CS-E4600
Algorithmic methods for data mining

Aristides Gionis
Dept of Computer Science

Slide set 5: Locality-sensitive hashing

reading assignment

LRU book : chapter 3

recall : finding similar objects

informal definition

two problems

1. similarity search problem

    given a set X of objects (off-line)

    given a query object q (query time)

    find the object in X that is most similar to q

2. all-pairs similarity problem

    given a set X of objects (off-line)

    find all pairs of objects in X that are similar

recall : warm up

let's focus on problem 1

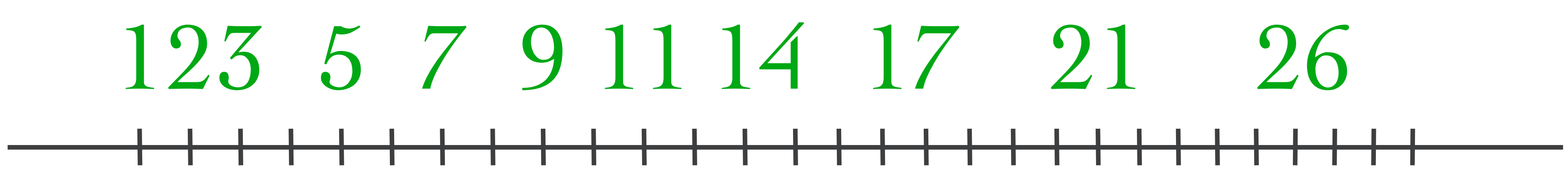
how to solve a problem for 1-d points?

example:

given X = { 5, 9, 1, 11, 14, 3, 21, 7, 2, 17, 26 }

given q=6, what is the nearest point of q in X?

answer: sorting and binary search!



warm up 2

consider a dataset of objects X (offline)

given a query object q (query time)

is q contained in X ?

warm up 2

consider a dataset of objects X (offline)

given a query object q (query time)

is q contained in X ?

answer : hashing !

warm up 2

consider a dataset of objects X (offline)

given a query object q (query time)

is q contained in X ?

answer : hashing !

running time ?

warm up 2

consider a dataset of objects X (offline)

given a query object q (query time)

is q contained in X ?

answer : hashing !

running time ? constant !

warm up 2

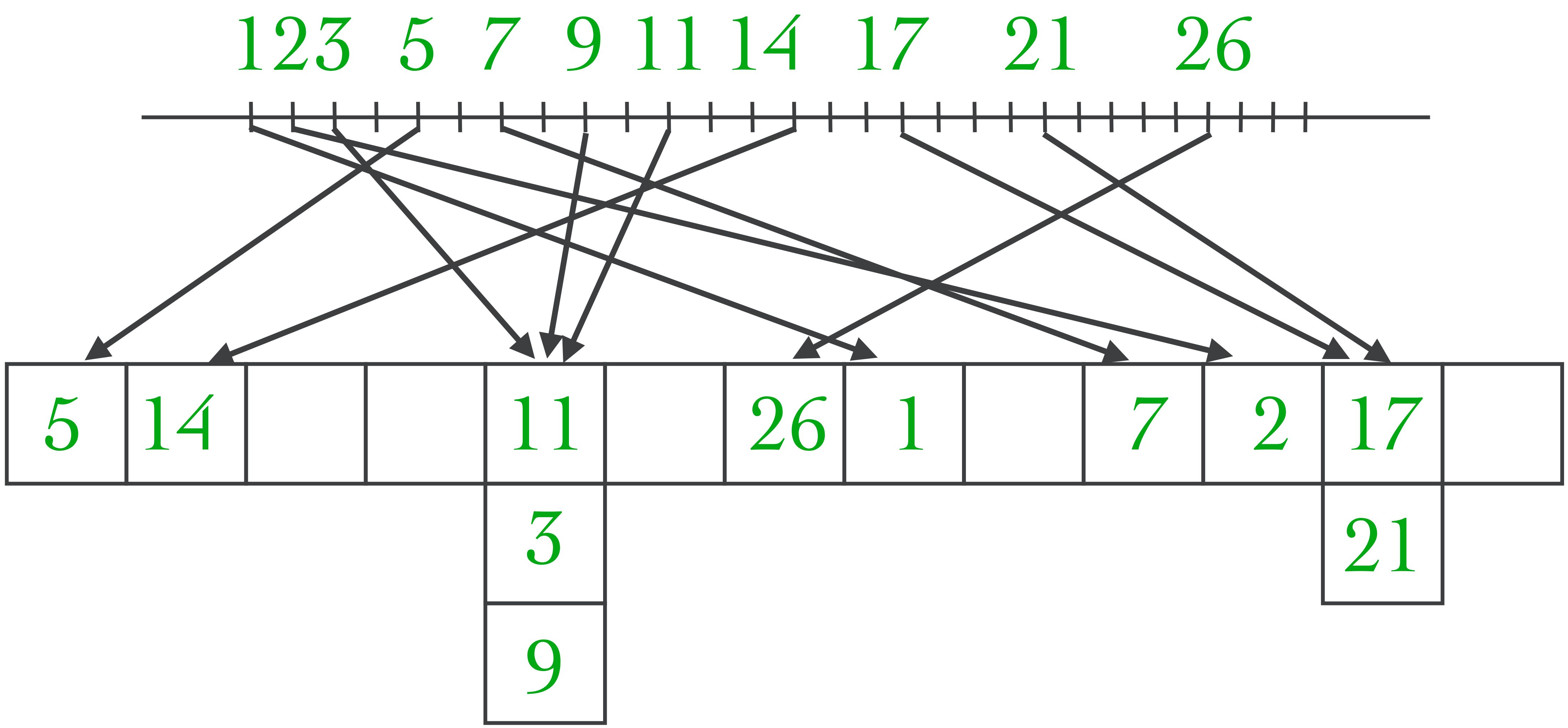
how we simplified the problem?

looking for exact match

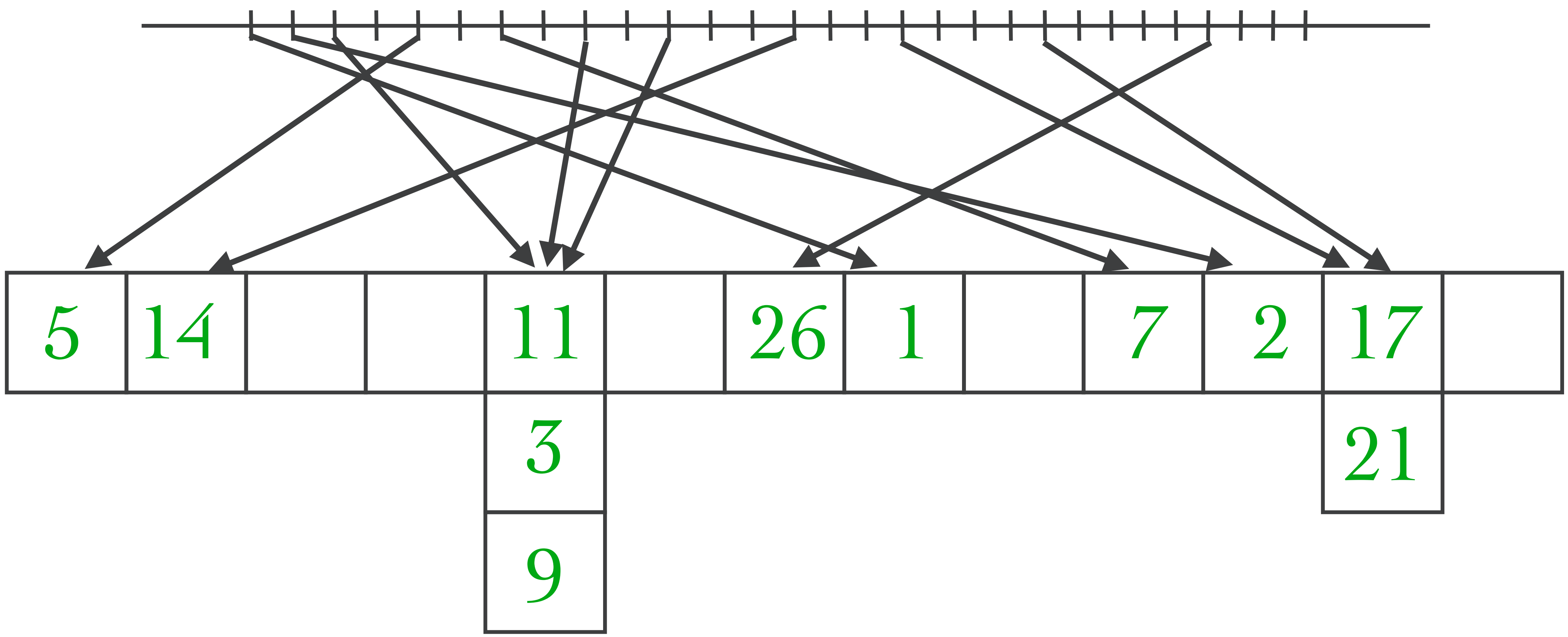
searching for similar objects does not work

searching by hashing

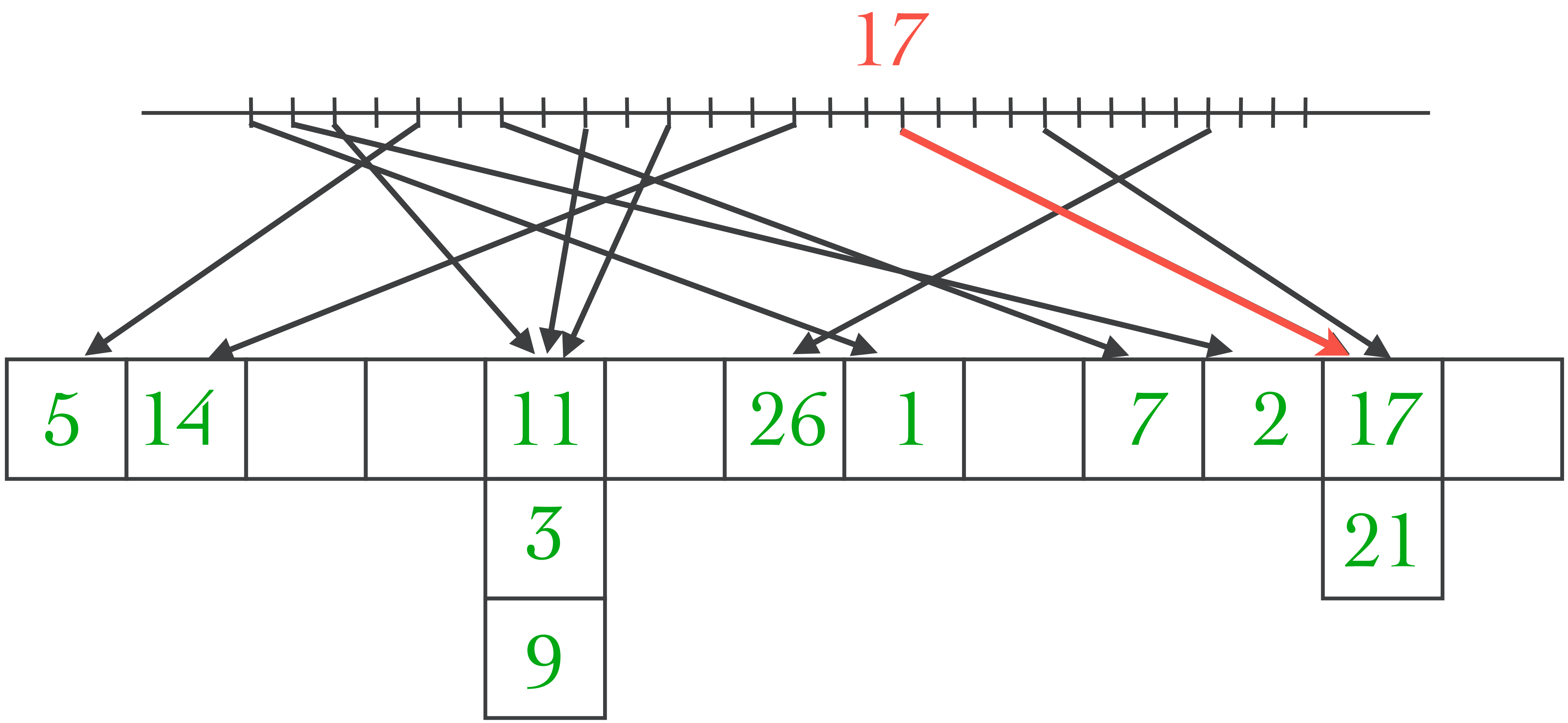
searching by hashing



searching by hashing

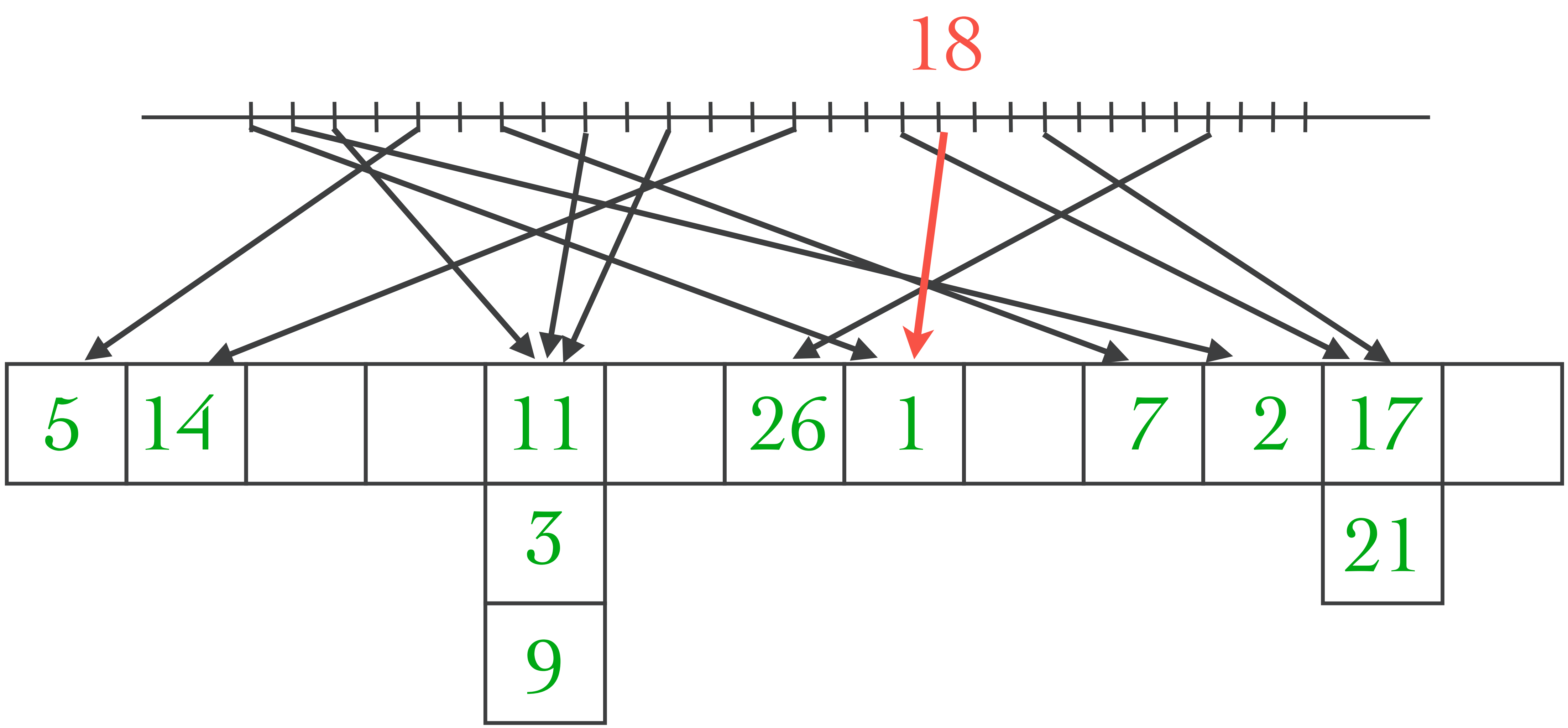


searching by hashing



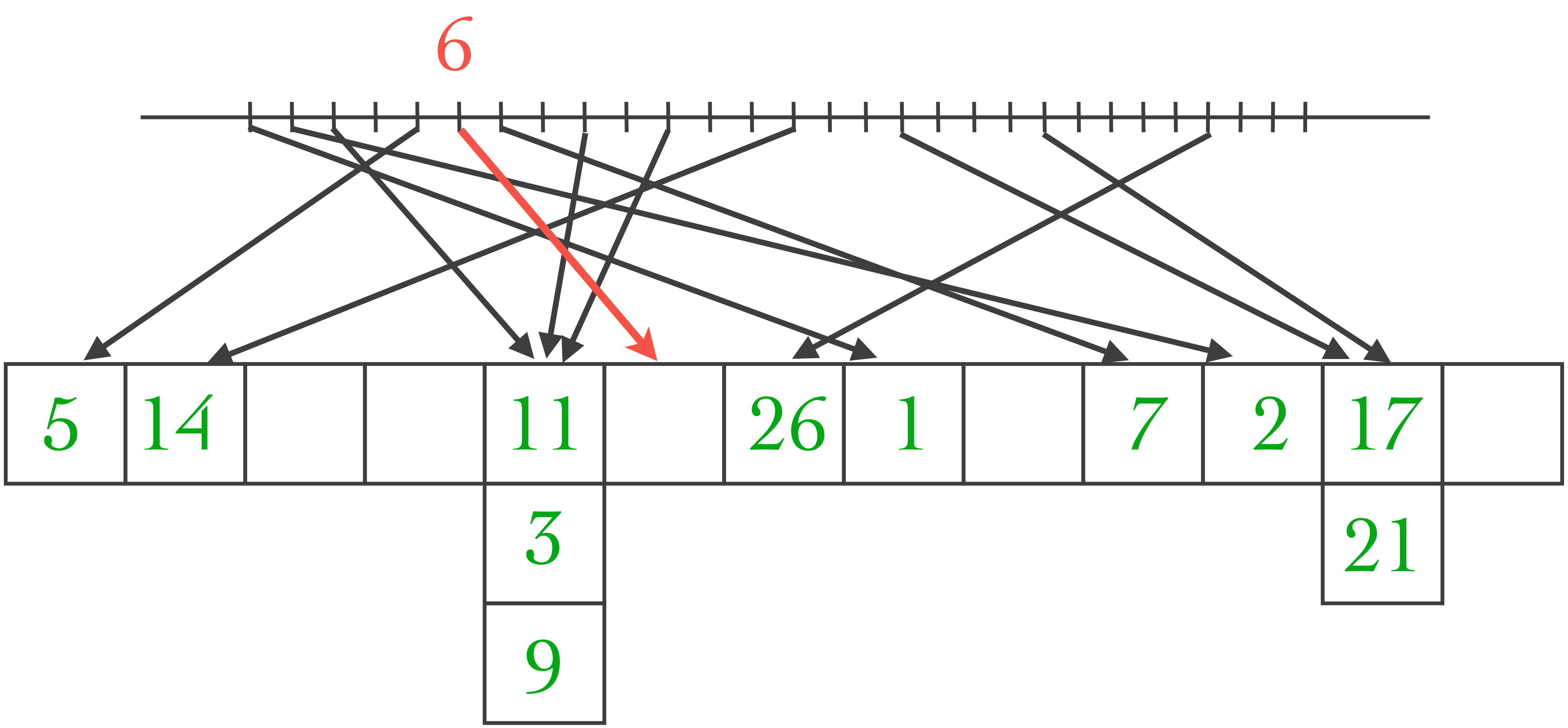
does 17 exist? yes

searching by hashing



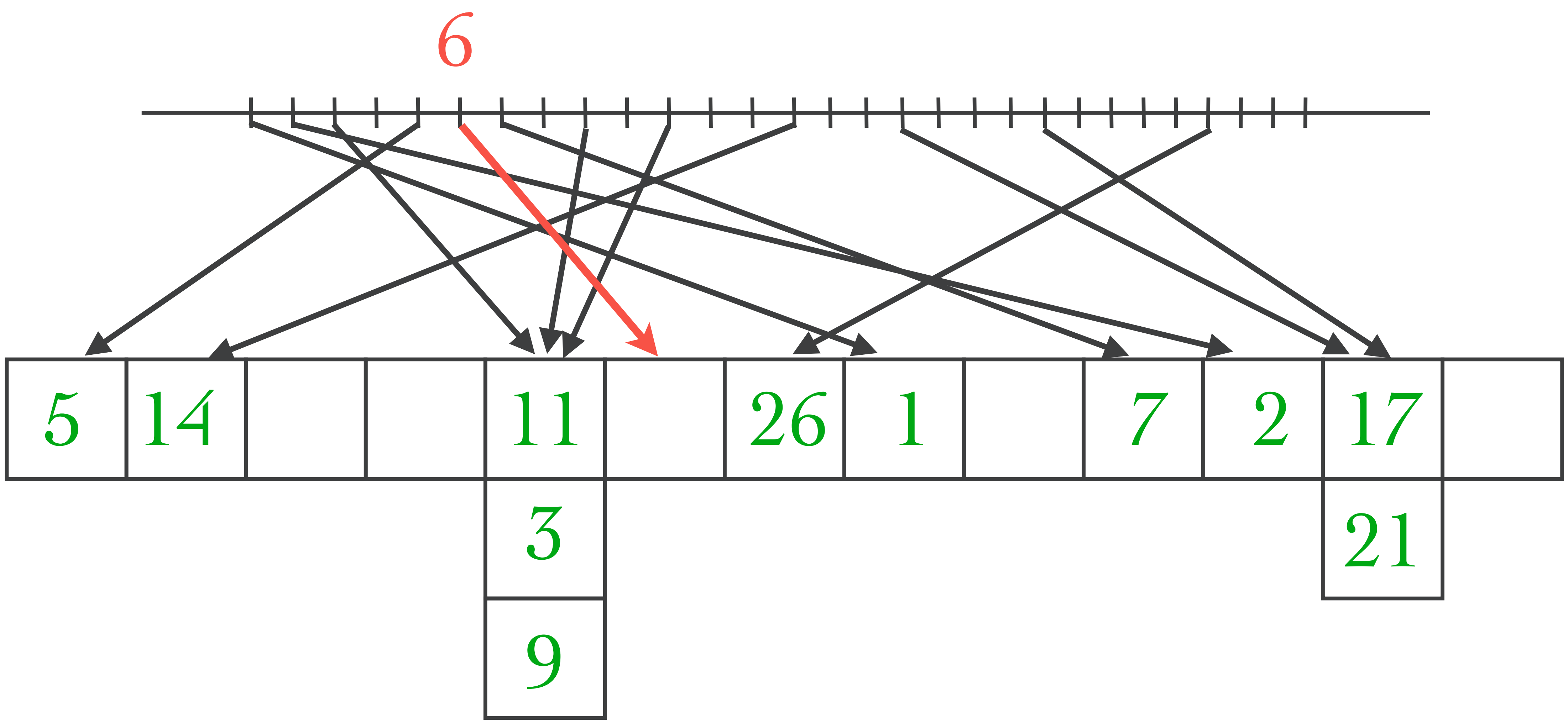
does 18 exist? no

searching by hashing



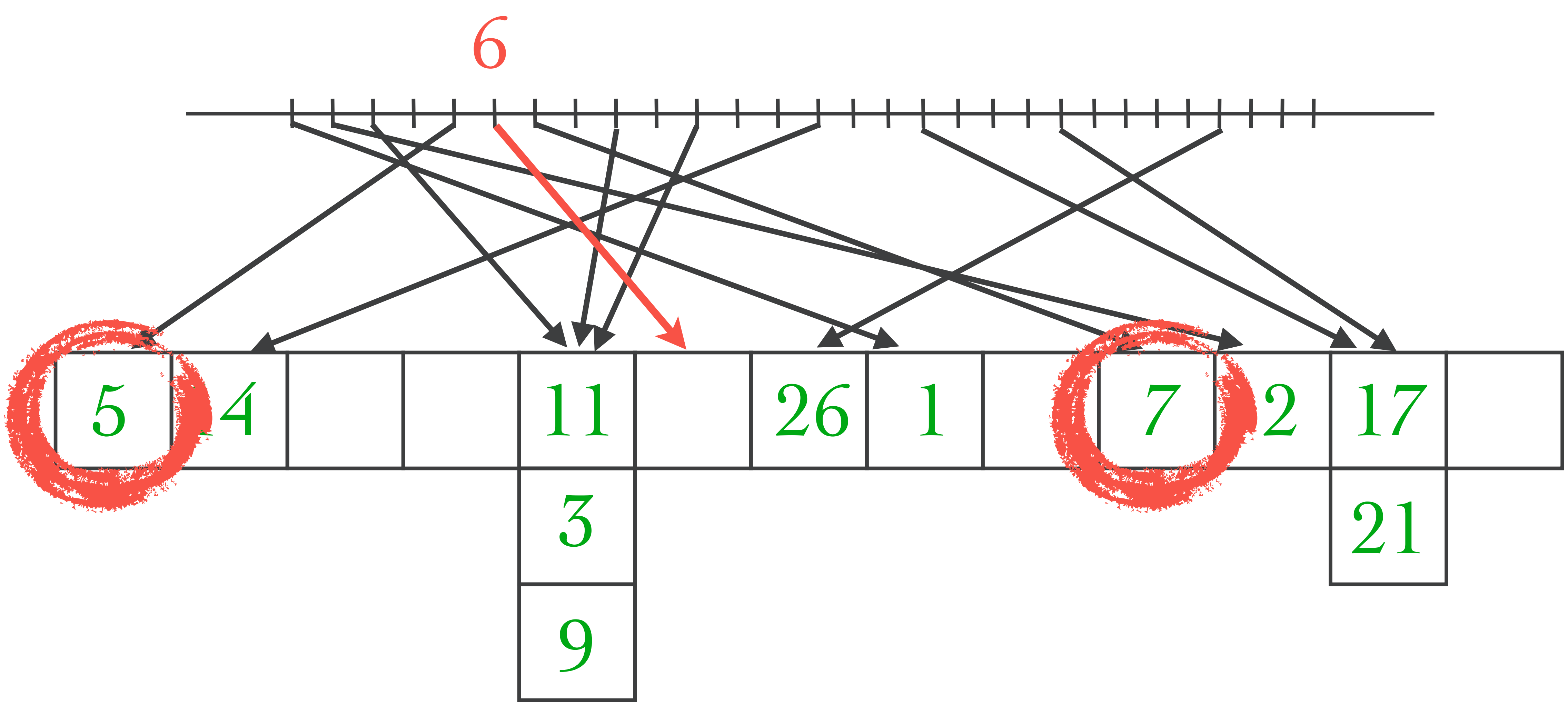
does 6 exist? no

searching by hashing



what is the nearest neighbor of 6?

searching by hashing



what is the nearest neighbor of 6?

recall : hash functions

perfect hash functions

recall : hash functions

perfect hash functions

provide an 1-to-1 mapping of objects to bucket ids

any two distinct objects are mapped to different buckets

no collisions!

drawback: hash function requires as many bits as the number of objects to be hashed

recall : hash functions

perfect hash functions

provide an 1-to-1 mapping of objects to bucket ids

any two distinct objects are mapped to different buckets

no collisions!

drawback: hash function requires as many bits as the number of objects to be hashed

universal hash functions

recall : hash functions

perfect hash functions

provide an 1-to-1 mapping of objects to bucket ids

any two distinct objects are mapped to different buckets

no collisions!

drawback: hash function requires as many bits as the number of objects to be hashed

universal hash functions

family of hash functions

for any two distinct objects the probability of collision is $1/n$

prob. is over the choice of a hash function in the family

very simple and inexpensive, e.g., $h(x) = ax+b \bmod q$

a collision-resolution mechanism is needed, e.g., chaining

searching by hashing

should be able to locate similar objects

searching by hashing

should be able to locate similar objects

locality-sensitive hashing

- collision probability for similar objects is high enough
- collision probability of dissimilar objects is low

searching by hashing

should be able to locate similar objects

locality-sensitive hashing

- collision probability for similar objects is high enough
- collision probability of dissimilar objects is low

randomized data structure

- guarantees (running time and quality) hold in expectation (with high probability)
- see : randomized algorithms

locality-sensitive hashing

focus on the problem of approximate nearest neighbor

given a set X of objects (off-line)

given accuracy parameter e (off-line)

given a query object q (query time)

find an object z in X, such that

$$d(q,z) \leq (1+e)d(q,x) \quad \text{for all } x \text{ in } X$$

locality-sensitive hashing

somewhat easier problem to solve: approximate near neighbor

given a set X of objects (off-line)

given accuracy parameter e and distance R (off-line)

given a query object q (query time)

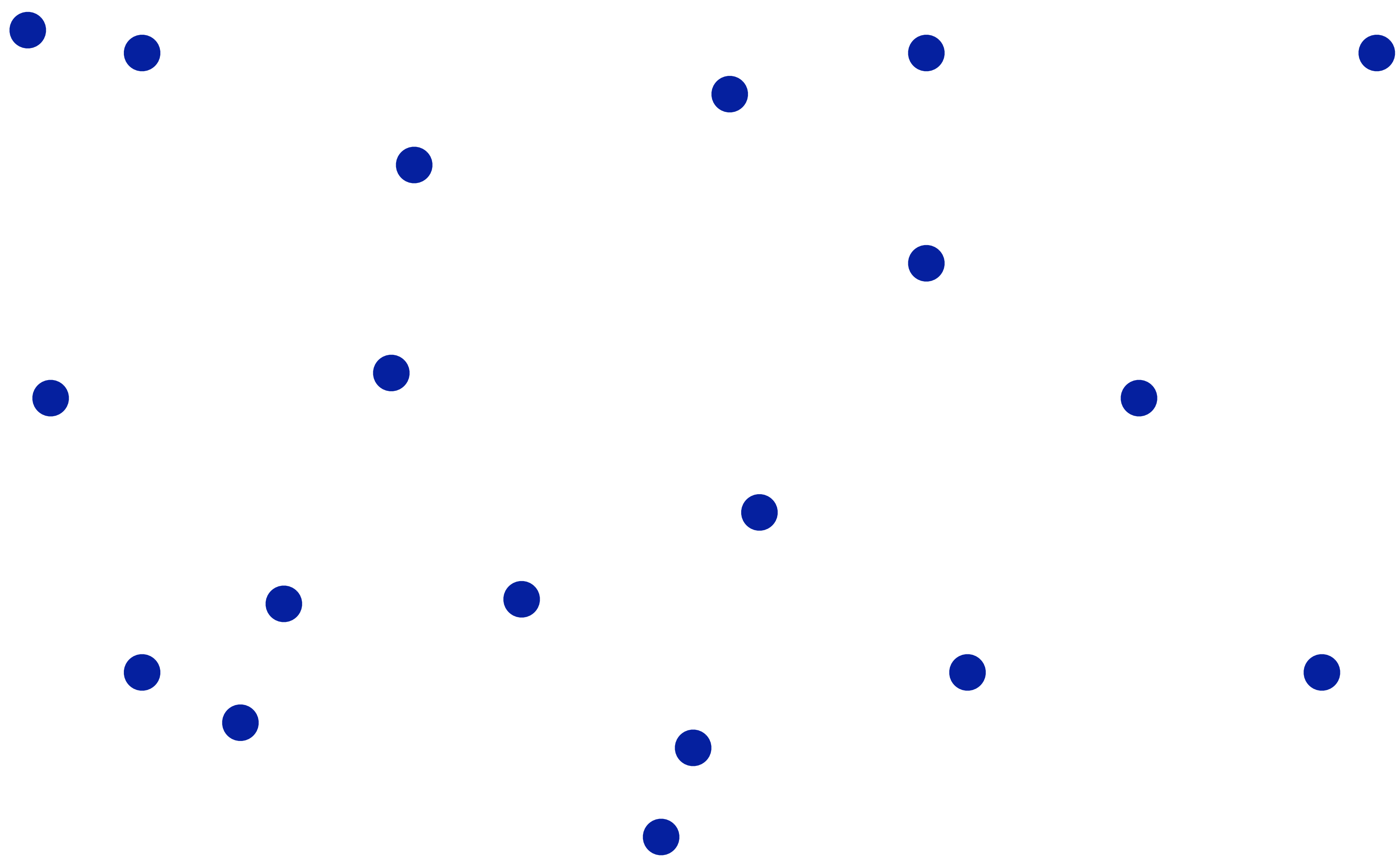
if there is object y in X s.t. $d(q,y) \leq R$

then return object z in X s.t. $d(q,z) \leq (1+e)R$

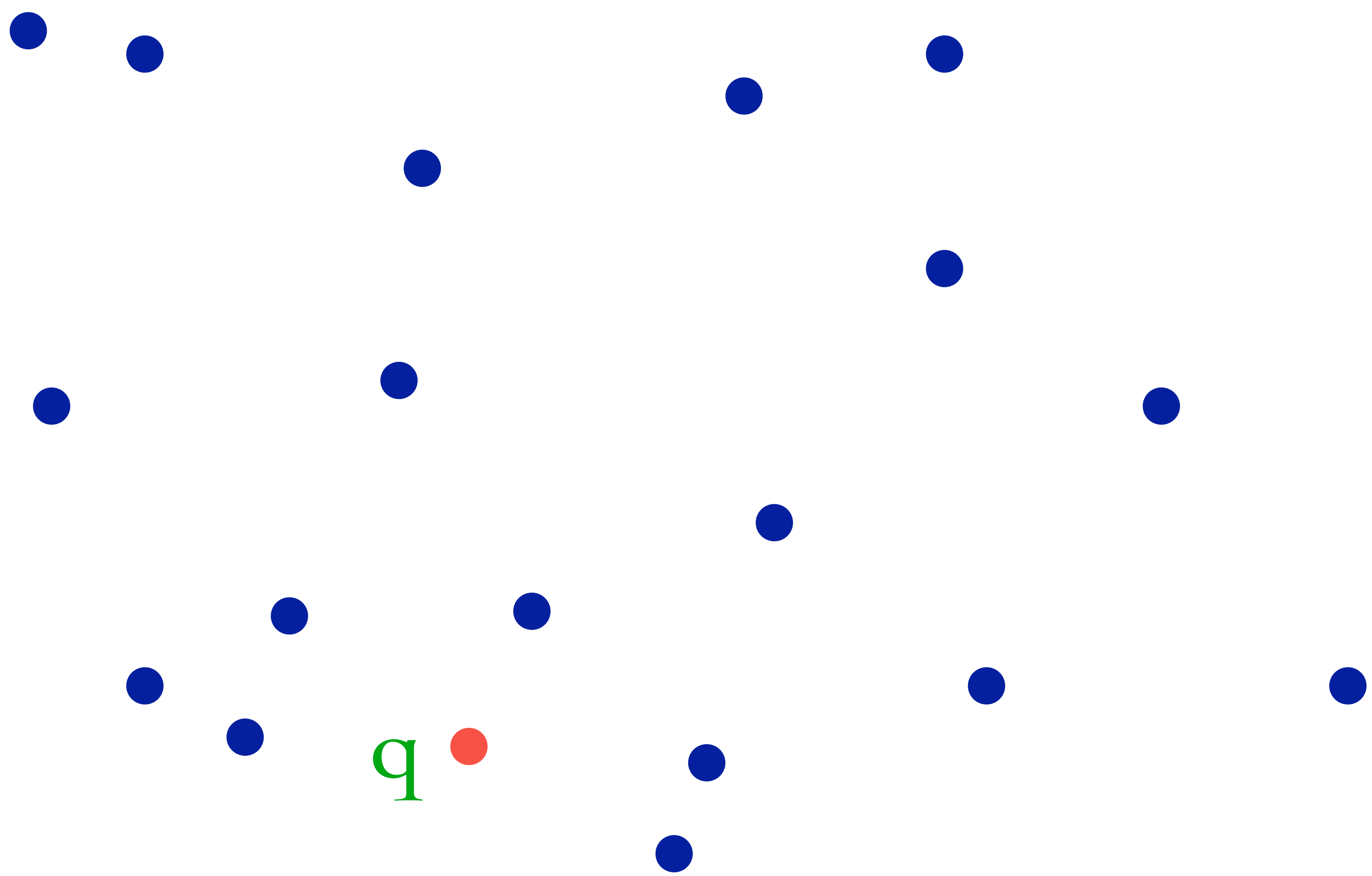
if there is no object y in X s.t. $d(q,z) \geq (1+e)R$

then return no

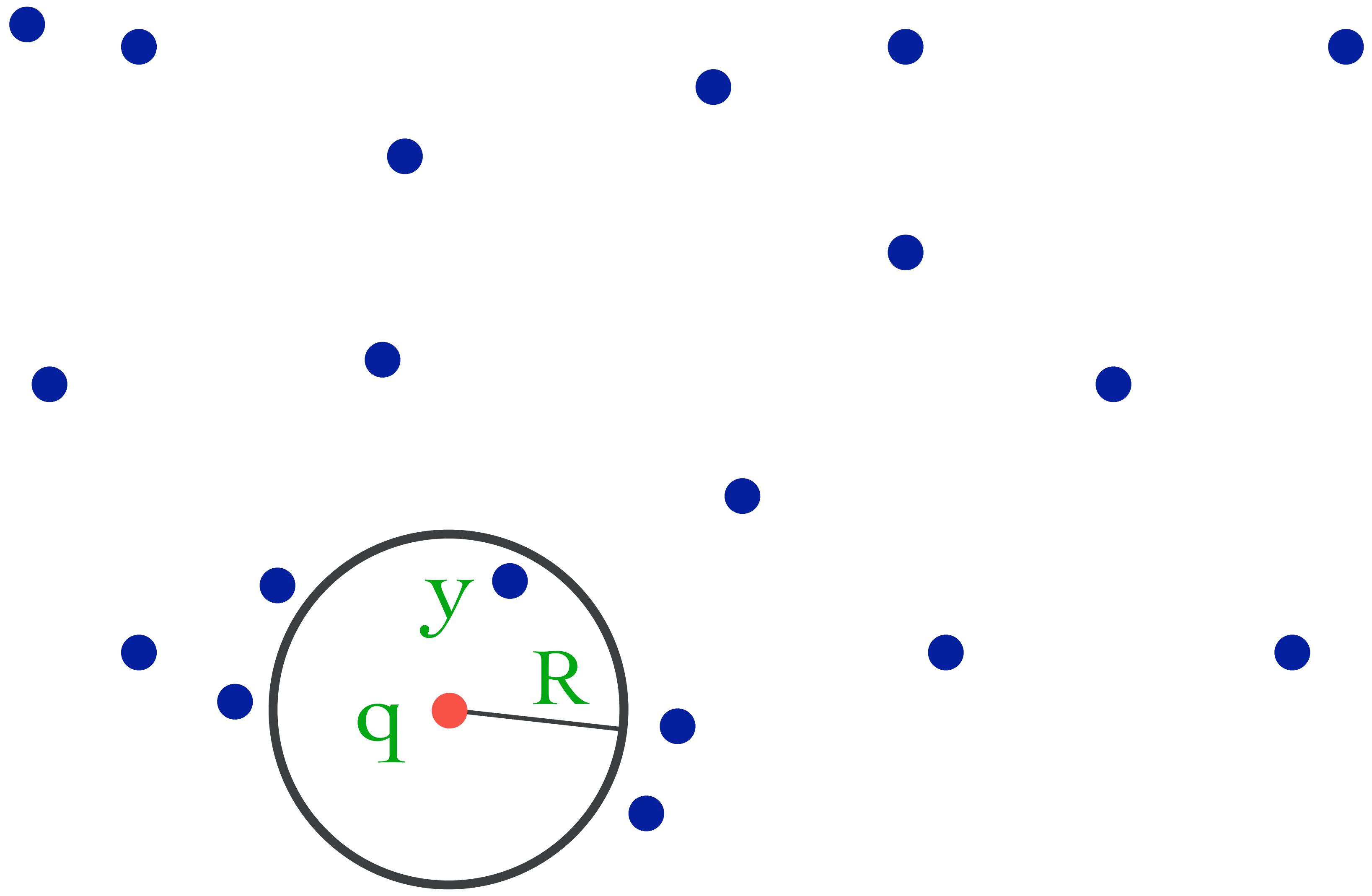
approximate near neighbor



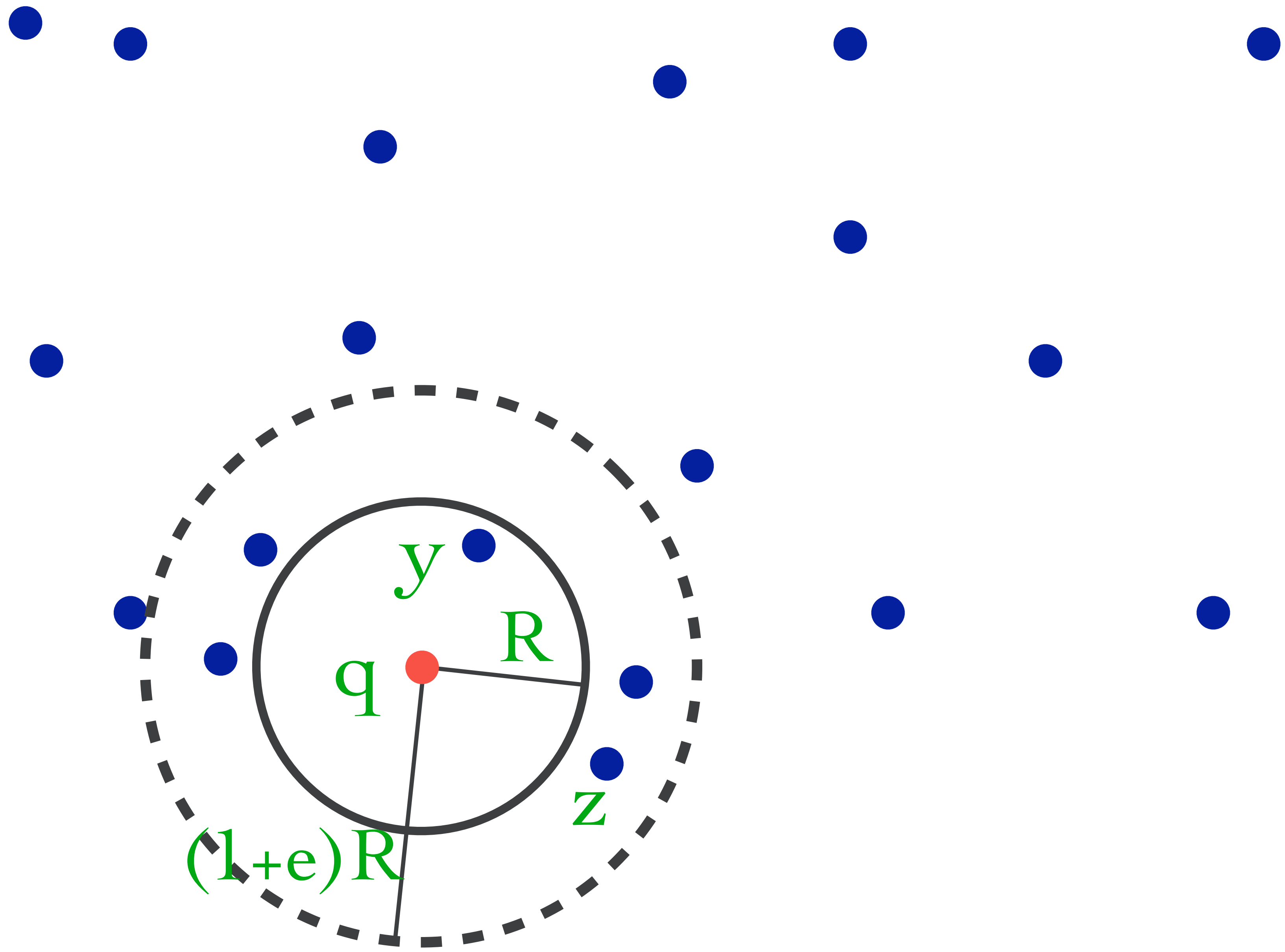
approximate near neighbor



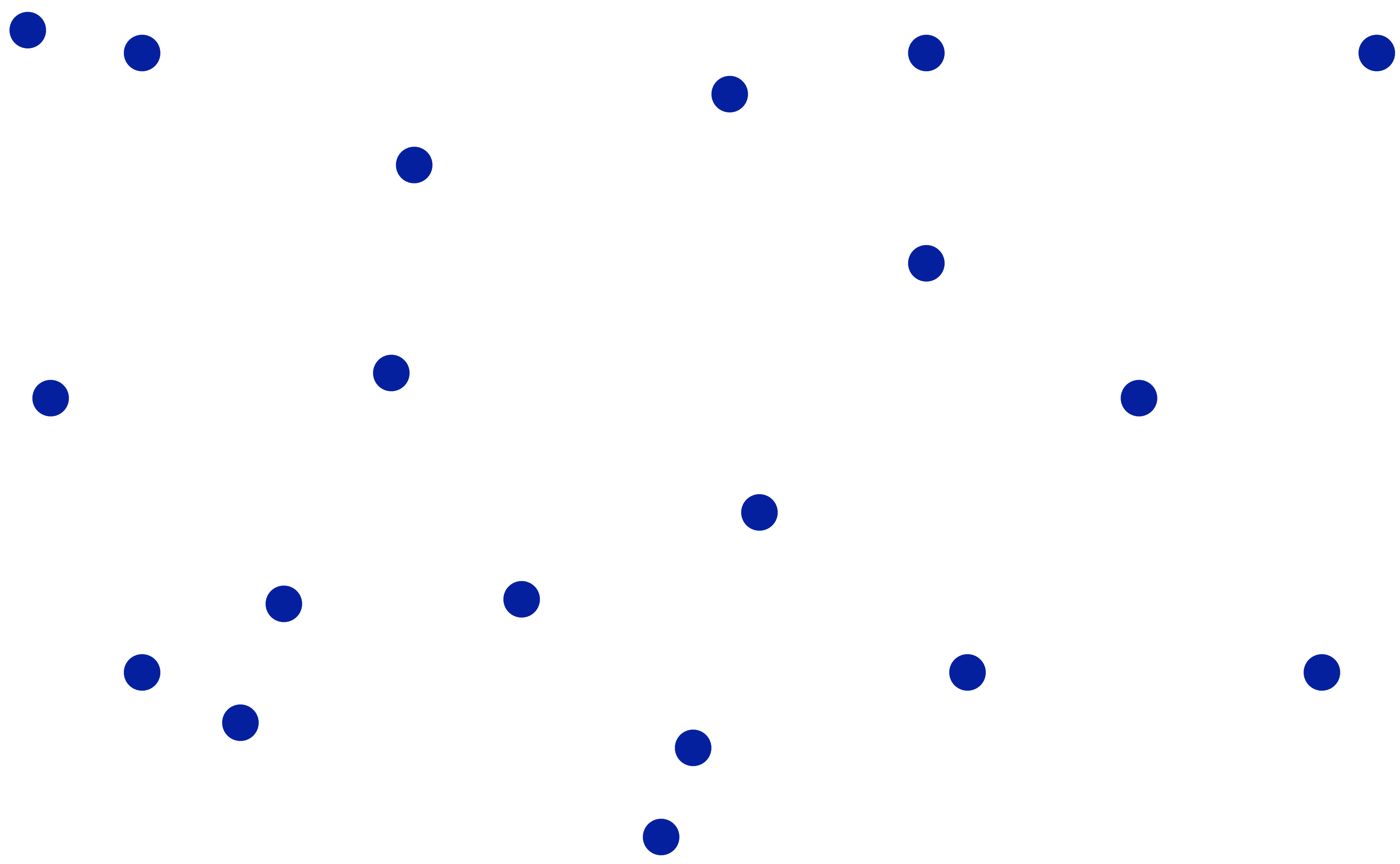
approximate near neighbor



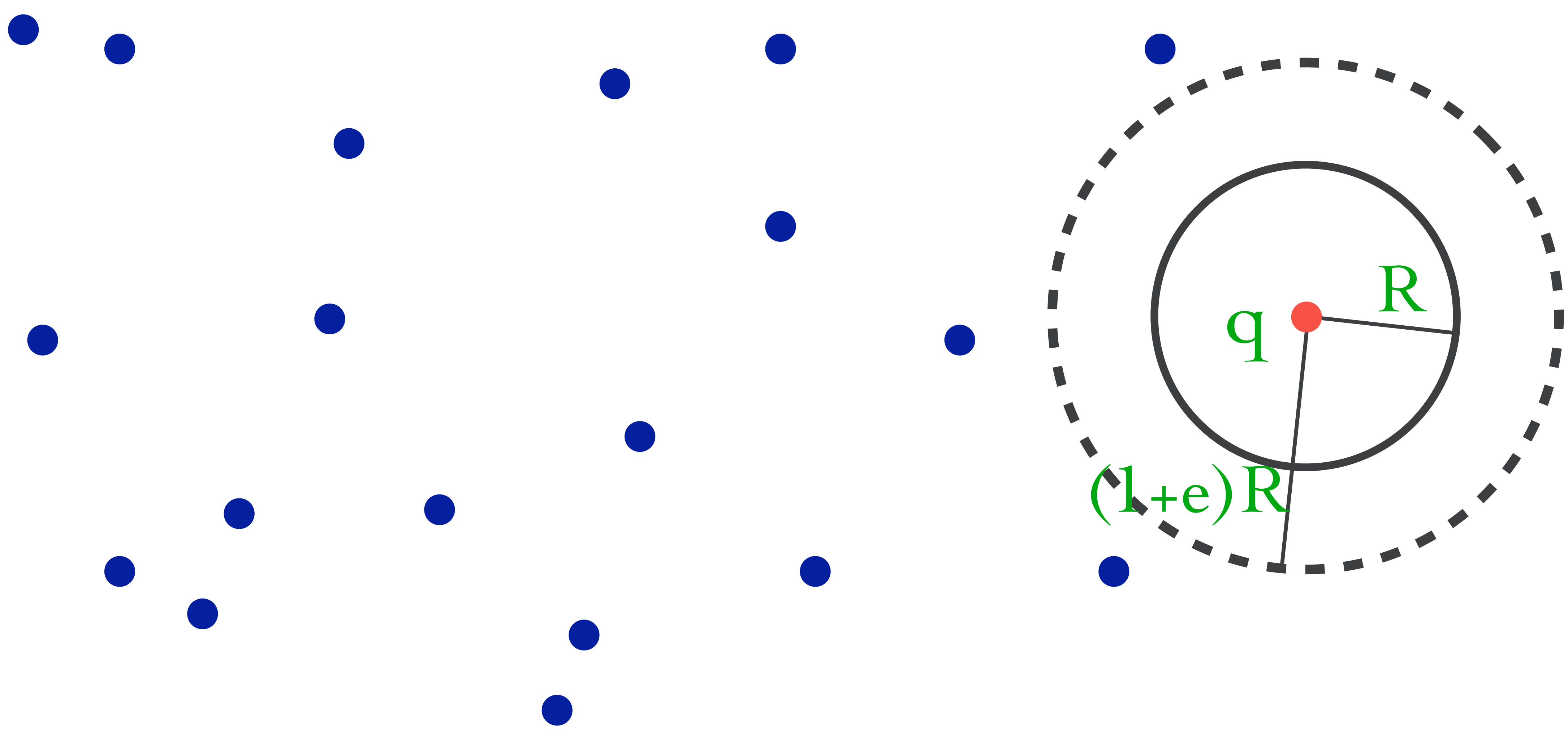
approximate near neighbor



approximate near neighbor



approximate near neighbor



approximate near(est) neighbor

approximate nearest neighbor can be reduced to approximate near neighbor

how?

approximate near(est) neighbor

approximate nearest neighbor can be reduced to approximate near neighbor

how?

let d and D the smallest and largest distances

build approximate near neighbor structures for

$$R = d, (1+e)d, (1+e)^2d, \ldots, D$$

approximate near(est) neighbor

approximate nearest neighbor can be reduced to approximate near neighbor

how?

let d and D the smallest and largest distances

build approximate near neighbor structures for

$$R = d,\ (1+e)d,\ (1+e)^2 d,\ \ldots,\ D$$

how to use ?

approximate near(est) neighbor

approximate nearest neighbor can be reduced to approximate near neighbor

how?

let d and D the smallest and largest distances

build approximate near neighbor structures for

$$R = d,\ (1{+}e)d,\ (1{+}e)^2 d,\ \ldots,\ D$$

how to use ?

how many?

approximate near(est) neighbor

approximate nearest neighbor can be reduced to approximate near neighbor

how?

let d and D the smallest and largest distances

build approximate near neighbor structures for

$$R = d, (1+e)d, (1+e)^2d, \ldots, D$$

how to use ?

how many? $O(\log_{1+e}(D/d))$

to think about..

to think about..

for query point q

search all approximate near neighbor structures with

$$R = d, (1+e)d, (1+e)^2 d, \ldots, D$$

return a point found in the non-empty ball with the smallest radius

answer is an approximate nearest neighbor for q

locality-sensitive hashing for approximate near neighbor

focus on vectors in $\{0,1\}^d$

binary vectors of d dimension

distances measured with Hamming distance

$$d_H(x, y) = \sum_{i=1}^{d} |x_i - y_i|$$

definitions for Hamming similarity

$$s_H(x, y) = 1 - \frac{d_H(x, y)}{d}$$

locality-sensitive hashing for approximate near neighbor

a family F of hash functions is called (s_1, s_2, p_1, p_2)-sensitive if for any two objects x and y

if $s_H(x,y) \geq s_1$, then $\Pr[h(x)=h(y)] \geq p_1$

if $s_H(x,y) \leq s_2$, then $\Pr[h(x)=h(y)] \leq p_2$

probability over selecting h from F

$s_1>s_2$ and $p_1>p_2$

locality-sensitive hashing for approximate near neighbor

vectors in $\{0,1\}^d$, Hamming similarity $s_H(x,y)$

locality-sensitive hashing for approximate near neighbor

vectors in $\{0,1\}^d$, Hamming similarity $s_H(x,y)$

consider the hash function family:

sample the i-th bit of a vector

locality-sensitive hashing for approximate near neighbor

vectors in $\{0,1\}^d$, Hamming similarity $s_H(x,y)$

consider the hash function family:

sample the i-th bit of a vector

probability of collision

$$Pr[h(x)=h(y)] = s_H(x,y)$$

locality-sensitive hashing for approximate near neighbor

vectors in $\{0,1\}^d$, Hamming similarity $s_H(x,y)$

consider the hash function family:

sample the i-th bit of a vector

probability of collision

$$\Pr[h(x)=h(y)] = s_H(x,y)$$

$(s_1, s_2, p_1, p_2) = (s_1, s_2, s_1, s_2)$-sensitive

$s_1>s_2$ and $p_1>p_2$, as required

locality-sensitive hashing for approximate near neighbor

obtained $(s_1, s_2, p_1, p_2) = (s_1, s_2, s_1, s_2)$-sensitive function

gap between p_1 and p_2 too small

amplify the gap:

stack together many hash functions

probability of collision for similar objects decreases

probability of collision for dissimilar objects decreases more

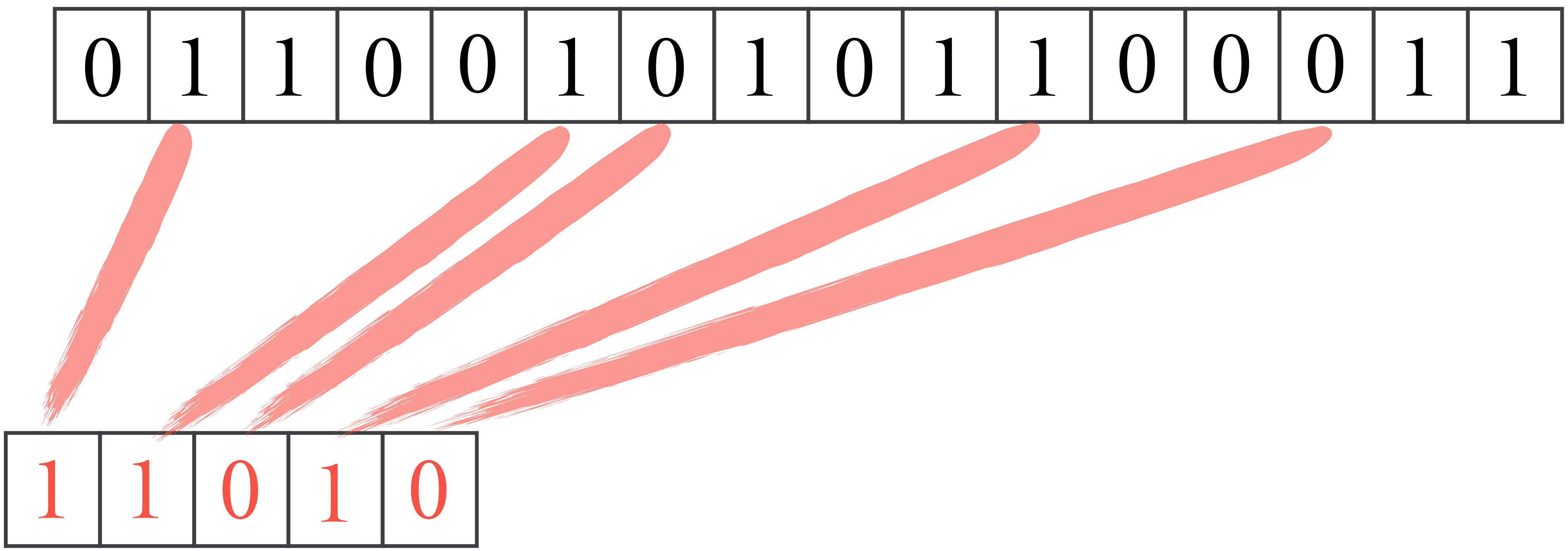
repeat many times

probability of collision for similar objects increases

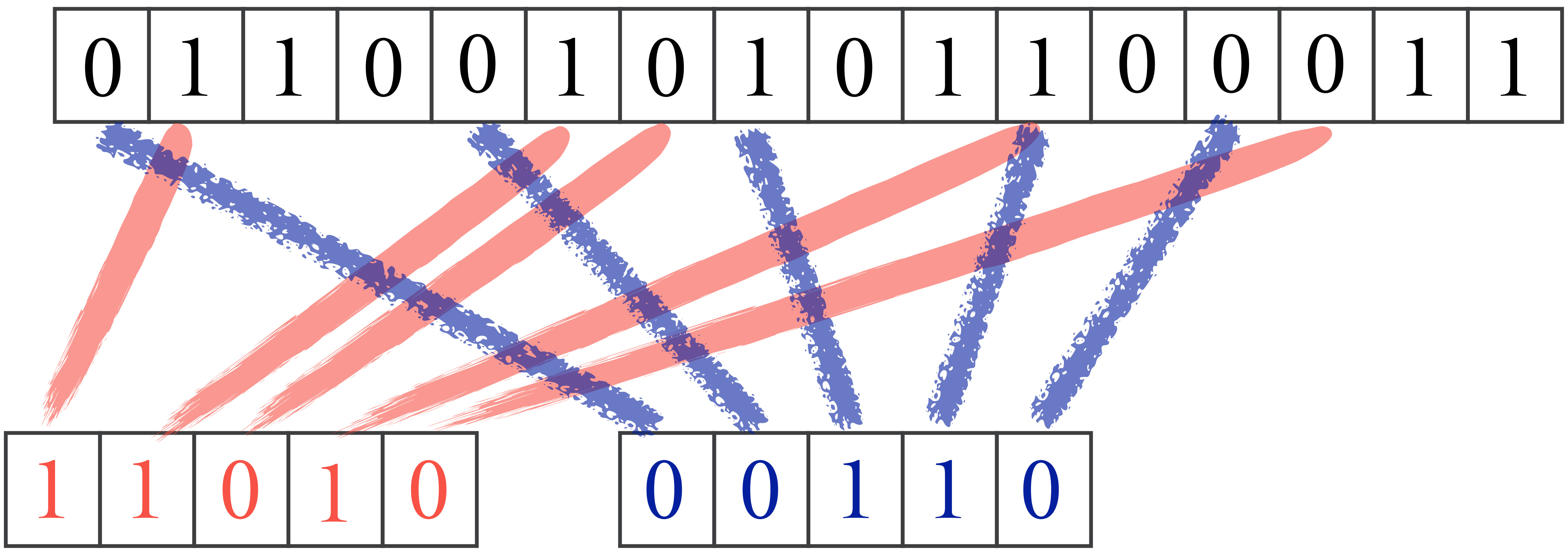
locality-sensitive hashing

0	1	1	0	0	1	0	1	0	1	1	0	0	0	1	1

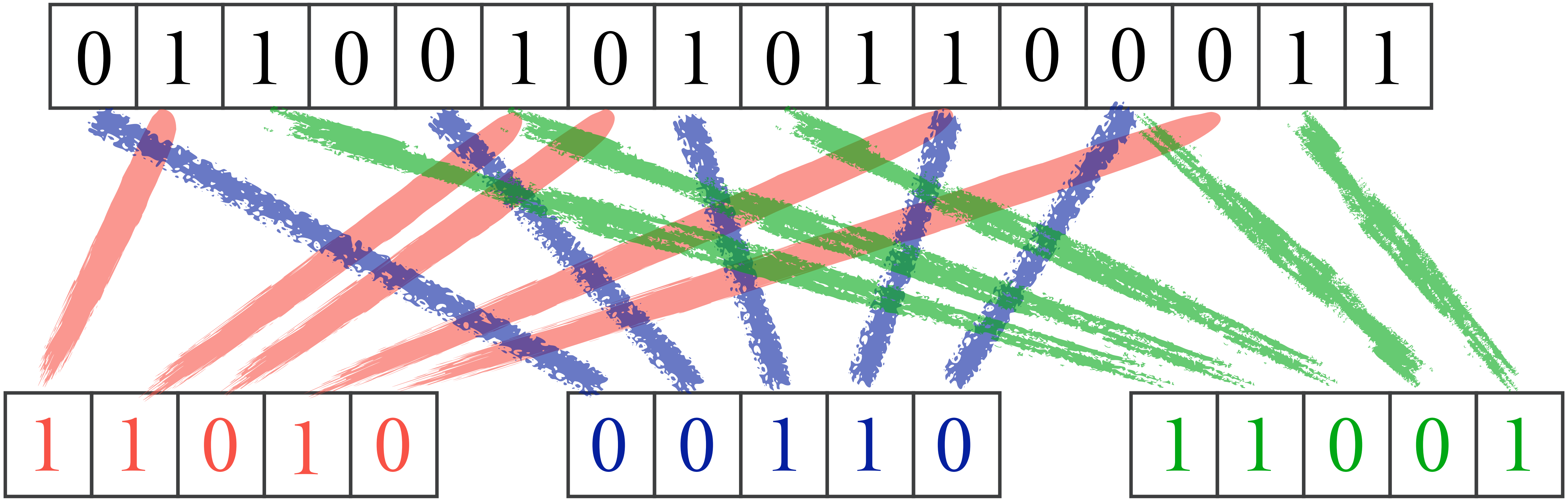
locality-sensitive hashing



locality-sensitive hashing

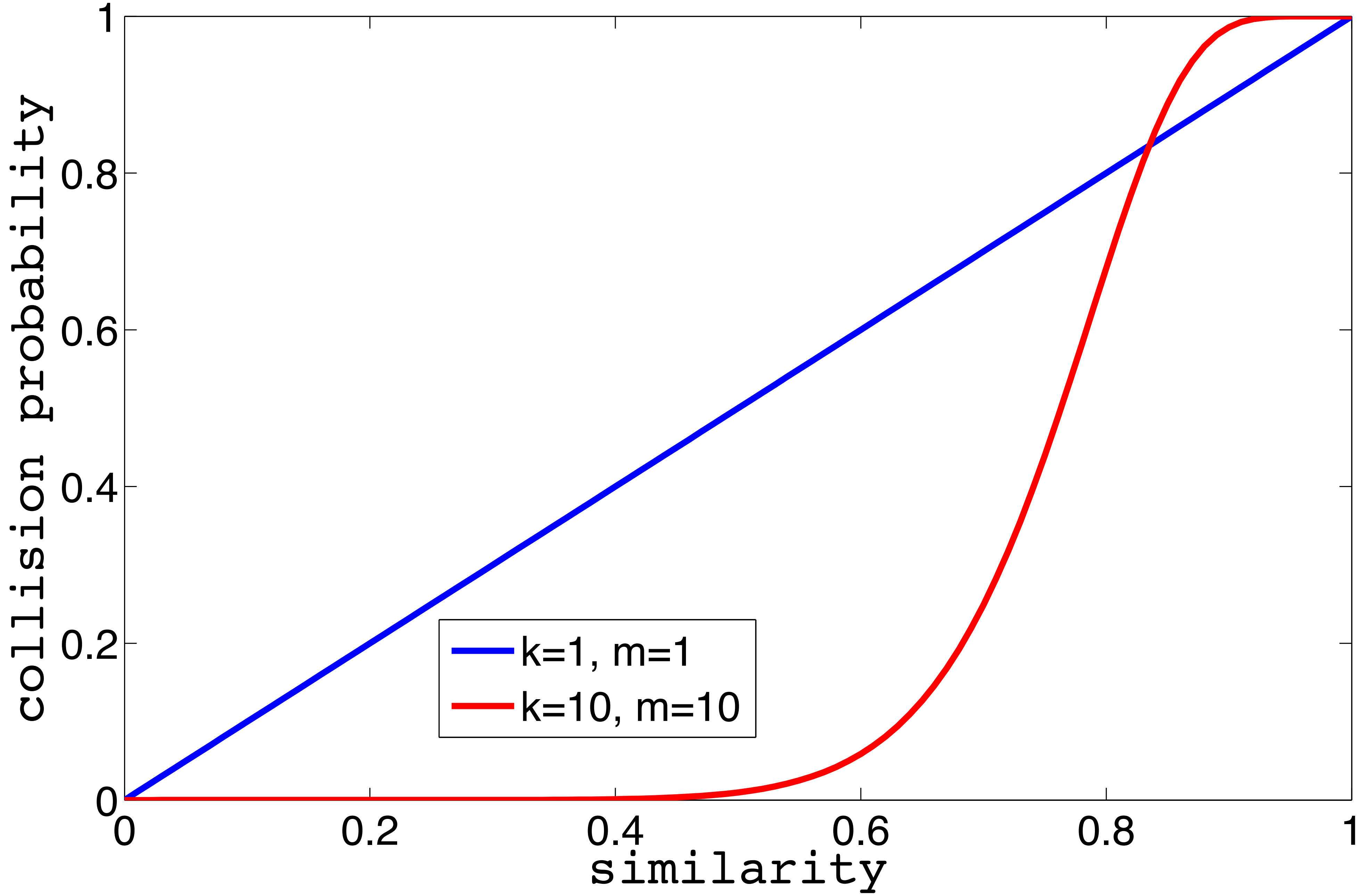


locality-sensitive hashing



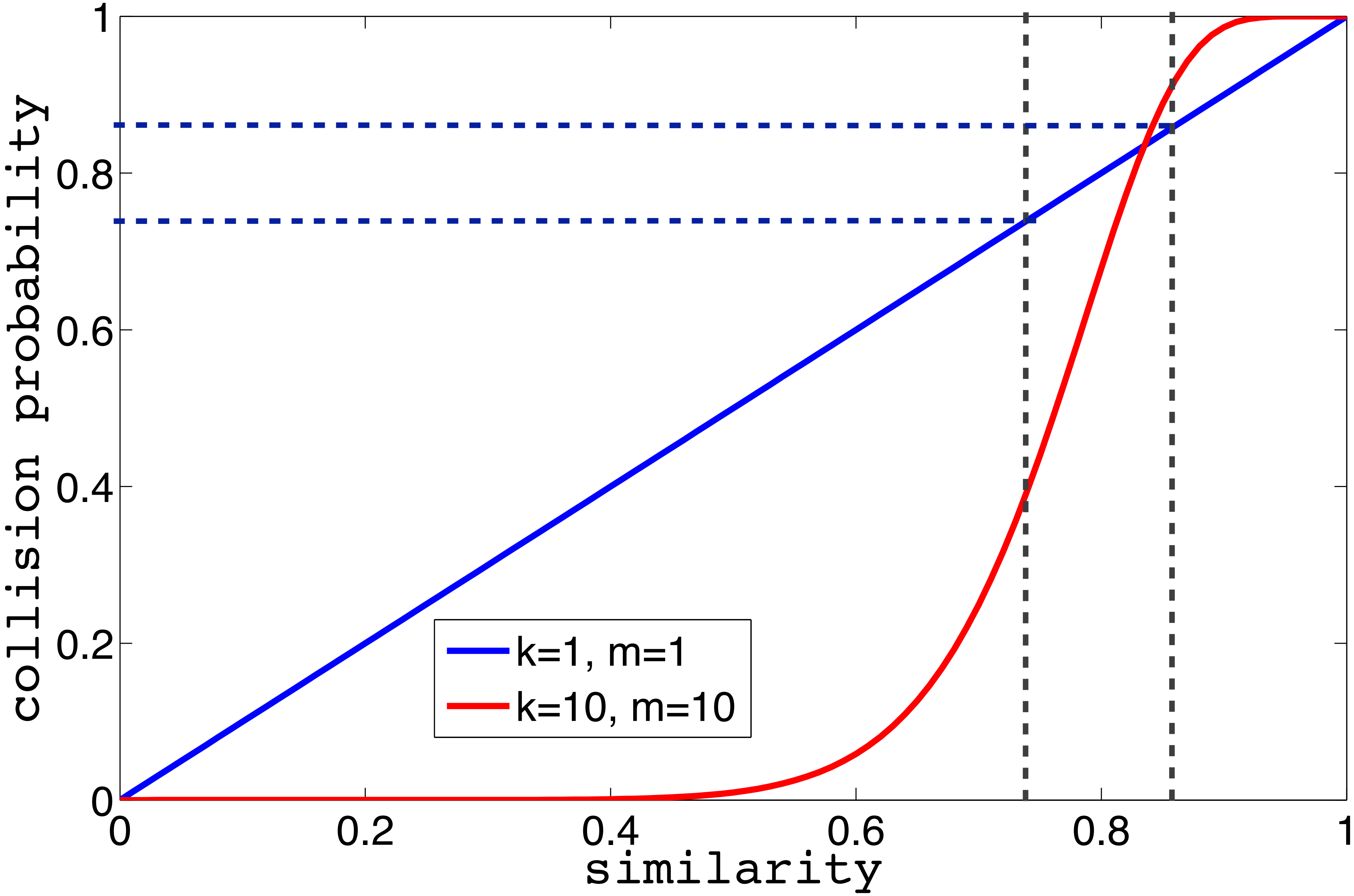
probability of collision

$$Pr[h(x) = h(y)] = 1 - (1 - s^k)^m$$



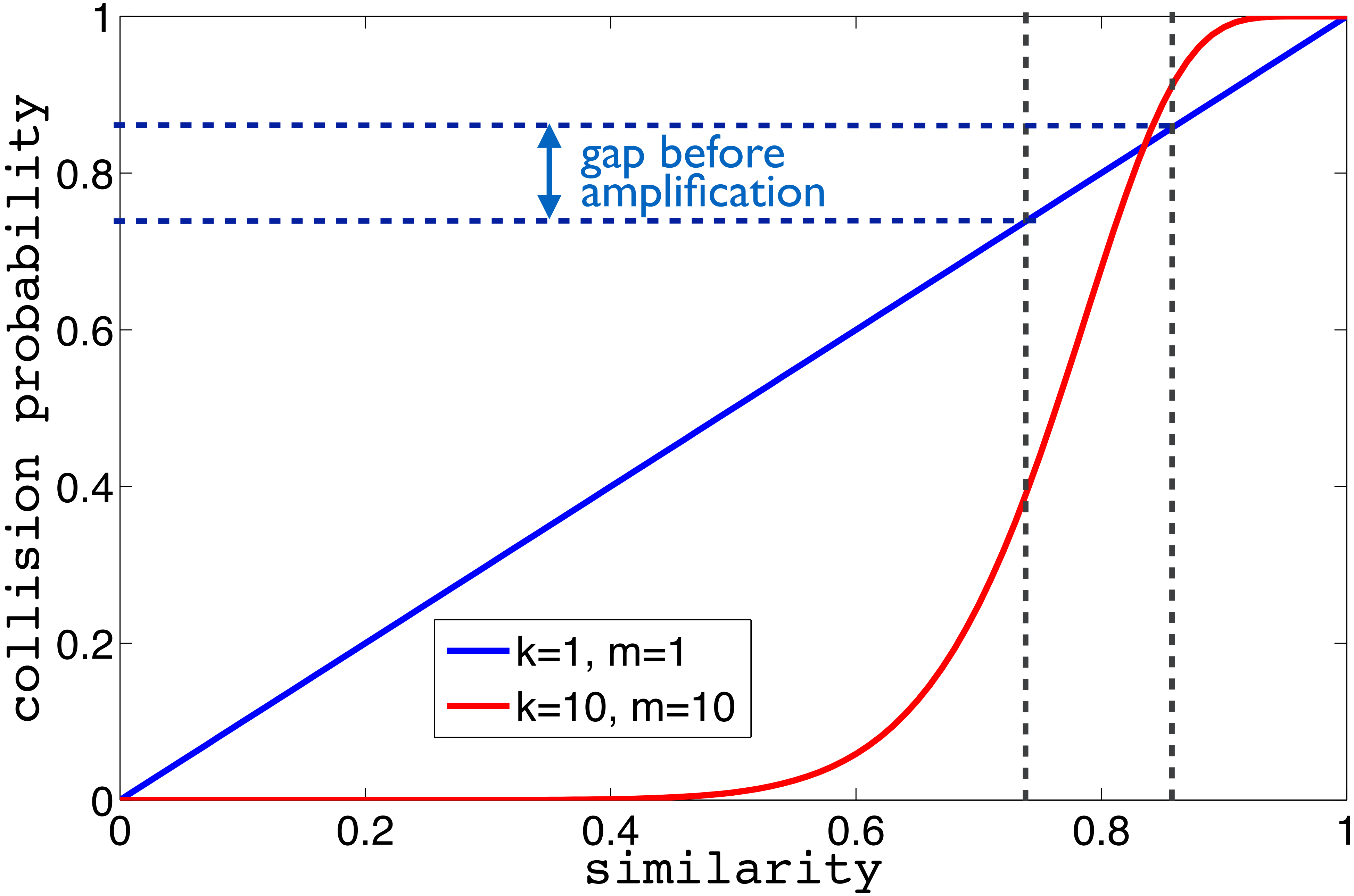
probability of collision

$$Pr[h(x) = h(y)] = 1 - (1 - s^k)^m$$



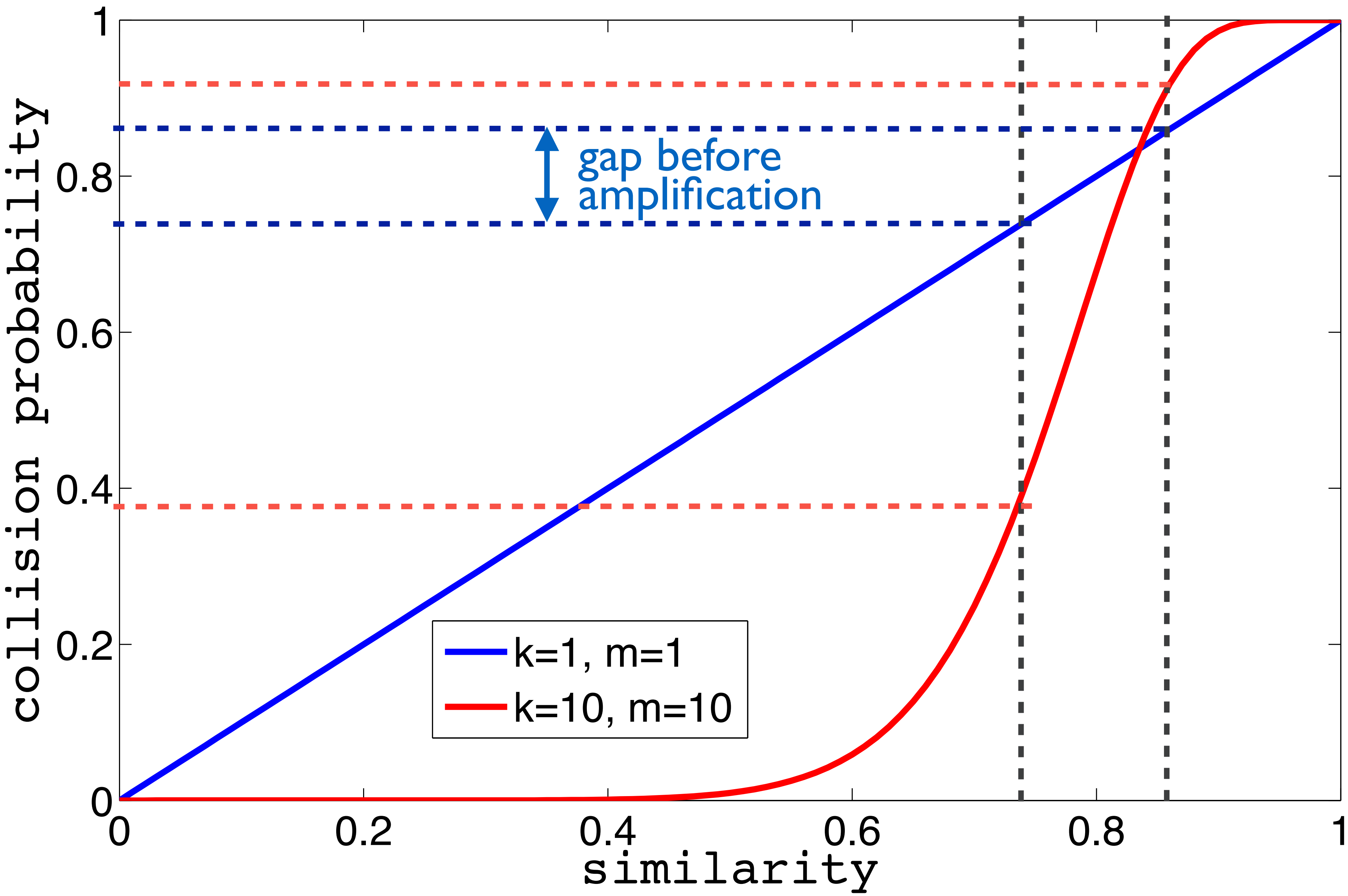
probability of collision

$$Pr[h(x) = h(y)] = 1 - (1 - s^k)^m$$



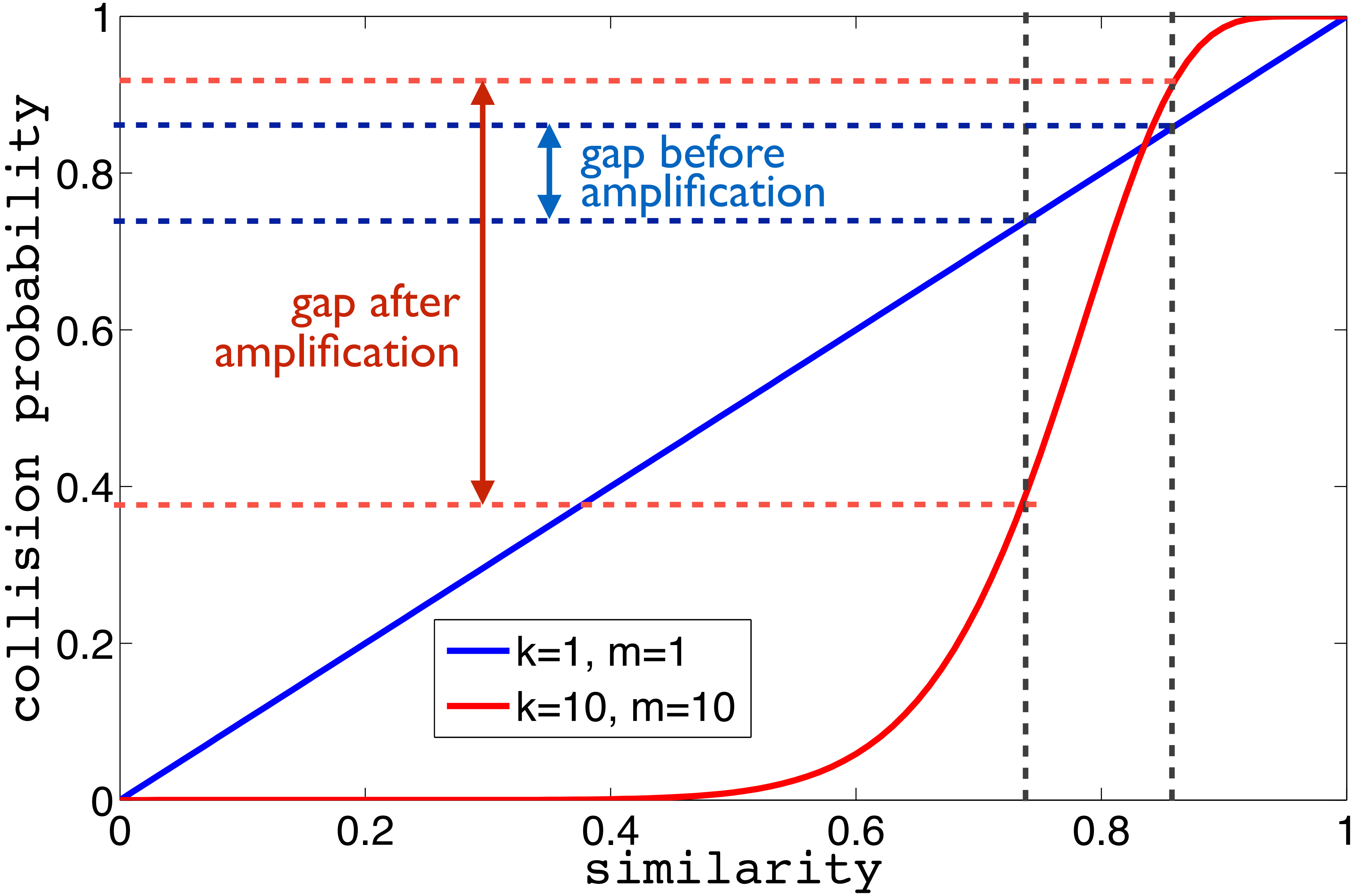
probability of collision

$$Pr[h(x) = h(y)] = 1 - (1 - s^k)^m$$



probability of collision

$$Pr[h(x) = h(y)] = 1 - (1 - s^k)^m$$



alto University

applicable to both similarity-search problems

1. similarity search problem

   hash all objects of X (off-line)

   hash the query object q (query time)

   filter out spurious collisions (query time)

2. all-pairs similarity problem

   hash all objects of X

   check all pairs that collide and filter out spurious ones (off-line)

locality-sensitive hashing for binary vectors

similarity search

```
preprocessing
input: set of vectors X
     for i=1...m times
          for each x in X
               form xi by sampling k random bits of x
               store x in bucket given by f(xi)
```

locality-sensitive hashing for binary vectors

similarity search

```
preprocessing
input: set of vectors X
     for i=1...m times
          for each x in X
               form xi by sampling k random bits of x
               store x in bucket given by f(xi)

query
input: query vector q
     Z = ∅
     for i=1...m times
          form qi by sampling k random bits of q
          Zi = { points found in the bucket f(qi) }
          Z = Z U Zi
     output all z in Z such that sH(q,z) ≥ s
```

locality-sensitive hashing for binary vectors
all-pairs similarity search

```
all-pairs similarity search
input: set of vectors X
     P = ∅
     for i=1...m times
          for each x in X
               form xi by sampling k random bits of x
               store x in bucket given by f(xi)
          Pi = { pairs of points colliding in a bucket }
          P = P U Pi
     output all pairs p=(x,y) in P such that sH(x,y) ≥ s
```

real-valued vectors

similarity search for vectors in R^d

quantize : assume vectors in $[1 \ldots M]^d$

real-valued vectors

similarity search for vectors in R^d

quantize : assume vectors in $[1...M]^d$

idea 1 : represent each coordinate in binary

real-valued vectors

similarity search for vectors in R^d

quantize : assume vectors in $[1...M]^d$

idea 1 : represent each coordinate in binary

sampling a bit does not work

think of 0011111111 and 0100000000

real-valued vectors

similarity search for vectors in R^d

quantize : assume vectors in $[1...M]^d$

idea 1: represent each coordinate in binary

sampling a bit does not work

think of 0011111111 and 0100000000

idea 2 : represent each coordinate in unary !

real-valued vectors

similarity search for vectors in R^d

quantize : assume vectors in $[1...M]^d$

idea 1: represent each coordinate in binary

sampling a bit does not work

think of 0011111111 and 0100000000

idea 2 : represent each coordinate in unary !

too large space requirements?

but do not have to actually store the vectors in unary

generalization of the idea

what might work and what not?

sampling a random bit is specific to binary vectors and Hamming distance / similarity

amplifying the probability gap is a general idea

generalization of the idea

consider object space X and a similarity function s

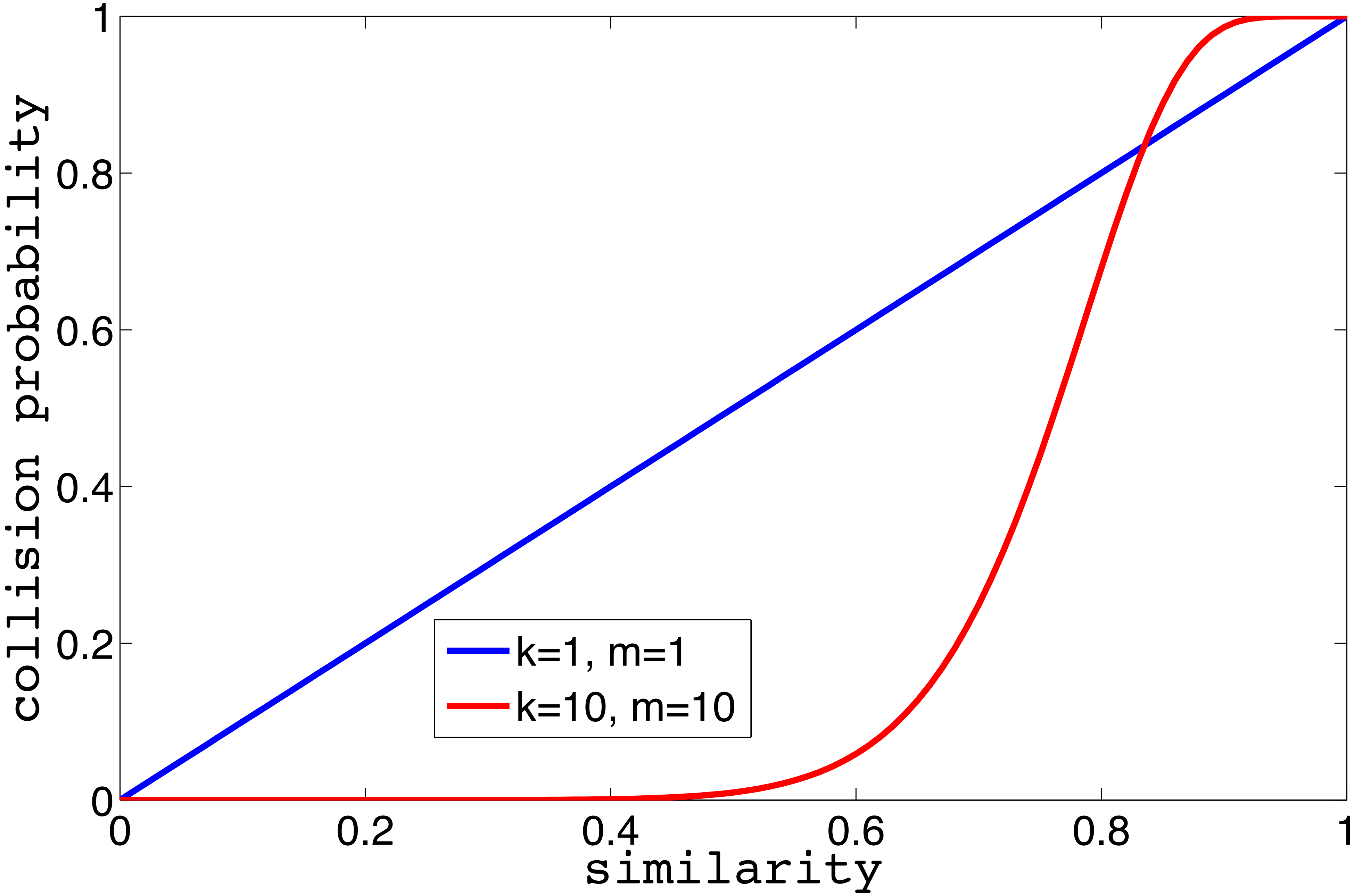
assume that we are able to design a family of hash functions such that

$$\Pr[h(x)=h(y)] = s(x,y), \text{ for all } x \text{ and } y \text{ in } X$$

we can then amplify the probability gap by stacking k functions and repeating m times

probability of collision

$$Pr[h(x) = h(y)] = 1 - (1 - s^k)^m$$



locality-sensitive hashing — generalization

similarity search

```
preprocessing
input: set of vectors X
     for i=1...m times
          for each x in X
               stack k hash functions and form
               xᵢ = h₁(x)...hₖ(x)
               store x in bucket given by f(xᵢ)
```

locality-sensitive hashing — generalization

similarity search

```
preprocessing
input: set of vectors X
     for i=1...m times
          for each x in X
               stack k hash functions and form
               xᵢ = h₁(x)...hₖ(x)
               store x in bucket given by f(xᵢ)

query
input: query vector q
     Z = ∅
     for i=1...m times
          stack k hash functions and form qᵢ = h₁(q)...hₖ(q)
          Zᵢ = { points found in the bucket f(qᵢ) }
          Z = Z ∪ Zᵢ
     output all z in Z such that sH(q,z) ≥ s
```

core of the problem

for object space X and a similarity function s

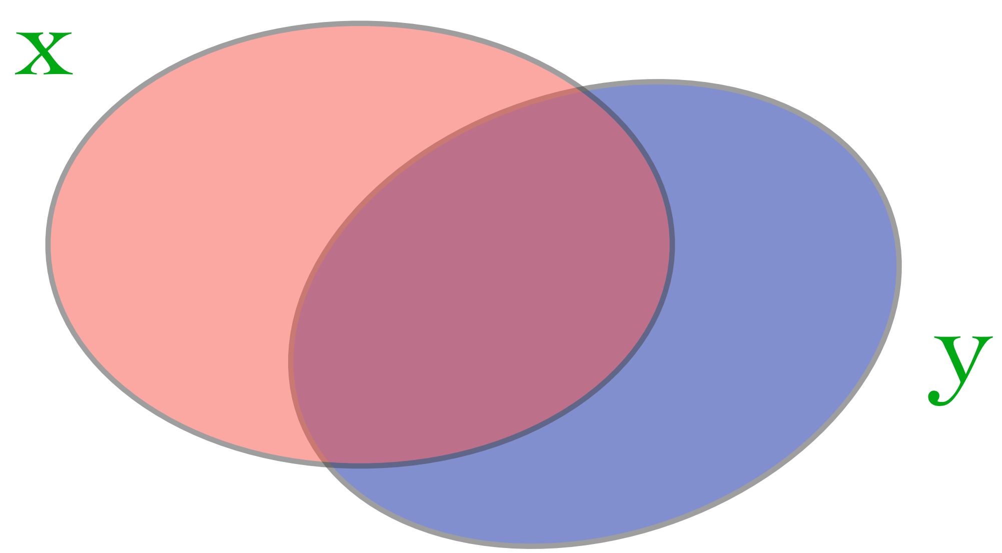
find family of hash functions such that :

$$Pr[h(x)=h(y)] = s(x,y), \text{ for all } x \text{ and } y \text{ in } X$$

what about the Jaccard coefficient?

set similarity $J(x, y) = \frac{|x \cap y|}{|x \cup y|}$

in Venn diagram:



objective

consider ground set U

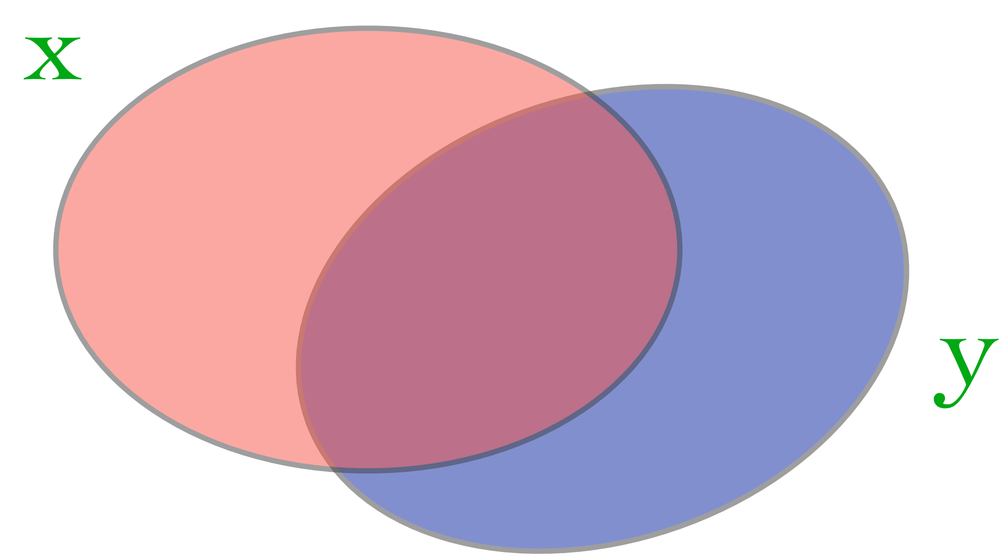
want to find hash-function family F such that

each set $x \subseteq U$ maps to h(x)

and Pr[h(x)=h(y)] = J(x,y),

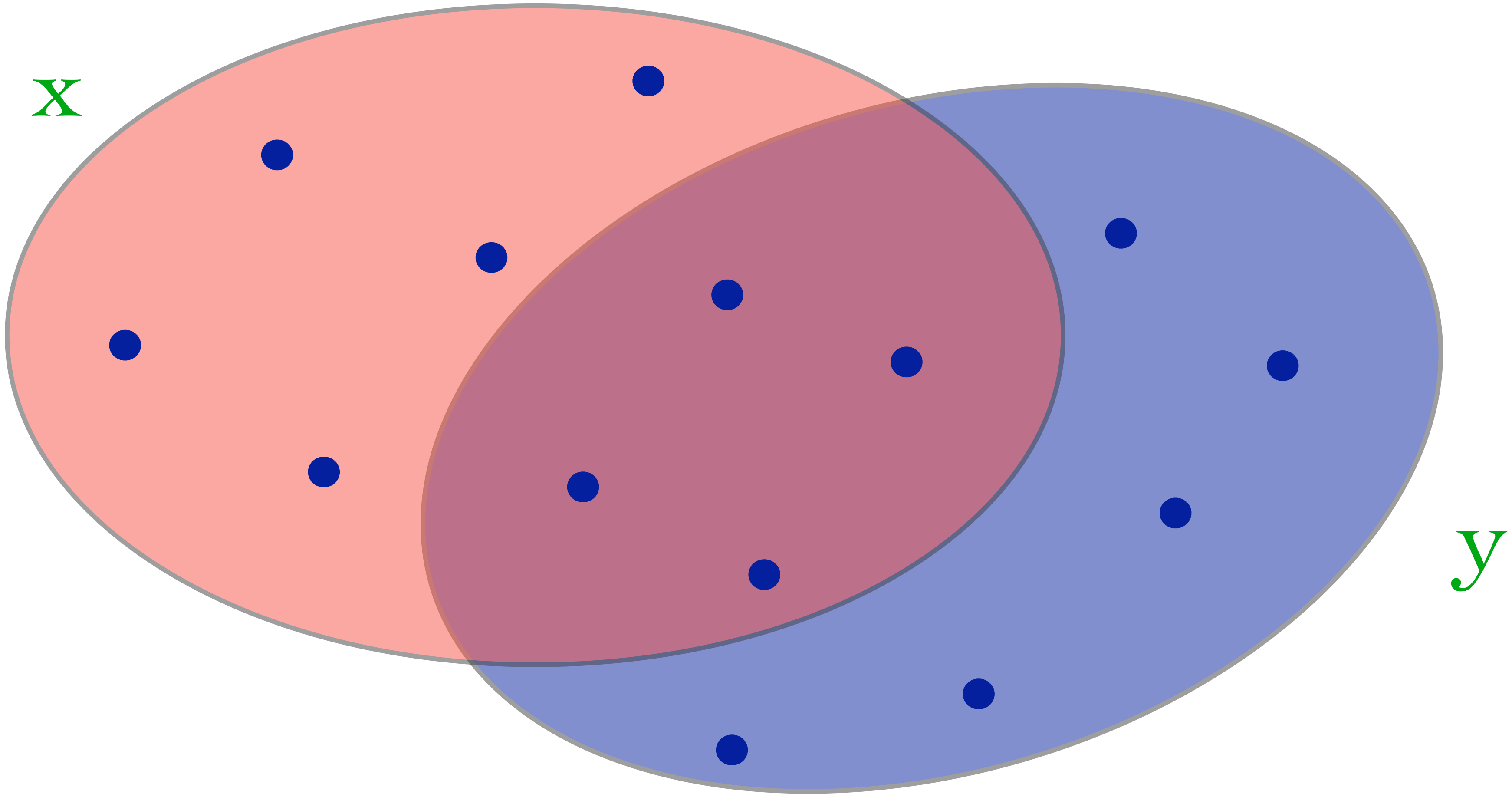
for all x and y in X

$$J(x,y) = \frac{|x \cap y|}{|x \cup y|}$$

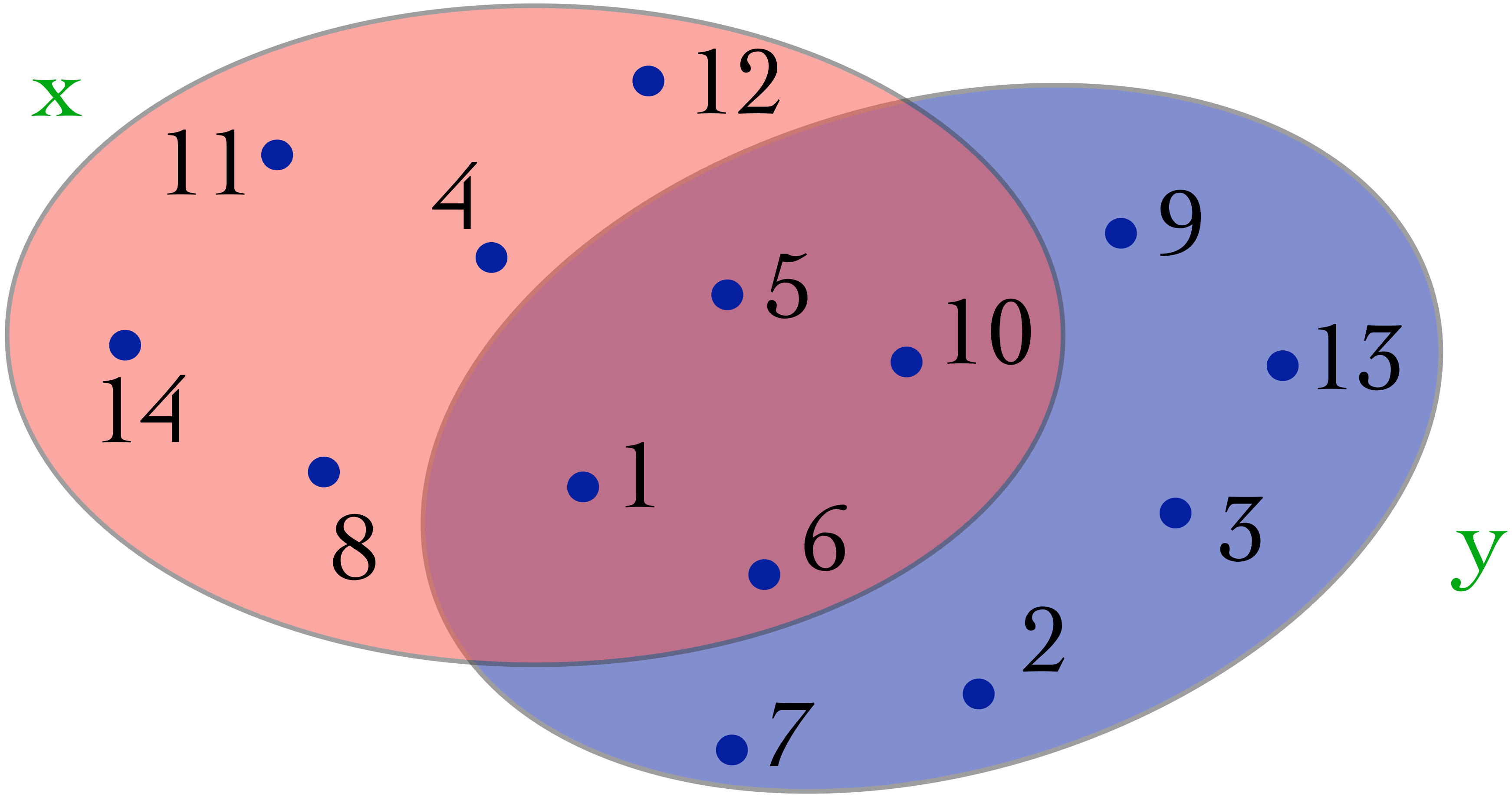


h(x) is also known as sketch

LSH for Jaccard coefficient

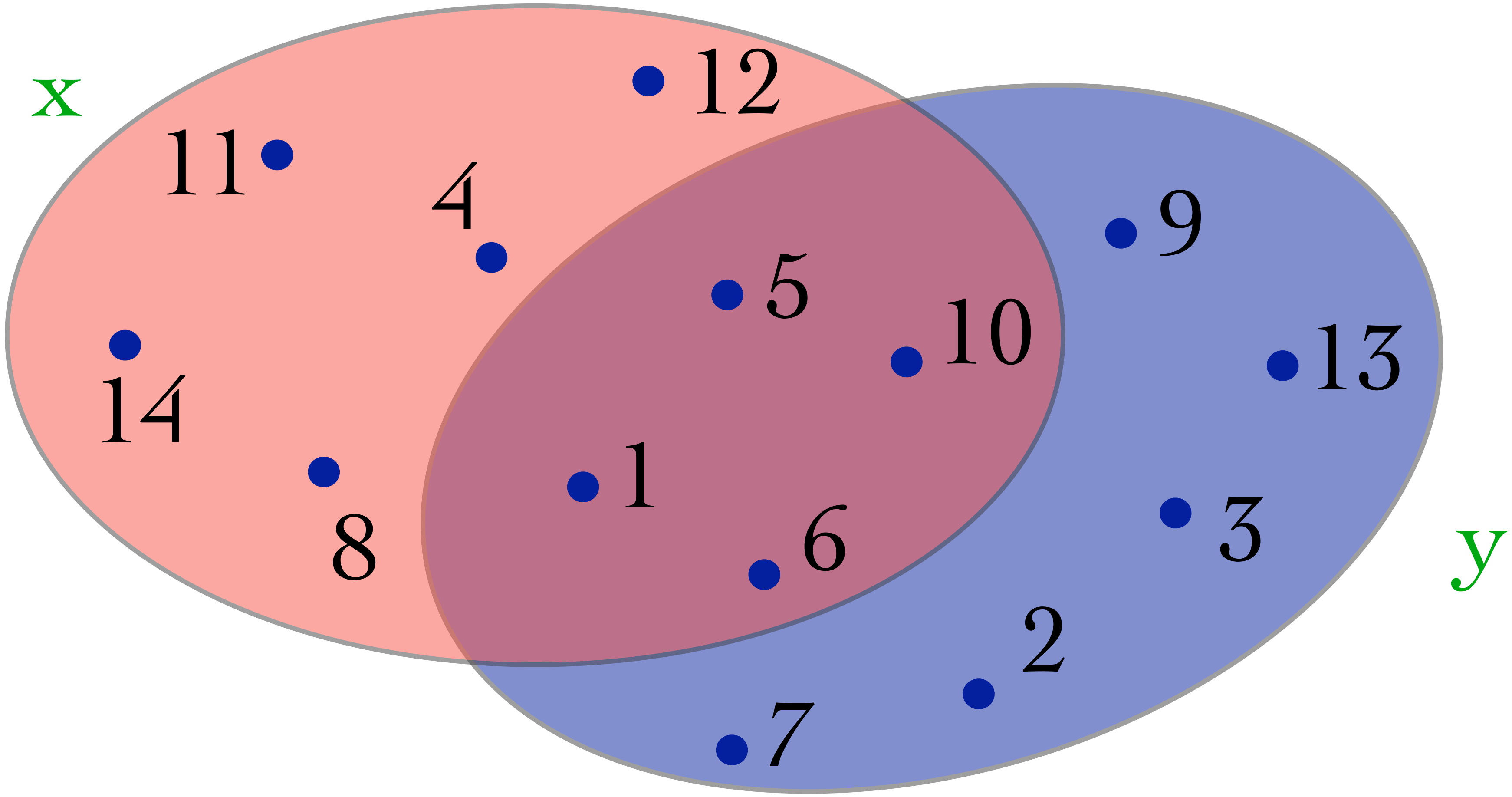


LSH for Jaccard coefficient



assume that the elements of U are randomly ordered

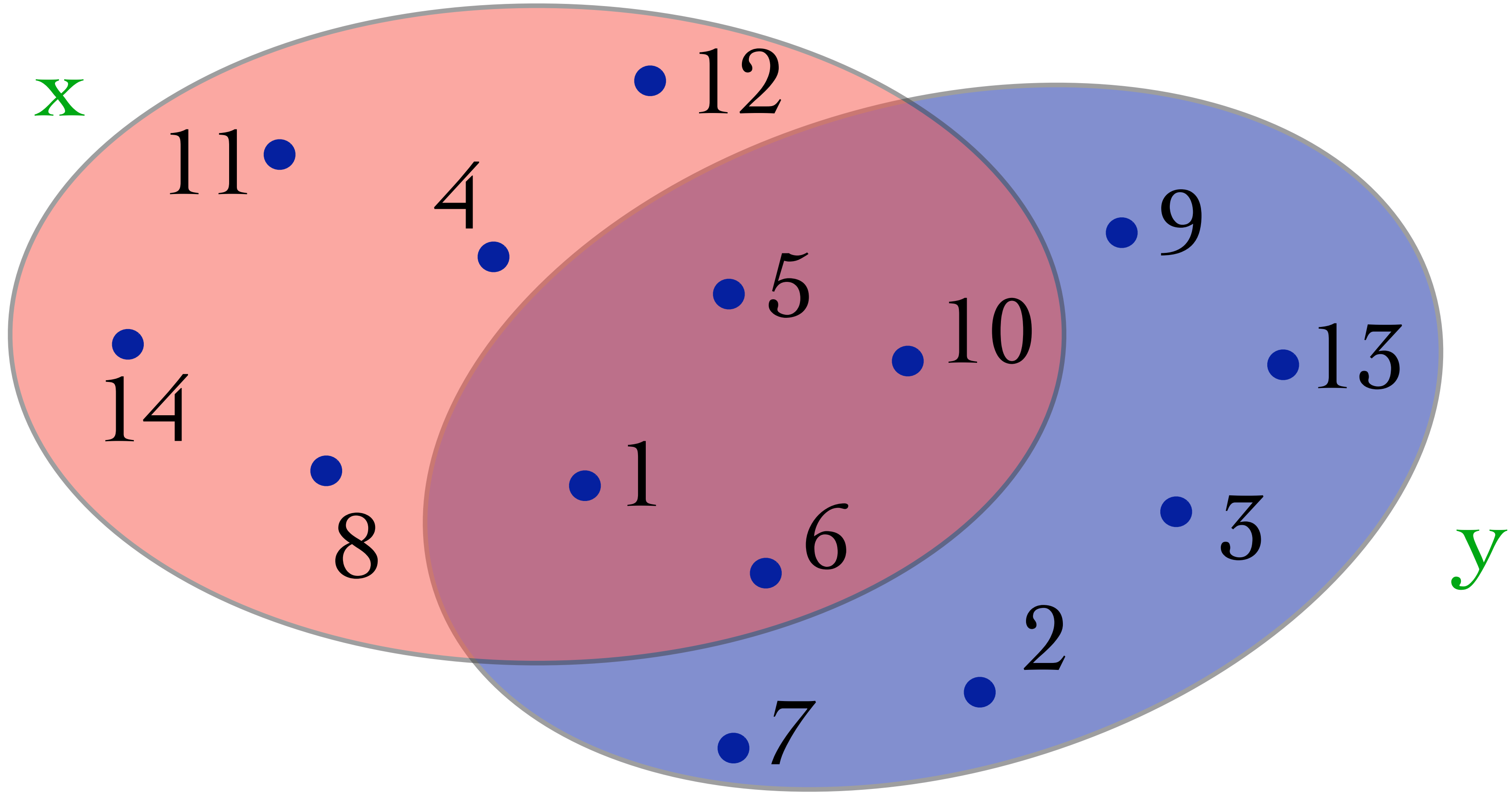
LSH for Jaccard coefficient



assume that the elements of U are randomly ordered

for each set look which element comes first in the ordering

LSH for Jaccard coefficient



assume that the elements of U are randomly ordered

for each set look which element comes first in the ordering

the more similar two sets, the more likely that the same element comes first in both

LSH for Jaccard coefficient

consider ground set U of m elements

consider random permutation $r : U \rightarrow [1...m]$

for any set $x = \{ x_1,...,x_k \} \subseteq U$ define

$$h(x) = \min_i \{ r(x_i) \}$$

(the minimum element in the permutation)

LSH for Jaccard coefficient

consider ground set U of m elements

consider random permutation $r : U \rightarrow [1...m]$

for any set $x = \{ x_1,...,x_k \} \subseteq U$ define

$$h(x) = \min_i \{ r(x_i) \}$$

(the minimum element in the permutation)

then, as desired

$$Pr[h(x)=h(y)] = J(x,y), \text{ for all } x \text{ and } y \text{ in } X$$

LSH for Jaccard coefficient

consider ground set U of m elements

consider random permutation $r : U \to [1...m]$

for any set $x = \{ x_1,...,x_k \} \subseteq U$ define

$$h(x) = \min_i \{ r(x_i) \}$$

(the minimum element in the permutation)

then, as desired

$Pr[h(x)=h(y)] = J(x,y)$, for all x and y in X

prove it !

LSH for Jaccard coefficient

scheme known as min-wise independent permutations

extremely elegant but impractical

LSH for Jaccard coefficient

scheme known as min-wise independent permutations

extremely elegant but impractical

why ?

LSH for Jaccard coefficient

scheme known as min-wise independent permutations

extremely elegant but impractical

why ?

keeping permutations requires a lot of space

in practice small-degree polynomial hash functions can be used

leads to approximately min-wise independent permutations

finding similar documents

problem : given a collection of documents, find pairs of documents that have a lot of common text

applications

- identify mirror sites or web pages
- plagiarism
- similar news articles

finding similar documents

problem easy when want to find exact copies

how to find near-duplicates?

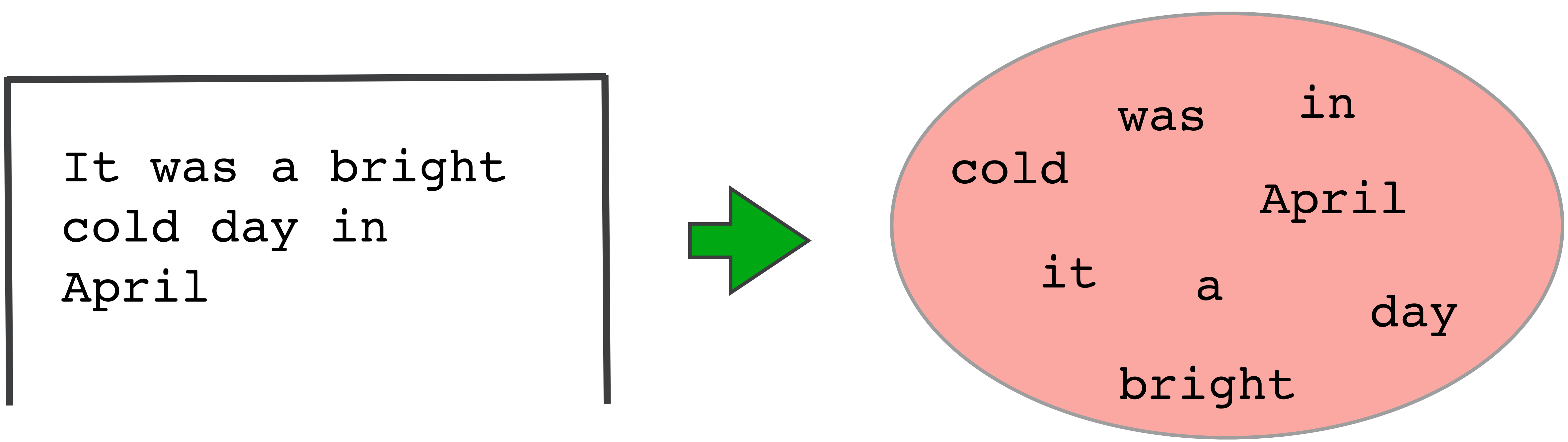
finding similar documents

problem easy when want to find exact copies

how to find near-duplicates?

represent documents as sets

bag of word representation



Aalto University

shingling

```
It was a bright cold day in April
```

document

shingling

```
It was a bright cold day in April
```

document

```
It was a bright
   was a bright cold
       a bright cold day
         bright cold day in
                cold day in April
```

shingles

shingling

```
It was a bright cold day in April
```

document

```
It was a bright
    was a bright cold
           a bright cold day
             bright cold day in
                    cold day in April
```

shingles

```
It was a bright
a bright cold day
cold day in April
was a bright cold
bright cold day in
```

bag of shingles

finding similar documents: key steps

shingling: convert documents (news articles, emails, etc) to sets

optimal shingle length?

LSH: convert large sets to small sketches, while preserving similarity

compare the signatures instead of the actual documents

locality-sensitive hashing for other data types?

angle between two vectors?

(related to cosine similarity)

other applications

image recognition, face recognition, matching fingerprints, etc.

next lectures

concentration bounds and tail inequalities

mining data streams