In order to prepare this submission, the following resource(s) has been used:

- Discussion with fellow students: *Victor Gan*, *Sarah Huber*, and *Evan Peng*.

1. Two halloween trick-or-treaters share a haul with n treats. They have agreed on a value for each treat, given by the positive integers $x_1, x_2, \ldots, x_n$. They want to split the treats into two sets of equal value. Design an algorithm that, given the values x_i, determines whether it is possible for the trick-or-treaters to fairly split the treats and if so, outputs a fair split. Your algorithm should run in time $O(nT)$, where T is the sum of the x_i's.
   Answer. Let $P(i, j)$ denote whether there exists a subset of the first j elements which sum to i. The answer to the question will be at $P(\frac{T}{2}, n)$. If we have an equal split of the treats, each should have the sum $\frac{T}{2}$ in order to be fair; thus it suffices to find one subset in the whole set which satisfies this condition.

   For the base case $P(0, 0) = true$ because it satisfies the proposition. For each $P(0, j) = true$ because we can always have an empty subset; and for each $P(i \geq 1, 0) = false$, because the base set will be empty and we can't satisfy the condition. Needless to say for $P(i < 0, j < 0) = false$.

   For other $P(i, j)$ we have:

$$P(i,j) = max \begin{cases} P(i - x_j, j-1) & \text{if If we decided to pick the jth element} \\ P(i, j-1) & \text{if If we decided to leave the jth element} \end{cases}$$

   If we have $P(\frac{T}{2}, n) = true$ then we know a fair split exists, and by tracing back the decisions we've made during filling the table, we can produce the target set.

2. Your task is to write pseudocode that makes precise the informal description provided in the paper for finding all paths from node 1 to node N that have length at most $f(1) + e$.
   Answer. This is technically a bounded DFS search! Following the same terminology as the paper, let's say $f(i)$ is the length of the shortest path from i to N. Furthermore, let's assume $E(i, j)$ is the weight of the directed edge from i to j. $E(i, j) = \infty$ if no such edge exists. The following function tries to solve the problem using the proposed method.

```
void boundedDFS( node v, distance d, stack s, bound b)
    if v == N then
        print stack elements as an answer;

    for each u in the set of directly reachable neighbors of v
        if d + E(v, u) + f(u) < b then
            s.push(u)
            boundedDFS( u, d + E(v, u), s, b)
            s.pop()
        end
end function
```

   Note that even though the pseudocode is a recursive function, the traversing order of the nodes is the same as the method proposed in that paper; and it's basically the same. And also since the graph is a DAG, no node marking is necessary.

3. A substring α of a string S is called a tandem array of β-called the base-if α consists of more than one consecutive copy of β.

   (a) Give an example to show that two maximal tandem arrays of a given base β can overlap.
   Answer. S = CABCABC while β=CABC.

   (b) Give an $O(n)$ time algorithm that takes a string S of length n and a base β of length at most n as input, finds every maximal tandem array of β in S, and outputs the pair (s, k) for each occurrence.
   Answer. With some modifications, we can use the Knuth-Morris-Pratt string search algorithm which has the time complexity $O(n)$. Instead of searching S for β, we define a string $\gamma = \beta^i$, where $i|\beta| < n$. In other words, we concat β to itself as long as the length of the new string is less than length of S and call the resulting string γ. Then we perform KMP string search algorithm on S and γ. The only difference is that, if we fail to match the i-th character (or matched all the characters) in γ, if $i > |\beta|$, we print the location of start of the match and $i/|\beta|$ as the pair (s, k). This algorithm will generate all the maximal tandem strings of base β as it tries to identify the longest matching characters over γ.

4. Given two strings X and Y of lengths n and m respectively, describe an efficient algorithm to compute the best global alignment of all pairs of circular shifts of input strings x and y over the alphabet A, C, G, T. Assume that the alignment is scored using a matrix δ just as in Exercise 6.26 (page 197) of the Algorithms textbook of Dasgupta, Papadimitriou and Vazirani, or as in the class lecture notes from October 21.

   Answer. Let's assume $X = x_0x_1x_2 \dots x_n$ and $Y = y_0y_1y_2 \dots y_m$, we construct X' and Y' by concatenating the first $n-1$ characters of X to X and the first $m-1$ characters of Y to Y. More formally, $X' = x_0x_1x_2 \dots x_nx_0x_1x_2 \dots x_{n-1}$ and $Y' = y_0y_1y_2 \dots y_ny_0y_1y_2 \dots y_{m-1}$. Note that each contagious substring of length n in X' and m in Y', represent one possible shifts. We can find the best alignment for X' and Y' in $(2m-1)(2n-1) = O(mn)$ using the previously discussed algorithm for finding the best alignment. Using the discussed method, we will end up with a $(2m-1) \times (2n-1)$ table of maximum possible values. Imagine selecting a m by n box in the bigger table. This m by n rectangle, represents a smaller maximum alignment problem with one possible shift of X and one possible shift of Y, except that the values are filled according to the bigger problem with X' and Y'. Obviously this subproblem must be optimum in terms of using the characters represented by the rectangle (can be proved by contradiction against the assumption that the bigger rectangle has to be optimum). Let's say $T(i, j)$ and $T(i+m-1, j+n-1)$ are the values at the upper left corner and bottom right corner of the rectangle. We claim that the maximum value brought by this rectangle is $T(i+m-1, j+n-1) - T(i-1, j-1)$. The idea is simple, $T(i+m-1, j+n-1)$ means the best alignment with the first $i+m-1$ characters of X' and $j+n-1$ characters of Y'. The same argument goes for $T(i-1, j-1)$, thus by subtracting the values, we can find the best alignment for those particular substrings. Consequently, we only have to locate the rectangle with the maximum value. Such a rectangle could be located while filling the table $T(i, j)$, by simply subtracting the best value found from $T(i-m, j-n)$ and keeping track of the maximum rectangle so far.