Problem 1

(a)

Algorithm:

1. Connect all the possible routes of hiking by connecting the n POIs with m routes with the weight of 1(If there is a route from m1 to m2, connect them with the weight of 1) so that we can turn this problem into a directed graph with all edges having size of 1.
2. Run the maximum flow network algorithm (either Ford-Fulkerson or Goldberg) and store it inside variable maximumFlow
3. If the value of maximumFlow is greater than or equal to the input value k, return “yes”
4. If not (the value of maximumFlow is less than the input k), return “no”

Runtime:

The runtime of the algorithm is same with the maximum flow network algorithm (let’s say Ford Fulkerson algorithm). The connecting of the m routes can be done in constant time and after running the maximum flow algorithm, it is simply comparison with the input value k, which is also constant time. Constant time can be cancelled when calculating the big-oh time. According to <https://stackoverflow.com/questions/33565995/time-complexity-of-the-ford-fulkerson-method-in-a-flow-network-with-unit-capacit>, the runtime of the Ford-Fulkerson algorithm is O(mf), where m is the number of edges and k is the value of maximal flow.

Proof:

My algorithm basically turns the question into the maximum flow network program to solve the sign problem. Hiking is an activity where there exists one ending point and one starting point. We can think of the starting point (the starting POI) as the source node (node s) of the maximum flow problem and the ending point (the ending POI) as the sink node (node t). Since the nodes (POIs) are connected by edges with size 1 if they are connected, we can run the maximum flow algorithm (either Ford-Fulkerson or Goldberg algorithm). The maximum flow algorithm will return the maximum amount of flow (sum of weights), which is the number of black-boxes needed, because every path from one POI to another POI the hiker can take has the size of 1 (despite the length and difficulty of the path) and the maximum network flow algorithm ensures that you consider every possible path from the source node and the sink node. After considering every path, the maximum flow algorithm returns the maximum amount of flow that can flow from the node s to node t, which is also the maximum number of black boxes required.

Since the runtime of the sign problem is O(m\*f), which is a polynomial time algorithm, meeting the expectation of the problem.

(b)

We can reduce the sign problem into the set cover problem.

1. Consider the graph that has the POIs connected according to whether there exists a path between them (paths are connected and are edges)
2. We can simply write this question as a vertex problem we covered in class. The vertex problem is question whether there is a vertex of size k (given by user) that covers all the nodes (has at least one endpoint to all the nodes in the graph). We can treat the sign problem as vertex problem because the sign problem is looking for number of vertices (edges) that covers all the nodes so that we can place the black boxes to those edges, with the minimum number of edges. If the vertices account for all the nodes, it means that we can place the black boxes on those edges (paths). Since the sign problem requires hikers to move from one POI to another POI (move from one node to another node), the vertex problem can be treated same as the sign problem.
3. According to the lecture slides given in class, it is proven that we can reduce the vertices problem into set cover problem.
4. Since we can treat the sign problem as the vertices problem and can reduce the vertices problem into set cover problem, we can reduce the sign problem into the set cover problem.

Example)

1. Is there a sign cover problem of size <= 2?

b

a

e

k

j

f

i

g

c

h

d

1. Define corresponding set cover problem
2. Find a set cover problem of size
3. Find corresponding solution for sign problem.

This is a hiking problem where the graph is directed from the lower to upper, according to their layers, so we only write the edges that it can reach by going up the mountain, not down the mountain (If node 5 covers node 2 and if node 2 covers node 1, it means that node 5 can cover node 1).If ignore the starting node since the starting node naturally includes every root to the destination, meaning that it includes all the vertices, which always gives the answer of the question to 1, which is incorrect in most cases.

U = {a, b, c, e, f, g, i, j, k}

S1 = {None}

S2 = {a}

S3 = {e}

S4 = {b}

S5 = {g, k, f, a, e, b}

S6 = {c, i, j, a, e, b}

S7 – Ignore Starting Point

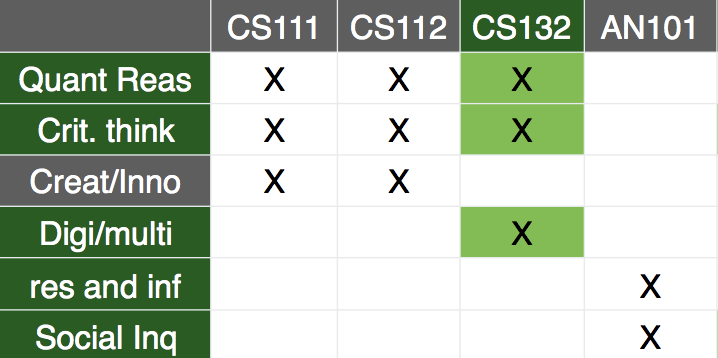
Method:

1. Choose S5, which has highest size that covers g, k, f, a, e, b
2. Choose S6, which has highest size that covers c, i, j, a, e, b
3. Since S5 and S6 can cover all the vertices, they are chosen as the nodes that have the minimum size, which has size 2.
4. The nodes S5 and S6 are chosen, which means that from the starting point (node S7), we should choose the path that can reach S5 and S6, which are h and d – size 2
5. This means that the sign cover problem for this question should also return 2, which it does, since the maximum network flow algorithm returns 2 for the question.

We can also reduce the set cover problem into sign problem.

1. Consider the set cover problem where we need to cover the entire set with the minimum number of classes (the example given in class)
2. Convert the set cover problem into a graph so that there is a directed edge from a class to the hub if the class covers that hub unit. (This will create a bipartite graph since classes will only direct to hubs)
3. Create source and sink node so that we can convert it into a maximum flow question.
4. By running the Ford-Fulkerson algorithm, the total number of classes will be returned.

Is there a set cover of size <= 3?



1. Define a corresponding sign problem
2. Find a sign problem of size 3
3. Find corresponding solution for set cover problem

For the graph above, (was too complicated to draw the weights). The weight of the following graph is, from a to f to t, the weight is 1. From the classes 1 to 4 to hubs a to f, the weight is very large number, let’s say 10000. From the source node to classes 1 to 4, the weight is according to the number of hub courses it can take. In other words, if class 1 (class CS 111) has three hub courses, the weight of that edge is 3.

If we conduct the maximum network flow algorithm, we will return back 6, which means that the four classes can cover all the 6 required hub units. Class 1 will connect to hubs a, b, and c, giving 3 to the sink node. Class 2 will do the same with class 1. Class 3 will connect to hubs a, b, and d. However, since a and b are already taken hub units, we simply get the value of 1 from hub d from class 3. Class 4 connects to hub units e and f, giving the sink node 2 more flows. After running the algorithm, we can realize that class 2 is same as class 1 in terms of the hub units, which means that it is unnecessary, despite the large number of hub units it provides. Therefore, we can conclude from this question that there are 6 hub units taken with 3 classes: classes 1, 3, and 4, which is the same if we have conducted the set cover problem.

Since the set cover problem can be reduced into sign cover problem and the sign cover problem can be reduced to the set cover problem, we can conclude that the sign placement problem is equivalent to the set cover problem.

Problem 3

The subset sum problem is a problem which there exists a set {a1,a2,…,an,b} and {x1,x2,…,xn} where a1x1 + a2x2 + … + anxn = b. The subset of xs are the values only of 0 and 1. According to the question, we assume that there exists a polynomial algorithm for the solution to the subset sum problem. With the information, (according to the hint), we can check if there is a solution to the input (0,a2,…,an,b), using the subset sum polynomial algorithm. If the algorithm returns a possible solution, this means that x1 equals 0 because it means that a1x1 equals 0, meaning that x1 equals 0. If there is no solution for the subset sum algorithm for x1 = 0, it means that x1 could equal 1. Assuming that x1 equals 1, a1 \* x1 = a1, implying that we there must exist a2x2 + … + anxn = b – a1, after simplifying and rearranging the problem. In other words, we run the polynomial algorithm for (a2,…,an,b-a1) and (x2,…,xn). Here, if there is a solution, it means that x1 equals 1. However, if a2x2 + … + anxn does not equal b – a1, it means that there is no solution to this problem since x1 is neither 0 or 1.

For each combination of a and x, we ran the polynomial algorithm 2 times (for a1…an with x1…xn and a2…an with x2…xn) and have down three comparisons. Since running polynomial algorithm twice is still a polynomial algorithm and the comparisons whether there is an answer to the question or not is simple constant time, this means that we have a polynomial algorithm to find a 0-1 sequence of (x1,x2,…,xn), given that there exists a polynomial algorithm for the subset sum problem.