CS 141 F20
DISCUSSION WEEK 5

Dynamic programming
Midterm problems

MIDTERM PROBLEM 3B

(B): It first solves three subproblems of size $n-1$, then combines the solutions of the subproblems in constant time.

$$T(n) = 3T(n-1) + \Theta(1)$$

Recurrence tree:

There is $\Theta(1) = c$ at the top. There are 3 nodes in the level below it, each with c cost, thus $3c$ in total. In the level below that, it would be $9c$.

Size of n decreases by 1 at each level and number of node multiples by 3. There would be a total of n levels. At the lowest level, there would be 3^{n-1}, thus $c * 3^{n-1}$ cost, at the lowest level. (**Recall:** n^{th} term of GP series is ar^{n-1}).

Total cost of the tree = c + 3c + 9c + ... $+c * 3^{n-1}$

$$= c * \frac{3^n - 1}{3-1} = c * \frac{3^n - 1}{2} = \Theta(3^n)$$

MIDTERM PROBLEM 3B

(B): It first solves two subproblems of size $n-2$, then combines the solutions of the subproblems in constant time.

$$T(n) = 2T(n-2) + \Theta(1)$$

$$\Theta(1) = c$$
$$T(1) = c$$

$$T(n) = 2\{2T(n-4) + c\} + c$$

$$= 4T(n-4) + c\{2+1\}$$

$$= 4\{2T(n-6) + c\} + c\{2+1\}$$

$$= 8T(n-6) + c\{2^2 + 2 + 1\}$$

$$= 2^i T(n - 2*i) + c\{2^{i-1} + ...2 + 1\}$$

$$n - 2*i = 1$$
$$\Rightarrow 2*i = n-1$$
$$\Rightarrow i = (n-1)/2 \approx n/2$$

$$= 2^{n/2}T(1) + c\{2^{n/2\ -1} + ...2 + 1\}$$

$$= c * 2^{n/2} + c * \frac{2^{n/2} - 1}{2-1}$$

$$= c * 2^{n/2} + c * \{2^{n/2} - 1\}$$

$$= c * 2^{n/2} + c * 2^{n/2} - c$$

$$= c * 2^{n/2+\ 1} - c$$

$$= \Theta(2^{n/2})$$

MIDTERM PROBLEM 4

Proof for optimal substructure

S is the set of all the students. Let B be the optimal solution for problem with S students. Student a and b travel in the same boat. S' is a subset of students, where $S = S' \bigcup \{a, b\}$. B' would be the optimal solution to the subproblem S'. $B = B' + 1$.

Lets assume that B' is not the optimal solution to the problem S'. Then there would be another solution B'', such that $B'' < B'$, which would be optimal for S' . Now, if we add the boat with $\{a, b\}$ to the solution of B'', then we will get $B'' + 1$ for S. Now, $B'' + 1 \leq B' + 1 => B'' + 1 \leq B$. Thus we got a more optimal solution, which is a contradiction. Thus, our assumption is wrong. Optimal substructure is thus proved.

MIDTERM PROBLEM 4

Proof for greedy choice

Assume that we have an optimal solution to this problem where we do not pair student i with student j. Instead, we pair student i with student k.

There are two possible scenarios here. First one is the scenario where $k = -1$. In that case, by looking at the rowling algorithm, we know that student j is the heaviest student whose weight is less than $T - w_i$ ($j = -1$ if no such student exists). If j and k are both equal to -1, then the rowling algorithm's solution and the optimal solution are identical. Otherwise, rowling algorithm gives us a better solution than the alleged optimal solution in our assumption and we reach a contradiction.

Second scenario is when $k \neq -1$. Since student j is the heaviest student with $w_j < T - w_i$, if we take student k out of the boat and put student j in the boat instead, we will utilize the capacity of the boat more efficiently ($T \geq w_i + w_j > w_i + w_k$). This contradicts our starting assumption since we found a better solution.$\square$

MIDTERM PROBLEM 5

- CLRS 9.2

MIDTERM PROBLEM 6

- Incorrect assignment of 0/1

- After forming the tree
  - Heavier branches should be assigned 1.
  - Lighter branches should be assigned 0.

DYNAMIC PROGRAMING

A simplified case

- Overall weight limit: 8 lb
- Item 1: 5 lb, $150
- Item 2: 4 lb, $100
- Item 3: 1 lb, $10

- Solution 1: Item 1 + Item 3 * 3, value: $180
- Solution 2: Item 2 * 2, value: $200

- Greedy strategy does not provide the optimal solution
- A naïve solution? Try all possibilities!

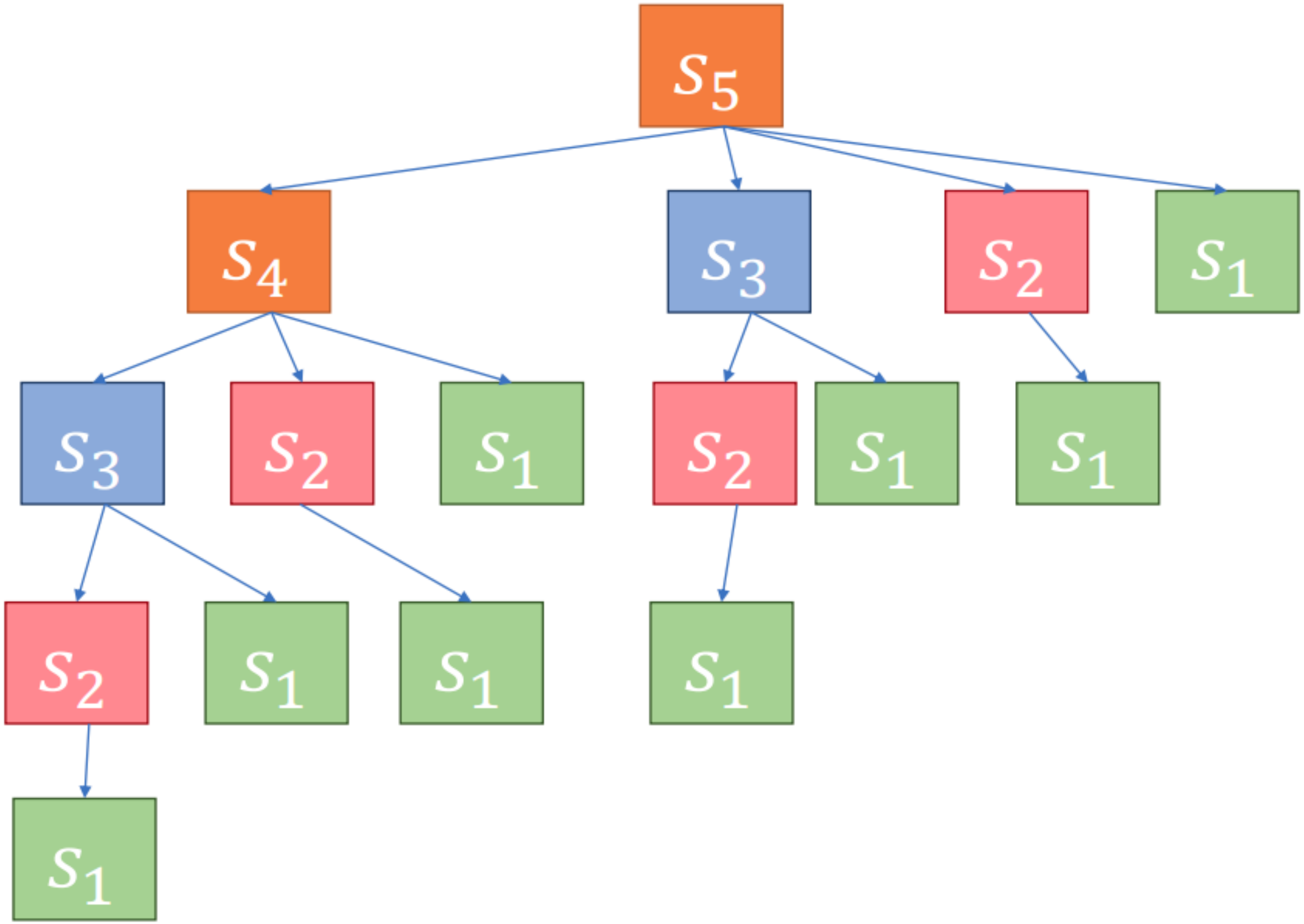
A naïve algorithm

```
int suitcase(int leftWeight) {
   int curBest = 0;
   foreach item (weight, value)
      if (leftWeight >= weight)
         curBest = max(curBest, suitcase(leftWeight - Weight) + value);
   return curBest;
}

answer = suitcase(50);
```

Execution Recurrence Tree

Assume we have for items with weights 1,2,3,4, and the overall weight is 5



A DP algorithm

```
int suitcase(int leftWeight) {
   if (ans[leftWeight] != -1) return ans[leftWeight];
   int curBest = 0;
   foreach item (weight, value)
      if (leftWeight >= weight)
         curBest = max(curBest, suitcase(leftWeight - Weight) + value);
   return ans[leftWeight] = curBest;
}

int ans[50] = {-1, … , -1};
answer = suitcase(50);
```

Recursive Solution

- **Define s_i as the maximum value you can get for a total weight of i**
- **We can express s_i as the following recurrence:**

The best value with
$\boldsymbol{i - w_j}$ weight

$$s_i = \max \begin{cases} 0 \\ \max\limits_{(w_j, v_j) \text{ is an item}} \left\{ s_{i-w_j} + v_j \right\} \mid i > w_j \end{cases}$$

Trying all
possible items

- **Final answer is s_n**

ISSUE WITH CURRENT APPROACH

- Can we pick an item more than once?
- If so, do we need to change the solutions discussed earlier?

- If we use just S_j , it is not possible to know if an item has been used before
- Let's also keep track of the items.

- Let $S_{i,j}$ be the optimal value for total max weight of *i* using only first *j* item

HOW DO WE GET $S_{I,J}$?

- There are 2 options for getting the optimal value for weight *i* and for first *j* items:
  - Item *j* is picked: $S_{i,j} = S_{i - Wj, j-1} + v_j$
  - Item *j* is not picked: $S_{i,j} = S_{i, j-1}$
  - Which one is the best??
    - We pick the maximum of both

Another way to state the relationship of $s_{i,j}$

- **The recurrence:**

$$s_{i,j} = \max \begin{cases} s_{i,j-1} & \\ s_{i-w_j,j-1} + v_j & i \geq w_j \end{cases}$$

- **The boundary:** $s_{i,0} = 0$

- **The recurrence cannot be circular**
  - You can not have **states** a, b, and c that computing a relied on b, b on c, and c on a

EXAMPLE

- Assume we have one copy of each of these items {($5,2lbs),($4,1lbs),($3, 1lbs)} and capacity W=3.
- Let us fill the dynamic programming table $\left(Caclulate\ all\ S_{\{i,j\}}\ for\ i \leq 3 (which\ is\ W) and\ j \leq 3\ (total\ number\ of\ items)\right)$

$$s_{i,j} = \max \begin{cases} s_{i,j-1} & \\ s_{i-w_j,j-1} + v_j & i \geq w_j \end{cases}$$

i: cols j:rows	0	1	2	3
0	0	0	0	0
1	0	0	5	5
2	0	4	5	9
3	0	4	7	9