README : Vectors and Matrices

Vishal Bindal
2018CS50425

vdim

To Prove: vdim [e1;e2;e3;...en] = n
Proof by induction
Base case: For n=0, vdim [ ] returns 0
Induction hypothesis: Assume vdim [e1;e2;..ek] = k
Induction step: vdim e::[e1;e2;...ek] = 1 + (vdim [e1;e2;...ek]) = 1+k

Time complexity = O(n), n is the dimension of the vector

mkzerov

To Prove: mkzerov n = [0.;0.;...0.] (n times)
Proof by induction
Base case: For n=0, mkzerov 0 returns [ ]
Induction hypothesis: Assume mkzerov k = [0.;0.;...0.] (k times)
Induction step: mkzerov 0.0::[0.;0.;...0. (k times)] = [0.;0.;...0. (k+1 times)]
Time complexity = O(n), n is the dimension of the vector

isvzerov

To Prove: isvzerov [e1;e2;e3;...en] = true if e1=e2..=en=0. else false
Proof by induction
Base case: For n=0, isvzerov [ ] returns true
Induction hypothesis: Assume isvzerov [e1;e2;..ek] = true
Induction step: isvzerov e::[e1;e2;...ek] = true if e is 0.0 else false
Time complexity = O(n), n is the dimension of the vector

addv

To Prove: add [e1;e2;...en] [f1;f2;...fn] = [e1+f1;e2+f2;....en+fn]
Proof by induction
Base case: For n=0, addv [ ] [ ] returns [ ]
Induction hypothesis: Assume add [e1;e2;...ek] [f1;f2;...fk] = [e1+f1;e2+f2;....ek+fk]
Induction step: add e::[e1;e2;...ek] f::[f1;f2;...fk] = (e+f)::[e1+f1;e2+f2;....ek+fk]

Time complexity = O(n), n is the dimension of the vector

scalarmultv

To Prove: scalarmultv s [e1; e2; ...en] = [s *. e1; s *. e2;... s *. en]
Proof by induction
Base case: For n=0, scalarmultv s [ ] returns [ ]
Induction hypothesis: Assume scalarmultv s [e1; e2; ...ek] = [s *. e1; s *. e2;... s *. ek]
Induction step: scalarmultv s e::[e1; e2; ...ek] = (s *. e)::[s *. e1; s *. e2;... s *. ek]
Time complexity = O(n), n is the dimension of the vector

dotprodv

To Prove: dotprodv [e1;e2;...en] [f1;f2;...fn] = e1 *. f1 +. e2 *. f2 +. … en *. fn
Proof by induction
Base case: For n=0, dotprodv [ ] [ ] = 0
Induction hypothesis: Assume dotprodv [e1;e2;...ek] [f1;f2;...fk] = e1 *. f1 +. e2 *. f2 +. … ek *. fk
Induction step: dotprodv e::[e1;e2;...ek] f::[f1;f2;...fk] = e *. f +. e1 *. f1 +. e2 *. f2 +. … ek *. fk
Time complexity = O(n), n is the dimension of the vector

validmatrix

Method : Check if each row (float list) of the matrix has equal length.
Proof can be shown by induction, similar to isvzerov
Time complexity = O(n*m), where n is no of rows, m is the max length of rows

mdim

Method : Rows is the length of the (float list) list. Columns is the length of the first element (float list) of this list. Since length function is correct, mdim is also correct.
Time complexity = O(n+m), where n is no of rows, m is the length of each row

mkzerom

To Prove: mkzerom n m= [ls; ls;...ls] (n times), where ls = [0.;0. … 0.] (m times)
Proof by induction
Base case: For n=m=0, mkzerom returns [ ]
Induction hypothesis: Assume mkzerom k m= [ls; ls;...ls] (k times), where ls = [0.;0. … 0.] (m times)
Induction step: mkzerom k+1 m = (mkzerov m)::[ls; ls;...ls]
Since mkzerov is correct, mkzerom is also correct

Time complexity = O(n*m), where n is no of rows, m is the max length of rows

iszerom

To Prove: isvzerom [e1;e2;e3;...en] = true if e1=e2..=en= [0.;0. … 0.] else false
Proof by induction
Base case: For n=0, isvzerom [ ] returns true
Induction hypothesis: Assume isvzerom [e1;e2;..ek] = true

Induction step: isvzerom e::[e1;e2;...ek] = true if (iszerov e) is true else false
Since iszerov is correct, iszerom is also correct.
Time complexity = O(n*m), where n is no of rows, m is the max length of rows

mkunitm

Helper functions

idrow, which returns a row of the unit matrix

To Prove: idrow i n = [0.;0.;...1.;...0.] (n times), 1. at the iTH position
Proof by induction
Base case: For n=0, idrow returns [ ]
Induction hypothesis: Assume idrow i k = [e1; e2; … ek]
Induction step: idrow i+1 k+1 = 1.0::[e1;e2..ek] if i+1 is 1 else 0.0::[e1;e2;..ek].
Since i can become 1 only once as it is being changed at each step, 1. will appear only once at the required position

mkrows, which generates a list of all rows
Recursively call idrow for all rows, can be proved by induction

Calls mkrows. Since mkrows is correct, it is also correct.
Time complexity = O(n*m), where n is no of rows, m is the max length of rows

isunitm

Helper functions

issquare, which returns whether a matrix is a square matrix
Calls mdim and checks whether no of rows = no of columns

iscorrow, which checks if given row is correct (i.e. 1 at given pos and 0 at others)
Proof similar to idrow above, except in this case check the existing entries and return true/false accordingly

isidentity, which recursively checks all rows by calling iscorrow
Can be proved easily by induction

Calls issquare and isidentity, and returns true if both true, else false
Time complexity = O(n*m), where n is no of rows, m is the max length of rows

addm

Recursively add row by row by calling addv
Since addv is correct, this is also correct
Time complexity = O(n*m), where n is no of rows, m is the max length of rows

scalarmultm

Recursively scale each row of matrix by calling scalarmultv
Since scalarmultv is correct, this is also correct
Time complexity = O(n*m), where n is no of rows, m is the max length of rows

transm

To Prove: transm [[a11;a12;...a1m];[a21;a22;...a2m];...[an1;an2;...anm]] = [[a11;a21;..an1];...[a1m;a2m;..anm]]
Proof by induction
Base case: For n=m=0, transm returns [ ]
Induction hypothesis: transm [[a11;a12;...a1k];[a21;a22;...a2k];...[an1;an2;...ank]] = [[a11;a21;..an1];...[a1k;a2k;..ank]] for some k
Induction step: transm [a1::[a11;a12;...a1k]; a2::[a21;a22;...a2k];...an::[an1;an2;...ank]] = (a1; a2; … an) :: [[a11;a21;..an1];...[a1k;a2k;..ank]]
This happens as (map hd rows) returns a list containing only the first element of each list in rows. xs::(map tl rows) returns the remaining matrix.
Time complexity = O(n*m), where n is no of rows, m is the max length of rows

removei

Removes ith element of given list
Time complexity: O(n), where n is length of list

detm

Helper functions:

> **iterm**: Returns over all the elements of the first row, and calls term for each of them, adding the result for every term. Correctness can easily be proved by induction.
> **term**: Calculates element*sign*(detm of matrix after removing rows and cols pertaining to the element). For removing cols, removei is used.
> Sign is passed by iterm, and is multiplied by -1 at each step of recursion (over elements of first row of matrix)

T(n) = n * T(n-1) for an n X n matrix
Hence **time complexity** = O(n!), where n is no of elements in the list.
Limitation: Time complexity can be improved to O(n^3) by using a better algorithm.
Trade-off: Efficiency will be improved, but the implementation will become harder to understand.

multm

Take transpose of second matrix, and then take dot product of rows to generate the product matrix.
if first matrix is m X n and second matrix is n X q,

then transpose = O(n*q)
Time for each element of product matrix = O(n) (Taking dot product of 2 n-dimension vectors)
No of elements in product matrix = m*q;
Time complexity = O(n*q)+ O(n*m*q) = O(n*m*q)

cofact

compute cofactor matrix of given n X n matrix
For a given row, compute matrix without that row in O(n) time
For each element in the row, remove the column corresponding to that element in O(n) time and calculate determinant of matrix in O((n-1)!)
Time complexity = (n*O(n)) *(n*O(n)*O((n-1)!)) = O(n! * n^3)

invm

Compute transpose of cofactor matrix, and scale each element by 1/determinant
Time complexity = O(n! * n^3) + O(n^2) + O(n!) + O(n^2) = O(n! * n^3)
Limitation: Time complexity can be improved by using a better algorithm
Trade-off: Efficiency will increase by using algorithm, but will become more complex

crossprodv

Make a 2 X 3 matrix of the 2 vectors, and take determinant of 3 submatrices to yield the 3 components of cross product
Time complexity = O(1), since the length and no of vectors are fixed

crossprodgen

In the (n-1) X n matrix of the n-1 vectors, take determinant of n submatrices of (n-1)X(n-1) to yield n components of cross product
Time complexity = O(n*(n-1)!) = O(n!)