Exercise 1

If we have the following values: $x[1, 1]$, $y[2, 3]$, the distance function:
$d_{cos}(x,\ y)\ =\ 1 - \frac{cos(x,y)\ +\ 1}{2}\ =\ 1 - \frac{1+1}{2} = 0$.

From this example we can see that it does not satisfy the following property: $d(x,y)\ =\ 0 <=> x\ =\ y$, which means that it's **not a metric**

Exercise 2

$D(x,y)\ =\ \frac{d(x,y)}{d(x,y)\ +\ 1}$

2.1

- Non Negativity

  $d(x,y)\ >=\ 0$ because it's a metric of a metric space

  so, whichever positive value that we add for **k**, it will always be a positive value $=> \frac{d(x,y)}{d(x,y)+1} >= 0$ **-> non negativity holds**

- Isolation

  Suppose $D(x,y)\ =\ 0$. Then $D(x,y)\ =\ \frac{d(x,y)}{d(x,y)+1}$, which implies that $d(x,y)\ =\ 0$.

  Since d is a metric on $X => x = y$.

  If x = y, then:

  $D(x,y)\ =\ \frac{d(x,y)}{d(x,y)+1}\ =\ \frac{0}{0+1}\ =\ \frac{0}{1}\ =\ 0$

  => $d(x,y) = 0\ if\ and\ only\ x = y$

- Symmetry

  $D(x,y)\ =\ \frac{d(x,y)}{d(x,y)+1}\ =\ \frac{d(y,x)}{d(y,x)+1}\ =\ D(y,x)$

  => **symmetry is satisfied**

- Triangular Inequality

$$D(x,z) \ = \ \frac{d(x,z)}{d(x,y)+1} \leq \frac{d(x,y)}{d(x,y)+1} + \frac{d(y,z)}{d(y,z)+1}$$

we can say that:

$a = d(x,\ z)$

$b\ = d(x,\ y)$

$c\ =\ d(y,\ z)$

$a * (b+1) * (c+1) \leq b * (a+1) * (c+1) + c * (a+1) * (b+1)$

$= a * (bc+b+c+1) \leq b * (ac+a+c+1) + c * (ab+a+b+1)$

$abc+ab+ac+a \leq abc+ab+bc+b\ +abc+ac+bc+c$

$a \leq b+2bc+abc+c$

$0 \leq -a+b+c+2bc+abc$

from $-a+b+c$ we can get $a \leq b+c$ which is:

$d(x,\ z) \leq d(x,\ y)\ +\ d(y,\ z)$

We know that **d** is a metric which means $-a+b+c$ is >0.

The $2bc+abc$ is also positive, so

$$\frac{d(x,z)}{d(x,y)+1} \leq \frac{d(x,y)}{d(x,y)+1} + \frac{d(y,z)}{d(y,z)+1}$$

=> **triangular inequality is satisfied**

=> $D(x,y)\ =\ \frac{d(x,y)}{d(x,y)+1}$ is also metric on X

2.2

Even if we change the $k > 0$, D will still be a metric

2.3

k plays role as a scale factor and it helps to normalize $d(x,\ y)$ in a range of [0,1].

With higher values for **k**, the distance gets closer to 0 and with lower values of **k**, the distance gets closer to 1.

2.4
The new metric can be used to convert a distance metric to yield values in range [0, 1].

Exercise 3

$d(b, A) = min_{a \in A} d(b, a)$ and $d(a,\ B)\ =\ min_{b \in B} d(a, b)$ =>
$dh(B, A) = max_{a \in A} d(b, A)$ and $dh(A, B) = max_{b \in B} d(a, B)$
and $Dh(A, B)\ =\ max\{dh(A, B),\ dh(B, A)\}$

- Non negativity
  d is a metric on X => $d(x, y)\ >=\ 0$ for every x, y that belong in X.
  From here we get that $d(a,\ B)\ >=\ 0\ and\ d(b,\ A) >= 0$.
  From here it follows that $dh(A, B)\ >= 0\ and\ dh(B, A)\ >= 0$
  => $Dh(A, B)\ >= 0$ => non negativity holds.

- Symmetry
  $Dh(A,\ B)\ =\ max\{dh(A,\ B),\ dh(B,\ A)\}\ =\ max\{dh(B,\ A),\ dh(A,\ B)\}\ =$
  $Dh(B,\ A)\ =>$ symmetry holds

- Isolation
  If $Dh(A,\ B)\ =\ 0$, then $dh(A,\ B)\ =\ dh(B,\ A)\ =\ 0$
  For every $a \in A$, $dh(a,\ B)\ =\ 0$.
  We have $min_{b \in B} dh(a,\ b)\ =\ 0$ for every $a \in A$.
  B is a finite set => $b = a$.
  $A \subset B$ and $B \subset A$ => $A = B$.
  If $A = B$, $Dh(A,\ B)\ =\ 0\ =>\ Dh(A,\ B)\ =\ 0$ if and only if $A = B$.

- Triangular inequality
  If we have 3 sets of finite points: A, B and C,
  the triangular inequality is defined as:
  $d(A,B) \leq d(A,C) + d(C, B)$
  For any $a \in A$ we have:
  $d(a, B) = min_{b \in B}(d(a, b) \leq min_{b \in B} d(a, c) + d(c, b))$ for every $c \in C$
  $= d(a, c) + min_{b \in B} d(c, b)$ for every $c \in C$
  $= d(a, c) + d(c, B)$ for every $c \in C$
  $\leq d(a, c) + d(C, B)$ for every $c \in C$

  If we minimize over **c**, we get:
  $d(a, B) \leq d(a, C) + d(C, B)$

  With this we have shown that for every $a \in A$
  $d(a, B) \leq d(a, C) + d(C, B)$

  If we maximize over **a** on the left and right side, we get:
  $d(A, B) \leq d(A, C) + d(C, B)$

With all that said, **it's a metric**.

Exercise 4

4.1

We can treat the paths as strings.
The first string X can be the first walk and it will consist of the vertices of the first walk.

The second string Y can be the second walk and it will consist of the vertices of the second walk.
This way we can use the string edit distance (Levenshtein distance) to compare the walks.

4.2
The Levenshtein distance use three operations: insertion, deletion and substitution.
In our case the distance will be the minimum number of operation needed to transform the first walk to the second.
The lower the distance, closer the two walks are.

4.3

- Non negativity
  The distance is defined as the minimum number of operations, thus, the distance can't be negative

- Coincidence
  If two strings X and Y are equal, then there is no need for any transformation so the distance would be 0.
  If strings X and Y are not equal, then we must do minimum one operation which mean the distance can't be 0.

- Symmetry
  If $N = (x1, x2, x3, \dots ,xn)$ is the minimum number of transformations that we need to do in order to transform string X to string Y, then we can use $N_{reverse} = (xn, xn-1, xn-2, \dots, x1)$ to transform Y to X.
  and $N = N_{reverse}$ => symmetry holds.

- Triangular inequality

If we have three strings: X, Y and Z.
Let:
$N_{X,Y}$ - minimum number of transformations from X to Y
$N_{X,Z}$ - minimum number of transformations from X to Z
$N_{Z,Y}$ - minimum number of transformations from Z to Y

Triangular inequality states that:
$N_{X,Y} \leq N_{X,Z} + N_{Z,Y}$

For contradiction, we can assume that:
$N_{X,Y} > N_{X,Z} + N_{Z,Y}$
which means that now the smallest distance is:
$N_{X,Z} + N_{Z,Y}$. We can replace that with $newN_{X,Y}$ and we get:
$N_{X,Y} > newN_{X,Y}$ which is a contradiction because $N_{X,Y}$ should be the smallest distance but it's not.

4.4
Let X be the first walk which consists of the vertices of the first walk.
Let Y be the second walk which consists of the vertices of the second walk.

```
$levenshtein(X, Y)$
    $if\ (length(X) < length(Y))$
        $levenshtein(Y, X)$
    $if\ (length(Y) == 0)$
        $return\ length(Y)$

  $previousRow = range(length(Y) + 1)$
  $for\ (i, c1\ in\ enumerate(Y))$
      $currentRow = [i + 1]$
      $for\ (i, c2\ in\ enumerate(X)) :$
```

```
            $insertions = previousRow[j + 1] + 1$
            $deletions = currentRow[j] + 1$
            $substitutions = previousRow[j] + (c1 != c2)$
            $currentRow.append(min(insertions, deletions, substitutions))$
        $previousRow = currentRow$
    $return\ previousRow[-1]$
```

The algorithm calculates the minimum changes required in order to transform path X to path Y.
The smaller the number of changes, the more similar the two walks are.

The complexity of this algorithm is **O(XY)**.

4.5
I don't know

Exercise 5.

```
$d = dimensions$
$n = vectors$

$M[n\ x\ d]$ matrix with rows as vectors and columns as dimensions

$len_d = d$
$len_n = n$
$distance = []$
$index = []$

$for\ d\ in\ len_d :$
```

```
        $min = M[0][d]$
        $max = M[1][d]$

        $if\ min > max:$
            $temp = min$
            $min = max$
            $max = temp$
        $for\ n\ in\ len_n:$
            $if\ M[n][d] < min:$
                $min = M[n][d]$
                $index_{min} = indexOf(M[n][d])$
            $if\ M[n][d] > max:$
                $max = M[n][d]$
                $index_{max} = indexOf(M[n][d])$

        $distance.append(|\ min - max\ |)$
        $index.append([index_{min},\ index_{max}])$

    $max_{dist} = max(distance)$
    $max_{pos} = indexOf(distance(max_{dist}))$

    $first_{vector} = index[max_{pos}][0]$
    $second_{vector} = index[max_{pos}][1]$

    return $first_{vector}$ , $second_{vector}$
```

The furthest vectors are: $first_{vector}$ and $second_{vector}$

The complexity is **O(nd).**

The algorithm takes the set of vectors and puts it in a matrix M[n x d]

where n is the number of vectors and d is the number of dimensions.
We iterate over each column and get the min and max values and take the difference. We do that for all the dimensions.
Then we end up with a vector of distances and take the highest distance and find from which vectors it was calculated.
We have to iterate over the number of vectors and the number of dimensions so the complexity is O(nd).

Exercise 6.

6.1
The new distance function d_l can speed up the computation in a way that if the distance $d_l(x_i, q)$ is greater than the current minimum value computed by $d(x_i, q)$ then we don't need to do the calculation $d(x_i, q$ for the new point.

6.2
```
$dmin <- d(x_1 q)$
$x^* <- x_1$
$for\ i = 2, ... , n$
    $candidate = d_l(x_i, q)$
    $if\ (candidate > dmin)$
        $pass$
    $else$
        $dtmp <- d(x_i, q)$
        $if\ (dtmp < dmin)$
        $dmin <- dtmp$
        $X^* <- x_i$
$return\ x^*$
```

6.3

We want to come up with a lower bound distance functions that are as large as possible because in that case we won't have to calculate the expensive distance function $d(x_i, q)$ many times.

6.4

If we have two strings: s1 and s2, $d(s1, s2) = |length(s1) - length(s2)|$ can be a lower bound distance function.

If the difference of the lengths of s1 and s2 is large, then the distance function will give large value, which is what we want, so we won't have to do the more expensive distance function.

The limitation of this function is that if the lengths of the strings are similar or same, then the lower bound function will give low values which won't help us much and we will have to do the expensive distance function.