Homework #4

Chris Dellomes
Professor: Ray Toal
CMSI 282: Algorithms
Loyola Marymount University

May 4, 2015

1.

node	0	1	2	3	4	5	6	7	8	9
S	0	0	0	0	0	0	0	0	0	0
A	∞	7	7	7	7	7	7	7	7	7
B	∞	∞	11	11	11	11	11	11	11	11
C	∞	6	5	5	5	5	5	5	5	5
D	∞	∞	8	7	7	7	7	7	7	7
E	∞	6	6	6	6	6	6	6	6	6
F	∞	5	4	4	4	4	4	4	4	4
G	∞	∞	∞	9	8	8	8	8	8	8
H	∞	∞	9	7	7	7	7	7	7	7
I	∞	∞	∞	8	8	8	8	8	8	8

2. The professor's method is not valid because it fails with certain negative values. For example, Let there be three vertices a, b, and c. Let there be edges (a, b), (b, c), and (a, c) where the first two edges are each valued at -1 and the edge between a and c is valued at -3. Following the professor's method, we add 4 to each edge so that the first two edges are worth 3 and the third edge is worth 2. The professor's method states then that the shortest path from a to c is the edge (a, c) at a value of 1. However, in the original situation, the shortest path from a to c is through edges (a, b) and (b, c) at a value of -2.

3. Every edge in the graph has a length in the range (0, W) which means that all distance values are in the range (0, W($|V|$ -1)) since the shortest path has at most $|V|$ - 1 edges with each edge being of weight of at most W. We implement a heap on the distance values by making an array of size W($|V|$ - 1) + 2 indexed by all possible distance values with each index of the array, i, a pointer to a linked list of values where distance = i. Thus, insertion operations are constant time since we iterate through the array till the appropriate linked list is found to append the incoming element to. The implementation of the heap on distance values makes the time complexity of makeheap $O(|V|)$. When running Dijkstra's algorithm, the heap's min value is repeatedly increasing, which means the deletemin operation only looks at each array element once looking for the minimum. This makes the complexity of deletemin $O(W|V|)$. For the decreasekey operation, we insert a new copy of the incoming element into the corresponding value linked list without needing to remove previous copies. There are at most $|E|$ decreasekey operation instances, which makes the total complexity of all decreasekey operations $O(|E|)$. Summing the three time complexities involved in Dijkstra's algorithm, we find that $O(|V|) + O(W|V|) + O(|E|)$ $= O(W|V| + |E|)$.

4. (a)

vertex included	edge included	cost
A	-	0
B	AB	1
C	BC	3
G	CG	5
D	GD	6
F	DF	7
H	FH	8
E	AE	12

(b)

(A,B)
(A,B) & (D, G)
(A,B) & (F, G), (D, G)
(A,B) & (F, G), (D, G), (G,H)
(A,B), (B,C) & (F, G), (D, G), (G,H)
(A,B), (B,C), (F, G), (D, G), (G,H)
(A,B), (A, E), (B,C), (F, G), (D, G), (G,H)

5.
```
#algorithm at http://en.wikipedia.org/wiki/Subset_sum_problem
def subset_sum(input_list, input_sum):
    s = [0]
    p = 5
    c = 2 ** -p
    if input_sum == 0:
        return True
    for i in input_list:
        t = []
        for y in s:
            t.append(i + y)

        union_of_s_and_t = s
        union_of_s_and_t += t
        union_of_s_and_t.sort()
        s = []
        k = union_of_s_and_t[0]
        s.append(k)
        for z in union_of_s_and_t:
            if k + c * input_sum / len(input_list) < z and z not in s:
                union_of_s_and_t[0] = z
                s.append(z)

    for i in s:
        if (i > 0 and (1 - c) * input_sum <= i <= input_sum) or \
        (i < 0 and (1 - c) * input_sum >= i >= input_sum):
            return True
    return False
```