COT5405 Analys orithms

LECTURE 14-15

Dynamic Programming vs Greedy Algorithms

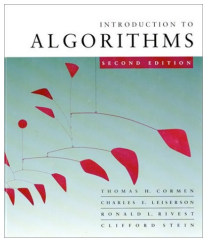
n Multiplication

lection Proble

tructure

- Opt
- Greedy Selection
- Knapsack Problem

Prof. Alper Üngör

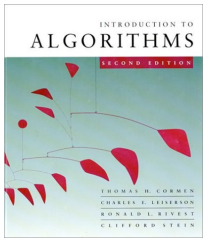


Matrix Chai tiplication

Given a sequence (chain) of n matrices $A_1, A_2, \ldots, A_n$, where A_i is a $p_{i-1} \times p_i$ matrix

Compute their 1 using the minimum number of scala ications

Find a parenthesization that minimizes the number of multiplications



Optimal Su ure

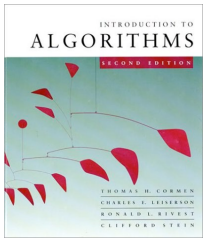
Notation. Let $A_{i,j} = A_i \cdot \ldots \cdot A_j$ for $i \le j$

- Consider an optimal parenthesization for $A_{i,j}$

Say it splits at , so $A_{i,j} = (A_i \cdot \ldots \cdot A_k)(A_{k+1} \cdot \ldots \cdot A_j)$

- Then, the pare efix $A_i \cdot \ldots \cdot A_k$

within the opti f $A_{i,j}$ must be an optimal parenthesization of



Optimal Su ure

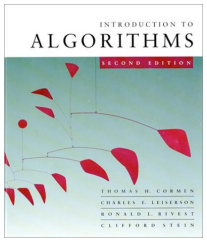
Notation. Let $A_{i,j} = A_i \cdot \ldots \cdot A_j$ for $i \le j$

- Consider an optimal parenthesization for $A_{i,j}$

Say it splits at k, so $A_{i,j} = (A_i \cdot \ldots \cdot A_k)(A_{k+1} \cdot \ldots \cdot A_j)$

- Then, the pare efix $A_i \cdot \ldots \cdot A_k$ within the opti f $A_{i,j}$ must be an optimal parenthesization of

(**Proof**. Suppose it is not optimal, then there exists a better parenthesization for $A_{i,k}$. **Copy and paste** this parenthesization into the parenthesization for $A_{i,j}$. This yields a better parenthesization for $A_{i,j}$. Contradiction.)



Dynamic pr ming

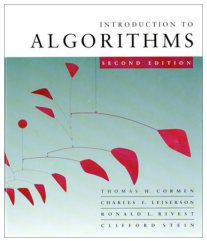
$m[i,j]$ = minimum number of scalar multiplications to compute A_{ij}. We want to compute $m[1,n]$

$A_{i,j}$ $\cdot A_j)$

p_j

Recurrence for optimal substructure:

- $m[i,i] = 0$ for $i=1,2,\ldots,n$
- $m[i,j] = \min_{i \le k < j}\{m[i,k]+m[k+1,j] + p_{i-1}\, p_k\, p_j\}$



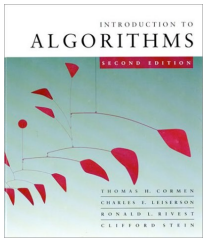
Naive or Re Approach

- Enumerate all possible paranthesizations
- Implement the described recursion directly

The runtime is $\Omega(2^n)$

- Overlapping subproblems!

There are only $O(n^2)$ different problems

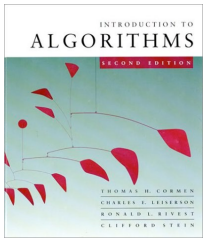


Dynamic Pr ming

Fill the 2 dimensional $m[i,j]$-table bottom-up

For the constru arenthesization, use an addition ords that value of k for which the minimum is found and stored in $m[i,j]$

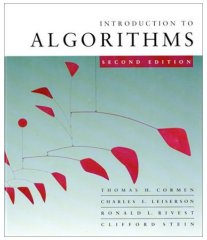
- $m[1,n]$ is the desired value



MatrixChai . Example

$A_1, \ldots A_6$ with sizes 8x10, 10x4, 4x1, 1x8, 8x4, 4x6

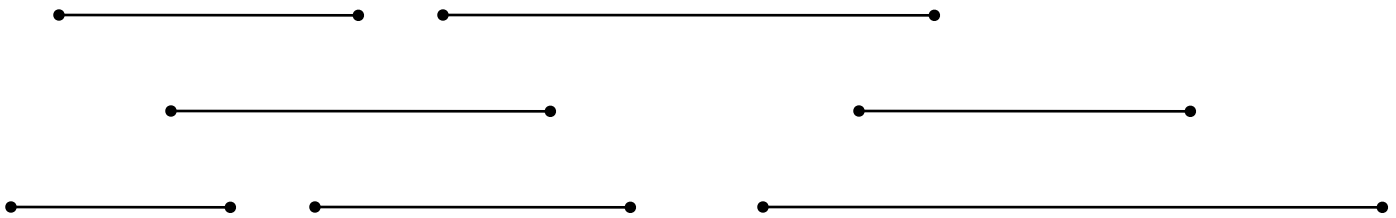
Nice Visualization/Animaiton of this Algorithm:
http://www.brian-borowski.com/Software/Matrix/
http://www.cs.auckland.ac.nz/software/AlgAnim/mat_chain.html#mat_chain_anim

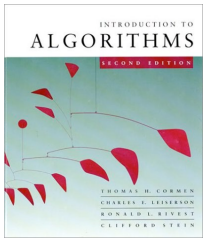


Activity Sel roblem

- Input: Set S of n activities, $a_1, a_2, \ldots, a_n$.
  - » s_i = start time of activity i.
  - » f_i = finish time of activity i.
- Output: Subs ber of compatible activities.
  - » Two activities are compatible, i als don't overlap.

Example:



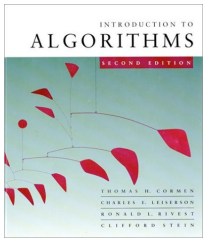


Optimal Su ure

- Assume activities are sorted by finishing times.
  - » $f_1 \le f_2 \le \ldots \le f_n$.
- Suppose an optimal solution includes activity a_k.

  - » This generate
  - » Selecting fro patible with one another, and that finish before a_k start, $a_1, \ldots, a_{k-1}$ mpatible with a_k).

  - » Selecting from $a_{k+1}, \ldots, a_n$, activities compatible with one another, and that start after a_k finishes.
  - » The solutions to the two subproblems must be optimal.
    - Prove using the cut-and-paste approach.



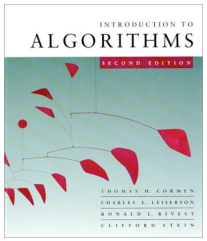
Recursive fo tion

- Let S_{ij} = subset of activities in S that start after a_i finishes and finish before a_j starts.

- Subproblems: mber of mutually compatible acti

- Let $c[i, j]$ = size of maximu et of mutually compatible activities in S_{ij}.

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \phi \\ \max_{i<k<j} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \phi \end{cases}$$

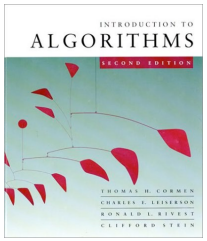


Can we do b

Theorem. Consider any non-empty subproblem S_{ij} , and a_m be the activity in S_{ij} with earliest finish time. Then,

i) Activity a_m is used in some maximum size subset of mutually compa

ii) The first sub $_m$



Can we do b

Theorem. Consider any non-empty subproblem S_{ij} , and a_m be the activity in S_{ij} with earliest finish time. Then,

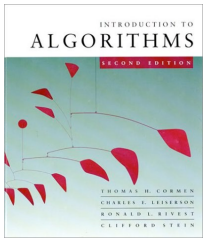
i) Activity a_m is used in some maximum size subset of mutually compa

ii) The first sub S_{im}

Proof. (ii) Suppose S_{im} is non

ere exists some activity a_k such that $f_i \le s_k < f_k \le s_m < f_m$. Then a_k is also in S_{ij} and it has earlier finish time than a_m.

Contradiction.



Can we do b

Theorem. Consider any non-empty subproblem S_{ij} , and a_m be the activity in S_{ij} with earliest finish time. Then,

i) Activity a_m is used in some maximum size subset of mutually compa

ii) The first sub

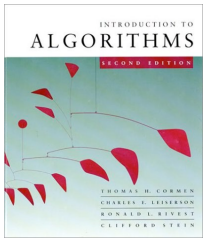
im

Proof. (i) Let A_{ij} be an opt sol ij . let a_k be the activity with earliest finish in A_{ij} . If $a_k = a_m$, we are done.

Otherwise, construct a new solution

$$A'_{ij} = A_{ij} - \{a_k\} + \{a_m\}$$

which is also an optimum feasible solution.



Implication

Theorem. Consider any non-empty subproblem S_{ij} , and a_m be the activity in S_{ij} with earliest finish time. Then,

i) Activity a_m is used in some maximum size subset of mutually compa

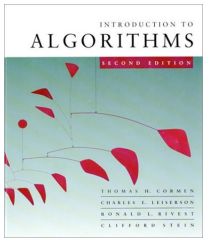
ii) The first sub

im

Implication

ii) solve only one of the two of subproblems.

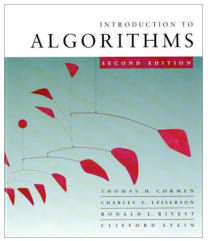
i) a simple top-down approach. pick the job with the earliest finish time. **Greedy Algorithm**!



Recursive G Algorithm

Recursive-Activity-Selector (*s, f, i, j*)

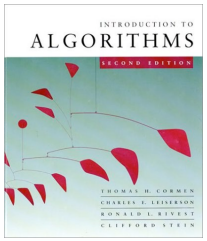
1. $m \leftarrow i+1$
2. **while** m
3. **do** m
4. **if** $m < j$
5. **then return** $\{a_m\} \cup$ Recursive-Activity-Selector(s, f, m, j)
6. **else** return ϕ



Iterative Gr lgorithm

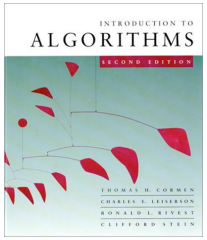
Greedy-Activity-Selector (*s*, *f*)

```
1.  n ← length[s]
2.  A ← [a1]
3.  i ← 1
4.  for m ←
5.      do if  sm ≥ fi
6.      then  A ← A ∪{am}
7.            i ← m
8.  return A
```



Recap of Gr trategy

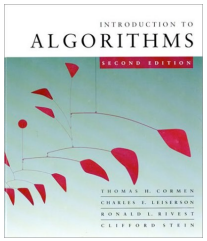
- Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
- Prove that there's always an optimal solution that makes the gree greedy choice is always safe.
- Show that greedy choice and olution to subproblem ⇒ optimal solution to the problem.
- Make the greedy choice and **solve top-down**.
- May have to preprocess input to put it into greedy order.
  - » Example: Sorting activities by finish time.



Why not use e time?

- **Matrix Chain Multiplication Problem.**
  Greedy Strategy: do the leftmost multiplication first;
  do the rightmost multiplication first;
  do the cheapest
  do the product rst;

- **Longest Common Subsequence.**
  Greedy Strategy: ???

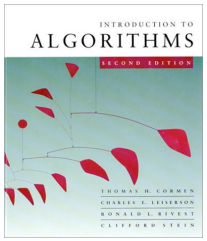


Knapsack P

- Given a knapsack with weight $W > 0$.
  A set S of n items with weights $w_i > 0$ and
  benefits $b_i > 0$ for $i = 1, \ldots, n$.

- $S = \{ (item_1, \quad),$
  $\ldots, (item_n, w_n, b_n) \}$

- Find a subset of the items which does not exceed the weight W of the knapsack and maximizes the benefit.



0/1 Knapsac lem

Determine a subset *A* of { 1, 2, …, *n* } that satisfies the following:

$$\max \sum_{i \in A} b_i \text{ where } \sum_{i \in A} w_i \leq W$$

In 0/1 knapsack a specific item is either selected or not

Variations of the problem

- Fractions are allowed. This applies to items such as:
  - bread, for which taking half a loaf makes sense
  - gold dust
- No fractions.
  - 0/1 (1 bro …)
  - Allows put more type in knapsack
    - 5 pairs of socks
    - 10 gold bricks
  - More than one knapsack, etc.
- First 0/1 *knapsack* problem will be covered then the Fractional *knapsack* problem.

Brute

- Generate all 2^n subsets
- Discard all subsets whose sum of the weights exceed *W (not feasible)*
- Select the ma e remaining (feasible) sub

- What is the run time?
  O($n\ 2^n$), Omega(2^n)
- Lets try the obvious greedy strategy .

Example wit ce"

S = { ($item_1$, 5, \$70), ($item_2$,10, \$90), ($item_3$, 25, \$140) } , W=25

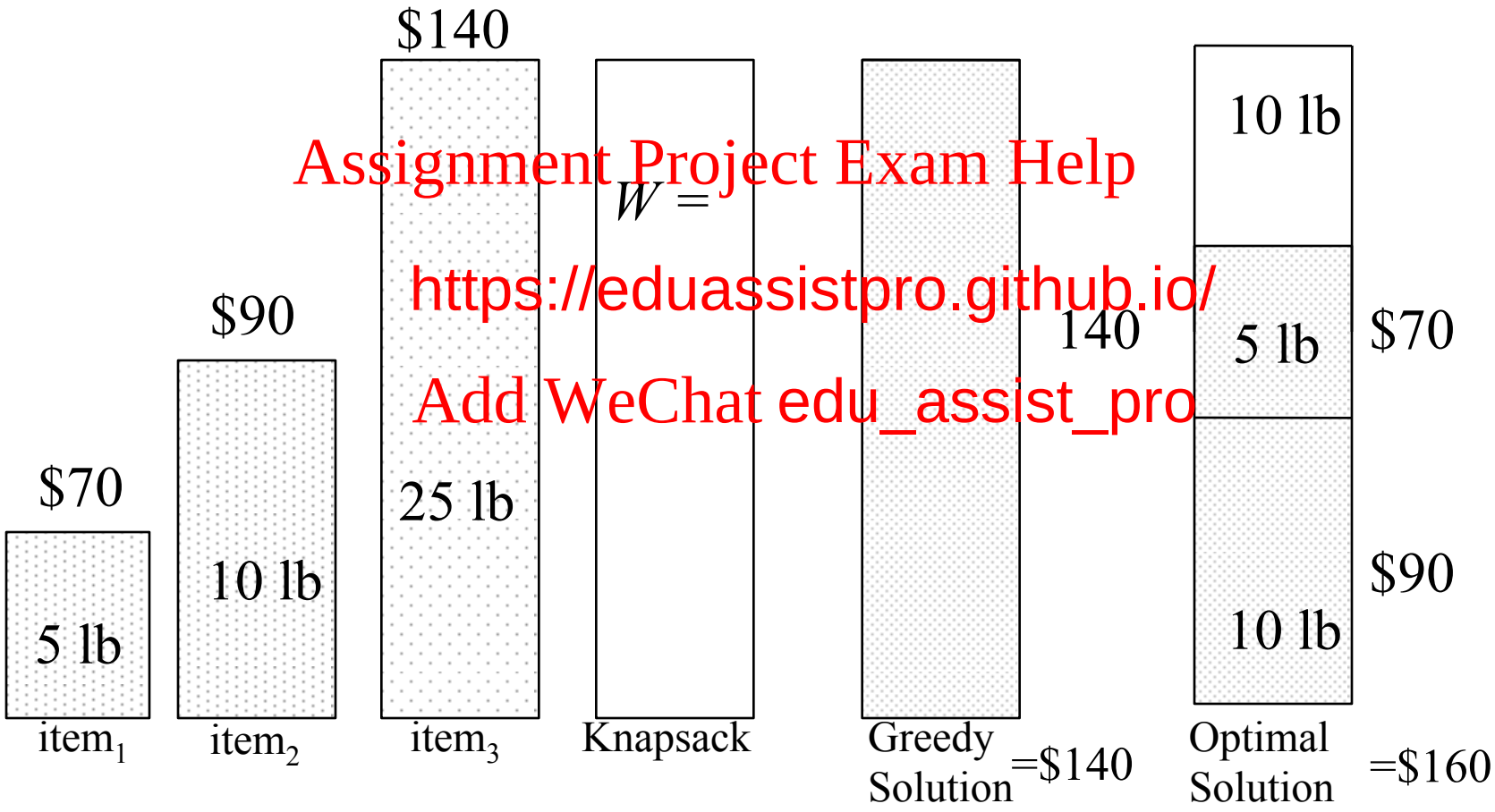
- Subsets:

1. {}
2. { ($item_1$, 5, \$70) } Profit=\$70
3. { ($item_2$,10, \$90 90
4. { ($item_3$, 25, \$14 140
5. { ($item_1$, 5, \$70), ($item_2$,10, \$90) } ****
6. { ($item_2$,10, \$90), ($item_3$, 25, \$140) } exceeds W
7. { ($item_1$, 5, \$70), ($item_3$, 25, \$140) } exceeds W
8. { ($item_1$, 5, \$70), ($item_2$,10, \$90), ($item_3$, 25, \$140) } exceeds W

Greedy 1: Selection criteria — *most beneficial* item.

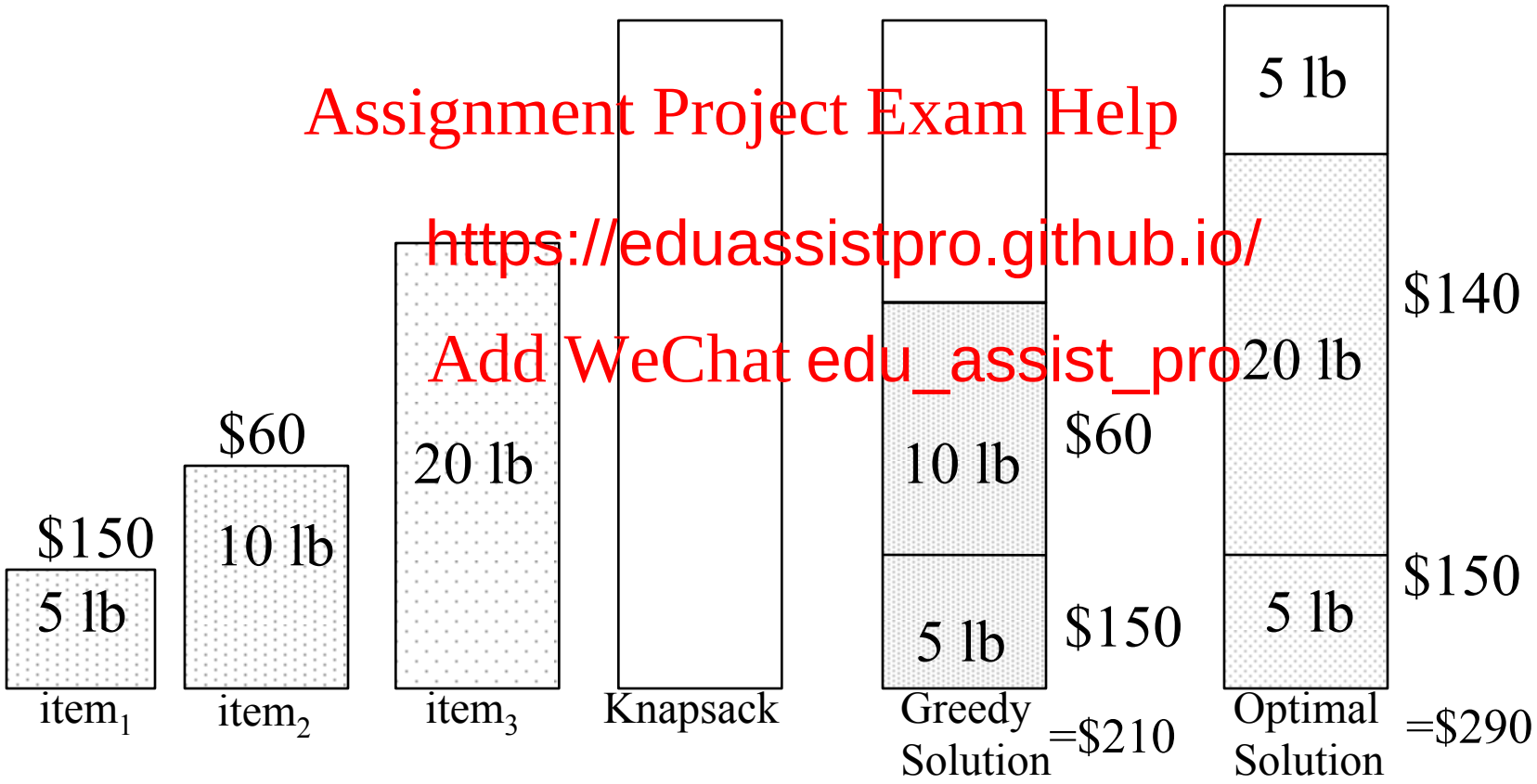
Counter Example:

S = { (*item*$_1$, 5, \$70), (*item*$_2$,10, \$90), (*item*$_3$, 25, \$140) }



Greedy 2: Selection crit *um weight* item
Counter Example:

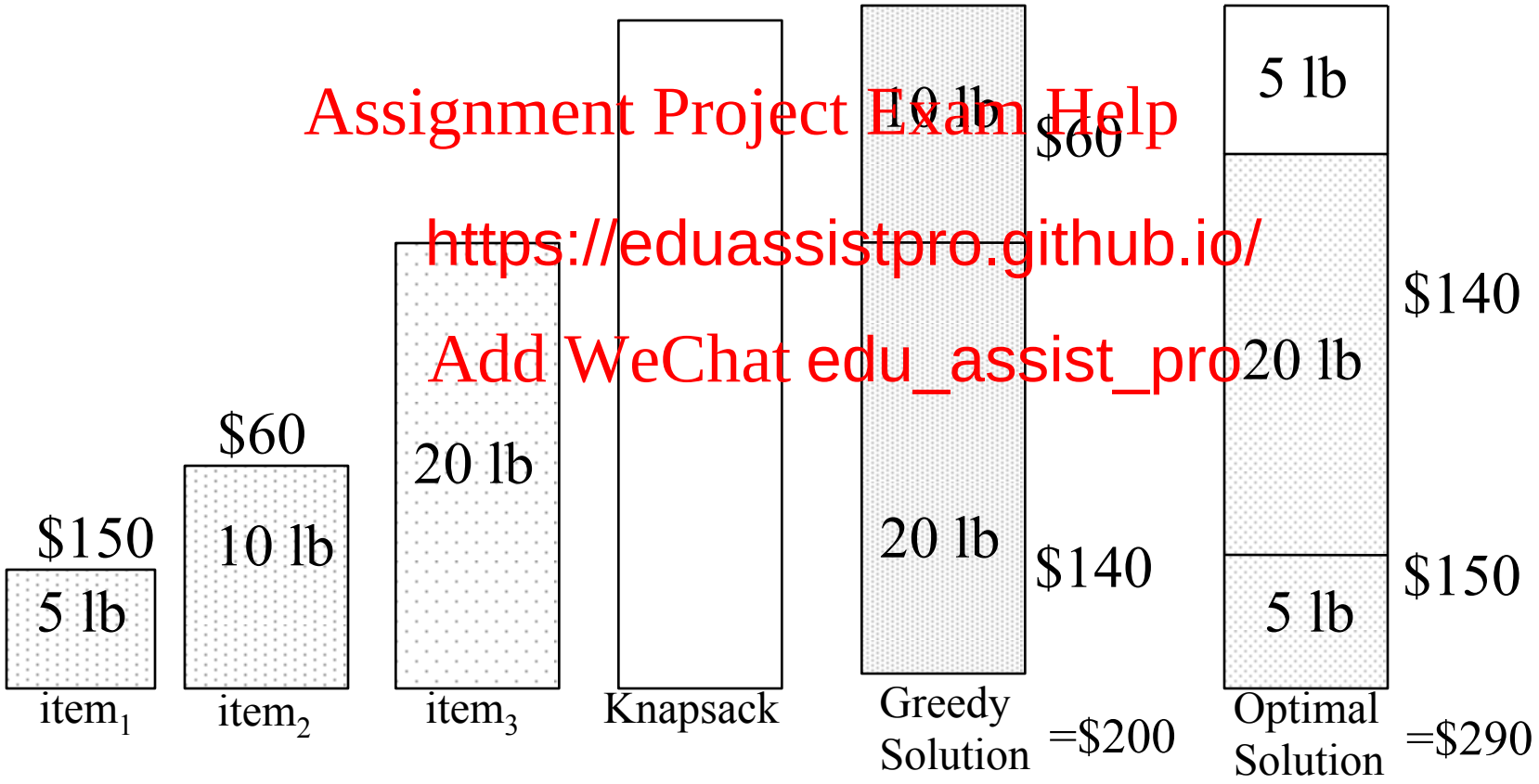
$S = \{ (item_1, 5, \$150), (item_2, 10, \$60), (item_3, 20, \$140) \}$



Greedy 3: Selection cri *mum weight* item

Counter Example:

$S = \{ (item_1 , 5, \$150), (item_2 ,10, \$60), (item_3, 20, \$140) \}$



Greedy 4: Selection criteria: *benefit per unit* item Counter Example

S = { ($item_1$, 5, \$50), ($item_2$, 20, \$140) ($item_3$,10, \$60), }