James Sweetman (jts2939)
Jung Yoon (jey283)
Unique #51835

Assignment #6

Q1.

Given an input of a1,...an, we will use a boolean 3D array, A[M+1][M+1][n+1] where M = ai (the individual sum of each of the three sets). This means that A[M][M][n] is true if there exists a solution. Each individual cell A[A,B,C] is true if and only if there exist two disjoint subsets such that when comprised of elements in {1,...C}, the sum of the elements in one subset is A and the sum of the elements in the other subset is B.

Base Case:
M[0,0,0]=1
M[A,B,0] = 0 when A+B>0

Recursive Definition:

M[A,B,C] = M[A,B-ai,C-1] || M[A-ai,B,C-1] || M[A,B,C-1]

Q2.

First, let's recognize that this is a variation of the subset sum problem with the changes being that:

1) each solution must have *n* elements
2) we are looking for solutions sets where *W* = the sum of the square of each element instead of just the sum of elements.

From what has been given, we can assume that the largest element in the set of possible elements would be floor(sqrt(*W*)). That is, the set of possible elements given *n* and *W* would be when in ascending order: *R* = {0,...,floor(sqrt(*W*))}. We would, then, need to find all of the possible subsets of *R* (which we will call *S*) such that the subset has n elements and such that the sum of the square of the elements in S equals *W*.

Data Structures: We will have an array holding each of the possible elements in R and we will also use a 2D array in the form s[*n*][*K*] as our main data structure that we will fill in a bottom-up tatic. We iterate over each element in R and also iterate through every element in the s array.

Base Case:

- returns 0 when K = 0 and n = 0
- returns 0 when K=0 and n!=1
- returns 1 when K=0 and n==1

Recursive:

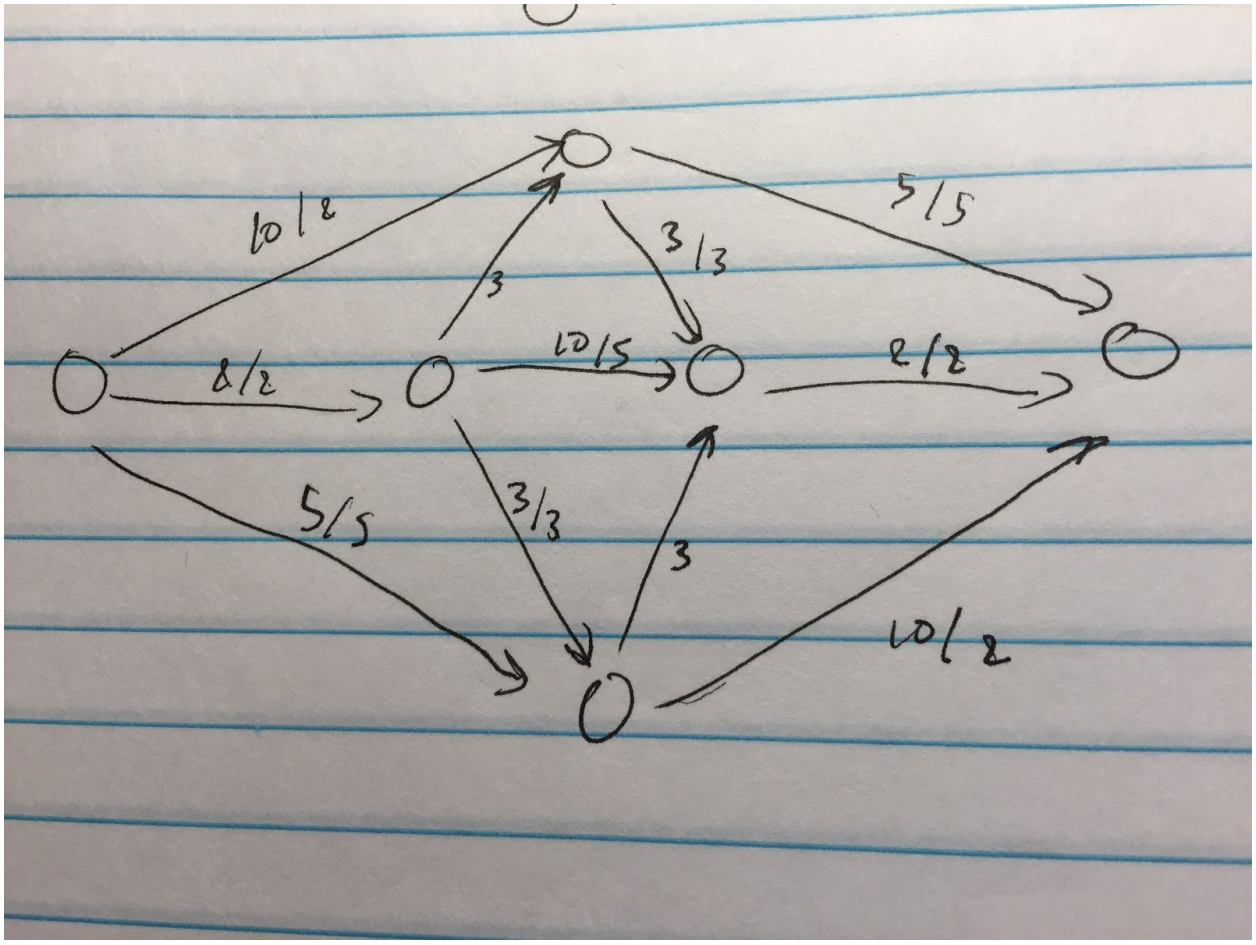
We start off with current_num = K and subtract that number. If it is possible to get a sum of that number (array element corresponding to that number is not zero) then add it to the array element corresponding to the current number.

Q3.

a. {s}, {u,v,t}
   {s,v}, {u,t}
   {s,u,v},{t}
b. Min. capacity is 4 with cut {s,u} {v,t}

Q4.

a. 18, no. As you can see, this is one flow that is greater than 18 (not implying it is the max. flow though).



b. {s and the top node, V - s and top node}, capacity is 21

Q5.

Adjust the problem by building a network flow. There is a node v_k for every client k and node u_j for every base station j. Connect the bases and edges if the client is within the range of the base station. Connect a source s to each of the client nodes with an edge of capacity 1. Connect each of the base stations to sink t with an edge of capacity L. Now we can apply Ford-Fulkerson and if there is a way to connect all clients to base stations, then there will be a s-t flow with value n.