In case the upper limit (5) to the number of bunnies didn't exist, the above brute force solution would clearly become exponential, as it involves calculating all permutations of all lengths of the set of bunnies. The following recursive approach can possibly be used in such cases. The approach works for most cases except when there is a zero-cycle in the graph. I plan to provide an algorithm that will detect a zero cycle and subsequently solve the problem at hand, but I have not coded it yet.

Why get rid of negative weights?

The recurrence used is

$$T(v,t) = \{v\} \bigcup \max_{u \in Adj[v]} \{T(u, t - w_{uv})\}$$

, with the mentioned bases cases. In the recurrence for t to hit the base case it must eventually decrease to either 0 or $delta[v]$ for some v. However with negative weights it's possible that the t keeps on increasing with successive recursion calls (or get stuck in a loop of values), thereby causing a recursion overflow.

After the transformation with Johnson's method we have a directed graph with only *non-negative* edge weights, and we can work on this graph from here on.

What's up with zero cycles in the transformed graph?

After the previous transformation is completed, the zero edges do not bother us much. We have ensured that t doesn't increase anymore, and if not all edges have 0 weight, t would eventually decrease. What creates a difficulty here is not the zero edges but the zero cyles. For a plain example say $(u - v)$ is a zero cycle, meaning both $w_{uv} = w_{vu} = 0$. From our recurrence, we need $T(u,t)$ to calculate $T(v,t)$, and $T(v,t)$ to calculate $T(u,t)$; and there we are stuck in a loop. So we must ensure that the graph does not contain zero cycles at all.

Approach 1 (failed) : Use Johnson's to transform all the non-negative weights to strictly positive weights without disturbing the shortest paths

Say there exists a mapping $f : V \to \mathbb{R}$, such that $\bar{w}_{uv} = w_{uv} + f(u) - f(v) > 0, \forall$ u and v, with a strict greater than unlike the usual Johnson's method; where $\bar{w}$ represents the weights after transformation. We will show that such a mapping is not always possible and will construct a example countering the approach.

fix a u, we must have $f(u) + w_{uv} > f(v), \forall\, v$

for a particular $v \neq u$, we must also have $f(v) + w_{vu} > f(u)$

Combining the above equations we have, $f(v) - w_{uv} < f(u) < f(v) + w_{vu}$

Choose $w_{uv} = w_{vu} = 0$ (the choice is valid as non-negative weights are allowed),

then, $-w_{uv} < f(u) - f(v) < w_{uv} \implies |f(u) - f(v)| < w_{uv} = 0$, which is a contradiction! Or zero cycles of length 2 would not allow the application of Johnson's algorithm to tranform non-negative edge weights of a graph to strictly positive weights.

Approach 2 (not coded) : Shrink all the zero cycles to a single vertex and work on the transformed graph

A brief description of the algorithm would be as follows.

- *identify the zero cycles*. How? Fix a source s. And remember the edge weights are non-negative now. If $(v_1, \cdots, v_k)$ is a zero cycle, i.e. $w_{v_i v_{i+1}} = 0, \forall\, i$, then the shortest distance $\delta(s, v_i) = \delta(s, v_j)$ for all $1 \leq i, j \leq k$, or all the veritces of a zero cyle would be equidistant from any source. Therefore our search space is strictly reduced now. We only need to run a single source shortest path (say dijkstra) from any random source, club the vertices which have equal distances, and search for a zero cycle inside them.

- Say $(uv \cdots z)$ is one such group of k equidistant vertices. Here, in the subgraph of these k vertices we can either run a DFS to obtain all cycles, and check for zero-sums, or we can obtain the $(k-1)!$ circular permutatioins and check for consecutive zero weights. The value of k would determine what is more efficient.

- Once such a cycle is detected, shrink the cycle into a single vertex, and for a vertex outside the cycle, the edge weight is updated to its shortest distance from the cycle. One particular way to do this would be the following. If $(uv \cdots z)$ is the zero cycle with u as the smallest vertex (assuming vertices are indexed), then in the adjacency matrix treat u as the entire cycle. For any vertex q outside the cycle, update w_{uq} and w_{qu} as the shortest distance of q from and to the cycle. In other words, for example, $w_{uq} = \min_{v \in cycle} w_{vq}$. Now pretend like the other vertices in the cycle aren't there. Or treat $w_{vq} = \infty$ for all q and for $v \in cycle$ but $v \neq u$. Now the transformed graph doesn't contain zero cycles anymore.

- From here we are good to go, there's nothing stopping t from reducinig, no increasing, no getting stuck in loops, t must reduce, and eventually hit the base cases. Now run the recursion to obtain the set $T(n-1, time\_limit)$. If vertex u is present in the set, expand it to include all the vertices of the cycles. And we're done!