Greedy Algorithms

Lecturer: Kelin Luo
Department of Computer Science and Engineering
University at Buffalo
Def. In an optimization problem, our goal of is to find a valid solution with the minimum cost (or maximum value).
Def. In an optimization problem, our goal is to find a valid solution with the minimum cost (or maximum value).

Trivial Algorithm for an Optimization Problem
Enumerate all valid solutions, compare them and output the best one.
Def. In an **optimization problem**, our goal is to find a valid solution with the minimum cost (or maximum value).

**Trivial Algorithm for an Optimization Problem**

Enumerate all valid solutions, compare them and output the best one.

- However, trivial algorithm often runs in **exponential** time, as the number of potential solutions is often exponentially large.
**Def.** In an *optimization problem*, our goal of is to find a valid solution with the minimum cost (or maximum value).

**Trivial Algorithm for an Optimization Problem**

Enumerate all valid solutions, compare them and output the best one.

- However, trivial algorithm often runs in exponential time, as the number of potential solutions is often exponentially large.
Def. In an optimization problem, our goal of is to find a valid solution with the minimum cost (or maximum value).

Trivial Algorithm for an Optimization Problem

Enumerate all valid solutions, compare them and output the best one.

- However, trivial algorithm often runs in exponential time, as the number of potential solutions is often exponentially large.
- $f(n)$ is a polynomial if $f(n) = O(n^k)$ for some constant $k > 0$.
Def. In an optimization problem, our goal of is to find a valid solution with the minimum cost (or maximum value).

Trivial Algorithm for an Optimization Problem
Enumerate all valid solutions, compare them and output the best one.

- However, trivial algorithm often runs in exponential time, as the number of potential solutions is often exponentially large.
- $f(n)$ is a polynomial if $f(n) = O(n^k)$ for some constant $k > 0$.
- convention: polynomial time = efficient
**Def.** In an optimization problem, our goal is to find a valid solution with the minimum cost (or maximum value).

**Trivial Algorithm for an Optimization Problem**

Enumerate all valid solutions, compare them and output the best one.

- However, trivial algorithm often runs in exponential time, as the number of potential solutions is often exponentially large.
- \( f(n) \) is a polynomial if \( f(n) = O(n^k) \) for some constant \( k > 0 \).
- convention: polynomial time = efficient

**Goals of algorithm design**
Def. In an optimization problem, our goal of is to find a valid solution with the minimum cost (or maximum value).

Trivial Algorithm for an Optimization Problem
Enumerate all valid solutions, compare them and output the best one.

- However, trivial algorithm often runs in exponential time, as the number of potential solutions is often exponentially large.
- $f(n)$ is a polynomial if $f(n) = O(n^k)$ for some constant $k > 0$.
- Convention: polynomial time = efficient

Goals of algorithm design
1. Design efficient algorithms to solve problems
Def. In an optimization problem, our goal is to find a valid solution with the minimum cost (or maximum value).

Trivial Algorithm for an Optimization Problem
Enumerate all valid solutions, compare them and output the best one.

- However, trivial algorithm often runs in exponential time, as the number of potential solutions is often exponentially large.
- \( f(n) \) is a polynomial if \( f(n) = O(n^k) \) for some constant \( k > 0 \).
- convention: polynomial time = efficient

Goals of algorithm design
1. Design efficient algorithms to solve problems
2. Design more efficient algorithms to solve problems
Common Paradigms for Algorithm Design

- Greedy Algorithms: shortest path problem
- Divide and Conquer: merge-sort, binary search
- Dynamic Programming: shortest path problem, Fibonacci number
Greedy algorithm properties

Greedy algorithms are often for optimization problems. They often run in polynomial time due to their simplicity: easy to come up with, easy to analyze running time. Hard to see correctness. Mostly, it is not correct. E.g.
Greedy algorithm properties

- Greedy algorithms are often for optimization problems.
Greedy algorithm properties

- Greedy algorithms are often for optimization problems.
- They often run in polynomial time due to their simplicity: easy to come up with, easy to analyze running time.
Greedy algorithm properties

- Greedy algorithms are often for optimization problems.
- They often run in polynomial time due to their simplicity: easy to come up with, easy to analyze running time.
- Hard to see correctness. Mostly, it is not correct. E.g. \( \min f(x) \)
## Greedy Algorithm

- Build up the solutions in steps
- At each step, make an **irrevocable** decision using a “reasonable” strategy

---

**Def.** A strategy is **safe** if there is always an optimum solution that agrees with the decision made according to the strategy.
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an **irrevocable** decision using a “reasonable” strategy

Analysis of Greedy Algorithm

- **Safety**: Prove that the reasonable strategy is “safe”
- **Self-reduce**: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an **irrevocable** decision using a “reasonable” strategy

Analysis of Greedy Algorithm

- **Safety**: Prove that the reasonable strategy is “safe” (**key**)
- **Self-reduce**: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem (**usually easy**.)
**Greedy Algorithm**
- Build up the solutions in steps
- At each step, make an **irrevocable** decision using a “reasonable” strategy

**Analysis of Greedy Algorithm**
- **Safety:** Prove that the reasonable strategy is “safe” (*key*)
- **Self-reduce:** Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem (*usually easy*)

**Def.** A strategy is safe: there is always an optimum solution that agrees with the decision made according to the strategy.
Outline

1. Toy Example: Box Packing
2. Interval Scheduling
   - Interval Partitioning
3. Offline Caching
   - Heap: Concrete Data Structure for Priority Queue
4. Data Compression and Huffman Code
5. Summary
6. Exercise Problems
## Box Packing

**Input:** \( n \) boxes of capacities \( c_1, c_2, \cdots, c_n \)

\( m \) items of sizes \( s_1, s_2, \cdots, s_m \)

Can put at most 1 item in a box

Item \( j \) can be put into box \( i \) if \( s_j \leq c_i \)

**Output:** A way to put as many items as possible in the boxes.

---

### Example:

- Box capacities: 60, 40, 25, 17, 12
- Item sizes: 45, 41, 20, 19, 16
Box Packing

**Input:** $n$ boxes of capacities $c_1, c_2, \cdots, c_n$

$m$ items of sizes $s_1, s_2, \cdots, s_m$

Can put at most 1 item in a box

Item $j$ can be put into box $i$ if $s_j \leq c_i$

**Output:** A way to put as many items as possible in the boxes.

**Example:**

- Box capacities: 60, 40, 25, 17, 12
- Item sizes: 45, 41, 20, 19, 16
- Can put 3 items in boxes: 45 $\rightarrow$ 60, 20 $\rightarrow$ 40, 16 $\rightarrow$ 25
Box Packing

**Input:** \( n \) boxes of capacities \( c_1, c_2, \cdots, c_n \)

\( m \) items of sizes \( s_1, s_2, \cdots, s_m \)

Can put at most 1 item in a box

Item \( j \) can be put into box \( i \) if \( s_j \leq c_i \)

**Output:** A way to put as many items as possible in the boxes.

Example:

- Box capacities: 60, 40, 25, 17, 12
- Item sizes: 45, 41, 20, 19, 16
- Can put 3 items in boxes: 45 → 60, 20 → 40, 16 → 25
- Can put 4 items in boxes: 45 → 60, 20 → 40, 19 → 25, 16 → 17
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an \textit{irrevocable} decision using a “reasonable” strategy

Q: Take box 1. Which item should we put in box 1?
A: The item of the largest size that can be put into the box.
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an irretrievable decision using a “reasonable” strategy

Designing a Reasonable Strategy for Box Packing

- Q: Take box 1. Which item should we put in box 1?
**Greedy Algorithm**

- Build up the solutions in steps
- At each step, make an **irrevocable** decision using a “reasonable” strategy

**Designing a Reasonable Strategy for Box Packing**

- Q: Take box 1. Which item should we put in box 1?
- A: The item of the largest size that can be put into the box.
Analysis of Greedy Algorithm

- Safety: Prove that the reasonable strategy is “safe”
- Self-reduce: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem
Analysis of Greedy Algorithm

- **Safety:** Prove that the reasonable strategy is “safe”
- **Self-reduce:** Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

**Lemma** The strategy that put into box 1 the largest item it can hold is “safe”: There is an optimum solution in which box 1 contains the largest item it can hold.
Analysis of Greedy Algorithm

- Safety: Prove that the reasonable strategy is “safe”
- Self-reduce: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

**Lemma** The strategy that put into box 1 the largest item it can hold is “safe”: There is an optimum solution in which box 1 contains the largest item it can hold.

- Intuition: putting the item gives us the easiest residual problem.
Analysis of Greedy Algorithm

- **Safety**: Prove that the reasonable strategy is “safe”
- **Self-reduce**: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

**Lemma** The strategy that put into box 1 the largest item it can hold is “safe”: There is an optimum solution in which box 1 contains the largest item it can hold.

- **Intuition**: putting the item gives us the **easiest residual problem**.
- **formal proof via exchanging argument**:
**Lemma** There is an optimum solution in which box 1 contains the largest item it can hold.
**Lemma**  There is an optimum solution in which box 1 contains the largest item it can hold.

**Proof.**

- Let $j =$ largest item that box 1 can hold.
**Lemma** There is an optimum solution in which box 1 contains the largest item it can hold.

**Proof.**
- Let $j =$ largest item that box 1 can hold.
- Take any optimum solution $S$. If $j$ is put into Box 1 in $S$, done.
**Lemma** There is an optimum solution in which box 1 contains the largest item it can hold.

**Proof.**

- Let $j = \text{largest item that box 1 can hold}$.
- Take any optimum solution $S$. If $j$ is put into Box 1 in $S$, done.
- Otherwise, assume this is what happens in $S$:

\begin{align*}
S: & \quad \text{box 1} \\
& \quad \text{item } j \\
& \quad \cdots \\
\end{align*}
**Lemma** There is an optimum solution in which box 1 contains the largest item it can hold.

**Proof.**

- Let $j =$ largest item that box 1 can hold.
- Take any optimum solution $S$. If $j$ is put into Box 1 in $S$, done.
- Otherwise, assume this is what happens in $S$:

```
box 1
S':

S:  item j'    item j
```

- $s_{j'} \leq s_j$, and swapping gives another solution $S'$
**Lemma** There is an optimum solution in which box 1 contains the largest item it can hold.

**Proof.**

- Let $j = \text{largest item that box 1 can hold.}$
- Take any optimum solution $S$. If $j$ is put into Box 1 in $S$, done.
- Otherwise, assume this is what happens in $S$:

  ![Diagram](image)

  - $s_{j'} \leq s_j$, and swapping gives another solution $S'$
  - $S'$ is also an optimum solution. In $S'$, $j$ is put into Box 1. □
Notice that the exchanging operation is only for the sake of analysis; it is not a part of the algorithm.
Notice that the exchanging operation is only for the sake of analysis; it is not a part of the algorithm.

**Analysis of Greedy Algorithm**

- **Safety**: Prove that the reasonable strategy is “safe”
- **Self-reduce**: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem
Notice that the exchanging operation is only for the sake of analysis; it is not a part of the algorithm.

Analysis of Greedy Algorithm

- **Safety**: Prove that the reasonable strategy is “safe”
- **Self-reduce**: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

- **Trivial**: we decided to put Item $j$ into Box 1, and the remaining instance is obtained by removing Item $j$ and Box 1.
**Generic Greedy Algorithm**

1. **while** the instance is non-trivial **do**
2. make the choice using the greedy strategy
3. reduce the instance

**Greedy Algorithm for Box Packing**

1. \( T \leftarrow \{1, 2, 3, \cdots, m\} \)
2. **for** \( i \leftarrow 1 \) **to** \( n \) **do**
3. **if** some item in \( T \) can be put into box \( i \) **then**
4. \( j \leftarrow \) the largest item in \( T \) that can be put into box \( i \)
5. print(“put item \( j \) in box \( i \)”)
6. \( T \leftarrow T \setminus \{j\} \)
Why “Safety” + “Self-reduce” $$\implies$$ Optimality?

- Let $\text{BP}(B, T)$ denote a box-packing instance.
- $\phi(1, 2, \ldots, m) \mapsto \{1, 2, \ldots, n, \text{NULL}\}$ denote packing strategy. e.g., $\phi(2) = 3$ means item 2 is put into box 3.
- $\text{val}(\phi) :=$ the number of items packed by $\phi$.
- $\phi_g$: the packing strategy obtained by greedy algorithm.

**Proof.**

- **Base case:** When $|B| = 1$ or $|T| = 1$.
- **Inductive case:** (Hypothesis) Assume Greedy alg solves $\text{BP}(B', T')$ optimally for $|B'| = n - 1$ and $|T'| = m - 1$. 
Why “Safety” + “Self-reduce” $\implies$ Optimality?

Proof.

(Induction) Wlog, let $\pi$ be the optimal solution matches our greedy sol on $\text{BP}(B, T)$, saying $\pi(j) = 1$.

By self-reduce: $\text{BP}(B \setminus \{1\}, T \setminus \{j\})$ is a smaller BP instance.

$\pi$ and $\phi_g$ onto $\text{BP}(B \setminus \{1\}, T \setminus \{j\})$, denoted as $\pi'$ and $\phi'_g$.

By Inductive hypothesis, $\phi'_g$ is the optimal sol for $\text{BP}(B \setminus \{1\}, T \setminus \{j\})$.

$\text{val}(\pi) \geq \text{val}(\phi_g) = 1 + \text{val}(\phi'_g) \geq 1 + \text{val}(\pi') = \text{val}(\pi)$.
Generic Greedy Algorithm

1: while the instance is non-trivial do
2: make the choice using the greedy strategy
3: reduce the instance

Greedy Algorithm for Box Packing

1: $T \leftarrow \{1, 2, 3, \cdots, m\}$
2: for $i \leftarrow 1$ to $n$ do
3: if some item in $T$ can be put into box $i$ then
4: $j \leftarrow$ the largest item in $T$ that can be put into box $i$
5: print(“put item $j$ in box $i$”)
6: $T \leftarrow T \setminus \{j\}$
Running time

Generic Greedy Algorithm
1: while the instance is non-trivial do
2: make the choice using the greedy strategy
3: reduce the instance

Greedy Algorithm for Box Packing
1: \( T \leftarrow \{1, 2, 3, \ldots, m\} \)
2: for \( i \leftarrow 1 \) to \( n \) do
3: if some item in \( T \) can be put into box \( i \) then
4: \( j \leftarrow \) the largest item in \( T \) that can be put into box \( i \)
5: print("put item \( j \) in box \( i \)")
6: \( T \leftarrow T \setminus \{j\} \)
### Generic Greedy Algorithm

1. **while** the instance is non-trivial **do**
2. make the choice using the greedy strategy
3. reduce the instance

### Greedy Algorithm for Box Packing

1. \( T \leftarrow \{1, 2, 3, \cdots, m\} \)
2. **for** \( i \leftarrow 1 \) to \( n \) **do**
3. \[ \text{if some item in } T \text{ can be put into box } i \text{ then} \]
4. \[ j \leftarrow \text{the largest item in } T \text{ that can be put into box } i \]
5. \[ \text{print(“put item } j \text{ in box } i”)} \]
6. \[ T \leftarrow T \setminus \{j\} \]

- With sorted item-sizes and box-capacities, running time is \( O(\max\{n, m\}) \).
Generic Greedy Algorithm

1: while the instance is non-trivial do
2: make the choice using the greedy strategy
3: reduce the instance

Lemma  Generic algorithm is correct if and only if the greedy strategy is safe.
Generic Greedy Algorithm

1: while the instance is non-trivial do 
2: make the choice using the greedy strategy 
3: reduce the instance

Lemma  Generic algorithm is correct if and only if the greedy strategy is safe.

- Greedy strategy is safe: we will not miss the optimum solution
Generic Greedy Algorithm

1: while the instance is non-trivial do
2: make the choice using the greedy strategy
3: reduce the instance

Lemma  Generic algorithm is correct if and only if the greedy strategy is safe.

- Greedy strategy is safe: we will not miss the optimum solution
- Greedy strategy is not safe: we will miss the optimum solution for some instance, since the choices we made are irrevocable.
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an irrevocable decision using a “reasonable” strategy
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an *irrevocable* decision using a “reasonable” strategy

Analysis of Greedy Algorithm

- Safety: Prove that the reasonable strategy is “safe”
- Self-reduce: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem
Greedy Algorithm

- Build up the solutions in steps
- At each step, make an **irrevocable** decision using a “reasonable” strategy

Analysis of Greedy Algorithm

- Safety: Prove that the reasonable strategy is “safe”
- Self-reduce: Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem

**Def.** A strategy is “safe” if there is always an optimum solution that is “consistent” with the decision made according to the strategy.
let $S$ be an arbitrary optimum solution.

if $S$ is consistent with the greedy choice, done.

otherwise, show that it can be modified to another optimum solution $S'$ that is consistent with the choice.
let $S$ be an arbitrary optimum solution.
if $S$ is consistent with the greedy choice, done.
otherwise, show that it can be modified to another optimum solution $S'$ that is consistent with the choice.

The procedure is not a part of the algorithm.
Outline

1. Toy Example: Box Packing
2. Interval Scheduling
   - Interval Partitioning
3. Offline Caching
   - Heap: Concrete Data Structure for Priority Queue
4. Data Compression and Huffman Code
5. Summary
6. Exercise Problems
Interval Scheduling

**Input:** \( n \) jobs, job \( i \) with start time \( s_i \) and finish time \( f_i \)

\( i \) and \( j \) are compatible if \([s_i, f_i)\) and \([s_j, f_j)\) are disjoint

**Output:** A maximum-size subset of mutually compatible jobs
Interval Scheduling

**Input:** $n$ jobs, job $i$ with start time $s_i$ and finish time $f_i$

$i$ and $j$ are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ are disjoint

**Output:** A maximum-size subset of mutually compatible jobs
Which of the following strategies are safe?
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size?
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
Greedy Algorithm for Interval Scheduling

Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs?
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!

![Diagram showing interval scheduling with jobs placed at different intervals.]
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
  - Schedule the job with the smallest size? No!
  - Schedule the job conflicting with smallest number of other jobs? No!
  - Schedule the job with the earliest finish time?
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
  - Schedule the job with the smallest size? No!
  - Schedule the job conflicting with smallest number of other jobs? No!
  - Schedule the job with the earliest finish time? Yes!
Greedy Algorithm for Interval Scheduling

Which of the following strategies are safe?

- Schedule the job with the smallest size? No!
- Schedule the job conflicting with smallest number of other jobs? No!
- Schedule the job with the earliest finish time? Yes!

![Diagram showing intervals and scheduling]

0 1 2 3 4 5 6 7 8 9
Greedy Algorithm for Interval Scheduling

- Which of the following strategies are safe?
  - Schedule the job with the smallest size? No!
  - Schedule the job conflicting with smallest number of other jobs? No!
  - Schedule the job with the earliest finish time? Yes!

![Diagram of interval scheduling](image-url)
Lemma  It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

Proof.
Lemma  It is safe to schedule the job \( j \) with the earliest finish time: There is an optimum solution where the job \( j \) with the earliest finish time is scheduled.

Proof.

- Take an arbitrary optimum solution \( S \)

\[ S: \]
Greedy Algorithm for Interval Scheduling

**Lemma**  It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

**Proof.**

- Take an arbitrary optimum solution $S$
- If it contains $j$, done

$S$: [ ] [ ] [ ] [ ]
**Lemma**  It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

**Proof.**

- Take an arbitrary optimum solution $S$
- If it contains $j$, done

$S$: [ ] [ ] [ ] [ ] [ ]

$j$: [ ]
Greedy Algorithm for Interval Scheduling

**Lemma** It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

**Proof.**
- Take an arbitrary optimum solution $S$
- If it contains $j$, done
- Otherwise, replace the first job in $S$ with $j$ to obtain another optimum schedule $S'$.

$S$: 

$j$: 

---

$S$: 

$j$: 

---
**Lemma**  It is safe to schedule the job $j$ with the earliest finish time: There is an optimum solution where the job $j$ with the earliest finish time is scheduled.

**Proof.**
- Take an arbitrary optimum solution $S$
- If it contains $j$, done
- Otherwise, replace the first job in $S$ with $j$ to obtain another optimum schedule $S'$.