CS6890 HW04

Jonathan Arndt

July 31 2017

Contents

0.1 Problems

0.1.1 Problem 1

See TableauSimplex.java attached.

0.1.2 Problem 2

Utilizes TableauSimplex.java

```
2000acre
A and B crops
A: 1 person-day of labor and $90 of capital for each acre planted
B: 2 person-day of labor and $60 of capital for each acre planted

A produces $170 revenue/acre
B produces $190 revenue/acre

maximize 170A+190B
A:  1+2   <= 3000
B:  90+60 <= 150000

System.out.println(
  getMatrixString(
      TableauSimplex.solveSimplexTableau(
              new double[][]{
                  {1,2,1,0,3000},
                  {90,60,0,1,150000},
                  {-170,-190,0,0,0}
              }
            ),new String[][]{
                  {"x1", "x2", "s1", "s2", "P"},
                  {"Crop A", "Crop B","Profit"}
        })
  );

Produces this output:

        x1  x2  s1     s2      |  P
Crop A  0   1   3/4    -1/120  |  1000
Crop B  1   0   -1/2   1/60    |  1000
-----------------------------------------
Profit  0   0   115/2  5/4     |  360000
```

Part 1.

100 more person-days:

```
System.out.println(
  getMatrixString(
      TableauSimplex.solveSimplexTableau(
              new double[][]{
                  {1,2,1,0,3100},
                  {90,60,0,1,150000},
                  {-170,-190,0,0,0}
              }
          ),new String[][]{
                  {"x1", "x2", "s1", "s2", "P"},
                  {"Crop A", "Crop B","Profit"}
       })
  );
```

Produces this output:

```
        x1  x2  s1     s2      |  P
Crop A  0   1   3/4    -1/120  |  1075
Crop B  1   0   -1/2   1/60    |  950
-----------------------------------------
Profit  0   0   115/2  5/4     |  365750
```

Part 2.

$100 more available capital

```
System.out.println(
  getMatrixString(
      TableauSimplex.solveSimplexTableau(
              new double[][]{
                  {1,2,1,0,3000},
                  {90,60,0,1,150100},
                  {-170,-190,0,0,0}
              }
          ),new String[][]{
                  {"x1", "x2", "s1", "s2", "P"},
                  {"Crop A", "Crop B","Profit"}
       })
  );
```

```
Produces this output:

        x1  x2  s1     s2      |  P
Crop A  0   1   3/4    -1/120  |  6445/6
Crop B  1   0   -1/2   1/60    |  2855/3
-----------------------------------------
Profit  0   0   115/2  5/4     |  365875
```

Part 3.

Baseline: 360000
Increase person-hours by 100: 365750
Increase capital by $100: 365875

0.1.3 Problem 3

See TableauSimplex.java

```
Large and Small muffins
Large: 4oz dough, 2oz bran
Small: 1oz dough, 1oz bran
300oz dough, 160oz bran

maximize .25L+.1S
4+1<=300
2+1<=160

System.out.println(getMatrixString(
    TableauSimplex.solveSimplexTableau(
            new double[][]{
                {4,1,1,0,300},
                {2,1,0,1,160},
                {-.25,-.1,0,0,0}
            }
    ),new String[][]{
            {"x1", "x2", "s1", "s2", "P"},
            {"Large", "Small","Profit"}
    }));

Results in:
        x1  x2  s1    s2    |  P
Large   1   0   1/2   -1/2  |  70
Small   0   1   -1    2     |  20
-------------------------------------
```

```
Profit  0   0   1/40  3/40  |  39/2
```

Part 1.

```
double d = Double.MIN_VALUE;
int bran = 300;
while(true){
    Ratio[][] tableau = TableauSimplex.solveSimplexTableau(
            new double[][]{
                    {4, 1, 1, 0, bran++},
                    {2, 1, 0, 1, 160},
                    {-.25, -.1, 0, 0, 0}
            }
    );
    double q;
    if((q=tableau[tableau.length-1][tableau[0].length-1].getDoubleValue()) <= d)
        break;
    d = q;
}
System.out.println("Bran max with dough 160 is: "+(bran-1));

Results in:
Bran max with dough 160 is: 321
```

Part 2.

```
double d = Double.MIN_VALUE;
int dough = 160;
while(true){
    Ratio[][] tableau = TableauSimplex.solveSimplexTableau(
            new double[][]{
                    {4, 1, 1, 0, 300},
                    {2, 1, 0, 1, dough++},
                    {-.25, -.1, 0, 0, 0}
            }
    );
    double q;
    if((q=tableau[tableau.length-1][tableau[0].length-1].getDoubleValue()) <= d)
        break;
    d = q;
}
System.out.println("Bran max with dough 160 is: "+(dough-1));

Results in:
```

```
Bran max with dough 160 is: 301
```

0.1.4 Problem 4

See TableauSimplex.java

```
250mg of Calcium
500mg of phosphorous
9mg of iron
apples,orances,bannanas

minimize calories

minimize 60a+50o+90b
calcium:      10a+40o+60b = 250
phosphorous:  10a+20o+30b = 500
iron:         .3a+.2o+.6b = 9

10a+40o+60b <= 250
-10a-40o-60b <= 250
10a+20o+30b <= 500
-10a-20o-30b <= 500
.3a+.2o+.6b <= 9
-.3a-.2o-.6b <= 9


System.out.println(getMatrixString(
    TableauSimplex.solveSimplexTableau(
            new double[][]{
                    {10,40,60,1,0,0,0,0,0,250},
                    {-10,-40,-60,0,1,0,0,0,0,-250},
                    {10,20,30,0,0,1,0,0,0,500},
                    {-10,-20,-30,0,0,0,1,0,0,-500},
                    {.3,.2,.6,0,0,0,0,1,0,9},
                    {-.3,-.2,-.6,0,0,0,0,0,1,-9},
                    {-60,-50,-90,0,0,0,0,0,0,0}
            }
    ),new String[][]{
            {"Apples", "Oranges", "Bananas", "s1","s2","s3","s4","s5","s6", "P"},
            {"Calcium", "-Calcium","Phosphorus","-Phosphorus","Iron","-Iron","Calori
    }));

Results in:
```

	Apples	Oranges	Bananas	s1	s2	s3	s4	s5	s6	P
Calcium	1	4	6	1/10	0	0	0	0	0	25
-Calcium	0	0	0	1	1	0	0	0	0	0
Phosphorus	0	-20	-30	-1	0	1	0	0	0	250
-Phosphorus	0	20	30	1	0	0	1	0	0	-250
Iron	0	-1	-6/5	-3/100	0	0	0	1	0	3/2
-Iron	0	1	6/5	3/100	0	0	0	0	1	-3/2
Calories	0	190	270	6	0	0	0	0	0	1500

No Apples, 190 oranges and 270 bananas

0.2 Source Code

0.2.1 TableauSimplex.java

```
package assignment;

import utilities.Ratio;

import java.util.HashSet;
import java.util.Set;

public class TableauSimplex {

    public static void main(String[] args) throws Exception {
//          new MatrixSimplex(new String[][]{
//                  {"p", "c", "s1", "s2", "P"},
//                  {"p", "c", "P"}
//          },new int[][]{
//                  {4,5,1,0,0},
//                  {1,3,0,1,0},
//                  {8,12,0,0,1}
//          },new int[]{20*16,15*16,20*12},
//                  new int[]{-3,-4,0,0,0,0}).doSimplex();

        System.out.println(getMatrixString(
                TableauSimplex.solveSimplexTableau(
                        new double[][]{
                                {4,1,1,0,300},
                                {2,1,0,1,160},
                                {-.25,-.1,0,0,0}
                        }
                ),new String[][]{
                        {"Large", "Small", "s1", "s2", "P"},
                        {"Dough", "Bran","Profit"}
                }));
//          double d = Double.MIN_VALUE;
//          int dough = 160;
//          while(true){
//              Ratio[][] tableau = TableauSimplex.solveSimplexTableau(
//                      new double[][]{
//                              {4, 1, 1, 0, 300},
//                              {2, 1, 0, 1, dough++},
//                              {-.25, -.1, 0, 0, 0}
//                      }
```

```
//              );
//              double q;
//              if((q=tableau[tableau.length-1][tableau[0].length-1].getDou
//                  break;
//              d = q;
//          }
//          System.out.println("Bran max with dough 160 is: "+(dough-1));
    }

    private static void printMatrix(double[][] doubles) {
        for (int i = 0; i < doubles.length; i++){
            for (int j = 0; j < doubles[i].length; j++)
                System.out.print(doubles[i][j]+"\t");
            System.out.println();
        }

    }

    private static String getMatrixString(Ratio[][] doubles) throws Excep
        String[][] names = new String[2][];
        names[0] = new String[doubles[0].length];
        names[1] = new String[doubles.length];
        String c = "x";
        for (int i = 0; i < names.length; i++) {
            for (int j = 0; j < names[i].length; j++)
                if(j == names[i].length - 1){
                    if(i == 0)  names[i][j] = "bs";
                    else names[i][j] = "z";
                }else
                    names[i][j] = c+""+j;
            c = "y";
        }
        return getMatrixString(doubles,names);
    }
    public static String getMatrixString(Ratio[][] doubles, String[][] na
        if(isNotSameLength(doubles))
            throw new Exception("Matrix is not Same length");

        int[] length = new int[doubles[0].length+1];
        for (int i = 0; i < length.length; i++)
            length[i] = Integer.MIN_VALUE;
        for (int i = 0; i < doubles.length; i++)
            for (int j = 0; j < doubles[i].length; j++)
                length[j] = Math.max(length[j],(doubles[i][j]+"").length(
```

```
        for (int i = 0; i < names[0].length; i++)
            length[i] = Math.max(length[i],names[0][i].length());
        for (int i = 0; i < names[1].length; i++)
            length[length.length-1] = Math.max(length[length.length-1],na

        String s = "";
        int lenghtSum = 4, k = doubles.length;
        for (int i = 0; i < length.length; i++)
            lenghtSum+=length[i]+2;
        for (int i = 0; i < doubles.length; i++) {
            if(i == 0) {
                s+=String.format("%-" + (length[length.length-1]+2)+"s","
                for (int j = 0; j < names[0].length; j++)
                    if(j == names[0].length - 1)
                        s+="| "+String.format("%-" + (length[j] + 2) + "
                    else
                        s += String.format("%-" + (length[j] + 2) + "s",
                s+="\n";
            }
            if(i == doubles.length - 1){
                for (int j = 0; j < lenghtSum; j++)
                    s+="-";
                s+="\n";
            }
            for (int j = 0; j < doubles[i].length; j++) {
                if(j == 0)
                    s+=String.format("%-"+(length[length.length-1]+2)+"s"
                if(j == doubles[i].length - 1)
                    s+="| ";
                s += String.format("%-" + (length[j] + 2) + "s", doubles[
            }
            s+="\n";
        }
        return s;
    }

    private static boolean isNotSameLength(Ratio[][] doubles) {
        int l = doubles[0].length;
        for (int i = 1; i < doubles.length; i++)
            if(doubles[i].length != l)
                return true;
        return false;
    }
```

```
public TableauSimplex(String[][] coefficientMatrix, int[][] basisMatr
    this.coefficientMatrix = coefficientMatrix;
    this.basisMatrix = basisMatrix;
    this.b = b;
    this.cost = cost;
}

/**
 * coefficientMatrix is like this
 * [
 *  [x1,x2,x3,s1,s2,P],
 *  [x1,x3,P]
 * ]
 *
 * basisMatrix is like this
 * [
 *  [2,3,2,1,0,0],
 *  [1,1,2,0,1,0]
 * ]
 */
private String[][] coefficientMatrix;
private int[][] basisMatrix;
/**
 * b is the solutions to the matrix like this
 * [0,0,0,1000,800]
 *
 * cost is like this:
 * [-7,-8,-10,0,0,0]
 */
private int[] b, cost;


/**
 * tableau has the following format:
 *           z    x1       x2       ... xn       RHS
 *    xb,1   0    v1,1     v1,2     ... v1,n     w1
 *    xb,2   0    v2,1     v2,2     ... v2,n     w2
 *    ...    ...  ...      ...      ... ...      ...
 *    xb,n   0    vm,1     vm,2     ... vm,n     wm
 *       z   0    z1-c1    z2-c2    ... zn-xn    cbw
 *
 * coefficients
 *
 * basis
```

```
 * cost  bT
 */
public void doSimplex(){
    Ratio[][] tableau = createTableau();
    try {
        System.out.println(getMatrixString(solveSimplexTableau(tablea
    } catch (Exception e) {
        System.out.println("No Solution");
    }
}

public Ratio[][] createTableau() {
    int r = coefficientMatrix[0].length + 1, c = basisMatrix.length +
    double[][] tableau = new double[c][r];
    for (int i = 0; i < basisMatrix.length; i++)
        for (int j = 0; j < basisMatrix[i].length; j++)
            tableau[i][j] = basisMatrix[i][j];
    for (int i = 0; i < cost.length; i++)
        tableau[c - 1][i] = cost[i];
    for (int i = 0; i < b.length; i++)
        tableau[i][r - 1] = b[i];
    return convertToRatios(tableau);
}

/** coefficientMatrix is like this
 * [
 *  [x1,x2,x3,s1,s2,P],
 *  [x1,x3,P]
 * ]
 */
private void printResults(double[][] tableau, String[][] coefficientM
    int v = tableau[0].length - 1, k = coefficientMatrix[0].length -
    String[] c1 = coefficientMatrix[1], nc1 = new String[coefficientM
    Set<String> s = new HashSet<String>();
    for(String i : coefficientMatrix[1])
        s.add(i);
    for (int i = 0; i < coefficientMatrix[0].length; i++)
        if(!s.contains(coefficientMatrix[0][i]))
            nc1[k--] = coefficientMatrix[0][i];
    for (int i = 0; i < c1.length; i++)
        nc1[i] = c1[i];

    for (int i = 0; i < nc1.length; i++)
        System.out.println(nc1[i]+" = "+(tableau.length - 1 < i ? 0 :
```

```
}

/**
 * is z optimal
 * what is the EV
 * min ratio test departing variable (dv)
 * pivot on the dv
 *
 */

/**
 * Row reduction of simplex tableau
 * @param tableau example:
 *                [[2,3,2,1,0,0|1000],
 *                 [1,1,2,0,1,0|800],
 *                 [-7,-8,-10,0,0,1|0]]
 *                last column is the z, last row is always solutions
 */
public static Ratio[][] solveSimplexTableau(double[][] tableau) throw
    checkMatrix(tableau);
    Ratio[][] ntableau = convertToRatios(tableau);
    return solveSimplexTableau(ntableau);
}public static Ratio[][] solveSimplexTableau(Ratio[][] ntableau) thro
    int iteration = 0;
    while (lastColumnHasNegative(ntableau)){
        int pivotColumn = getPivotColumnIndex(ntableau);
        if(isNoSolution(ntableau,pivotColumn))
            throw new Exception("No Solution to tableau");
        int pivotRow = getPivotRowIndex(ntableau, pivotColumn);
        ntableau = makePivotOne(ntableau,pivotColumn,pivotRow);
        ntableau = makePivotRowZero(ntableau,pivotColumn,pivotRow);

        if(iteration++ > iterationMax){
            if(deepEquals(ntableau,lastMatrix) || deepEquals(ntableau
                if(repeatMatrix++ > repeatMatrixMax)
                    throw new Exception("No Solution: Tableau got rep
            if(which == 1) {
                lastMatrix = deepCopyIntMatrix(ntableau);
                which = 0;
            }else{
                secondToLastMatrix = deepCopyIntMatrix(ntableau);
                which = 1;
            }
        }
```

```
        }
        return ntableau;
    }

    private static Ratio[][] convertToRatios(double[][] tableau) {
        Ratio[][] ratios = new Ratio[tableau.length][tableau[0].length];
        for (int i = 0; i < tableau.length; i++)
            for (int j = 0; j < tableau[i].length; j++)
                ratios[i][j] = new Ratio(tableau[i][j]);
        return ratios;
    }

    private static int repeatMatrix = 0, repeatMatrixMax = 100, iterationM
    private static Ratio[][] lastMatrix;
    private static Ratio[][] secondToLastMatrix;

    public static Ratio[][] deepCopyIntMatrix(Ratio[][] input) {
        if (input == null)
            return null;
        Ratio[][] result = new Ratio[input.length][];
        for (int r = 0; r < input.length; r++)
            result[r] = input[r].clone();
        return result;
    }
    public static boolean deepEquals(Ratio[][] a, Ratio[][] b){
        if(a == null && b == null)
            return true;
        if( a == null || b == null)
            return false;
        if(a.length != b.length)
            return false;
        for (int i = 0; i < a.length; i++)
            for (int j = 0; j < a[i].length; j++){
                if(a[i].length != b[i].length)
                    return false;
                if(a[i][j] != b[i][j])
                    return false;
            }
        return true;
    }

    private static void checkMatrix(double[][] tableau) throws Exception
        if(MatrixSimplex.isNotSameLength(tableau))
            throw new Exception("Matrix not same length");
```

```
}

public static boolean isNoSolution(Ratio[][] tableau, int pivotColumn
    for (int i = 0; i < tableau.length; i++)
        if(tableau[i][pivotColumn].getDoubleValue() > 0)
            return false;
    return true;
}


public static boolean lastColumnHasNegative(Ratio[][] tableau){
    for(Ratio i : tableau[tableau.length - 1])
        if(i.getDoubleValue() < 0) return true;
    return false;
}
public static int getPivotColumnIndex(Ratio[][] tableau){
    Ratio smallest = new Ratio(Integer.MAX_VALUE);
    int index = -1;
    Ratio[] tableauLastRow = tableau[tableau.length - 1];
    for(int i = 0; i < tableauLastRow.length; i++ ){
        Ratio v = tableauLastRow[i];
        if(v.getDoubleValue() < smallest.getDoubleValue()){
            smallest = v;
            index = i;
        }
    }
    return index;
}
public static int getPivotRowIndex(Ratio[][] tableau, int pivotColumn
    Ratio smallest = new Ratio(Integer.MAX_VALUE);
    int lastRowIndex = tableau[0].length - 1, index = -1;
    for(int i = 0; i < tableau.length - 1; i++){
        if(tableau[i][pivotColumnIndex].getDoubleValue() == 0)
            continue;
        Ratio v = tableau[i][lastRowIndex].divide(tableau[i][pivotColu
        if(v.getDoubleValue() < smallest.getDoubleValue()){
            smallest = v;
            index = i;
        }
    }
    return index;
}
public static Ratio[][] makePivotOne(Ratio[][] tableau, int pivotColu
    Ratio v = tableau[pivotRowIndex][pivotColumnIndex];
```

```
        if(v.getDoubleValue() == 1)
            return tableau;
        for(int i = 0; i<tableau[pivotRowIndex].length; i++)
            tableau[pivotRowIndex][i]=tableau[pivotRowIndex][i].divide(v)
        return tableau;
    }
    public static Ratio[][] makePivotRowZero(Ratio[][] tableau, int pivotC
        Ratio v = tableau[pivotRowIndex][pivotColumnIndex];
        if(v.getDoubleValue() == 0)
            throw new IllegalArgumentException("Pivot value can't be zero
        for(int i = 0; i<tableau.length; i++){
            if(i == pivotRowIndex)
                continue;
            Ratio w = new Ratio(-1).divide(v).multiply(tableau[i][pivotCo
            for(int j = 0; j<tableau[i].length; j++)
                tableau[i][j]=tableau[i][j].add(w.multiply(tableau[pivotR
        }
        return tableau;
    }
}
```

0.2.2 Ratio.java

```
package utilities;

public class Ratio {
    int numerator, denominator;

    public Ratio(int numerator, int denominator) {
        basicInit(numerator,denominator);
    }public Ratio(int numerator) {
        this.numerator = numerator;
        this.denominator = 1;
    }public Ratio(double value){
        int count = 0;
        while ((int)value != value) {
            value *= 10;
            count++;
        }
        int n = (int) value, d = (int) Math.pow(10,count);
        int v = gcd(n,d);
        this.numerator = n/v;
        this.denominator = d/v;
    }public Ratio(String fraction){
        if(!fraction.contains("/")){
            Ratio r = parseDecimal(fraction);
            this.numerator = r.numerator;
            this.denominator = r.denominator;
            return;
        }
        String[] split = fraction.split("/");
        int v1 = Integer.parseInt(split[0]), v2 = Integer.parseInt(split[
        basicInit(v1,v2);
    }

    private void basicInit(int numerator, int denominator) {
        if(denominator == 0)
            throw new IllegalArgumentException("Denominator must not be 0
        int v = gcd(numerator, denominator);
        this.numerator = numerator/v;
        this.denominator = denominator/v;
    }

    private Ratio parseDecimal(String fraction) {
        double d = Double.parseDouble(fraction);
```

```
        return new Ratio(d);
    }

    public double getDoubleValue(){
        return numerator/new Double(denominator);
    }

    public int getNumerator() {
        return numerator;
    }

    public void setNumerator(int numerator) {
        this.numerator = numerator;
    }

    public int getDenominator() {
        return denominator;
    }

    public void setDenominator(int denominator) {
        this.denominator = denominator;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Ratio ratio = (Ratio) o;

        if (numerator != ratio.numerator) return false;
        return denominator == ratio.denominator;
    }

    @Override
    public int hashCode() {
        int result = numerator;
        result = 31 * result + denominator;
        return result;
    }

    @Override
    public String toString() {
        if(denominator < 0){
```

```
            denominator*=-1;
            numerator*=-1;
        }
        if(denominator == 1)
            return numerator+"";
        return numerator+"/"+denominator;
    }

    public Ratio add(Ratio r){
        int v = lcm(denominator,r.denominator);
        int v1 = v/denominator, v2 = v/r.denominator;
        return new Ratio(v1*numerator+v2*r.numerator,v);
    }public Ratio plus(Ratio r){
        return add(r);
    }public Ratio add(int r){
        return this.add(new Ratio(r));
    }public Ratio add(double r){
        return this.add(new Ratio(r));
    }public Ratio add(String r){
        return this.add(new Ratio(r));
    }

    public Ratio subtract(Ratio r){
        int v = lcm(denominator,r.denominator);
        int v1 = v/denominator, v2 = v/r.denominator;
        return new Ratio(v1*numerator-v2*r.numerator,v);
    }public Ratio minus(Ratio r){
        return subtract(r);
    }public Ratio subtract(int r){
        return this.subtract(new Ratio(r));
    }public Ratio subtract(double r){
        return this.subtract(new Ratio(r));
    }public Ratio subtract(String r){
        return this.subtract(new Ratio(r));
    }

    public Ratio multiply(Ratio r){
        return new Ratio(r.numerator*numerator,r.denominator*denominator)
    }public Ratio multiply(int r){
        return new Ratio(r*numerator,denominator);
    }public Ratio multiply(double r){
        return this.multiply(new Ratio(r));
    }public Ratio multiply(String r){
        return this.multiply(new Ratio(r));
```

```
    }

    public Ratio divide(Ratio r){
        return new Ratio(numerator*r.denominator,denominator*r.numerator)
    }public Ratio divide(int r){
        return this.divide(new Ratio(r));
    }public Ratio divide(double r){
        return this.divide(new Ratio(r));
    }public Ratio divide(String r){
        return this.divide(new Ratio(r));
    }

    private int gcd(int a, int b){
        while(a!=0 && b!=0){ // until either one of them is 0
            int c = b;
            b = a%b;
            a = c;
        }
        return a+b; // either one is 0, so return the non-zero value
    }private int lcm(int a, int b){
        return a * (b / gcd(a, b));
    }
}
```