CS141: Intermediate Data Structures and Algorithms

Greedy Algorithms

Yan Gu

How to be greedy?

- **Only care about the immediate reward for any decision make!**
- **I have a few homework assignments to do, which one should I start first?**
  - (For simplicity, we assume you can always get full score using a certain time)
  - A. Work on the one with the earliest deadline!
  - B. Work on the one that worth the highest points!
  - C. Work on the easiest one that requires the least time!
  - D. Work on the hardest one that requires the most time!
  - Etc.

- **Which one do you like most?**

How to be greedy?

- **Today is Oct 5th.**

141 programming II Due Oct 13th 2 points Takes 5 days	141 written II Due Oct 20th 5 points Takes 8 days
Homework in course A Due Oct 18th 3 points Takes 4 days	Homework in course B Due Oct 8th 1 points Takes 3 days

(If there's another set of assignments with deadline/points, the performance of each greedy strategy may be different.)

A. Deadline first: 6pts in total
Oct 5 - 7: HW in course B, 1pt!
Oct 8 - 12: 141 programming II, 2pts!
Oct 13 - 16: HW in course A, 3pts!
(missed the deadline of 141 written II on 20th)

B. Highest score first: 8pts in total
Oct 5 - 12: 141 written II, 5pts!
(missed the deadline of HW in course B on 8th)
(missed the deadline of 141 programming II on 13th)
Oct 13 - 16: HW in course A, 3 pts!

C. Shortest first: 4pts in total
Oct 5 - 7: HW in course B, 1pt!
Oct 8 - 11: HW in course A, 3 pts!
(missed the deadline of 141 programming II on 13th)
(missed the deadline of 141 written II on 20th)

D. Longest first: 8pts in total
Oct 5 - 12: 141 written II, 5pts!
(missed the deadline of HW in course B on 8th)
(missed the deadline of 141 programming II on 13th)
Oct 13 - 16: HW in course A, 3pts!

How to be greedy?

- **Today is Oct 5th.**

141 programming II Due Oct 13th 2 points Takes 5 days	141 written II Due Oct 20th 5 points Takes 8 days
Homework in course A Due Oct 18th 3 points Takes 4 days	Homework in course B Due Oct 8th 1 points Takes 3 days

A. Deadline first: 6pts in total

B. Highest score first: 8pts in total

C. Shortest first: 4pts in total

D. Longest first: 8pts in total

One better solution than all above: 9pts in total
Oct 5 - 7: HW in course B, 1pt!
Oct 8 - 11: HW in course B, 3pts!
(missed the deadline of 141 programming II on 13th)
Oct 12 - 19: 141 written II, 5pts!

(If there's another set of assignments with deadline/points, the performance of each greedy strategy may be different.)

Optimization Problems

- **A class of problems in which we are asked to**
  - Find a **set** (or a **sequence**) of “**items**”
  - That satisfy some **constraints** and simultaneously optimize (i.e., **maximize** or **minimize**) some **objective function**

- **A sequence of tasks with workload/deadline/reward, maximize reward while finish before deadline**
  - Items: tasks; constraints: finish before deadline; optimize: total reward
- **A set of products with weight/value, put into a bag of a certain weight limit and, maximize value**
  - Items: products; constraints: weight limit; optimize: total value
- **A file in computer, encode/compress it to minimize the length**
  - Items: codes; constrains: original file recoverable; optimize: code length
- **Shortest-paths, minimum spanning tree, etc.**

Being greedy?

- **Only care about the immediate reward!**
  - When making a decision, always choose the “best” based on a certain criterion
- **May lose the overall earnings in a long-term…**
  - Conclusion: Plan ahead when you work on homework assignments!
  - (and don’t give up programming assignments of 141)
  - Greedy solution is not necessarily optimal!
- **Sometimes greedy may also be good enough?**
  - When you can prove it!

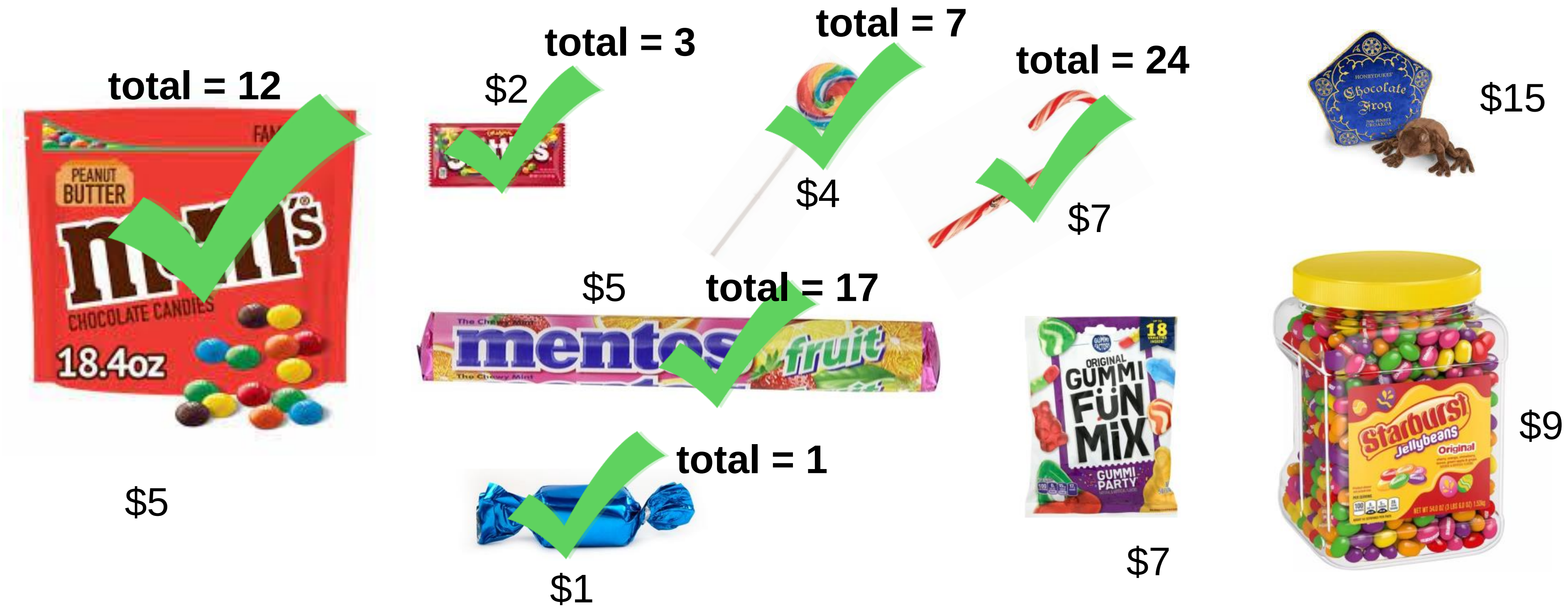
Example: Buying Gifts

Buying gifts

- Yihan is going to buy candies for 141 students
- Her budget is s dollars
- There are n candies in store, with price $p[i]$ each
- She wants all candies to be different
- She wants to buy as many candies as possible

Buying gifts

- Lowest price first!
- Consider the budget $s = 30$
- Can buy 6 items in total

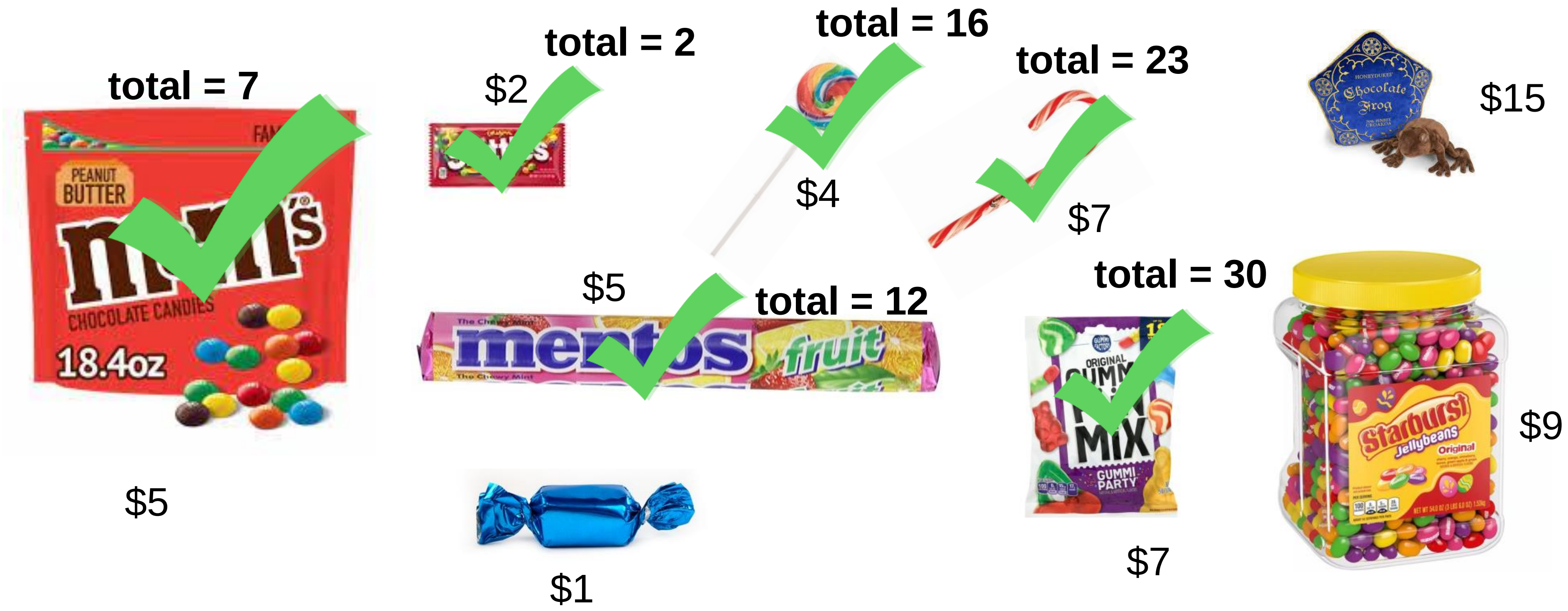


Buying gifts

- **Other solutions with 6 candies?**

Buying gifts

- **Other solutions with 6 candies?**



Buying gifts: the first decision

- **Buying the $1 candy is never a bad idea**
- **If you don’t buy it in an optimal solution, you can always substitute any chosen candy with the cheapest one!**
  - So why don’t we buy the cheapest candy? It never hurts!

Buying gifts: a greedy algorithm

- **If possible, but the cheapest available candy**
- **Repeat until no candies left or leftover money is insufficient**

Prove the optimality of a greedy algorithm

- **To prove optimality of a greedy strategy, we have to prove the following two properties**

1. **Greedy Choice: The greedy choice is part of the optimal answer**
2. **Optimal Substructure: The optimal solution to the big problem contains the optimal solution to the sub-problem**
   - After making the first choice,
     - The final best solution is first choice + best solution for the rest of (compatible) input
     - We can solve the same optimization problem recursively!

Buying gifts: revisit the greedy choice

- **The cheapest candy is always in ONE OF the optimal solutions**
  - If not, I can always substitute any chosen candy to the cheapest one, and this is still an optimal solution!

Prove the optimality of a greedy algorithm

- **To prove optimality of a greedy strategy, we have to prove the following two properties**

1. **Greedy Choice: The greedy choice is part of the answer**
2. **Optimal Substructure: The optimal solution to the big problem contains the optimal solution to the sub-problem**
   - After making the first choice,
     - The final best solution is first choice + best solution for the rest of (compatible) input
     - We can solve the same optimization problem recursively!

Buying gifts: optimal substructure

- **Global optimal solution is the cheapest candy + ONE OF the optimal solutions for the subproblem without the cheapest candy**

Buying gifts: optimal substructure

- **Global optimal solution is the cheapest candy + ONE OF the optimal solutions for the subproblem without the cheapest candy**
  - Assume to the contrary that the optimal solution is $1 candy + another solution

Buying gifts: optimal substructure

- **Global optimal solution is the cheapest candy + ONE OF the optimal solutions for the subproblem without the cheapest candy**
  - Assume to the contrary that the optimal solution is \$1 candy + another solution
  - Then \$1 candy + optimal solution for the rest is no worse

Prove the optimality of a greedy algorithm

- **To prove optimality of a greedy strategy, we have to prove the following two properties**

1. **Greedy Choice: The greedy choice is part of the answer**
2. **Optimal Substructure: The optimal solution to the big problem contains the optimal solution to the sub-problem**
    - After making the first choice,
        - The final best solution is first choice + best solution for the rest of (compatible) input
        - We can solve the same optimization problem recursively!

Buying gifts: a greedy algorithm

- **If possible, but the cheapest available candy // greedy choice**
- **Repeat until no candies left or run out of money // optimal substructure**

Example: Kayaking!

Kayaking

- n 141 students go kayaking
- **Each kayak can take 1 or 2 person(s)**
  - Same price
  - Same weight limit w
- **Given the weight of all students $a[i]$**
- **What's the smallest number of kayaks needed?**

Kayaking

- a[] = 1, 3, 5, 6, 8, 10, 12, 16, 18, 19
- w = 20

Kayaking

- **a[] = 1, 3, 5, 6, 8, 10, 12, 16, 18, 19**
- **w = 20**

- **Solution 1:**
  - Start with the lightest student a[1], pair it with the heaviest s.t. they can be in one kayak
  - (1, 19)
  - (3, 16), (18)
  - (5, 12)
  - (6, 10)
  - (8)
  - 6 in total

Kayaking

- **a[] = 1, 3, 5, 6, 8, 10, 12, 16, 18, 19**
- **w = 20**

- **Solution 2:**
  - Start with the heaviest student a[n], pair it with the heavies s.t. they can be in one kayak
  - (19, 1)
  - (18)
  - (16, 3)
  - (12, 8)
  - (10, 6)
  - (5)
  - 6 in total

Kayaking

- **Actually, both will give you an optimal solution!**
- **Why????**
- **Take solution 1 as an example**
  - Start with the lightest student a[1], pair it with the heaviest s.t. they can be in one kayak

Prove the optimality of a greedy algorithm

- **To prove optimality of a greedy strategy, we have to prove the following two properties**

1. **Greedy Choice: The greedy choice is part of the answer**
2. **Optimal Substructure: The optimal solution to the big problem contains the optimal solution to the sub-problem**
   - After making the first choice,
     - The final best solution is first choice + best solution for the rest of (compatible) input
     - We can solve the same optimization problem recursively!

Kayaking: greedy choice

- **The greedy choice is part of the answer**
- **Start with the lightest student A_1 with weight a_1, pair it with the heaviest student (say X with weight x) s.t. they can be in one kayak**
- **Is (A_1, X) always part of ONE OF the optimal solutions?**
- **If not … assume A_1 is paired with student Y with weight $y < x$ (why?)**
  - X is either along, or paired with student Z
  - Then we can swap X and Y
    - (A_1, Y), (X, Z) ➔ (A_1, X), (Y, Z) or (A_1, Y), (X) ➔ (A_1, X), (Y)
  - We know $a_1 + x \leq w$
  - $y + z < x + z \leq w$, so kayak (Y, Z) is also valid
  - Hence, for any optimal solution, we can modify it and make (A_1, X) part of it

Kayaking: greedy choice

- **The greedy choice is part of the answer**
- **Or …, if we cannot find anyone to pair with A_1, then the kayak (A_1) must be in the optimal solution**

Prove the optimality of a greedy algorithm

- **To prove optimality of a greedy strategy, we have to prove the following two properties**

1. **Greedy Choice: The greedy choice is part of the answer**
2. **Optimal Substructure: The optimal solution to the big problem contains the optimal solution to the sub-problem**
   - After making the first choice,
     - The final best solution is first choice + best solution for the rest of (compatible) input
     - We can solve the same optimization problem recursively!

Kayaking: optimal substructure

- **Optimal Substructure: The optimal solution to the big problem contains the optimal solution to the sub-problem**
- **After we pair A_1 with X, what happens?**
- **Claim: the optimal solution of the problem with choosing (A_1, X) is**
  - (A_1, X) + optimal solution of assigning boats to the rest n-2 students
  - Why? Assume to the contrary it is not, it is (A_1, X) + solution B, where B is worse than the "optimal solution of assigning boats to the rest n-2 students".
  - Why don't we substitute B with the "optimal solution of assigning boats to the rest n-2 students"?
- **Similar for the case when we cannot pair A_1 with anyone**

Prove the optimality of a greedy algorithm

- **To prove optimality of a greedy strategy, we have to prove the following two properties**

1. **Greedy Choice: The greedy choice is part of the answer**
2. **Optimal Substructure: The optimal solution to the big problem contains the optimal solution to the sub-problem**
    - After making the first choice,
        - The final best solution is first choice + best solution for the rest of (compatible) input
        - We can solve the same optimization problem recursively!

Recap

- **The greedy strategy is widely used in optimization problems**
  - When making a decision, always choose the “best” based on a certain criterion
  - Easy to design and implement, but not necessarily optimal

- **To prove the optimality of your greedy algorithm, you need to show**
  - Your greedy choice is always part of ONE OF the optimal solutions
  - Optimal substructure so you can recursively apply greedy choice and get the entire solution

- **Programming HW 2 is ready, and Problems B, C, and D can be solved using greedy algorithms**
  - Can try them now!

About UCRPC

- **Many of you have attended the competition, and did pretty well**
  - Send me an email (ygu@ucr.edu) about your name and I will give you bonus candies
- **It seems many of you enjoy algorithms and programming**
  - We have a competition programming club that has weekly practice
  - 2h training on Saturday, 1h lecture/solution, free pizza
  - Join #comp-programming at CS@UCR slack channel for more weekly info
  - The goal for the club is to prepare for ACM ICPC (International Collegiate Programming Contest), one of the most well-renown competitions
- **In addition, if you have asked and answered questions in the first few lectures, please come and write down your names**