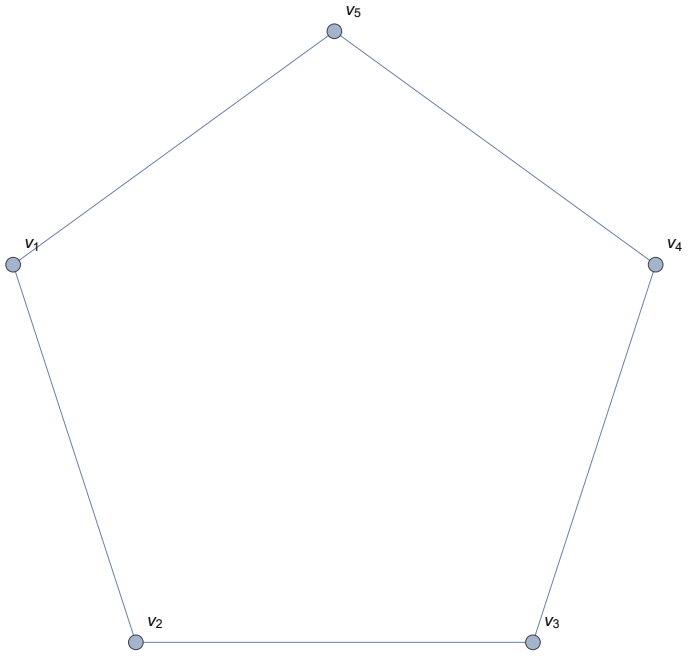
Nicole is running on a graph given by ‘edge’. She is now at vertex ‘x’ and needs to finish at home vertex ‘y’. Nicole wants to exercise as much as possible while she only has ‘n’ chocolate bars, each of which provides the energy for Nicole to run through a single edge. Yuan Yuan finds a route for Nicole to run.

For example, let ‘edge’ be given by its edges as follows.

```
edge = {a ⟼ b, b ⟼ c, c ⟼ d, d ⟼ e, e ⟼ a};
```

The graph looks like follows. We label the vertices using numbers so they are easily referred to using the Position function in the path function below.

```
AdjacencyGraph[AdjacencyMatrix[edge],
 VertexLabels → Table[i → vi, {i, First[Dimensions[AdjacencyMatrix[edge]]]}]]
```



The function ‘walk’ tells the number of possible routes from ‘x’ to ‘y’ traversing exactly ‘n’ edges on ‘edge’. ‘M’ is the adjacency matrix: Entry $M_{i,j} = 1$ if and only if there is an edge between vertices i and j. It is known that the (i,j) entry in the nth power of ‘M’ gives the number of n-walks from i to j.

```
walk[edge_, x_, y_, n_] :=
 MatrixPower[AdjacencyMatrix[edge], n][[x, y]]
```

The function ‘path’ is defined recursively. It gives a path randomly from ‘x’ to ‘y’ traversing exactly ‘n’ edges on ‘edge’, if there exists one.

```
path[_, x_, x_, 0] := ToString[v_x, StandardForm];
path[edges_, x_, y_, 1] := If[AdjacencyMatrix[edges][[x, y]] == 0,
    "",
    ToString[v_x, StandardForm] <> ToString[v_y, StandardForm]];
path[edges_, x_, y_, n_] := Module[
  {M = Normal[AdjacencyMatrix[edges]], z},
  z = RandomChoice[Level[Intersection[
       Position[M[[x]], w_ /; w ≠ 0],
       Position[MatrixPower[M, n - 1][[y]], w_ /; w ≠ 0]
      ], {2}]];
  ToString[v_x, StandardForm] <> path[edges, z, y, n - 1]
 ]
```

Try an example:

```
walk[edge, 1, 3, 4]
```

```
4
```

```
path[edge, 1, 3, 4]
```

$v_1v_2v_1v_2v_3$