CS350 Homework 4

Russell Miller

March 2, 2011

24.3-6

```
MOST-RELIABLE-PATH(G, source)
mark each vertex as distance infinity
mark each vertex's previous vertex as undefined
mark the source's distance as 0
put all of the vertices in a Queue
while the Queue is not empty
  find the vertex u with the least distance
  if that vertex is the target
    create a new sequence S
    while previous vertex of u is defined
      add u at the beginning of S
      continue to u's previous vertex
    remove that vertex from the Queue
  for each neighbor v of that vertex
    add the negative logs of u and (u,v)
    if the sum is less than the current value of v's distance
      set v's distance to that sum
      set v's previous vertex to u
```

15.2-1

For this problem I wrote the Matrix-Chain-Order and Print-Optimal- Parens functions in Python, using the pseudocode from the textbook. My code is below:

```
import sys

def main():
    # dimensions of each matrix
    a = [5,10,3,12,5,50,6]
    m,s = matrix_chain_order(a)
    for line in m:
        print(line)
    for line in s:
        print(line)
    print_opt_parens(s,1,6)
    sys.stdout.write('\n')
```

```
def matrix_chain_order(p):
    n = len(p) - 1
    m = []
    s = []
    for i in range(n):
        m.append([])
        s.append([])
        for j in range(n):
            m[i].append(0)
            s[i].append(0)
    for l in range(2,n+1):
        for i in range(1,n-l+2):
            j = i + l - 1
            m[i-1][j-1] = 9999
            for k in range(i,j):
                q = m[i-1][k-1] + m[k][j-1] + p[i-1]*p[k]*p[j]
                if q < m[i-1][j-1]:
                    m[i-1][j-1] = q
                    s[i-1][j-1] = k
    return m,s

def print_opt_parens(s,i,j):
    if i == j:
        sys.stdout.write("A_" + str(i) + " ")
    else:
        sys.stdout.write("(")
        print_opt_parens(s,i,s[i-1][j-1])
        print_opt_parens(s,s[i-1][j-1] + 1, j)
        sys.stdout.write(")")

if __name__=="__main__":
    main()
```

The output of the program is:

```
[0, 150, 330, 405, 1655, 2010]
[0, 0, 360, 330, 2430, 1950]
[0, 0, 0, 180, 930, 1770]
[0, 0, 0, 0, 3000, 1860]
[0, 0, 0, 0, 0, 1500]
[0, 0, 0, 0, 0, 0]
[0, 1, 2, 2, 4, 2]
[0, 0, 2, 2, 2, 2]
[0, 0, 0, 3, 4, 4]
[0, 0, 0, 0, 4, 4]
[0, 0, 0, 0, 0, 5]
[0, 0, 0, 0, 0, 0]
((A_1 A_2 )((A_3 A_4 )(A_5 A_6 )))
```

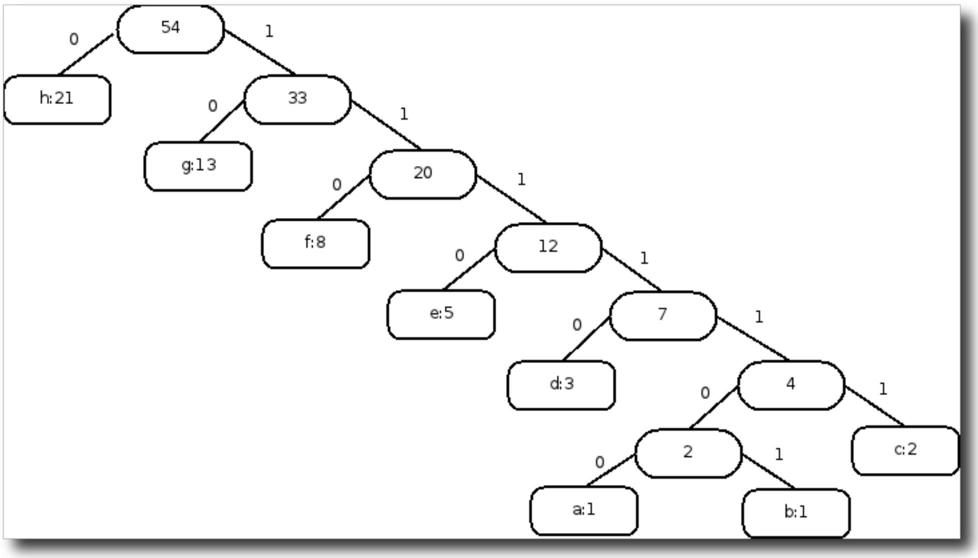
16.2-2

For this algorithm, I'm going to go with a few assumptions.
First of all, W is a multiple of 100. Also, each item is a value n * 100 for values of n up to and including n * 100 = W.
This builds a table like the one in class, and returns the very last value of the table.

```
KNAPSACK(W, n)
create new table K [1..n , 100..W]
set each value of column 1 to 100
set each value of row 100 to 100
for i = 2 to n
  for j = 200 to W
    max = -1
    current_sum = 0
    if i*100 <= j
      current_sum = put item i*100
      if j-i*100 > 0
        current_sum += K[i-1][j-i*100]
      endif
      if current_sum > max
        max = current_sum
      endif
    else if K[i-1][j] > max
      max = K[i-1][j]
    endif
  endfor
endfor
return K[n][W]
```

16.3-3

I used the algorithm in the book to build a tree, where the edges are the encodings.



So the encodings would be:

```
a: 1111100
b: 1111101
c: 111111
d: 11110
e: 1110
f: 110
g: 10
h: 0
```

The algorithm is able to handle any value of n. With this sequence of values, however, a very inefficient encoding is created.

16.3-9 - *I am not going to attempt this one.*