CSE 431/531: Algorithm Analysis and Design (Fall 2024)
Greedy Algorithms

Lecturer: Kelin Luo

Department of Computer Science and Engineering
University at Buffalo

Announcements: HW1 Due

- Due: Mon 16 Sep @ 11:59PM
- Late Email submission to Instructor (kelinluo@buffalo.edu) and Head TAs Xiaoyu Zhang (zhang376@buffalo.edu) and Bahadir (ialtun@buffalo@edu): due 18 Sep @ 11:59PM
- Typed submission
- Potential Grading scheme will be released over the weekend.

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 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.

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.

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 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

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 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
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 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

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

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: $45 \to 60, 20 \to 40, 16 \to 25$

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: $45 \to 60, 20 \to 40, 16 \to 25$
- Can put 4 items in boxes: $45 \to 60, 20 \to 40, 19 \to 25, 16 \to 17$

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

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.

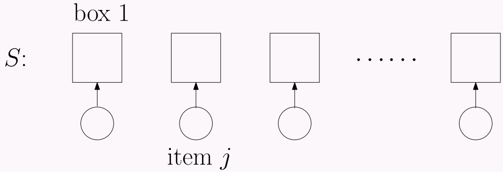
Proof.

- 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.

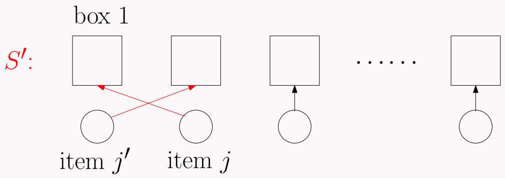
- Let $j =$ largest item that box 1 can hold.
- Take any optimum solution S. If j is put into Box 1 in S, done.
- Otherwise, assume this is what happens in S:



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:

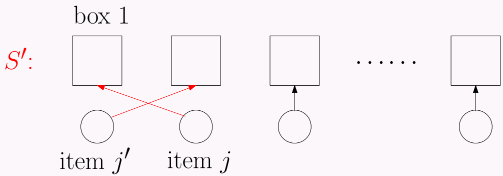


- $s_{j'} \leq s_j$, and swapping gives another solution S'

Lemma There is an optimum solution in which box 1 contains the largest item it can hold.

Proof.

- Let $j =$ 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:



- $s_{j'} \leq s_j$, and swapping gives another solution S'
- S' is also an optimum solution. In S', j is put into Box 1. $\square$

- 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

while the instance is non-trivial **do**
    make the choice using the greedy strategy
    reduce the instance

Greedy Algorithm for Box Packing

$T \leftarrow \{1, 2, 3, \cdots, m\}$
for $i \leftarrow 1$ to n **do**
    if some item in T can be put into box i **then**
        $j \leftarrow$ the largest item in T that can be put into box i
        print("put item j in box i")
        $T \leftarrow T \setminus \{j\}$

Running time

Generic Greedy Algorithm

while the instance is non-trivial **do**
  make the choice using the greedy strategy
  reduce the instance

Greedy Algorithm for Box Packing

$T \leftarrow \{1, 2, 3, \cdots, m\}$
for $i \leftarrow 1$ to n **do**
  if some item in T can be put into box i **then**
    $j \leftarrow$ the largest item in T that can be put into box i
    print("put item j in box i")
    $T \leftarrow T \setminus \{j\}$

Running time

Generic Greedy Algorithm

```
while the instance is non-trivial do
    make the choice using the greedy strategy
    reduce the instance
```

Greedy Algorithm for Box Packing

$T \leftarrow \{1, 2, 3, \cdots, m\}$
for $i \leftarrow 1$ to n **do**
    if some item in T can be put into box i **then**
        $j \leftarrow$ the largest item in T that can be put into box i
        print("put item j in box i")
        $T \leftarrow T \setminus \{j\}$

Running time

Generic Greedy Algorithm

while the instance is non-trivial **do**
    make the choice using the greedy strategy
    reduce the instance

Greedy Algorithm for Box Packing

$T \leftarrow \{1, 2, 3, \cdots, m\}$
for $i \leftarrow 1$ to n **do**
    if some item in T can be put into box i **then**
        $j \leftarrow$ the largest item in T that can be put into box i
        print("put item j in box i")
        $T \leftarrow T \setminus \{j\}$

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