[**E-commerce: Algorithm for calculating discounts**](http://stackoverflow.com/questions/16605357/e-commerce-algorithm-for-calculating-discounts)

The scenario is:

* e-commerce web-site
* lots of products
* lots of discounts mixed on these products

A product is identified by a unique ProductID and has a sales price. Very classic scenario. The product can also be in one or more discounts.

A discount can be of different types. One example of a discount is:

* Buy two or more within a set of products and get X percent off each product

A line item can only get one discount thus once a line item has been discounted it is not available for other discounts.

**Test Case Data:**

* Product-1: $10
* Product-2: $10
* Product-3: $50
* Product-4: $100

**Discount-A**: Buy two or more and get 20 % off the any of the following products

* Product-1
* Product-2
* Product-3
* Product-4

**Discount-B**: Buy product and get 50 % off the following product

* Product-3

**Test Scenario 1:**

**Basket**: containing line items with:

* Product-1
* Product-3
* Product-4

**Calculation #1**:

* Discount-A: Product-1, Product-3, Product-4 = $2 + $10 + $20 = $32
  + = $32 total saving

**Calculation #2**:

* Discount-A: Product-2, Product-4 = $2 + $20 = $22
* Discount-B: Product-3 = $25
  + = $22 + $25 = $47 total saving

Which means that a combination of **Discount-A** and **Discount-B** will give the best possible discount for the customer.

**Test Scenario 2:**

**Basket**: containing line items with:

* Product-3
* Product-4

**Calculation #1**:

* Discount-A: Product-3, Product-4 = $10 + $20 = $30
  + = $30 total saving

**Calculation #2**:

* Discount-B: Product-3 = $25
  + = $25 total saving

Which means that applying **Discount-A** will give the best possible discount for the customer.

In order to calculate the best discount for a given basket, literally all combinations of products and available discounts on these products must be evaluated.

Normally there is 30-40 line items in a basket each with 0-3 discounts each.

Basically I'm stuck with finding an efficient way to do this calculation.

Right now the algorithm I have for applying the discounts is something like:

* Clear discounts on the Basket
* Get all unique ProductID's for LineItems in the Basket
* Get all discounts available for these ProductID's
* For-Each Discount (unordered)
  + Apply the Discount if it is satisfied by non-discount flagged line items
    - Flag line items in discount as discounted

But this is not at all sufficient as it does not try out different combinations of line items / discounts

**SOLUTION TO THE PROBLEM**

Assuming that:

1. We can compute all available discounts based on your basket
2. Each product can only have a single discount applied to it
3. Each discount can only be used once

Then the problem becomes one that is called an [assignment](http://en.wikipedia.org/wiki/Assignment_problem) problem and can be optimally solved in O(n^3) using the [Hungarian algorithm](http://en.wikipedia.org/wiki/Hungarian_algorithm).

We will need to compute a matrix mat[x,y] containing the money saved if using discount a on product b. (If a discount does not apply, then set the money saved to 0.)

The Hungarian algorithm will compute the way of assigning discounts to products that saves the most money.

If we don't have the same number of discounts and products, then add dummy discounts (with zero savings) or dummy products (again with zero savings) until the number of discounts matches the number of products.

**CODE**

**Main.m**

%Hungaraian method to calculate best deal on an ecommerce website for a

%product under different discounts

%n=input('Enter the dimension of matrix\n')

mat=input('Enter the matrix\n')

[x,y]=size(mat)

if(x<y)

fprintf('Not a balanced matrix adding zeros to the row\n')

mat1=zeros(x+1,y);

for i=1:x

for j=1:y

mat1(i,j)=mat(i,j);

end

end

end

if(x>y)

fprintf('Not a balanced matrix adding zeros to the row\n')

mat1=zeros(x,y+1);

for i=1:x

for j=1:y

mat1(i,j)=mat(i,j);

end

end

end

if(x==y)

mat1=mat;

end

mat=mat1;

mat

[x,y]=size(mat);

% applying row operation

[rmin,rminpos]=min(mat,[],2);

for i=1:x

for j=1:y

mat(i,j)=mat(i,j)-rmin(i);

end

end

%applying column operation

[cmin,cminpos]=min(mat);

for j=1:y

for i=1:x

mat(i,j)=mat(i,j)-cmin(j);

end

end

mat

f=0;

result=0;

%this loop terminates when all 0s are eliminated from the matrix

%calling selecting zeros

mat=selectingZeros(mat,x,y);

for i=1:x

for j=1:y

if(mat(i,j)==-999)

mat(i,j)=0;

end

if(mat(i,j)==999)

result=result+mat1(i,j);

end

end

end

disp('the position with value 999 are selected as the optimal positions')

mat

disp('the orignal matrix');

mat1

fprintf('the optimal value is : %d',result);

**selectingZeros.m**

%this loop terminates when all 0s are eliminated from the matrix

function mat=selectingZeros(mat,x,y)

for i=1:x

rdone(i)=0;

end

for i=1:y

cdone(i)=0;

end

ze=1; %initially will check for 1 zero in column and row and increment this value in each successive iteration

while(1)

f=0;

for i=1:x

rzero(i)=0;

end

for i=1:y

czero(i)=0;

end

%check for no. of 0s in each row

rzero=countrowzero(mat,rzero);

for i=1:x

if(rzero(i)<=ze && rzero(i)>0)

for m=1:y

if(mat(i,m)==0)

mat(i,m)=999;

rdone(i)=1;

cdone(m)=1;

rzero(i)=rzero(i)-1;

for k=1:x

if(mat(i,k)==0)

mat(i,k)=-999;

rzero(i)=rzero(i)-1;

end

if(mat(k,m)==0)

mat(k,m)=-999;

rzero(k)=rzero(k)-1;

end

end

end

end

end

end

%applying column operation

%counting number of zero in each column

czero=countcolzero(mat,czero);

for j=1:y

if(czero(j)<=ze && czero(j)>0)

for m=1:x

if(mat(m,j)==0)

mat(m,j)=999;

czero(j)=czero(j)-1;

rdone(m)=1;

cdone(j)=1;

for k=1:x

if(mat(m,k)==0)

mat(m,k)=-999;

czero(k)=czero(k)-1;

end

if(mat(k,j)==0)

mat(k,j)=-999;

czero(j)=czero(j)-1;

end

end

end

end

end

end

for i=1:x

for j=1:y

if(mat(i,j)==0)

f=1;

break;

end

end

if(f==1)

break;

end

end

if(f~=1)

break;

end

ze=ze+1;

end

op=0;

for i=1:x

if(rdone(i)==0 || cdone(i)==0)

op=1;

end

end

if(op)

fprintf('Optimal solution is not reached..proceeding for finding the optimal solution\n')

mat

mat=optimizing(mat,x,y,rzero,czero);

else

fprintf('Optimal solution is reached..the End\n')

end

**optimizing.m**

function mat=optimizing(mat,x,y,rzero,czero)

%making 999 and -999as zeros and also counting the number of zeros

totzero=0;

for i=1:x

for j=1:y

if(mat(i,j)==999||mat(i,j)==-999)

mat(i,j)=0;

%counting total number of zeros in the mat

totzero=totzero+1;

end

end

end

%rline and cline corresponds to the lines we draw in the matrix such that

%it cover all the zeros

for i=1:x

rline(i)=0;

end

for i=1:y

cline(i)=0;

end

%drawing lines which cover all the zeros and terminates when total number

%of zeros become 0

while(totzero~=0)

rzero=countrowzero(mat,rzero);

czero=countcolzero(mat,czero);

[rmax,rmaxpos]=max(rzero);

[cmax,cmaxpos]=max(czero);

if(rmax>=cmax)

rline(rmaxpos)=1;

%making all the zeros in that row as 888

for i=1:x

if(mat(rmaxpos,i)==0)

mat(rmaxpos,i)=888;

totzero=totzero-1;

end

end

end

if (rmax<cmax)

cline(cmaxpos)=1;

%making all the zeros in that col as 888

for i=1:x

if(mat(i,cmaxpos)==0)

mat(i,cmaxpos)=888;

totzero=totzero-1;

end

end

end

end

%finding the minimum number in the matrix from uncovered region

minNum=999;

for i=1:x

if(rline(i)==1)

continue;

end

for j=1:y

if(cline(j)==1)

continue;

end

minNum=min(mat(i,j),minNum);

end

end

%making zeros in the matrix where it is 888 since intersection can be any

for i=1:x

for j=1:y

if(mat(i,j)==888)

mat(i,j)=0;

end

end

end

%adding minimun number from uncovered region to intersection and subs from

%uncovered

for i=1:x

for j=1:y

if(rline(i)==1&&cline(j)==1)

mat(i,j)=mat(i,j)+minNum;

end

if(rline(i)~=1&&cline(j)~=1)

mat(i,j)=mat(i,j)-minNum;

end

end

end

mat;

%calling the selecting zero to redo the process

mat=selectingZeros(mat,x,y);

end

**countcolzero.m**

function czero=countcolzero(mat,czero)

%check for no. of 0s in each col

%col operation

[x,y]=size(mat);

for i=1:x

for j=1:y

if(mat(i,j)==0)

czero(j)=czero(j)+1;

end

end

end

**countrowzero.m**

function rzero=countrowzero(mat,rzero)

%check for no. of 0s in each row

%row operation

[x,y]=size(mat);

for i=1:x

for j=1:y

if(mat(i,j)==0)

rzero(i)=rzero(i)+1;

end

end

end

**output**

Enter the matrix

[12,9,12,9;15,0,13,20;4,8,10,6]

mat =

12 9 12 9

15 0 13 20

4 8 10 6

x =

3

y =

4

Not a balanced matrix adding zeros to the row

mat =

12 9 12 9

15 0 13 20

4 8 10 6

0 0 0 0

mat =

3 0 3 0

15 0 13 20

0 4 6 2

0 0 0 0

Optimal solution is reached..the End

the position with value 999 are selected as the optimal positions

mat =

3 0 3 999

15 999 13 20

999 4 6 2

0 0 999 0

the orignal matrix

mat1 =

12 9 12 9

15 0 13 20

4 8 10 6

0 0 0 0

the optimal value is : 13Enter the matrix

[-12,-9,12,9;15,0,13,20;4,8,10,-6]

mat =

-12 -9 12 9

15 0 13 20

4 8 10 -6

x =

3

y =

4

Not a balanced matrix adding zeros to the row

mat =

-12 -9 12 9

15 0 13 20

4 8 10 -6

0 0 0 0

mat =

0 3 24 21

15 0 13 20

10 14 16 0

0 0 0 0

Optimal solution is reached..the End

the position with value 999 are selected as the optimal positions

mat =

999 3 24 21

15 999 13 20

10 14 16 999

0 0 999 0

the orignal matrix

mat1 =

-12 -9 12 9

15 0 13 20

4 8 10 -6

0 0 0 0

the optimal value is : -18