CS 381 – FALL 2019

Week 7.2, Wednesday, Oct 2

No PSOs next week (October Break)
Thursday/Friday PSOs are now ``office hours”
Look for Homework 4 tonight

Announcements

- Midterm Graded
  - Possible Points: 110 (+5 point bonus)
  - Max: 113 Mean: 89.23 Median: 92.5
  - Std Dev: 14.44
- No Curve Until the End of the Year
- Gradescope vs. Blackboard Scores
- Proofs: most challenging part of exam
  - Important for CS381!
  - How to judge whether a proof is rigorous/clear
    - Each claim should be easily verifiable (or falsifiable)
    - Consider changing the original problem statement to something false, but keeping the proof unchanged
      - E.g., set T(2)=50 so that we don't always have T(n) $\leq 10n^2$
      - It should be clear what part of the proof breaks down
        - If this is not clear it is a good sign your proof is not rigorous/clear

Grade distribution

- To determine the final grades, we ask questions like “How well did this student master the material?”
- Grading is *not* curved in the sense that the average is set or a fraction of the class must receive a certain grade.
  - We do not have a pre-defined mapping from completed work scores to a final grade.
- We use the standards given on next two slides as guidelines.
  - Adapted from U of Washington grading guidelines

- A+, A: Superior performance in all aspects with work exemplifying the highest quality. Unquestionably prepared for courses building on 381 and for graduate work.
- A-: Superior performance in most aspects; high quality work in the remainder. Prepared for courses building on 381 and graduate work.

- B+: High quality performance in all or most aspects. Considered prepared for courses building on 381.
- B: High quality performance in some; satisfactory performance in the remainder. Good chance of success in courses building on 381.

- B-: Satisfactory performance. Evidence of sufficient learning to succeed in courses building on 381.
- C+: Satisfactory performance in most of the course. Evidence of sufficient learning to succeed in courses building on 381 with effort.

- C: Evidence of learning but generally marginal performance. Not considered prepared for courses building on 381.

- D+, D: Demonstrated minimal learning and low quality performance in all aspects.
- D-: Little evidence of learning. Poor performance in all aspects.
- F: Complete absence of evidence of learning.

6.4 Knapsack Problem

Knapsack Problem (Greedy)

Knapsack problem.

- Given n objects and a "knapsack."
- Item i weighs w_i > 0 kilograms and has value v_i > 0.
- Knapsack has capacity of W kilograms.
- Goal: fill knapsack so as to maximize total value.

Ex: { 3, 4 } has value 40.

```
W = 4-2
  = 2
```

#	value	weight	ratio
1	1	1	1
2	6	2	3
3	18	5	3.6
4	22	6	3.66..
5	28	7	4

Greedy: repeatedly add item with maximum ratio v_i / w_i.
Ex: { 5, 2, 1 } achieves only value = 35 $\Rightarrow$ greedy not optimal.

Dynamic Programming: False Start

Def. OPT(i) = max profit subset of items 1, …, i.

- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 }
  - In this case we have OPT(i) = OPT(i-1) ☺

- Case 2: OPT selects item i.
  - accepting item i does not immediately imply that we will have to reject other items

    $OPT(i) = v_i$? → underestimate, could pick other items

    $OPT(i) = v_i + OPT(i-1)$? → overestimate, capacity reduced!
  - without knowing what other items were selected before i, we don't even know if we have enough room for i

Conclusion. Need more sub-problems!

Dynamic Programming: Adding a New Variable

Def. OPT(i, w) = max profit subset of items 1, …, i with weight limit w.

- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 } using weight limit w
- Case 2: OPT selects item i.
  - new weight limit = w – w_i
  - OPT selects best of { 1, 2, …, i–1 } using this new weight limit

$$OPT(i, w) = \begin{cases} 0 & \text{if } \ i = 0 \\ OPT(i-1, w) & \text{if } \ w_i > w \\ \max\{\, OPT(i-1, w), \quad v_i + OPT(i-1, w-w_i) \,\} & \text{otherwise} \end{cases}$$

Clicker Question

W + 1

	0	1	2	3
ϕ	0	0	0	0
{ 1 }	0	1	1	1
{ 1, 2 }	0	1	x1	x2

n + 1

W = 3

Item	Value	Weight
1	1	1
2	6	2

What are the missing values in the DP table?

A. (x1=1, x2=6)
B. (x1=5, x2=7)
C. (x1=6, x2=9)
D. (x1=1, x2=1)
E. (x1=6, x2=7)

Clicker Question

W + 1 →

	0	1	2	3
ϕ	0	0	0	0
{ 1 }	0	1	1	1
{ 1, 2 }	0	1	x1	x2

n + 1 ↓

W = 3

Item	Value	Weight
1	1	1
2	6	2

What are the missing values in the DP table?

A. (x1=1, x2=6)
B. (x1=5, x2=7)
C. (x1=6, x2=9)
D. (x1=1, x2=1)
E. (x1=6, x2=7)

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{\,OPT(i-1,w),\ \ v_i + OPT(i-1,w-w_i)\,\} & \text{otherwise} \end{cases}$$

Knapsack Problem: Bottom-Up

Knapsack. Fill up an n-by-W array.

```
Input: n, W, w1,…,wN, v1,…,vN

for w = 0 to W
   M[0, w] = 0

for i = 1 to n
   for w = 1 to W
      if (wi > w)
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi ]}

return M[n, W]
```

Knapsack Algorithm

W + 1

n + 1

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

W = 11

OPT: { 4, 3 }
value = 22 + 18 = 40

OPT(5,11)= max{OPT(4,11), 28+OPT(4,4)}
= OPT(4,11) → item 5 not selected

OPT(4,11) = 22+OPT(3,5) → item 4 selected

Knapsack Problem: Running Time

Running time. Θ(n W).

- Not polynomial in input size!
  - Only need $\log_2 W$ bits to encode each weight
  - Problem can be encoded with $O(n \log_2 W)$ bits
- "Pseudo-polynomial."
- Decision version of Knapsack is NP-complete. [Chapter 8]

Knapsack approximation algorithm. There exists a poly-time algorithm that produces a feasible solution that has value within 0.01% of optimum. [Section 11.8]

Basic Matrix Multiplication

- **Inputs:** matrices A (m x n), B (n x r)
  - Assume A[i,j] and B[i,j] are integers
- **Output:** D = AB (m x r)
  - Standard algorithm uses **mnr** integer multiplications
  - (Faster with Strassen's Algorithm --- Divide & Conquer)
  - For this problem suppose we use standard algorithm
- **Observation 1:** Matrix Multiplication <u>is not</u> commutative
  - i.e. BA≠AB (in fact BA might not even be well defined)
- **Observation 2:** Matrix Multiplication <u>is</u> associative
  - i.e. ABC = (AB)C=A(BC)

Basic Matrix Multiplication

- **Inputs:** matrices A (m x n), B (n x r), C (r x k)
- **Observation 2:** Matrix Multiplication is associative
  - i.e. ABC = (AB)C=A(BC)

  - Option 1: Compute D=(AB) (m x r) then DC
    - Total Multiplications: mnr + mrk
  - Option 2: Compute D =(BC) (n x k) then AD
    - Total Multiplications: nrk + mnk
- Suppose m=100, n=100, r=500 and k=5
  - Option 1 → 5.25 million integer multiplications
  - Option 2 → 0.3 million integer multiplications

Given matrices $A_1, A_2, \ldots, A_n$, place parenthesis minimizing the total number of multiplications.

$$((A_1 A_2) A_3) \; (A_4 A_5)$$

Notation: Matrix A_i is a p_{i-1} by p_i matrix

→ The product $A_i A_{i+1}$ is well defined p_{i-1} by p_{i+1} matrix

→ The product $A_{i+1} \ldots A_n$ is a p_i by p_n matrix

Thus, $A_i(A_{i+1} \ldots A_n)$ is well defined p_{i-1} by p_n matrix

→ The product $A_1 A_2 \ldots A_{i-1}$ is a p_0 by p_{i-1} matrix

Thus, $(A_1 A_2 \ldots A_{i-1})A_i$ is well defined p_0 by p_i matrix

Given matrices $A_1, A_2, \ldots, A_n$, place parenthesis minimizing the total number of multiplications.

$$((A_1 A_2) A_3)\ (A_4 A_5)$$

Observation: Exponentially many ways to place parenthesis!

→Brute force solutions will not work!

→Dynamic Programming?

Given matrices $A_1, A_2, \ldots, A_n$, place parenthesis minimizing the total number of multiplications.

$$((A_1 A_2) A_3)\ (A_4 A_5)$$

1. An optimal solution to matrix chain contains optimal subsolutions.
2. Fill an n×n table **m** where m[i,j] = minimum number of multiplications for generating $A_i \times A_{i+1} \times \ldots \times A_j$

$$\mathbf{m[i,j] = \min \{m[i,k] + m[k+1,j] + p_{i-1} \times p_k \times p_j\}}$$

with i<j and i≤k<j

$$(A_i \times A_{i+1} \times \ldots A_k) \times (A_{k+1} \times \ldots \times A_{j-1} \times A_j)$$