Goal: choose as many disjoint intervals as possible, (i.e., without any overlap)

Algorithm:
1. 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 \( f_i \)).

**OPTIMALITY PROOF**

Consider an input \( A[1..n] \)
- Let \( G \) be the greedy solution
- Let \( O \) be an optimal solution
- "Greedy stays ahead" argument
  - Intuition: out of the given set of intervals, greedy picks as many as optimal

**VISUAL EXAMPLE**

How to compare \( G \) and \( O \)?
Imagine reordering \( O \) to match \( G \)!

**CRUCIAL:** We are NOT assuming the optimal algorithm uses the same sort order.
We are merely imagining reordering the intervals chosen by the optimal algorithm so we can easily compare their finish times to intervals in \( G \).
REORDERING O BY INCREASING FINISH TIME

<table>
<thead>
<tr>
<th>O</th>
<th>O₁</th>
<th>O₂</th>
<th>O₃</th>
<th>O₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>O'</td>
<td>O₁'</td>
<td>O₂'</td>
<td>O₃'</td>
<td>O₄'</td>
</tr>
</tbody>
</table>

Now O' and G are both ordered by increasing finish time.
This ordering helps us leverage what we know about G in our comparison with O'.
Argue for a prefix of the intervals sorted this way, G chooses as many as O'.

COMPARING O' WITH G

Looks like $f(G₁) \leq f(O₁')$ and $f(G₂) \leq f(O₂')$ ... and $f(Gᵢ) \leq f(Oᵢ')$ for all i?
If this trend holds in general, then out of the intervals with finish time $\leq f(Oᵢ')$, G chooses as many intervals as O'.

PROVING LEMMA: $f(Gᵢ) \leq f(Oᵢ')$ FOR ALL i

<table>
<thead>
<tr>
<th>O'</th>
<th>O₁'</th>
<th>O₂'</th>
<th>O₃'</th>
<th>O₄'</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>O₁</td>
<td>O₂</td>
<td>O₃</td>
<td>O₄</td>
</tr>
</tbody>
</table>

Base case: $f(G₁) \leq f(O₁')$ since G chooses the interval with the earliest finish time first.

PROVING LEMMA: $f(Gᵢ) \leq f(Oᵢ')$ FOR ALL i

<table>
<thead>
<tr>
<th>O'</th>
<th>O₁'</th>
<th>O₂'</th>
<th>O₃'</th>
<th>O₄'</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>O₁</td>
<td>O₂</td>
<td>O₃</td>
<td>O₄</td>
</tr>
</tbody>
</table>

Inductive step: assume $f(Gᵢ₋₁) \leq f(Oᵢ₋₁')$. Show $f(Gᵢ) \leq f(Oᵢ')$.
- Since O' is feasible, $f(Oᵢ₋₁') \leq s(Oᵢ')$
- So $f(Gᵢ₋₁) \leq s(Oᵢ')$
- So G can choose $Oᵢ'$ if it has the smallest finish time
- So $f(Gᵢ) \leq f(Oᵢ')$

USING THIS LEMMA

<table>
<thead>
<tr>
<th>O'</th>
<th>O₁'</th>
<th>O₂'</th>
<th>O₃'</th>
<th>O₄'</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>O₁</td>
<td>O₂</td>
<td>O₃</td>
<td>O₄</td>
</tr>
</tbody>
</table>

- Suppose $|O'| > |G|$ to obtain a contradiction
  - So if G chooses k intervals, O' chooses at least k + 1
By the lemma, $f(Gₖ) \leq f(Oₖ)$
Since O' is feasible, $f(Oₖ') \leq s(Oₖ₊₁')$
But then G can, and would, pick $Oₖ₊₁'$.
- Contradiction!

So $G$ is optimal.

A DIFFERENT PROOF

"Slick" ad-hoc approaches are sometimes possible...

2023-09-25
Let \( F = \{ f_1, \ldots, f_n \} \) be the finishing times of the intervals in \( X \).

No interval finishes strictly to the left

\[ f_{i_1} \quad \text{greedy} \quad f_{i_2} \quad \ldots \quad f_{i_n} \]

No interval starts strictly to the right

\[ f_{i_1} \quad \text{would be chosen by greedy (contradiction)} \quad f_{i_2} \quad \ldots \quad f_{i_n} \]

So, in addition to the intervals in \( X \), 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 \( X \).

QED

POSSIBLE GREEDY STRATEGIES FOR KNAPSACK PROBLEMS

**Strategy 1:** consider items in decreasing order of profit

(i.e., we maximize the local evaluation criterion \( p_i \))

Let’s try an example input

- Profits \( P = [20, 50, 100] \)
- Weights \( W = [10, 20, 100] \)
- Weight limit \( M = 10 \)

Algorithm selects last item for 100 profit

Looks optimal in this example
POSSIBLE GREEDY STRATEGIES FOR KNAPSACK PROBLEMS

Strategy 2: consider items in increasing order of weight [i.e., we minimize the local evaluation criterion \(w_i\)]

Counterexample
- Profits \(P = [20,50,100]\)
- Weights \(W = [10,20,10]\)
- Weight limit \(M = 10\)
Algorithm selects first item for 20 profit
- It could select half of second item, for 25 profit!

POSSIBLE GREEDY STRATEGIES FOR KNAPSACK PROBLEMS

Strategy 3: consider items in decreasing order of weight divided by profit [i.e., we maximize local evaluation criterion \(p_i/w_i\)]

Let’s try our first example input
- Profits \(P = [20,50,100]\)
- Weights \(W = [10,20,10]\)
- Weight limit \(M = 10\)
Profit divided by weight
- \(P/W = [2,2.5,0]\)
Algorithm selects last item for 100 profit (optimal)

POSSIBLE GREEDY STRATEGIES FOR KNAPSACK PROBLEMS

Strategy 3: consider items in decreasing order of profit divided by weight [i.e., we maximize local evaluation criterion \(p_i/w_i\)]

Let’s try our second example input
- Profits \(P = [20,50,100]\)
- Weights \(W = [10,20,100]\)
- Weight limit \(M = 10\)
Profit divided by weight
- \(P/W = [2,2.5,1]\)
Algorithm selects second item for 25 profit (optimal)

INFORMAL FEASIBILITY ARGUMENT
(SHOULD BE GOOD ENOUGH TO SHOW FEASIBILITY ON ASSESSMENTS)

Feasibility: all \(x_i\) are in \([0,1]\) and total weight is \(\leq M\)

Either everything fits in the knapsack, or:
- When we exit the loop, weight is exactly \(M\)
- Every time we write to \(x_i\) it’s either 0, 1 or \((M – \text{weight})/w_i\) where \(\text{weight} + w_i > M\)

Rearranging the latter we get \((M – \text{weight})/w_i < 1\)

And \(\text{weight} \leq M\), so \((M – \text{weight})/w_i \geq 0\)
So, we have \(x_i \in [0,1]\)
MINOR MODIFICATION TO FACILITATE FORMAL PROOF

Pocket Change (p.1

Weight = 0

for i = 1 to n

if [Weight + w[i] = M then

Weight = M

else

X[i] = 1

Weight = Weight + w[i]

break

end if

end for

return X

Optional slide, just for your notes

Does NOT change behaviour of the algorithm at all

FORAL PROOF

In the invariant, we have for all

Case 1: weight + wi \leq M

Case 2: weight + wi > M

WTP: \text{weight} = \sum_i w_i x_i \leq M

and weight = \sum_i w_i x_i \leq M

Base case. Initially weight = 0 and \text{weight} = 0.

- So weight = \sum_i w_i \cdot 0 = \sum_i w_i x_i = M

- Inductive step.

Suppose invariant holds at start of iteration i.

- Let weight', x' denote values of weight, x_i at end of iteration i.

- Prove invariant holds at end of iteration i.

- i.e., \text{weight} = \sum_i w_i x_i \leq M

Optimal Weight and Weight.

OPTIMALITY – AN EXCHANGE ARGUMENT

For simplicity, assume that the profit / weight ratios are all distinct, so

\frac{p_1}{w_1} > \frac{p_2}{w_2} > \ldots > \frac{p_n}{w_n}

Suppose the greedy solution is \(X = (x_1, \ldots, x_n)\) and the optimal solution

in \(Y = (y_1, \ldots, y_n)\).

We will prove that \(X = Y\), i.e., \(x_j = y_j\) for \(j = 1, \ldots, n\). Therefore there

is a unique optimal solution and it is equal to the greedy solution.

Suppose \(X \neq Y\). To obtain a contradiction.

Pick the smallest integer \(j\) such that \(x_j \neq y_j\). If \(X\) and \(Y\) are identical up to \(x_j\) and \(y_j\), respectively.
What's the relationship between $x_j$ and $y_j$?

Can we have $y_j > x_j$?

No! Greedy would take more of item $j$ if it could.

Since item $j$ is worth more per unit weight, replacing even a tiny amount of items with item $j$ will improve the solution.

So, we remove an infinitesimal $\delta > 0$ of weight of item $k$, and add $\delta$ weight of item $j$.
**Feasibility of** $Y'$

To show $Y'$ is feasible, we show $y'_j \geq 0, y'_j \leq 1$ and $\text{weight}(Y') \leq M$

Let's show $y'_k \geq 0$

- By definition, $y'_k = y_k - \frac{\delta}{w_k}$
- So, $y'_k \geq 0 \iff y_k - \frac{\delta}{w_k} \geq 0 \iff y_k \geq \frac{\delta}{w_k}$
- And we know $y_k$ and $w_k$ are both positive
- So, this constrains $\delta$ to be smaller than this positive number
- Therefore, it is possible to choose positive $\delta$ s.t. $y'_k \geq 0$

Existence proof: but a non-constructive one

**Feasibility of** $Y''$

To show $Y''$ is feasible, we show $y''_j \geq 0, y''_j \leq 1$ and $\text{weight}(Y'') \leq M$

Now let's show $y''_j \leq 1$

- By definition, $y''_j = y_j + \frac{\delta}{w_j}$
- So, $y''_j \leq 1 \iff y_j + \frac{\delta}{w_j} \leq 1 \iff (1 - y_j)w_j \geq \delta$
- Recall $y_j < 1$, so $y_j < 1$, which means $(1 - y_j) > 0$
- So, this constrains $\delta$ to be smaller than some positive number

**Superiority of** $Y''$

Finally, we compute $\text{profit}(Y'')$

\[
\text{profit}(Y'') = \text{profit}(Y') + \sum_k \left[ \frac{y''_k - y'_k}{w_k} \right] \frac{\delta}{w_k} - \sum_k \frac{\delta}{w_k}
\]

- Since $j$ is before $k$, and we consider items with more profit per unit weight first, we have $\frac{\delta}{w_j} > \frac{\delta}{w_k}$
- So, if $\delta > 0$ then $\sum_k \left[ \frac{y''_k - y'_k}{w_k} \right] \frac{\delta}{w_k} > 0$
- Since we can choose $\delta > 0$, we have $\text{profit}(Y'') > \text{profit}(Y')$

The idea is to show that $Y''$ is feasible, and $\text{profit}(Y'') > \text{profit}(Y')$.

This contradicts the optimality of $Y'$ and proves that $X = Y'$.

To show $Y''$ is feasible, we show $y''_j \geq 0, y''_j \leq 1$ and $\text{weight}(Y'') \leq M$
WHAT IF ELEMENTS DON'T HAVE DISTINCT PROFIT/WEIGHT RATIOS?

OPTIMALITY PROOF WITHOUT DISTINCTNESS

- There may be many optimal solutions
  - Key idea: Let $Y$ be an optimal solution that matches $X$ on a maximal number of indices
  - Observe: if $X$ is really optimal, then $Y = X$
  - Suppose not for contradiction
    - We will modify $Y$, preserving its optimality, but making it match $X$ on one more index (a contradiction!)

- Let $\delta(j)$ be the difference in profit-to-weight ratio on index $j$.
  - Remove some $\delta(j)$ such that $\delta(j) = \frac{y(j) - x(j)}{w(j)}$. Let $\delta(j)$ be the same.
  - With the goal of making the solutions equal on index $k$.

- Optimal solution $Y'$
  - Suppose $\delta = w(2x_j - y_j)$.
    - In this case, $\delta = w(2x_j - y_j)$ and $w(3x_j - y_j)$.
      - If $\delta = w(2x_j - y_j)$, we would have $y_j = x_j$.

- Greedy solution $X$
  - Fraction we should add $w(y_j - x_j)$ to make solutions equal on index $j$.
    - Weight to add: $w(y_j - x_j)$.
  - Optimal solution $Y$
    - Let $\delta = \min\{\delta(k) = w(3x_k - y_k), w(x_k - y_k)\}$, $\delta = 0$.
      - Fraction we should add $w(y_k - x_k)$ to make solutions equal on index $k$.
    - Weight to add: $w(y_k - x_k)$.
To show $Y'$ is feasible, we show $\text{weight}(Y') \leq M$ and $y'_k \geq 0, y'_j \leq 1$.

### Weight

- We move $\delta$ weight from item $k$ to item $j$. This does not change the total weight! So $\text{weight}(Y') = \text{weight}(Y) = M$.

### Profit of $Y'$

$\text{profit}(Y') = \text{profit}(Y) + \frac{\delta}{w_j} p_j - \frac{\delta}{w_k} p_k = \text{profit}(Y) + \delta \left( \frac{p_j}{w_j} - \frac{p_k}{w_k} \right)$

Since $j$ is before $k$, and we consider items with more profit per unit weight first, we have $\frac{p_j}{w_j} \geq \frac{p_k}{w_k}$.

Since $\delta > 0$ and $\frac{p_j}{w_j} \geq \frac{p_k}{w_k}$, we have $\delta \left( \frac{p_j}{w_j} - \frac{p_k}{w_k} \right) \geq 0$.

Since $Y$ is optimal, this cannot be positive.

So $Y'$ is a new optimal solution that matches $X$ on one more index than $Y$.

Contradiction: $Y$ matched $X$ on a maximal number of indices.

### Feasibility of $Y'$

- Showing $y'_k \geq 0$
  - By definition, $y'_k = y_k - \frac{\delta}{w_k} \geq 0$ iff $\delta \leq y_k w_k$.
  - But $\delta$ is the minimum of $w_j (y_j - y_j)$ and $w_k (y_k - x_k) \leq w_k y_k$.
  - And $w_k (y_k - x_k) \leq w_k y_k$ so $\delta \leq y_k w_k$.

- Showing $y'_j \leq 1$
  - $y'_j = y_j + \frac{\delta}{w_j} \leq 1$ iff $\frac{\delta}{w_j} \leq 1 - y_j$ iff $\delta \leq w_j (1 - y_j)$ (rearranging).

- $\delta \leq w_j (x_j - y_j)$ (definition of $\delta$).

- And $w_j (x_j - y_j) \leq w_j (1 - y_j)$ (by feasibility of $X$, i.e., $x_j \leq 1$).

### Summarizing Exchange Arguments

- If inputs are distinct
  - So there is a unique optimal solution.
  - Let $O \neq G$ be an optimal solution that beats greedy.
  - Show how to change $O$ to obtain a better solution.

- If not
  - There may be many optimal solutions.
  - Let $O \neq G$ be an optimal solution that matches greedy on as many choices as possible.
  - Show how to change $O$ to obtain an optimal solution $O'$ that matches greedy for even more choices.

### Notes

- I don't think we will have time to get past here.
  - But if we do, great, we can catch up.

---

**Interval Colouring**
PROBLEM: INTERVAL COLOURING

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

Feasible solution: A $c$-colouring is a mapping $c : A \rightarrow \{1, \ldots, c\}$
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

MORE EXAMPLES

Example

57

Not feasible!

58

Example

Same colour, but not disjoint.
OK!

59

Example

Example: ORDER MATTERS!

Consider intervals in the order they are given in the input:
$A_1, A_2, \ldots, A_{10}$

EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!

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 $i$th 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!

EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!

EXAMPLE: ORDER MATTERS!
ORDER MATTERS!

- Used 4 colours
- Can we do better?

- Pre-sort intervals by increasing start time

- Example: Order matters!
EXAMPLE: ORDER MATTERS!
EXAMPLE:
ORDER MATTERS!

Used 3 colours

Can we do better?

Initial state

EXAMPLE:
RUNNING GREEDY

\[ \text{is} \in \{0, \ldots, d-1\} \]

Color before the loop (in grey colour)

Color after the loop (in black colour)
EXAMPLE: RUNNING GREEDY

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

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

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

Is \( \text{finish}[3] \leq s \)?
Yes. We can reuse colour 3.

Is \( \text{finish}[4] \leq s \)?
Yes. We can reuse colour 1.

Is \( \text{finish}[5] \leq s \)?
No. We cannot reuse colour 1.

Is \( \text{finish}[6] \leq s \)?
No. We cannot reuse colour 2.

Is \( \text{finish}[7] \leq s \)?
Yes. We can reuse colour 3.

Is \( \text{finish}[8] \leq s \)?
No. We cannot reuse colour 1.

Is \( \text{finish}[9] \leq s \)?
Yes. We can reuse colour 1.

Is \( \text{finish}[10] \leq s \)?
No. We cannot reuse colour 1.
Let $F_{c}$ be the first interval that has colour $c$.
Let $L_{c}$ be the last interval that has colour $c$ and starts before $F_{c}$ ends.
We prove $F_{c}$ overlaps every interval $L_{c}$ for all $c < 2$.

And so on, and so forth...

Correctness of the Algorithm

The correctness of this greedy algorithm can be proven inductively as well as by a "slick" method—we give the "slick" proof.

Let $D$ denote the number of colours used by the algorithm.

**Theorem:**
Let $F_{c}$ be the first interval that has the last colour $D$.
Let $L_{c}$ be the last interval that has colour $c$ and starts before $F_{c}$ ends.
We prove $F_{c}$ overlaps every interval $L_{c}$ for all $c < 2$.

If $D = 1$ the proof is immediate, so suppose $D ≥ 2$.

**Proof:**

1. We prove $F_{c}$ overlaps every interval $L_{c}$ for all $c < 2$.
2. We prove $F_{c}$ overlaps every interval $L_{c}$ for all $c < 2$.
3. We prove $F_{c}$ overlaps every interval $L_{c}$ for all $c < 2$.
4. We prove $F_{c}$ overlaps every interval $L_{c}$ for all $c < 2$.

**TIME COMPLEXITY?**

Total $O(n \log n + md)$

Could be $O(n \log n)$ if only a constant number of colours are needed

Could be $O(n^2)$ if all colours are used

Most accurate complexity statement is $O(n \log n + m)$ where $D$ is # colours used.

What inefficiencies exist in this algorithm?
Could we make it faster with clever data structure usage?
IMPROVING THIS ALGORITHM

Current greedy algorithm:
- For each interval $A_i$, compare its start time $s_i$ with the $finish[c]$ times of all colours introduced so-far
- Why? Looking for some $finish[c]$ time that is earlier than $s_i$
- We are doing linear search... Can we do better?
Use a priority queue to keep track of the earliest $finish[c]$ at all times in the algorithm
- Then we only need to look at minimum element

EXAMPLE: HEAP-BASED ALGORITHM

Iteration i=1
Check heap minimum
Empty, so a new colour is needed

Min element: NULL
Heap

Iteration i=2
Check heap minimum
Check if finish time 3 is before $s_2$
No. New colour!

Min element: finish of time 3
Heap

Iteration i=3
Check heap minimum
Check if finish time 3 is before $s_3$
No. New colour!

Min element: finish of time 3
Heap

Iteration i=4
Check heap minimum
Check if finish time 3 is before $s_4$
No. New colour!

Min element: finish of time 3
Heap
EXAMPLE: HEAP-BASED ALGORITHM

Min element: time of task 3
Heap: finish of tasks 4 and 5

Iteration i = 3
Check heap minimum: No. New colour!

Check if finish time 3 is before $s_3$

Iteration i = 4
Check heap minimum: Yes. Reuse colour, delete and insert new finish time into heap!

Iteration i = 5
Check heap minimum: Yes. Reuse colour, delete and insert new finish time into heap!

Iteration i = 6
Check heap minimum: No. New colour!

Iteration i = 7
Check heap minimum: No. New colour!

Iteration i = 8
Check heap minimum: No. New colour!

Iteration i = 9
Check heap minimum: No. New colour!

Iteration i = 10
Check heap minimum: No. New colour!
EXAMPLE: HEAP-BASED ALGORITHM

Min element: time 2

Heap: finish at time 9

Check heap minimum

Min element: NULL

Check if finish time 5 is before \( s_5 \)

Iteration \( i = 5 \)

Finish at time 7

Finish at time 9

Yes. Reuse colour, deleteMin and insert new finish time into heap!

Finish at time 11

And so on, and so forth...

O(1)

O(\log D)

O(\log S)

where \( S = \text{size(priority queue)} \)

Preprocess(A[1..n])

Sort A by increasing start time

let \( s_i \) be the start times in \( A \)

For i = 1 to n

if \( s_i < \text{startMin} \)

else

\( \text{colour}(1) = \text{d} \)

\( \text{h}.\text{insert}(\text{f1}, \text{colour}(1)) \)

\( \text{h}.\text{deleteMin}() \)

\( \text{h}.\text{insert}(\text{f1}, \text{colour}(1)) \)

\( \text{h}.\text{deleteMin}() \)

\( \text{h}.\text{insert}(\text{f1}, \text{colour}(1)) \)

\( \text{h}.\text{deleteMin}() \)

\( \text{h}.\text{insert}(\text{f1}, \text{colour}(1)) \)

\( \text{h}.\text{deleteMin}() \)

return d

\( \Theta n \log n + \Theta(n \log D) \)

\( \Theta(n \log n) \) since \( n \geq D \), \( D(\log n) \)