CIS 606 Homework 7 Fall 2019

Name: **Reza Shisheie**

Login ID: **reshishe**

1. Give an integer length x, and the integer cutting points $x_1, x_2, ..., x_n$ (where $1 \leq x_i \leq x$ for $1 \leq i \leq n$), find the minimum cost of cutting a piece of iron bar of length x at the n points at distance $x_1, x_2, ..., x_n$ from the left-hand end (given that cutting a piece of bar of length y at any cost y). Devise a polynomial -time algorithm for the problem using dynamic programing technique.

   The solution to this problem is based on the location of cut. The bar starts at position $l = 0$ which is denoted by l_0 and ends at position $l = L$ which is full length and is denoted by l_n.

   A visual diagram of the bar with cuts is shown in Figure.1.

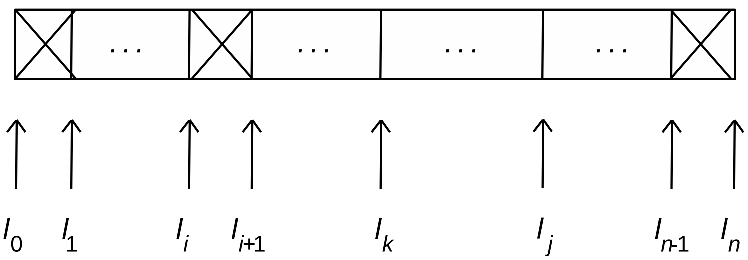


Figure 1: Cutting points. It starts from 0 and cuts through the bar to n

   Cutting points for a piece of rod which starts at position l_i and ends at position l_j is located at l_k. Therefore $i < k < j$.

   Let $m[i, j]$ be dynamic variable which is the minimum cost of cuts from point i o point j.

$$m[i,j] = \begin{cases} 0 & \text{if } j - i = 0 \text{ or } j - i = 1 \\ m[i,k] + m[k,j] + l_j - l_i & \text{otherwise} \end{cases} \tag{1}$$

   where $l_j - l_i$ is the length of rod from position i to j.

Two algorithms are provided to find optimal cut and then print where cuts should take place.

First algorithm –MIN-COST-CUT(L, c, l)– finds the most optimal cut, where L is the full length of the rod, c is the number of cuts, and l is the location of cuts.

The Second algoritgm –MIN-COST-CUT(L, c, l)–, prints the precedence of cuts.

MIN-COST-CUT(L, c, l)

```
// Reeconstruct l by adding l_0 = 0 and l_n = L to the beginning and end of l
l = [0, [l], l_n]
// n is the maximum number of pieces, which is number of cuts+1.
n = c + 1
// generate matrix m to accumulate cost and matrix s to store location of cut and initilize to 0.
m = zero(n × n)
s = zero(n × n)
for l_c = 2 → n
    for i = 0 → n − l_c
        j = i + l_c
        m[i, j] = ∞
        for k = i + 1 → j − 1
            q = m[i, k] + m[k, j] + l_j − l_i
            m[i, j] = ∞
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
print(m[0, n])
PRINT-CUT(l, s, i, j)
```

Time complexity of MIN-COST-CUT(L, c, l) is $O(n^3)$ since it has 3 internal loops. It is actually almost identical to the matrix multiplication time complexity.

PRINT-CUT(l, s, i, j)

```
// if bar length is 2, then just return where the cut last cut is. This is the minimum cutable piece
if (j − i) = 2
    return print(l_{s[i,j]})
// if bar length is 1 or 0, return nothing. Bar is not cuttable.
elseif (j − i) = 1 or i = j
    return ∅
// if bar length is larger than 2, then print where the last cut is and then cut the left and right sides
else
    print(l_{s[i,j]})
    PRINT-CUT(l, s, i, s[i, j])
    PRINT-CUT(l, s, s[i, j], j)
```

2. Write a program in Python 3 to solve previous problem.

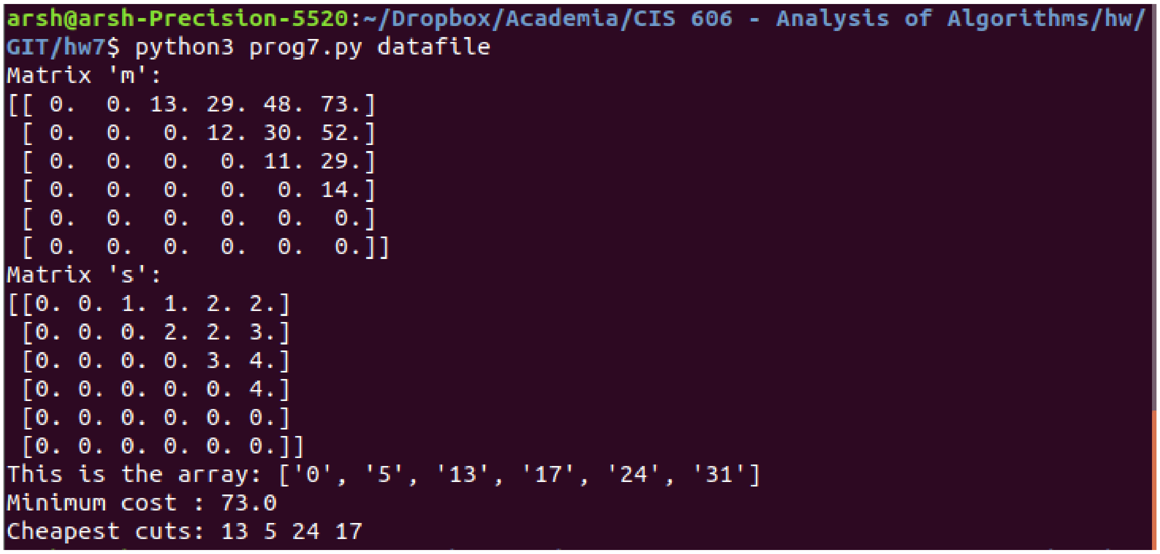
   A snippet of code is hown in Figure2. Notice the uses uses "*numpy*" library.

```python
import numpy as np
import sys

def print_s(l,s,i,j):
    # if bar length is 2, which is minimum length -> print the last cut
    if (j-i)==2:
        return print(l[int(s[i,j])], end = ' ')
    # if bar length is 1, no cut is possible
    if (j-i)==1 or i==j:
        return None
    else:
        # ptint current cut and then cut left and right pieces recursively
        print(l[int(s[i,j])], end = ' ')
        print_s(l,s,i,int(s[i,j]))
        print_s(l,s,int(s[i,j]),j)


# Taking arguments and make array "l"
arguments = (sys.argv)
f = open(arguments[1], "r")
line1 = f.readline().rstrip('\n').split(" ")
line2 = f.readline().rstrip('\n').split(" ")

# add "0" to beginning and "L" to the end
l=line2
l.insert(0,"0")
l.append(line1[0])

#make "m" and "s" matrices
m = np.zeros((6,6))
s = np.zeros((6,6))
n = len(l)-1

for l_c in range (2,n+1):
    for i in range(0,n-l_c+1):
        j=i+l_c
        m[i,j]=99990
        for k in range(i+1,j):
            q = m[i,k]+m[k,j]+(int(l[j])-int(l[i]))
            if q<m[i,j]:
                m[i,j]=q
                s[i,j]=k
print("Matrix 'm':")
print(m)
print("Matrix 's':")
print(s)
# show the array with start, end, and location of cuts
print("This is the array: "+str(l))
# print minimum cost
print("Minimum cost : "+ str(m[0,n]) )
# print locations of cust using function "print_s" recusrsively
print("Cheapest cuts:", end = ' ')
print_s(l,s,0,n)
print("")
```

Figure 2: Snippet of code

Cheapest cut costs 73 and the precedence of cuts are:

$[13, 5, 24, 17]$ which is $13 \rightarrow 5 \rightarrow 24 \rightarrow 17$



```
arsh@arsh-Precision-5520:~/Dropbox/Academia/CIS 606 - Analysis of Algorithms/hw/
GIT/hw7$ python3 prog7.py datafile
Matrix 'm':
[[ 0.   0. 13. 29. 48. 73.]
 [ 0.   0.   0. 12. 30. 52.]
 [ 0.   0.   0.   0. 11. 29.]
 [ 0.   0.   0.   0.   0. 14.]
 [ 0.   0.   0.   0.   0.   0.]
 [ 0.   0.   0.   0.   0.   0.]]
Matrix 's':
[[0. 0. 1. 1. 2. 2.]
 [0. 0. 0. 2. 2. 3.]
 [0. 0. 0. 0. 3. 4.]
 [0. 0. 0. 0. 0. 4.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
This is the array: ['0', '5', '13', '17', '24', '31']
Minimum cost : 73.0
Cheapest cuts: 13 5 24 17
```

Figure 3: Results

This is how cuts happen:

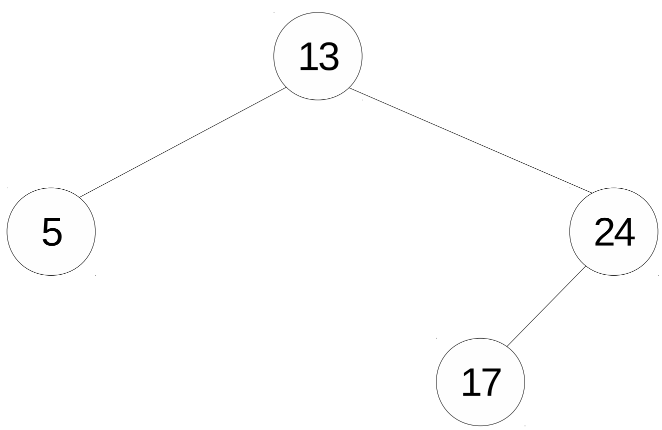


Figure 4: Precedence of cuts