COMPLETE SEARCH+
DYNAMIC PROGRAMMING + GREEDY

COMP 321 – McGill University
These slides are mainly compiled from the following resources.
- Professor Jaehyun Park’ slides CS 97SI
- Top-coder tutorials.
- Programming Challenges books.
Greedy

• It makes locally optimal choice at each step with the hope of finding the optimal solution.

• Key ingredients to make greedy works.
  • It has optimal sub-structures.
    • Optimal solution to the problem contains optimal solutions to the sub-problems.
  • It has a greedy property (remark: hard to prove its correctness!).
    • If we make a choice that seems best at the moment and solve the remaining subproblems later, we still reach optimal solution. We never have to reconsider our previous choices.
Greedy – example 1

- Suppose we have a large number of coins with different denominations, i.e. 25, 10, 5, and 1 cents. We want to make change with the least number of coins used.
  - Ex. if the denominations are {25, 10, 5, 1} cents and we want to make a change of 42 cents, we can do: $42-25 = 17 \rightarrow 17-10 = 7 \rightarrow 7-5 = 2 \rightarrow 2-1 = 1 \rightarrow 1-1 = 0$, of total 5 coins. This is optimal.
Greedy – example 1

- Lets check the key ingredients.
- It has optimal sub-structures.
  - We have seen that in the original problem to make 42 cents, we have to use 25+10+5+1+1.
  - This is an optimal 5 coins solution to the original problem!
  - Now, the optimal solutions to its sub-problems are contained in this 5 coins solution, i.e.
    - a. To make 17 cents, we have to use 10+5+1+1 (4 coins),
    - b. To make 7 cents, we have to use 5+1+1 (3 coins), etc
Greedy – example 1

• Lets check the key ingredients.
• It has a greedy property.
  • Given every amount $V$, we greedily subtract it with the largest denomination of coin which is not greater than this amount $V$. It can be proven that using other strategy than this will not lead to optimal solution.
Greedy – example 1

- PLEASE NOTE THAT this greedy algorithm **does not** work for all sets of coin denominations.
  - e.g. \{1, 3, 4\} cents. To make 6 cents with that set, a greedy algorithm would choose 3 coins \{4, 1, 1\} instead of the optimal solution using 2 coins \{3, 3\}. 
Greedy – example 2

- Problem:
  - Job j starts at $s_j$ and finishes at $f_j$.
  - Two jobs compatible if they don't overlap.
  - Goal: find maximum subset of mutually compatible jobs.
Greedy – example 2

- Greedy template. Consider jobs in some natural order.
- Take each job provided it's compatible with the ones already taken.
- [Earliest start time] Consider jobs in ascending order of $s_j$. 
Greedy – example 2

- Greedy template. Consider jobs in some natural order.
- Take each job provided it's compatible with the ones already taken.
- [Shortest interval] Jobs in ascending order of $f_j - s_j$. 

**counterexample for shortest interval**

[Image of counterexample with redacted text]
Greedy – example 2

• Greedy template. Consider jobs in some natural order.
• Take each job provided it's compatible with the ones already taken.
• [Fewest conflicts] For each job $j$, count the number of conflicting jobs $c_j$. Schedule in ascending order of $c_j$. 

counterexample for fewest conflicts
Greedy – example 2

- Greedy template. Consider jobs in some natural order.
- Take each job provided it's compatible with the ones already taken.
- [Earliest finish time] Consider jobs in ascending order of $f_j$.

```
EARLIEST-FINISH-TIME-FIRST (n, s_1, s_2, ..., s_n, f_1, f_2, ..., f_n)

SORT jobs by finish time so that $f_1 \leq f_2 \leq ... \leq f_n$

A ← ∅ ← set of jobs selected

FOR j = 1 TO n

    IF job j is compatible with A
        A ← A ∪ {j}

RETURN A
```
Greedy – example 3

- **Problem:**
  - Lecture $j$ starts at $s_j$ and finishes at $f_j$.
  - **Goal:** find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.
  - **Ex.** This schedule uses 4 classrooms to schedule 10 lectures.
Greedy – example 3

• Problem:
  • Lecture j starts at \( s_j \) and finishes at \( f_j \).
  • Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.
  • Ex. This schedule uses 3 classrooms to schedule 10 lectures.
Greedy – example 3

- Greedy template. Consider lectures in some natural order.
- Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.
- [Earliest finish time] Consider lectures in ascending order of $f_j$. (solution of the previous example.)
Greedy – example 3

- Greedy template. Consider lectures in some natural order.
- Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.
- [Shortest interval] Consider lectures in ascending order of $f_j - s_j$. 

(counterexample for shortest interval)
Greedy – example 3

- Greedy template. Consider lectures in some natural order.
- Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.
- [Fewest conflicts] For each lecture $j$, count the number of conflicting lectures $c_j$. Schedule in ascending order of $c_j$. 

(counterexample for fewest conflicts)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Greedy – example 3

- Greedy template. Consider lectures in some natural order.
- Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.
- [Earliest start time] Consider lectures in ascending order of $s_j$.

```
EARLIEST-START-TIME-FIRST (n, s_1, s_2, ..., s_n, f_1, f_2, ..., f_n)

SORT lectures by start time so that $s_1 \leq s_2 \leq ... \leq s_n$.

d ← 0 ← number of allocated classrooms

FOR j = 1 TO n
    IF lecture j is compatible with some classroom
        Schedule lecture j in any such classroom k.
    ELSE
        Allocate a new classroom $d + 1$.
        Schedule lecture j in classroom $d + 1$.
        $d ← d + 1$

RETURN schedule.
```
Greedy – example 4

Given $1 \leq C \leq 5$ chambers which can store 0, 1, or 2 specimens, $1 \leq S \leq 2C$ specimens, and $M$: a list of mass of the $S$ specimens, determine in which chamber we should store each specimen in order to minimize IMBALANCE.

$$\text{IMBALANCE} = \sum_{i=1}^{C} |X_i - A|$$

i.e. sum of differences between the mass in each chamber w.r.t $A$. where $X_i$ is the total mass of specimens in chamber

$$A = \left( \sum_{j=1}^{S} M_j \right) / C,$$

$A$ is the average of all mass over $C$ chambers.

**Example:**

- $C = 3$, $S = 4$, $M = \{5, 1, 2, 7\}$
- $A = (5 + 1 + 2 + 7) / 3 = 5$

**IMBALANCE =**

$$|5 - 5| + |2 - 5| + |1 - 5| = 7 + 3 + 4 = 14$$
Greedy – example 4

• Observations.
  • If there exists an empty chamber, at least one chamber with 2 specimens must be moved to this empty chamber! Otherwise the empty chambers contribute too much to IMBALANCE!

\[
A = 5
\]

\[
\text{IMBALANCE} = |(7+1)-5| + |(2+5)-5| + |0-5| = 3 + 2 + 5 = 10
\]
Greedy – example 4

• Observations. If $S > C$, then $S-C$ specimens must be paired with one other specimen already in some chambers.
Greedy – example 4

• Observations. If $S < 2C$, add dummy $2C-S$ specimens with mass 0.
  • For example, $C = 3, S = 4, M = \{5, 1, 2, 7\} \rightarrow C = 3, S = 6, M = \{5, 1, 2, 7, 0, 0\}$.
• Then, sort these specimens based on their mass such that $M_1 \leq M_2 \leq \ldots \leq M_{2C-1} \leq M_{2C}$
  • For example, $M = \{5, 1, 2, 7, 0, 0\} \rightarrow \{0, 0, 1, 2, 5, 7\}$. 
Greedy – example 4

By adding dummy specimens and then sorting them, a greedy strategy ‘appears’. We can now:

- Pair the specimens with masses $M_1$&$M_{2C}$ and put them in chamber 1, then
- Pair the specimens with masses $M_2$&$M_{2C-1}$ and put them in chamber 2, and so on . . .

If you swap any two specimens from two different chambers, you will always have worse/equal solution.
Tips

- Using Greedy solutions in programming contests is usually risky.
  - A greedy solution normally will not encounter TLE response, as it is lightweight, but tends to get WA response.
- Proving that a certain problem has optimal sub-structure and greedy property in contest time may be time consuming, so a competitive programmer usually do this.
  - Look at the input size. If it is ‘small enough’ for the time complexity of either Complete Search or Dynamic Programming, he will use one of these approaches as both will ensure correct answer.
  - Use Greedy solution if you know for sure that the input size is too large.
- A problem that seems extremely complicated on the surface might signal a greedy approach.
Wrapping the examples

Abridged problem statement: Given different models for each garment (e.g. 3 shirts, 2 belts, 4 shoes, ...), buy one model of each garment. As the budget is limited, we cannot spend more money than the budget, but we want to spend the maximum possible. But it is also possible that we cannot buy one model of each garment due to that small amount of budget.

The input consist of two integers $1 \leq M \leq 200$ and $1 \leq C \leq 20$, where $M$ is the budget and $C$ is the number of garments that you have to buy. Then, there are information of the $C$ garments. For a garment_id $\in [0 \ldots C-1]$, we know an integer $1 \leq K \leq 20$ which indicates the number of different models for that garment_id, followed by $K$ integers indicating the price of each model $\in [1 \ldots K]$ of that garment_id.

The output should consist of one integer that indicates the maximum amount of money necessary to buy one element of each garment without exceeding the initial amount of money. If there is no solution, print “no solution”.
Wrapping the examples

For example, if the input is like this (test case A):
M = 20, C = 3
3 models of garment_id 0 → 6 4 8 // see that the prices are not sorted in input
2 models of garment_id 1 → 5 10
4 models of garment_id 2 → 1 5 3 5

Then the answer is 19, which may come from buying the underlined items (8+10+1).
Note that this solution is not unique, as we also have (6+10+3) and (4+10+5).

However, if the input is like this (test case B):
M = 9 (very limited budget), C = 3
3 models of garment_id 0 → 6 4 8
2 models of garment_id 1 → 5 10
4 models of garment_id 2 → 1 5 3 5

Then the answer is “no solution” as buying all the cheapest models (4+5+1) = 10 is still > M.
Wrapping the examples

- **Approach 1: Complete Search.**
  - Start with `money_left = M` and `garment_id = 0`. Try all possible models in that `garment_id = 0` (max 20 models). If model `i` is chosen, then subtract `money_left` with model `i`'s price, and then recursively do the same process to `garment_id = 1` (also can go up to 20 models), etc. Stop if the model for the last `garment_id = C - 1` has been chosen.
  - If `money_left < 0` before we reach the last `garment_id`, prune this partial solution.
  - Among all valid combinations, pick one that makes `money_left` as close to 0 as possible yet still $\geq 0$. 
Wrapping the examples

• Approach 1: Complete Search.
• This solution works correctly, but very slow!
  • In the largest test case, garment id 0 have up to 20 choices; garment id 1 also have up to 20 choices; ...; and the last garment id 19 also have up to 20 choices. Therefore, Complete Search like this runs in $20 \times 20 \times \ldots \times 20$ of total 20 times in the worst case, i.e. $20^{20} = \text{a very very large number}$
Wrapping the examples

• Approach 2: Greedy.
  • Since we want to maximize the budget spent, why don’t we take the most expensive model in each garment_id which still fits our budget?
    • This greedy strategy ‘works’ for test cases A+B above and produce the same optimal solution (8+10+1) = 19 and “no solution”, respectively.
    • It also runs very fast, which is 20 + 20 + ... + 20 of total 20 times = 400 operations in the worst case.
    • But greedy does not work for many other cases.

\[
M = 12, \ C = 3
\]

3 models of garment_id 0 → 6 4 8
2 models of garment_id 1 → 5 10
4 models of garment_id 2 → 1 5 3 5

• Greedy wrongly reports “nosolution”. The optimal solution is actually (4+5+3 = 12), which use all our budget.
Wrapping the examples

- Approach 3: Top-Down DP.
- This problem has optimal sub-structures.
  - This is shown in Complete Search recurrence above: solution for the sub-problem is part of the solution of the original problem. Although optimal sub-structure are the same ingredient to make a Greedy Algorithm work, this problem lacks the ‘greedy property’ ingredient.
- This problem has overlapping sub-problems.
  - This is the key point of DP! The search space is actually not as big as $20^{20}$ analyzed in Complete Search discussion above as many sub-problems are actually overlapping!
    - Think in two models in certain garment_id with the same price $p$. 
Wrapping the examples

• Approach 3: Top-Down DP.
• how many distinct sub-problems (a.k.a. states) are there in this problem?
  • The answer is, only 201 X 20 = 4,020. As there are only 201 possible money_left (from 0 to 200, inclusive) and 20 possible garment_id (from 0 to 19, inclusive).
Wrapping the examples

- **Approach 3: Top-Down DP.**

- **Implementation** (we already have the recursive backtracking).
  1. Initialize a DP ‘memo’ table with dummy values, e.g. ‘-1’.
     - The dimension of the DP table must be the size of distinct sub-problems.
  2. At the start of recursive function, simply check if this current state has been computed before.
     - (a) If it is, simply return the value from the DP memo table, O(1).
     - (b) If it is not, compute as per normal (just once) and then store the computed value in the DP memo table so that further calls to this sub-problem is fast.