## CSE 417 Algorithms

Lecture 21, Autumn 2023
Dynamic Programming
Subset Sum etc.

1

#### **Announcements**

- · Homework Deadlines
  - HW 7: Due Wednesday, Nov 20
  - HW 8: Due Wednesday, Nov 27
  - HW 9: Due Friday, Dec 6
- · Dynamic Programming Reading:
  - 6.1-6.2, Weighted Interval Scheduling, Path Counting, Paragraph Layout
  - 6.4 Knapsack and Subset Sum
  - 6.6 String Alignment
  - 6.7\* String Alignment in linear space
  - 6.8 Shortest Paths (again)
  - 6.9 Negative cost cycles
    - How to make an infinite amount of money

\_

What is the largest sum you can make of the following integers that is ≤ 20

{4, 5, 8, 10, 13, 14, 17, 18, 21, 23, 28, 31, 37}

3

What is the largest sum you can make of the following integers that is ≤ 2000

{78, 101, 122, 133, 137, 158, 189, 201, 220, 222, 267, 271, 281, 289, 296, 297, 301, 311, 315, 321, 322, 341, 349, 353, 361, 385, 396 }

4

#### Subset Sum Problem

- Given integers {w<sub>1</sub>,...,w<sub>n</sub>} and an integer K
- Find a subset that is as large as possible that does not exceed K
- Dynamic Programming: Express as an optimization over sub-problems.
- New idea: Represent at a sub problems depending on K and n
  - Two dimensional grid

5

## **Subset Sum Optimization**

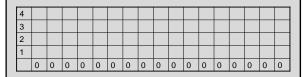
Opt[ j, K ] the largest subset of  $\{w_1,\,...,\,w_j\}$  that sums to at most K

 $Opt[j, K] = max(Opt[j-1, K], Opt[j-1, K-w_j] + w_j)$ 

6

#### Subset Sum Grid

Opt[ j, K] = max(Opt[ j - 1, K], Opt[ j - 1, K -  $w_i$ ] +  $w_i$ )



{2, 4, 7, 10}

#### Subset Sum Grid

 $Opt[j, K] = max(Opt[j-1, K], Opt[j-1, K-w_i] + w_i)$ 

| ſ | 4 | 0 | 2 | 2 | 4 | 4 | 6 | 7 | 7 | 9 | 10 | 11 | 12 | 13 | 14 | 14 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|
|   | 3 | 0 | 2 | 2 | 4 | 4 | 6 | 7 | 7 | 9 | 9  | 11 | 11 | 13 | 13 | 13 | 13 | 13 |
| I | 2 | 0 | 2 | 2 | 4 | 4 | 6 | 6 | 6 | 6 | 6  | 6  | 6  | 6  | 6  | 6  | 6  | 6  |
| Ī | 1 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2  | 2  | 2  | 2  | 2  | 2  | 2  | 2  |
| Ī |   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

 $\{2, 4, 7, 10\}$ 

.

#### Subset Sum Code

$$\label{eq:forj} \begin{split} &for \ j=1 \ to \ n \\ &for \ k=1 \ to \ W \\ &Opt[j, \ k] = max(Opt[j-1, \ k], \ Opt[j-1, \ k-w_j] + w_j) \end{split}$$

9

## **Knapsack Problem**

- · Items have weights and values
- The problem is to maximize total value subject to a bound on weght
- Items {I<sub>1</sub>, I<sub>2</sub>, ... I<sub>n</sub>}
  - Weights  $\{w_1, w_2, ..., w_n\}$
  - Values {v<sub>1</sub>, v<sub>2</sub>, ..., v<sub>n</sub>}
  - Bound K
- · Find set S of indices to:
  - Maximize  $\sum_{i \in S} v_i$  such that  $\sum_{i \in S} w_i \le K$

10

## **Knapsack Recurrence**

Subset Sum Recurrence:

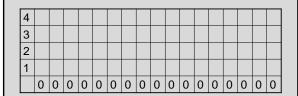
Opt[ j, K] = max(Opt[ j - 1, K], Opt[ j - 1, K -  $w_i$ ] +  $w_i$ )

Knapsack Recurrence:

11

## Knapsack Grid

Opt[ j, K] = max(Opt[ j - 1, K], Opt[ j - 1, K -  $w_i$ ] +  $v_j$ )



Weights {2, 4, 7, 10} Values: {3, 5, 9, 16}

12

## Knapsack Grid

Opt[ j, K] = max(Opt[ j - 1, K], Opt[ j - 1, K -  $w_i$ ] +  $v_i$ )

| 4 | 0 | 3 | 3 | 5 | 5 | 8 | 9 | 9 | 12 | 16 | 16 | 18 | 18 | 21 | 21 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
| 3 | 0 | 3 | 3 | 5 | 5 | 8 | 9 | 9 | 12 | 12 | 14 | 14 | 17 | 17 | 17 | 17 | 17 |
| 2 | 0 | 3 | 3 | 5 | 5 | 8 | 8 | 8 | 8  | 8  | 8  | 8  | 8  | 8  | 8  | 8  | 8  |
| 1 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3  | 3  | 3  | 3  | 3  | 3  | 3  | 3  | 3  |
| Г | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Weights {2, 4, 7, 10} Values: {3, 5, 9, 16}

13

## Alternate approach for Subset Sum

- Alternate formulation of Subset Sum dynamic programming algorithm
  - Sum[i, K] = true if there is a subset of  $\{w_1, \dots w_i\}$  that sums to exactly K, false otherwise
  - Sum [i, K] = Sum [i -1, K] **OR** Sum[i 1, K w<sub>i</sub>]
  - Sum [0, 0] = true; Sum[i, 0] = false for  $i \neq 0$
- To allow for negative numbers, we need to fill in the array between K<sub>min</sub> and K<sub>max</sub>

14

#### Run time for Subset Sum

- With n items and target sum K, the run time is O(nK)
- If K is 1,000,000,000,000,000,000,000,000 this is very slow
- Alternate brute force algorithm: examine all subsets: O(n2<sup>n</sup>)
- Point of confusion: Subset sum is NP Complete

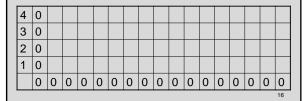
15

# Two dimensional dynamic programming

Subset sum and knapsack

Opt[ j, K] = max(Opt[ j - 1, K], Opt[ j - 1, K -  $w_j$ ] +  $w_j$ )

Opt[j, K] = max(Opt[j - 1, K], Opt[j - 1, K -  $w_i$ ] +  $v_i$ )



10

## Reducing dimensions

- Computing values in the array only requires the previous row
  - Easy to reduce this to just tracking two rows
  - And sometimes can be implemented in a single row
- · Space savings is significant in practice
- Reconstructing values is harder

17