Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

CSC373S: Algorithm Design, Analysis & Complexity

Tutorial 03

Monday January 30, 2017

based on notes by TA (LM 161 - Colin Li qiyang.li@mail.utoronto.ca)

Dynamic Programming

CLRS 15-4

Define Problem: n words, cost of a line = (# of extra spaces)^3), total cost = $\sum c_i$ (except last line)
Example:

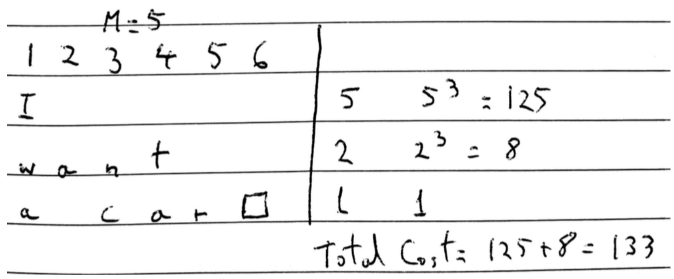

Figure 1: Visualizing an example problem

Greedy Solution: fit as many words possible for each line

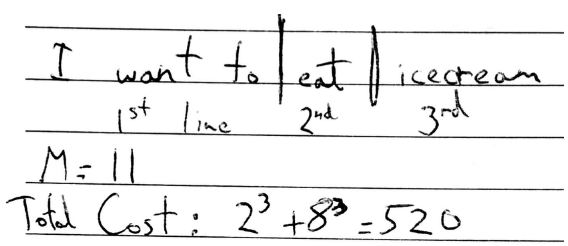

Figure 2: *Greedy Solution* example

However, we can find a better *optimal* solution (*see Figure 3 below*).
DP Solution

1. Define *semantic array & computational array*
   Semantic Array: extras$[i, j]$ = # of extra spaces if we fit i^{th} word, $i+1^{th}$ word, ..., j^{th} word in one line

   Computational Array: extras$[i, j] = M - (j - i) - \sum_{k=i}^{j} l_k$, where $(j - i)$ is the # of spaces between the words and the summation is the total length of all the words

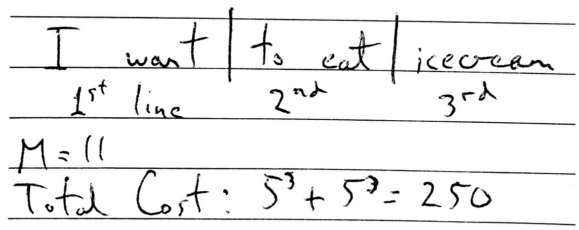

Figure 3: Better solution, showing failure of *Greedy Solution*

$$extras[i,j] = \begin{cases} extra[i,j-1] - l_j - 1 & (i < j) \\ M - l_i & (i = j) \end{cases}$$

$g(i,j)$ = cost of the line if we fit i^{th} word to j^{th} word

$$g[i,j] = \begin{cases} \infty, & \text{if } extras[i,j] < 0 \\ 0, & \text{if } extras[i,j] \geq 0, j = n \\ extras[i,j]^3, & \text{otherwise} \end{cases}$$

$f[i]$ = minimum total cost if we want to fit the first i words into one line / multiple lines.
$f[i] = \min(f[i-k] + g[i-k+1,i] | i \leq k \leq i)$

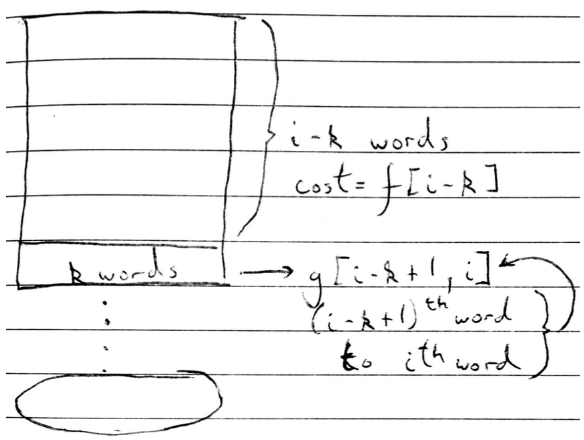

Figure 4: Visualization of $f[i]$ *in action*

Answer: $f(n)$

2. Algorithm:

```
def Solution():
  # O(n**2)
  for i = 1 to n:
    extras[i, i] = M - l_i
```

```

    for j = i + 1 to n:
      extras[i, j] = extras[i, j - 1] + l_j - 1

  # O(n**2)
  for i = 1 to n:
    for j = i to n:
      if extras[i, j] < 0:
        g[i, j] = float('inf')
      elif n = j:
        g[i, j] = 0
      else:
        g[i, j] = (extras[i, j]) ** 3

  # O(n**2)
  f[0] = 0
  for i = 1 to n:
    f[i] = float('inf')
    for k = 1 to i:
      if f[i - k] + g[i - k + 1, i] < f[i]:
        f[i] = f[i - k] + g[i - k + 1, i]
```

Runtime: $\mathcal{O}(n^2)$
Memory: $\mathcal{O}(n^2)$
However, this is a rather slow implementation (*see Figure 5 below*).

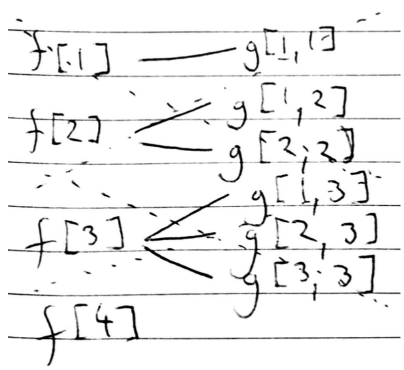

Figure 5: Why SOLUTION is so costly

Algorithm (IMPROVED):

```
def SimplifiedSolution():
  f[0] = 0
  for i = 1 to n:
    f[i] = float('inf')
    extras[i] = M - l_i
    for k = 1 to i:
```

```
extras[i - k + 1] = extras[i - k + 1] - l_i - 1
if extras[i - k + 1] < 0:
  g = float('inf')
elif n = i:
  g = 0
else:
  g = (extras[i - k + 1]) ** 3
if f[i - k] + g < f[i]
  f[i] = f[i - k] + g

  # p[i] = the last word in the previous line
  #        if we fit the first i words
  p[i] = i - k
```

Example Run:

```
# Changes in extras
i = 1
extras[1, 1] = extras[1]
i = 2
extras[1, 2] = extras[1]

# Changes in p
p[8] = 5, 1 to 2 first
p[5] = 2, 3 to 5 second
p[2] = 0, 6 to 8 last
```