PROBLEM SET 4 – Manav Bilakhia

CSc 250, Spring 2022
Assigned: Tuesday, Week 5
Due: Thursday, Week 6

Aaron G. Cass
Department of Computer Science
Union College

ALGORITHM DESIGN AND ANALYSIS

1. **The Knapsack Problem.** You are given a number W and a set $S[1..n]$ of n items along with weight and value functions w and v such that the weight of element $S[i]$ is $w(i)$ and its value is $v(i)$. Your goal is to fill the knapsack with the most valuable load of items from S. The restriction is that the knapsack can only hold items with weight totaling at most W.

   (a) Show that the problem has optimal substructure.

   (b) Give a dynamic programming algorithm that solves the problem. Prove that the algorithm works.

   (c) Prove that the running time is in $\Theta(nW)$. Is this polynomial or exponential? **[Be careful — see the book for a formal definition.]**

Answer:

[1]

(a) Optimal Substructure:

$$max \begin{cases} table[i-1][j] \\ v[i-1] + table[i-1][j - w[i-1]] \end{cases}$$

(b) KNAPSACK($W, w[1...n], v[1...n], s[1...n]$)
Input: takes an array of elements, an array with corresponding weights, an array with corresponding values and the total weight of the knapsack
Output: returns the highest possible value that the bag can hold

```
1: for i← 0 to n do
2:   for j← 0 to W do
3:     if i = 0 or j = 0 then
4:       table[i][j] ← 0
5:     else if w[i-1] ≤ j then
6:       table[i][j] ← MAX(v[i-1]+ table [i-1][j-w[i-1]], table [i-1][j])
7:     else
8:       table [i][j] ← table [i-1][j]
9: return table [n][W]
```

Proof by induction
Base Case: We know that table [i,0] = table [0,j] = 0 for all items and weights
Inductive hypothesis: Let us assume that the algorithm is correct for all values of the table[i,j] where table[i,j] $\leq$ table [i',j']. Therefore all the elements in the table are correct.
According to our inductive hypothesis, we know that all values in the table are computed correctly. Our algorithm now considers the most optimal value for an item i' in the knapsack as table [i'-1, j'-w(i')] + v(i') and for an item i' that is not in the knapsack as table [i'-1,j]. Hence the value of table[i',j'] is correct

(c) Our algorithm traverses through all possible weight capacities, we can represent this with the inequality $1 <= w <= W$ with N given elements Hence we know that the run time is $\Theta(nW)$. In this case, the input is of size W, which utilizes $2^{log_2 W}$ bits making W in $\Theta(nW)$ exponential.

2. **Printing Neatly.** [From [2], pg. 364, #15-2] Consider the problem of neatly printing a paragraph on a printer. The input text is a sequence of n words of lengths $l_1, l_2, ..., l_n$, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Note that $\forall k : l_k \leq M$ (i.e. every word is short enough to fit on a line by itself). Our criterion of "neatness" is defined as follows:

   - Let $A = A_1, A_2, ..., A_m$ be the neatly printed lines of output (where each A_i is a line of the output).
   - Let $c(A_q)$ be the number of extra space characters at the end of line A_q of the output. If A_q contains words i through j, where $i \leq j$, and we leave exactly one space between words, then $c(A_q)$ is the line width (M) minus the number of inter-word spaces ($j - i$) minus the sum of the lengths of the words we add ($\sum_{k=i}^{j} l_k$). Therefore, $c(A_q) = M - j + i - \sum_{k=i}^{j} l_k$, which must be non-negative so that the words fit on the line.
   - We wish to minimize the following over all possible answers A:

   $$\sum_{k=1}^{m-1} c(A_k)^3$$

   In other words, we minimize the sum of the cubes of the extra spaces on all lines **except the last line**. Note that this does **not produce the same results as simply minimizing the extra spaces**.

   (a) Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer, as defined above. The input is an array of n word lengths $l_1, l_2, \ldots, l_n$ and a line width M. Your algorithm can print word i with "print(i)" and can print a line-break with "linebreak".

   (b) Prove the correctness of the algorithm. In other words, prove that the problem has the optimal substructure you used in the algorithm.

   (c) Analyze the running time and space requirements of your algorithm.

Answer:

(a) COST(i, s)
Input: takes in a single word i, and the space remaining on a line s
Output: returns a table with the cost and paragraphs

```
1:  x ← (i, s)
2:  if i = length(words) then
3:     return (s + 1)^3
4:  else if x is in table then
5:     return table [x]
6:  else if s = M then
7:     c,paragraph ← cost(i+1,s-length(words[i])-1)
8:     table[x] ← (c, words[i] +paragraph)
9:     return table [x]
10: else if length(words[i])≤ s then
11:    cost1, paragraph1 ← cost(i+1, s-length(words[i])-1) // adding on the same line
12:    cost2, paragraph2 ← cost(i, M) // adding on a new line
13:    cost2 ← cost2 + (s + 1)^3
14:    if cost1 ≤ cost2 then
15:       paragraph ← words[i]+paragraph1
16:       table [x] ← (cost1, paragraph)
17:    else
```

```
        paragraph ← paragraph2
        table [x] ← (cost2, paragraph)
    return table [x]
else
    c, paragraph ← cost(i,M)
    c ← c + (s + 1)^3
    table [x] ← (c, paragraph)
    return table [x]
```

DRIVER(M)
Input: takes a single parameter M which is the character limit for a single line
Output: prints the most optimal paragraph

```
a,b ← cost(0,M)
print b
```

(b) Optimal Substructure:

$$cost(i, s) = min \begin{cases} cost(i+1, s - length(words[i]) - 1) \\ cost(i, M) \end{cases}$$

Proof:

We have written in our optimal substructure in such a way that there are only two penalties to decide from. Either add the word on the current line with another pre-existing word or place the word on a new line. The algorithm after processing for one line looks at the sub paragraph as an entirely new paragraph and uses the table to figure out the cost that has already been incurred before the start of the sub paragraph. We do this by first calling the word on i+1 words to be compute the cost of keeping the word on the same line with s available space and then compare it with the cost of putting the word in the next line and then computing the total cost of 2 lines. This underlines the sub problem at the most basic level which can then be extended to the entire paragraph.

(c) The time and space of this algorithm would be $\Theta(nM)$ where n is the number of words and M is the character limit for a particular line. We are also using as 2-dimensional table with dimensions $n * M$

HONOR CODE AFFIRMATION

I affirm that I have carried out my academic endeavors with full academic honesty
Manav Bilakhia

REFERENCES

[1] 0-1 knapsack problem: Dp-10, Mar 2022.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.