>
<th>Strategy 2</th>
<th>Sort the intervals in increasing order of duration. At any stage, choose the interval of minimum duration that is disjoint from all previously chosen intervals (i.e., the local evaluation criterion is $f_i - s_i$).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consider input:</td>
<td>$[0, 5), [6, 10), [4, 7)$.</td>
</tr>
</tbody>
</table>

We will show that **Strategy 3** (sort in increasing order of finishing times) always yields the optimal solution.
Algorithm: \textit{GreedyIntervalSelection}(\mathcal{A})
renames the intervals, by sorting if necessary, so that \( f_1 \leq \cdots \leq f_n \)
\( \mathcal{B} \leftarrow \{ A_1 \} \)
\( \text{prev} \leftarrow 1 \)
\text{comment: prev is the index of the last selected interval}
\textbf{for} \( i \leftarrow 2 \text{ to } n \)
\begin{cases} 
\text{if } s_i \geq f_{\text{prev}} \\
\text{then} \\
\quad \{ \mathcal{B} \leftarrow \mathcal{B} \cup \{ A_i \} \}
\quad \{ \text{prev} \leftarrow i \}
\end{cases}
\text{return } (\mathcal{B})

How to prove this is correct? (I.e., how can we show the returned solution is both feasible and optimal?)

Feasibility? Easy! We always choose an interval that starts after all other chosen intervals.

Optimality? Harder...

Time complexity: \( \in O(n \log n) \)
BRO I DON'T WANT PROOF

I WANT EVIDENCE
Let's demonstrate approach #1

GREEDY CORRECTNESS PROOFS

• Want to prove: greedy solution \( B \) is **correct** (feasible & optimal)
• Typically show **feasibility directly** and **optimality by contradiction**:  
  • Suppose solution \( O \) is better than \( B \)  
  • Show this necessarily leads to a contradiction  
• Two broad strategies for **deriving** this contradiction:

  1. **Greedy stays ahead**: show every choice in \( B \) is  
     “at least as good” as the corresponding choice in \( O \)  
  2. **Exchange**: show \( O \) can be improved by replacing some  
     choice in \( O \) with a choice in \( B \)
We give an induction proof.
Let $\mathcal{B}$ be the greedy solution,

$$\mathcal{B} = (A_{i_1}, \ldots, A_{i_k}),$$

where $i_1 < \cdots < i_k$.
Let $\mathcal{O}$ be any optimal solution,

$$\mathcal{O} = (A_{j_1}, \ldots, A_{j_\ell}),$$

where $j_1 < \cdots < j_\ell$.

Observe that $\ell \geq k$ since $\mathcal{O}$ is optimal.
We want to prove that $\ell = k$.

I.e., $B$ is a subsequence of the sorted intervals

And so is $\mathcal{O}$

Optimal must include at least as many intervals as greedy
Lemma 4.2 (Greedy stays ahead)

- \( f_{i_m} \leq f_{j_m} \) for \( m = 1, 2, \ldots \)

Proof.

Initial case \( m = 1 \). We have \( f_{i_1} \leq f_{j_1} \) since the greedy algorithm begins by choosing \( i_1 = 1 \). \( (A_1 \) has the earliest finishing time.\)

Induction assumption: \( f_{i_{m-1}} \leq f_{j_{m-1}} \). Consider \( A_{i_m} \) and \( A_{j_m} \). We have

\[
\begin{align*}
  s_{j_m} &\geq f_{j_{m-1}} \geq f_{i_{m-1}}. \\
  \leq f_{j_{m-1}} &\quad \text{(by I.H.)} \\
  \geq f_{j_{m-1}} &\quad \text{(since optimal solution is feasible, so intervals are disjoint)}
\end{align*}
\]

\( A_{i_m} \) has the earliest finishing time of any interval that starts after \( f_{i_{m-1}} \) finishes. Therefore \( f_{i_m} \leq f_{j_m} \). \( \square \)
Correctness Proof (cont.)

Recall

Greedy solution is \( B = (A_{i_1}, \ldots, A_{i_k}) \).
Optimal solution is \( O = (A_{j_1}, \ldots, A_{j_\ell}) \).

Now we complete the proof.

From the Lemma, we have \( f_{i_k} \leq f_{j_k} \).

Suppose that \( \ell > k \).

(to obtain a contradiction)

This completes the proof!
A DIFFERENT PROOF

“Slick” ad-hoc approaches are sometimes possible...
Let \( F = \{f_{i_1}, \ldots, f_{i_k}\} \) be the finishing times of the intervals in \( B \).

No interval finishes strictly to the left.
No interval starts strictly to the right.

No interval in is strictly between these points!

So, in addition to the intervals in \( B \), only the following types of intervals are possible:

- Contains \( f_{i_1} \)
- Contains \( f_{i_2} \)
- Contains \( f_{i_1} \) and \( f_{i_2} \)

Thus, every interval contains some finishing time in \( F \).
And, two intervals in \( O \) cannot contain the same element of \( F \).

So, there must be as many finishing times in \( F \) as there are intervals in \( O \). QED.
**PROBLEM: INTERVAL COLOURING**

**Instance:** A set $A = \{A_1, \ldots, A_n\}$ of intervals.
For $1 \leq i \leq n$, $A_i = [s_i, f_i)$, where $s_i$ is the start time of interval $A_i$ and $f_i$ is the finish time of $A_i$.

**Feasible solution:** A $c$-colouring is a mapping $\text{col} : A \rightarrow \{1, \ldots, c\}$ that assigns each interval a colour such that two intervals receiving the same colour are always disjoint.

**Find:** A $c$-colouring of $A$ with the minimum number of colours.

---

**Example**

- 7 intervals, 7 colours.
- Feasible, but not optimal.
MORE EXAMPLES

Example 1: 7 intervals, 6 colours. Feasible, but not optimal.
- Same color, but not disjoint...

Example 2: 7 intervals, 6 colours. Feasible, but not optimal.
- Same color, but disjoint. OK!

Example 3: 7 intervals, 2 colours. Optimal.
Greedy Strategies for Interval Colouring

As usual, we consider the intervals one at a time.

At a given point in time, suppose we have coloured the first \( i < n \) intervals using \( d \) colours.

We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \( d \) colours, then we introduce a **new colour** and \( d \) is increased by 1.

**Question:** In **what order** should we consider the intervals?
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE: ORDER MATTERS!**

Consider intervals in the order they are given in the input: \(A_1 \ldots A_{10}\)
We will colour the \((i + 1)\)st interval with \textbf{any permissible colour}. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a \textbf{new colour} and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE: ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**Example:**

**Order Matters!**
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

| \(A_1\) | 1 |
| \(A_2\) |   |
| \(A_3\) |   |
| \(A_4\) | 2 |
| \(A_5\) | 1 |
| \(A_6\) | 3 |
| \(A_7\) | 2 |
| \(A_8\) | 4 |
| \(A_9\) |   |
| \(A_{10}\) |   |

**EXAMPLE:**

**ORDER MATTERS!**

\[ x\)-axis\]
We will colour the \((i + 1)\text{st}\) interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**

Used **4 colours**

Can we do better?
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

### EXAMPLE: ORDER MATTERS!

Pre-sort intervals by increasing start time!
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**Example:**

**Order Matters!**

Pre-sort intervals by increasing start time!
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE: ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with **any permissible colour**. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE:**  
**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a **new colour** and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

ORDER MATTERS!
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

ORDER MATTERS!
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with \textbf{any permissible colour}. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a \textbf{new colour} and \(d\) is increased by 1.

**EXAMPLE:**

\textbf{ORDER MATTERS!}
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**EXAMPLE:**

**ORDER MATTERS!**
We will colour the \((i + 1)\)st interval with any permissible colour. If it cannot be coloured using any of the existing \(d\) colours, then we introduce a new colour and \(d\) is increased by 1.

**Example:**

ORDER MATTERS!

Used 3 colours

Can we do better?
Algorithm: GreedyIntervalColouring(A)

sort the intervals so that $s_1 \leq \cdots \leq s_n$
\[ d \leftarrow 1 \]
\[ \text{colour}[1] \leftarrow 1 \]
\[ \text{finish}[1] \leftarrow f_1 \]

for $i \leftarrow 2$ to $n$
\begin{align*}
\text{flag} &\leftarrow \text{false} \\
\text{c} &\leftarrow 1 \\
\text{while } c \leq d \text{ and } (\text{not } \text{flag}) \text{ do} \\
\quad \text{if } \text{finish}[c] \leq s_i \text{ then} \\
\quad \quad \{ \text{colour}[i] \leftarrow c \\
\quad \quad \quad \text{finish}[c] \leftarrow f_i \\
\quad \quad \quad \text{flag} \leftarrow \text{true} \\
\quad \quad \text{else } c \leftarrow c + 1 \\
\text{if not } \text{flag} \text{ then} \\
\quad d \leftarrow d + 1 \\
\quad \text{colour}[i] \leftarrow d \\
\quad \text{finish}[d] \leftarrow f_i \\
\end{align*}

return $(d, \text{colour})$

\[ \text{finish}[c] \text{ = finish time of last interval to receive colour } c \]

Consider interval $A_i = (s_i, f_i)$. If $s_i \geq \text{finish}[c]$, then we can give $A_i$ colour $c$ without breaking feasibility.

If we didn’t reuse a colour, use a **new colour**

Check if we can use any colour $c$ in $1..d$
EXAMPLE: RUNNING GREEDY

Initial state
EXAMPLE: RUNNING GREEDY

Code **before** the loop: just assign colour 1

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
<th>A9</th>
<th>A10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Code at the end of the loop: assign finish[1] = d

\( i = 1 \)
\( d = 1 \)
\( \text{finish}[1] = \)
While loop over $c$. Check if we can reuse a color in $1..d$

Is $\text{finish}[1] \leq s_2$?
No. We cannot reuse color 1.

Cannot reuse any colour. Create a new one!

EXAMPLE: RUNNING GREEDY
Is $\text{finish}[1] \leq s_2$?

No. We cannot reuse colour 1.

Cannot reuse any colour. Create a new one!

While loop over $c$. Check if we can reuse a color in $1..d$

EXAMPLE: RUNNING GREEDY
EXAMPLE: RUNNING GREEDY

While loop over \( c \). Check if we can reuse a color in \( 1..d \).

- \( i = 3 \)
- \( d = 2 \)
- \( \text{finish}[1] = \) pattern
- \( \text{finish}[2] = \) pattern

<table>
<thead>
<tr>
<th>A_1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_2</td>
<td>2</td>
</tr>
<tr>
<td>A_3</td>
<td></td>
</tr>
<tr>
<td>A_4</td>
<td></td>
</tr>
<tr>
<td>A_5</td>
<td></td>
</tr>
<tr>
<td>A_6</td>
<td></td>
</tr>
<tr>
<td>A_7</td>
<td></td>
</tr>
<tr>
<td>A_8</td>
<td></td>
</tr>
<tr>
<td>A_9</td>
<td></td>
</tr>
<tr>
<td>A_{10}</td>
<td></td>
</tr>
</tbody>
</table>

\( \text{Is finish}[1] \leq s_3? \)
- No. We cannot reuse colour 1.

\( \text{Is finish}[2] \leq s_3? \)
- No. We cannot reuse colour 2.

Cannot reuse any colour. Create new one.
### Example: Running Greedy

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

While loop over $c$. Check if we can reuse a color in $1..d$

- $i=3$
- $d=3$

- Is $\text{finish}[1] \leq s_3$?
  - No. We cannot reuse colour 1.

- Is $\text{finish}[2] \leq s_3$?
  - No. We cannot reuse colour 2.

- Cannot reuse any colour. Create new one.
While loop over c. Check if we can reuse a color in 1..d.

EXAMPLE: RUNNING GREEDY

Is \( \text{finish}[1] \leq s_4 \)?
Yes. We can reuse colour 1.
While loop over $c$. Check if we can reuse a color in $1..d$

**EXAMPLE:**

**RUNNING GREEDY**

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
<th>$A_9$</th>
<th>$A_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is $\text{finish}[1] \leq s_4$?

Yes. We can reuse colour 1.
While loop over \( c \).
Check if we can reuse a color in \( 1..d \).

**EXAMPLE: RUNNING GREEDY**

<table>
<thead>
<tr>
<th>( A )</th>
<th>( 1 )</th>
<th>( 2 )</th>
<th>( 3 )</th>
<th>( 4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_4 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_6 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_7 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_9 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_{10} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( i = 5 \)
- \( d = 3 \)
- \( \text{finish}[1] = \)
- \( \text{finish}[2] = \)
- \( \text{finish}[3] = \)

Is \( \text{finish}[1] \leq s_5 \)?
No. We **cannot** reuse colour 1.

Is \( \text{finish}[2] \leq s_5 \)?
No. We **cannot** reuse colour 2.

Is \( \text{finish}[3] \leq s_5 \)?
Yes. We **can** reuse colour 3.
EXAMPLE: RUNNING GREEDY

While loop over $c$. Check if we can reuse a color in $1..d$

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
<th>$A_9$</th>
<th>$A_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>finish[1]=</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>finish[2]=</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>finish[3]=</td>
</tr>
</tbody>
</table>

Is $\text{finish}[1] \leq s_5$?
No. We cannot reuse colour 1.

Is $\text{finish}[2] \leq s_5$?
No. We cannot reuse colour 2.

Is $\text{finish}[3] \leq s_5$?
Yes. We can reuse colour 3.
EXAMPLE: RUNNING GREEDY

While loop over c. Check if we can reuse a color in 1..d.

\[
\text{Is } f_{\text{finish}[1]} \leq s_6? \\
\text{No. We cannot reuse colour 1.}
\]

\[
\text{Is } f_{\text{finish}[2]} \leq s_6? \\
\text{Yes. We can reuse colour 2.}
\]
While loop over $c$. Check if we can reuse a color in $1..d$

**EXAMPLE:**
**RUNNING GREEDY**

Is $f_{\text{finish}}[1] \leq s_6$?

No. We **cannot** reuse colour 1.

Is $f_{\text{finish}}[2] \leq s_6$?

Yes. We **can** reuse colour 2.

And so on, and so forth…