데이터 구조론 팀프로젝트

LP SOLVER FOR FINANCIAL ENGINEERING

2019048704 김정헌

2018015414 박진혁

2021044257 안도현

<목 차>

프로젝트 목적 및 개요

프로젝트 개요

Financial manger가 아래의 상황에 사용할 수 있는 프로그램을 구현한다.

상황: 80000개의 각 문제 상황에 맞는 optimal solution, optimal value, 걸린 시간을 구한 뒤(변수는 최대 3000개), 각 문제를 푸는 시간, 풀렸다면 얻을 수 있는 정보들 그리고 추가적으로 제공하면 좋을 정보들을 제공한다.

프로젝트 목적

각 문제상황은 선정된 투자종목에 한하여 특정 수익률 이상을 가지면서, 손실을 최소화 할 수 있는 투자조합과 그 비율을 결정하는 문제이다. 즉, 아래와 같은 lp문제를 해결하는 것이다.

Optimization Problem 1

minimizing Value-at-Risk

$$\min_x var\_risk_\alpha(L(x)) \qquad \text{(CS.1)}$$

subject to

budget constraint

$$\sum_{i=1}^{I} x_i = 1 \qquad \text{(CS.2)}$$

portfolio return constraint

$$\sum_{i=1}^{I} r_i x_i \geq const \qquad \text{(CS.3)}$$

bounds on decision variables

$$0 \leq x_i \leq 2x_i^0,\ i = 1, \dots, I \qquad \text{(CS.4)}$$

문제 상황에서 이는 최대 변수 3000개를 가지는 LP 문제를 총 80000번 해결한 후, 이를 효율적인 자료구조로 저장한 후, 사용자가 원하는 조건에 따른 답을 LP문제를 통해 제공한다.

프로젝트 참가자 및 역할

프로젝트는 크게 3가지로 구성 되어있다.

LP 문제를 해결하는 부분, 해결된 LP solution의 크기가 매우 큼으로 이를 자료구조에 효율적으로 저장하는 부분, 사용자가 원하는 정보들을 제공하는 부분이다.

이에 맞추어 역할을 아래와 같이 구분하였다.

이름	역할
안도현	정보 제공 메소드 구현(JAVA)
박진혁	데이터구조 설계 및 구현(JAVA)
김정현	LP 풀이(python) 및 정보 제공 메소드 구현(JAVA)

LP solver(python) 구현 및 상세설명

자료 저장

```
left_coefficient = np.concatenate((np.ones((1,3000)),-np.ones((1,3000)),-cp_return.T,-np.eye(3000, dtype=int),np.eye(3000, dtype=int)),axis=0)
#left_coefficient라는 배열은 좌변 계수를 의미. 이 매트릭스 안에는 1x3000행렬에 전부 1, 1x3000행렬에 전부 -1, cp.return의 트랜스포즈한 음수값, 30

left_coefficient=matrix(left_coefficient) #LP Solve를 위해 행렬로 변환.

#create full RHS
constant=-0.0963007951203275
right_constraint = matrix([1,-1,constant,matrix(np.zeros((3000,1))),matrix(point_2x0)])

#위에서부터 1, -1, constant, 3000*1 전부 0, point_2x0 값이 들어가 있음.


#-----------------------------------------------------제약 조건 생성 완료-----------------------------------------------------#
```

파이썬으로 사용할 수 있는 cvxopt 패키지의 lp들을 solve해주는 solvers함수를 이용하기 위해 본 문제의 제약식들을 모두 포함하는 행렬을 생성한다. 이 행렬의 column 부분에는 본 문제의 변수인 x 3000개가, row 부분에는 constraint들이 배치된다. Solvers 함수의 특징은 1) 구하고자 하는 문제가 minimization이어야 하고, 2) 제약식의 부등호 방향이 "<" 이어야 한다는 것이다. 이 특징에 맞추어주기 위해 cp.return을 전치해주고 음수값을 씌우는 등의 조작이 필요했다.

CVXOPT를 이용해 결과값 출력 및 저장

```
f_result = open('result.txt',mode = 'w',encoding='utf-8')

for i in range(40000,50000): # 8만개 해결

    s_right_c=right_constraint
    s_cost=Cost[vars[i],:] #vars[i]엔 i번째 lp에 사용한 x들이 적혀있음.
    s_left_c=left_coefficient[:,vars[i]]


    start=time()
    sol=solvers.lp(s_cost,s_left_c,s_right_c, solver = 'glpk') #glpk를 사용해서 lp푸는 시간 최적화. glpk 사용 x = 하나당 최소 5초, glpk 사용 o = 하나당 최대 0.2초
    end=time() #끝난 시간
    timecost=end-start #걸린 시간.
```

문제를 풀기 전 solvers 함수가 요구하는 식들을 생성해주어야 한다. s_right_c행렬은 자료 저장 단계에서 생성한 right_constratint행렬을 그대로 가져오면 된다. right_constraint는 열이 한 개인 matrix이기 때문이다. s_cost행렬은 i번째 문제에서 사용한 x들이 적혀 있는 vars[i]를 이용하여 Cost[vars[i],:]을 저장한다. s_left_c도 마찬가지로 vars[i]를 이용하여 좌변계수 left_coefficent[:,vars[i]]를 저장한다. 그리고 3가지 행렬을 이용하여 LP들을 풀어냈다. 이후 푼 값을 sol에 저장하고, time 함수를 이용해 풀린 시간을 계산할 수 있었다.

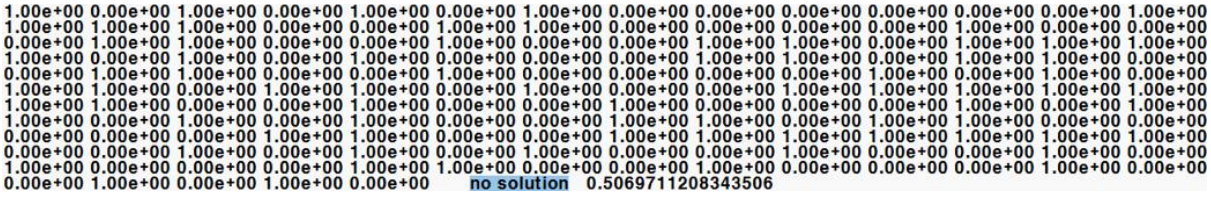
```
    if sol['x'] is None :
      Sol_result = matrix(np.ones((3000, 1)))#사용하지 않은 x들값에 1을 대입. 0이 아니라 1을 대입한 이유는 사용된 x중에서도 최적해가 0인 x들이 있기 때문에 구분하기 위함.
      Sol_result[vars[i],:] = 0
      Solution = str(Sol_result)
      Solution=Solution.replace("₩n","")
      Solution=Solution.replace("[","")
      Solution=Solution.replace("]","")
      f_result.write(Solution+'₩t')
      f_result.write("no solution"+'₩t') #infeasible 등의 이유로 해를 구하지 못하였을 때는 그 줄에 no solution만 적음
      f_result.write(str(timecost)+'₩n')

    else :
      Sol_result = matrix(np.ones((3000, 1))) #사용하지 않은 x들값에 1을 대입. 0이 아니라 1을 대입한 이유는 사용된 x중에서도 최적해가 0인 x들이 있기 때문에 구분하기 위함.
      Sol_result[vars[i],:]=sol['x'] #sol['x']는 사용한 x들의 optimal값만 들어있어서 사용하지 않은 x들 값을 반영하지 못함.
      Solution = str(Sol_result)
      Solution=Solution.replace("₩n","")
      Solution=Solution.replace("[","")
      Solution=Solution.replace("]","") #편의를 위해 한 줄에 LP 1개의 solution이 들어가게 편집함.
      opt_value=sol['x'].T*s_cost


      opt_value=str(opt_value)
      opt_value=opt_value.replace("[","")
      opt_value=opt_value.replace("]","")
      opt_value=opt_value.replace("₩n","")
      f_result.write(Solution+"₩t")
      f_result.write(opt_value+"₩t")
      f_result.write(str(timecost)+'₩n')#optimal solution, optimal value, 걸린 시간을 순서대로 적음

    print(i+1,'번째 LP Solved')

f_result.close()
```

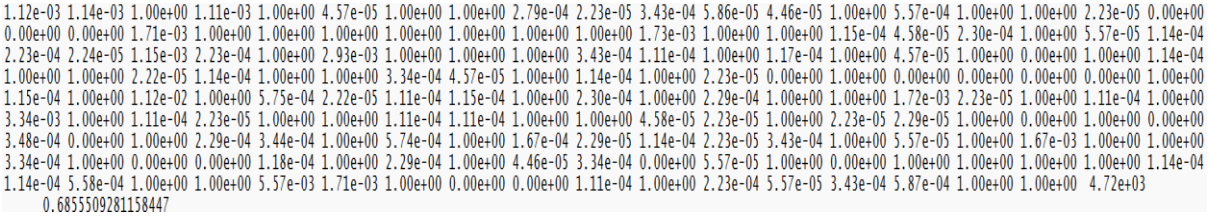
그런 뒤 infeasible 등의 이유로 해를 구하지 못하고 LP를 풀지 못하였을 때와 LP를 풀었을 때로 나누었다.

먼저 사용되지 않은 변수 x들의 값에는 1을 대입하였다. 고객에게 서비스를 제공할 때 사용한 변수와 사용하지 않은 변수를 구분할 수 있어야 하는데, 사용된 결정변수 중 최적해가 0인 경우도 있기 때문에 이와 구분하기 쉽게 하기 위해서다. 사용된 변수 x들의 합이 1인 상황에서 어떠한 경우에도 특정 변수의 값이 1일 수는 없기 때문에 1을 대입하는 방식을 선택했다. 후에 자바로 vars.txt를 따로 읽으면서 사용된 변수들을 확인하는 것보다 이 방식의 time cost가 더 적을 것이라 확신했다.

결과를 분석하는 중 sol['x']이 x값 한 개를 출력할 때마다 [x값] 줄바꿈을 하고 있음을 알아냈다. 이 역시 자바로 작업할 때의 편의성을 위해 제거하는 과정을 거쳤다. Optimal value는 s_cost와 sol['x']의 전치행렬을 곱함으로써 쉽게 구할 수 있었다.

그런 다음 LP가 풀리지 않은 문제들은 결정변수 x의 값(1 또는 0), 'no solution', 걸린 시간을 순차대로 적었다. 반면에 LP가 풀렸다면 optimal solution, optimal value, 걸린시간을 순차대로 적었다. 밑의 사진들은 각각 풀리지 않았을 때와 풀렸을 때의 결과값들이다.





자료구조 및 기능 구현 및 상세설명(JAVA)

구현 시 고려해야할 점

python으로 푼 LP solution의 형식

python으로 푼 LP solution은 아래와 같은 형식을 가지고 있다.

```
1.11e-04 1.00e+00 2.23e-04 5.57e-05 3.43e-04 5.87e-04 1.00e+00 1.00e+00	4.72e+03	0.6855509281158447
1.11e-04 5.58e-05 1.00e+00 5.57e-05 1.00e+00 1.00e+00 1.00e+00 1.00e+00	4.52e+03	0.7653014659881592
1.11e-04 1.00e+00 2.23e-04 1.00e+00 3.43e-04 5.87e-04 1.14e-04 0.00e+00	4.60e+03	0.778092622756958
1.11e-04 1.00e+00 2.23e-04 5.57e-05 1.00e+00 5.87e-04 1.14e-04 0.00e+00	4.95e+03	0.7070109844207764
1.00e+00 1.00e+00 1.00e+00 5.57e-05 3.43e-04 5.87e-04 1.00e+00 0.00e+00	5.15e+03	0.7422962188720703
1.11e-04 5.58e-05 2.23e-04 1.00e+00 1.00e+00 1.00e+00 1.14e-04 1.00e+00	4.81e+03	0.7263836860656738
```

한 행이 한 LP의 계산 결과를 나타낸다.

이 행은 LP에 사용되는 변수인 x1~x3000에 해당하는 LP solution을 (₩s로 구분하여) 열로 가지고 이후 tab(₩t)로 구분하여 LP solution value와 문제가 풀리는데 걸린 시간을 표시한다.

이후 다음 줄(₩n)에 다른 LP의 결과를 같은 표현방법으로 나타낸다.

즉, 전체 행렬은 80000*3002의 크기를 가진다.

각 해당하는 변수의 표현에 관해서는 아래와 같다.

1. LP solution variable

   모든 변수는 지수표현법을 사용 소수점을 포함 8자리를 사용한다.

   모든 변수가 사용되는 것이 아니기에 사용되지 않은 변수의 경우 1로 표현하였다. 이는 해결하고자 하는 LP문제의 특성 상 한 변수의 값이 1이 나올 수 없기 때문이다(첫번째 제약식).

   사용된 변수의 경우 값이 있다면 그 값으로, 값이 없는 경우(문제에 사용되었지만, Solution에 영향이 없는 값)는 0으로 표현하였다. 이는 특정 변수가 어떤 LP에 사용되었는지 차후에 사용자가 검색을 원할 경우 그 데이터를 제공하기 위해서이다.

2. Solution value

   풀린 경우는 지수표현법을 사용, 8자리로 표현한다.

   풀리지 않은 경우 no solution으로 표현한다.

3. Time

   문제가 풀리는 데 걸린 시간을 sec 단위로 표현한다.

자료구조 구현 및 상세설명

행렬을 그대로 이용 시, 즉, 80000*3000의 matrix를 그대로 표현할 경우 메모리를 매우 많이 사용할 수 밖에 없다. 또한 실제로 각 LP문제에서 사용하지 않은 변수들도 있기 때문에 필요한 데이터만 효율적으로 표현하기 위해 Sparse matrix로 자료구조를 구현하였다.

-MyIndex Class

```
class MyIndex {
    private int x=0; // nr of rows
    private int y=0; // nr of columns
    private int hashvalue = 0;

    public MyIndex (final int x, final int y)
    {
        this.x=x;
        this.y=y;
        hashvalue=((x+"")+(y+"")).hashCode();
    }

    public boolean equals (final Object obj)
    {
        if (obj instanceof MyIndex)
        {
            MyIndex index = (MyIndex) obj;
            return ((x==index.x) && (y==index.y));
        }
        else
            return false;
    }
    public int hashCode()
    {
        return hashvalue;
    }

}
```

MyIndex는 row랑 column을 받아 이를 hash로 변환해주는 코드로, 아래의 Sparse Matrix code에서 행과 열을 보다 효율적으로 저장하기 위해서 사용하였다.

-Sparse Matrix Class

```
public class SparseMatrix {

    private int rows;
    private int columns;
    private HashMap<MyIndex,Double> values;

    private boolean validIndex (final int row, final int column)
    {
        return (row>=0 && row<rows && column>=0 && column<columns);
    }


    public SparseMatrix(int rows, int columns)  {
        this.rows=rows;
        this.columns=columns;
        this.values=new HashMap<MyIndex,Double>();
    }


    public int getNumberOfRows() {
        return rows;
    }

        public int getNumberofActiveElements() {
                return values.size();
        }


    public int getNumberOfColumns() {
        return columns;
    }
```

SparseMatrix는 위에서 정의한 MyIndex를 이용해 value를 저장한다.

각 메소드의 기능과 설명은 아래와 같다.

1. private boolean validIndex

   처음 정의한 전체 행열의 크기를 벗어나는지 확인한다

   단 행열은 0부터 시작된다

2. 생성자

   Row, column을 정수타입으로 받아 해당 크기의 행렬을 생성한다.

3. public int getNumberOfRows()

   Row, 행의 개수 정수형으로 반환한다.

4. public int getNumberofActiveElements()

   value의 수를 반환한다

5. public int getNumberOfColumns()

   열의 개수를 정수형으로 반환환다.

6. public double getElement(int row, int column)

   특정 행,열에 value를 반환한다. 값이 존재하는 경우 해당 값을 double로 반환한다. 없는 경우 -1로 반환한다(0이 의미 있는 data이기에 -1을 반환)

7. public void setElement(int row, int column, double value)

   특정 행,열에 value를 넣는다. 이 때 값이 1인 경우, 우리의 case에서는 필요하지 않는 값이다. 따라서 value를 집어넣지 않는다. 이 경우 6로직에 의해서 -1이 반환된다.

자료구조에 LP solution을 읽고 저장하는 기능 구현 및 상세설명

- setvalue(): 값을 sparsematrix에 저장하는 부분

```
public static void setSvalue(int row, int column, Double dValue){
        // 1인 경우는 결과 txt파일에서 사용되지 않은 변수를 의미 그럼으로 저장할 필요없음, 1이 아닌경우만 저장
        if(dValue != 1) {
                //3001번째부분인 결과값을 반환 0에서 시작
                if(column==3000){
                        mat.setElement(row, 0, dValue);
                }
                else if(column==3001){
                        mat.setElement(row, 1, dValue);
                }
                else{
                        mat.setElement(row, column+2, dValue);
                }
        }
} //end of setSvalue method
```

행과 열을 받아서 저장하는 method. setElement와 동일하지만, Sparse matrix 저장하는 방식을 조금 변경하였다.

기존에는 각 행이 solution variable, solution value, solution time 순으로 이루어졌지만,

우리는 SparseMatrix를 통해서 value가 1인, 의미가 없는 값을 모두 제거하고 저장하기에, 각 행의 통일성 및 검색효율성을 높이기 위해 행의 저장 순서를 solution value, time, solution variable 순으로 변경하였다. 따라서 이런 변경된 형태에 맞게 열의 3000번째인 경우와 3001번째인 경우가 각각 value와 time임으로 해당하는 부분을 0번째 열과 1번째 열에 넣는다.

-getMatrix(): python으로 생성된 결과 txt를 읽어 값을 저장하는 부분

```
public static SparseMatrix getMatrix() {
                // Get file path using relative path
//              File directory = new File("./");
//              System.out.println(directory.getAbsolutePath());
                String filePath = "./data/result30000~40000.txt";
                File file = new File(filePath);
                Double dValue = 0.00;
                int rowCount = 0; //세로줄 (0,2)에서 0
                int columnCount = 0; //가로줄 (0,2)에서 2
                //read file using stream
                try(Scanner sc = new Scanner(new FileInputStream(file))){

                        while(sc.hasNext()){
//                              //read 1line by 1line
                                while(sc.hasNext()&&columnCount<3002){
                                        String sValue = sc.next();
                                        try{
                                                //sc.next는 기본적으로 string을 반환하기에 다시 double 타입
                                                dValue = Double.parseDouble(sValue);
                                                setSvalue(rowCount, columnCount, dValue);
                                        }//end of try
                                        catch (Exception e){
                                            //에러시 수행
//                                              e.printStackTrace(); //오류 출력
//                                               System.out.println("에러 있음" + sValue);
//                                           System.out.println("에러 있음:" + columnCount);
                                                // 0에 한없이 가까운 변수값은 0으로 반환 -> lp 문제에 사용됨
                                                //no Solution 경우에도 0 반환
                                                if(sValue.equals("no")){
                                                        dValue = 0.00;
```

```
                                        setSvalue(rowCount, columnCount, dValue);
                                    }
                                    else if(sValue.equals("solution")){
                                            columnCount --;
                                    }
                                    //format 길이 8 1.00+e04
                                    //1.14e-04-9.62e-20 인 경우의 error 처리위한 로직 뒷부분은
                                    else{
                                            String sValue1 = sValue.substring(0, 8);
                                            String sValue2 = sValue.substring(8, 17);
//                                          System.out.println("에러 있음" + sValue1);
//                                      System.out.println("에러 있음:" + sValue2);
                                            Double dValue1 = Double.parseDouble(sValue1);
                                            Double dValue2 = 0.00;
                                            setSvalue(rowCount, columnCount, dValue1);
                                            columnCount ++;
                                            setSvalue(rowCount, columnCount, dValue2);
                                    } //end of else statement
                                }//end of catch statement
                                columnCount ++; //3001까지 순회함
                            } //end of while
                            System.out.println("column: "+ columnCount);
                            columnCount = 0;
                            System.out.println("row는 " + rowCount);
                            rowCount ++;
                        } //end of while
                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return mat;
```

Python을 통해 얻은 결과 txt파일은 총 3002*80000 크기의 행렬과 같다. 또한 각 변수의 구분자가 공백 혹은 tab으로 되어있기 때문에 이를 모두 scanner를 이용해 읽는다.

따라서 기본적인 로직은 한 줄을 읽으면서 columnCount을 0부터 3001까지 증가(3002회)할 때까지 변수들을 읽고, 그 이상이라면, 다음줄로 판단하여 rowCount를 증가, 다음줄의 값을 넣는다.

세부적으로는 아래 사항을 고려하였다.

우선 scanner를 이용해 값을 불러오는 경우 모두 String type이다. 따라서 이를 Double.parseDouble()을 통해 double값으로 바꿔준다.

두번째 우리의 txt파일 안에는 1.00+e04와 같이 실수형 자료들도 있지만, 해가 없는 경우를 의미하는 no solution이나 1.14e-04-9.62e-20와 같이 나타나는 음수형 표현도 존재하였다. 해당 경우는 Double.parseDouble()에서 에러가 발생한다. 따라서 이를 try-catch처리하여 catch부분에서 에러를 다루었다.

No solution의 경우 이는 실제로 하나의 값이나 공백으로 나누어져 있어 scanner는 2개의 값으로 인식하는 문제가 있었기에, 우선 공백 앞부분이 no가 들어올 시 해당 값을 0으로 변경하여 값을 넣는다. 이후 solution이 오면 열의 개수를 세는 coulmnCount를 하나 줄여주어 문제를 해결했다.

1.14e-04-9.62e-20의 경우 실제로 2가지 값이지만, -표현으로 인해 공백이 인식되지 않아 한 값으로 인식되는 경우였다. 이 경우 지수표현법으로 숫자가 모두 8자리(.과 기호 포함)로 표현되었기에 subString을 이용해 2 값으로 분리하였다. 이후 앞의 값은 동일하게 실수형으로 변경후 값을 넣고, 뒤의 값, 음수부분은 변수로는 사용되었지만, 의미가 없기에 0으로 처리한 후 값을 넣었다.

사용자 인터페이스 및 함수 구현 및 상세설명(JAVA)

서비스 시작

```
public class InvestService {

        static SparseMatrix mat = ReadResult.GetMatrix();// InvestService의 matrix를 ReadResult로 저장한다.
        //public static이라 import나 객체 생성 없이 바로 사용 가능.

        //객체를 생성하면서 바로 서비스 시작
        public InvestService() {
                Scanner sc = new Scanner(System.in);

                while (true) {
                        System.out.println();
                        System.out.println("무엇에 대해 알고 싶습니까?");
                        System.out.println("(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료");
                        int menu = sc.nextInt();
                        if (menu == 0)
                                break;
```

사용자가 원하는 정보를 제공하는 서비스를 시작하기 위해서 많은 방식을 고려할 수 있었다. 본조에서는 InvestService의 생성자에 서비스를 구현하여, main 메소드에서 InvestService 타입의 객체를 하나 생성하기만 하면 바로 서비스가 시작되도록 구현하였다. 서비스를 시작하면 사용자는 다음과 같은 선택지를 가지게 된다. (특정 문제에 관한 정보, 특정 변수에 관한 정보, 서비스 종료) 사용자는 0을 입력하여 종료하기 전까지 계속해서 서비스를 사용할 수 있다. 이후 사용되는 메소드들은 Switch-case 조건문을 사용했다. If 조건문을 사용하는 것보다 코드의 가독성이 좋아지기 때문이다. 이후 Switch문을 다시 사용할 때에는 이중 Switch 문은 오류를 발생시키기 쉽기 때문에 ProblemService, VariableService 등의 메소드를 따로 만들어서 가독성이 떨어지는 것을 방지했다.

특정 문제

```
case 1:
        System.out.println("몇 번 문제(1~80000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.");
        var = sc.nextInt();
        System.out.println(var + "번 문제의 무엇에 대해 알고 싶습니까?");
        System.out.println("(1)문제의 시간 (2)문제의 풀렸는지 유무 (3)문제의 목적식값들 (4)문제의 최적해 값들 (5)전체 문제의 평균 걸린 시간");
        x = sc.nextInt();
        ProblemService(x,var);
        break;
```

1. 고객이 특정 문제에 대해서를 선택한다면 1부터 80000개의 문제 중 몇 번째 문제를 알고 싶어하는지 묻는다. 그리고 그 값을 변수 var에 저장한 뒤 그 문제에 대한 어떤 정보를 알고 싶은지 추가적으로 묻는다. 선택지는 다음과 같다. (그 문제의 시간, 그 문제가 풀렸는지 유무, 그 문제의 목적식값, 그 문재의 최적해 값들, 전체 문제의 평균 걸린 시간) 이후 이 선택지들은 변수 x에 저장되고, ProblemService를 실행한 뒤 Switch문을 빠져나간다.

선택 시 ProblemService라는 메소드가 실행된다. 첫번째 선택지를 선택했다면 그 문제의 걸린 시간을 반환한다. 반환하기 위해서는 데이터가 저장되 있는 행렬의 var에 해당하는 행과 시간이 적혀있는 두번째 열을 가져와야하기 때문에 mat.getElement(var,1)를 실행하고 그 문제가 풀리는데 걸린 시간을 출력한다. 두번째 선택지를 선택했다면 그 문제가 풀렸는지 유무에 대해 알려준다. 먼저 그 문제가 풀렸는지 알려주기 위해서는 목적식값을 알아야 하기 때문에 var에 해당하는 행과 첫번째 열을 가져온다. mat.getElement(var,0)를 실행한 뒤 그 목적식값이 0이라면 풀리지 않았다고 출력하고 0이 아니라면 풀렸다고 출력한다. 세번째 선택지를 선택했다면 목적식값을 반환한다. 그러기 위해서는 위에 언급한 것 처럼 mat.getElement(var,0)을 실행해서 출력한다. 4번째 선택지를 선택했다면 최적해 값들이 반환된다. 그러기 위해서는 행렬의 3번째 열부터 3002번째 열까지 다 출력해야 하기 때문에 while 문을 이용하였다. 마지막 선택지를 선택했다면 전체 문제의 평균 걸린 시간을 반환한다. 그러기 위해서는 80000개의 문제의 걸린 시간을 모두 더한 뒤 80000으로 나눈 값을 출력한다. 메소드는 아래와 같다.

```
public static void ProblemService(int x, int var) {
        var-=1; //행렬은 첫 칸이 0이기 때문에 조정해줘야함
        switch (x) {
        case 1:
                System.out.println(mat.getElement(var, 1));// 걸린 시간 반환
                break;
        case 2:
                System.out.println(mat.getElement(var, 0)==0 ? "풀리지 않았습니다" : "풀렸습니다");// 풀렸는지 유무 반환
                break;
        case 3:
                System.out.println(mat.getElement(var, 0));// 문제의 목적식값 반환
                break;
        case 4:
                int y = 2;// 2부터 3001까지 출력되게
                while (y <= 3001) {
                        System.out.println(mat.getElement(var, y));// 문제의 최적해 값들
                        y++;
                }
                break;
        case 5:
                int i = 0;
                double sum = 0;
                while (i < 80000)//resultsample로 돌릴 때에는 100을 사용
                        {
                        sum = sum + mat.getElement(i, 1);
                        i++;
                }
                System.out.println(sum / 80000);// 평균 걸린 시간.
                break;
```

특정 변수

```
case 2:
        System.out.println("몇 번 변수(1~3000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.");
        var = sc.nextInt();
        if (var == 0) {
                System.out.println("목적값이 특정 값보다 작은지, 큰지를 알려주는 서비스를 제공하고 있습니다.");
                inequality(0);
        } else {
                System.out.println(var + "번 변수 의 무엇에 대해 알고 싶습니까?");
                System.out.println("(1)풀린 문제들 번호 (2)풀린 문제들의 목적식 값들 (3)풀린 문제들 중 이 변수의 최적값 (4)특정 조건");
                x = sc.nextInt();
                VariableService(x,var);
        }
        break;
```

고객이 특정 변수에 관한 정보에 대해 물어본다면 1부터 3000개의 변수 중 몇 번째 변수에 대해 알고 싶은 지 묻는다. 그리고 그 값을 변수var에 저장한 뒤 그 변수에 대한 어떤 정보를 알고 싶은 지 추가적으로 묻는다. 선택지는 다음과 같다. (풀린 문제들의 번호, 풀린 문제들의 목적식 값들, 풀린 문제들 중 이 변수의 최적값, 특정 조건) 작동하는 방식은 특정 문제에서와 유사하다.

inequality

만약 목적식 값에 대해 알고 싶어 0을 입력했다면 목적값이 특정 값보다 큰지 알려주기 위해서 inequality()가 실행된다. 원하는 특정 값을 입력하고 그 값보다 큰 목적값들은 알고 싶으면 >을 입력하면 된다. 그 값보다 작은 목적값들을 알고 싶으면 <을 입력하면 된다. 만약 <를 입력했다면 목적값이 존재하면서 원하는 변수의 최적해가 특정값보다 작을 때 result3에 몇 번째 문제인지 그리고 그 최적해값을 추가한다. 이 과정을 행의 수만큼 for문을 돌린다. >를 입력했다면 반대로 특정값보다 클 때 result3에 추가한다. 그런 뒤에 조건에 맞는 변수의 최적해들을 출력한다.

```
private static void inequality(int var) {
        Scanner sc = new Scanner(System.in);
        if(var==0) {
                var-=1; //뒤에 var + 1 작업을 해주기 때문에 목적값을 구하려면 var를 -1로 설정해주어야함.
        }

        System.out.println("특정 값을 입력하세요.");
        double point = sc.nextDouble();
        String sign;
        String result3 = "";
        while (true) {
                System.out.println("이 값 이상인 값들을 원하면 >, 이하인 값들을 원하면 <를 입력하세요");
                sign = sc.next();
                if (sign.equals("<")) {
                        for (int i = 0; i < mat.getNumberOfRows(); i++) {
                                if (mat.getElement(i, var + 1) <= point) {
                                        if (mat.getElement(i, 0) != 0) {
                                                result3 += (i+1) + "번째 문제에서 조건에 맞는 이 변수의 최적해 : " + mat.getElement(i, var + 1) + "₩n";
                                        }
                                }
                        }
                        break;
```

```
            } else if (sign.equals(">")) {
                for (int i = 0; i < mat.getNumberOfRows(); i++) {
                    if (mat.getElement(i, var + 1) >= point) {
                        if (mat.getElement(i, 0) != 0) {
                            result3 += (i+1) + "번째 문제에서 조건에 맞는 이 변수의 최적해 : " + mat.getElement(i, var + 1) + "₩n";
                        }
                    }
                }
                break;
            } else {
                System.out.println("다시 입력하세요.");
            }
        }
        System.out.println("조건에 맞는 변수의 최적해들은 다음과 같습니다.");
        System.out.println(result3);
}// end inequality
```

변수에 대해 어떤 정보를 원하는지 선택했다면 VariableService라는 메소드가 실행된다. 첫번째 선택지를 선택했다면 골랐던 변수가 사용된 문제들 중 풀린 문제들을 반환한다. 그러기 위해서 목적값이 0이 아니면서 변수에 해당하는 특정값 또한 0 이상일때 result에 몇 번째 문제인지 추가된다. 이과정을 행의 수만큼 for문을 돌리고 result의 값을 출력한다. 두번째 선택지를 선택했다면 골랐던 변수가 사용된 문제들의 목적값들을 반환한다. case1 메소드에 추가적으로 목적식값만 추가하면 된다. 여기에 추가로 사용자가 이 상품에 투자했을 때 최대 수익률과 평균 수익률을 알고 싶어 할 것 같아 최대치와 평균치를 알려주는 메소드를 추가했다. (본 사진에는 미첨부) 세번째 선택지를 선택했다면 골랐던 변수가 사용된 문제들의 변수에 해당하는 최적식값들을 반환한다. 마찬가지로 case1 메소드에 추가적으로 최적식값만 추가하였다. 네번째 선택지를 선택했다면 선택한 변수가 특정 값보다 작은지 큰지 알려준다. 변수에 해당하는 최적값을 가지고 inequality(var)을 실행하였다.

```
public static void VariableService(int x, int var) {
        Scanner sc = new Scanner(System.in);
        switch (x) {
        case 1:
                String result = "";
                for (int i = 0; i < mat.getNumberOfRows(); i++) {
                        if (mat.getElement(i, var + 1) > 0) {
                                if (mat.getElement(i, 0) != 0) {
                                        result += (i+1) + " ";
                                }
                        }
                }
                System.out.print("이 변수가 사용된 문제들 중 ");
                System.out.println(result + " 번 째 문제들이 풀렸습니다.");
                // 변수가 사용된 문제들 중 풀린 문제들 반환
                break;
        case 2:
                String result1 = "";
                for (int i = 0; i < mat.getNumberOfRows(); i++) {
                        if (mat.getElement(i, var + 1) >= 0) {
                                if (mat.getElement(i, 0) != 0) {
                                        result1 += (i+1) + "번째 문제 최적값 : " + mat.getElement(i, 0) + "₩n";
                                }
                        }
                }
                System.out.println("풀린 문제들의 최적값들은 다음과 같습니다.");
                System.out.println(result1);
                // 풀린 문제들의 목적식 값들
                break;
```

```
case 3:
        String result2 = "";
        for (int i = 0; i < mat.getNumberOfRows(); i++) {
                if (mat.getElement(i, var + 1) >= 0) {
                        if (mat.getElement(i, 0) != 0) {
                                result2 += (i+1) + "번째 문제에서 이 변수의 최적해 : " + mat.getElement(i, var + 1) + "₩n";
                        }
                }
        }
        System.out.println("풀린 문제들 중 이 변수의 최적해는 다음과 같습니다.");
        System.out.println(result2);
        break;
        // 풀린 문제들 중 이 변수의 최적값
case 4:
        System.out.println("문제에서 이 변수가 특정 값보다 작은지, 큰지를 알려주는 서비스를 제공하고 있습니다.");
        inequality(var);
        break;
```

실행결과 및 결론

```
무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
1
몇 번 문제(1~80000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
1
1번 문제의 무엇에 대해 알고 싶습니까?
(1)문제의 시간 (2)문제의 풀렸는지 유무 (3)문제의 목적식값들 (4)문제의 최적해 값들 (5)전체 문제의 평균 걸린 시간
4
x1변수의 최적해 : 2.29E-4
x2변수의 최적해 : 5.71E-4
x3변수의 최적해 : 0.0
x4변수의 최적해 : 1.14E-4
x5변수의 최적해 : 0.0
x6변수의 최적해 : 1.11E-4
x7변수의 최적해 : 4.46E-5
x8변수의 최적해 : 4.57E-5
x9변수의 최적해 : 0.0114
x10변수의 최적해 : 1.14E-4
무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
1
몇 번 문제(1~80000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
1
1번 문제의 무엇에 대해 알고 싶습니까?
(1)문제의 시간 (2)문제의 풀렸는지 유무 (3)문제의 목적식값들 (4)문제의 최적해 값들 (5)전체 문제의 평균 걸린 시간
5
0.684904146194458

무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료


무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
1
몇 번 문제(1~80000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
1
1번 문제의 무엇에 대해 알고 싶습니까?
(1)문제의 시간 (2)문제의 풀렸는지 유무 (3)문제의 목적식값들 (4)문제의 최적해 값들 (5)전체 문제의 평균 걸린 시간
1
0.6855509281158447

무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
1
몇 번 문제(1~80000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
1
1번 문제의 무엇에 대해 알고 싶습니까?
(1)문제의 시간 (2)문제의 풀렸는지 유무 (3)문제의 목적식값들 (4)문제의 최적해 값들 (5)전체 문제의 평균 걸린 시간
2
풀렸습니다

무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
1
몇 번 문제(1~80000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
1
1번 문제의 무엇에 대해 알고 싶습니까?
(1)문제의 시간 (2)문제의 풀렸는지 유무 (3)문제의 목적식값들 (4)문제의 최적해 값들 (5)전체 문제의 평균 걸린 시간
3
4720.0
```

```
무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
2
몇 번 변수(1~3000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
1
1번 변수 의 무엇에 대해 알고 싶습니까?
(1)풀린 문제들 번호 (2)풀린 문제들의 목적식 값들 (3)풀린 문제들 중 이 변수의 최적값 (4)특정 조건
1
이 변수가 사용된 문제들 중
1
2
3
4
5
6
7
8
무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
2
몇 번 변수(1~3000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
1
1번 변수 의 무엇에 대해 알고 싶습니까?
(1)풀린 문제들 번호 (2)풀린 문제들의 목적식 값들 (3)풀린 문제들 중 이 변수의 최적값 (4)특정 조건
2
풀린 문제들의 최적값들은 다음과 같습니다.
1번째 문제 최적값 : 4720.0
2번째 문제 최적값 : 4520.0
3번째 문제 최적값 : 4600.0
4번째 문제 최적값 : 4950.0
5번째 문제 최적값 : 5150.0
6번째 문제 최적값 : 4810.0
7번째 문제 최적값 : 4900.0
8번째 문제 최적값 : 4560.0
9번째 문제 최적값 : 4870.0
10번째 문제 최적값 : 5170.0
11번째 문제 최적값 : 5480.0
12번째 문제 최적값 : 5540.0
13번째 문제 최적값 : 7330.0
14번째 문제 최적값 : 4700.0
15번째 문제 최적값 : 4970.0
16번째 문제 최적값 : 5010.0
```

```
무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
2
몇 번 변수(1~3000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
1
1번 변수 의 무엇에 대해 알고 싶습니까?
(1)풀린 문제들 번호 (2)풀린 문제들의 목적식 값들 (3)풀린 문제들 중 이 변수의 최적값 (4)특정 조건
3
풀린 문제들 중 이 변수의 최적해는 다음과 같습니다.
1번째 문제에서 이 변수의 최적해 : 2.29E-4
2번째 문제에서 이 변수의 최적해 : 2.29E-4
3번째 문제에서 이 변수의 최적해 : 2.29E-4
4번째 문제에서 이 변수의 최적해 : 2.29E-4
5번째 문제에서 이 변수의 최적해 : 2.29E-4
6번째 문제에서 이 변수의 최적해 : 2.29E-4
7번째 문제에서 이 변수의 최적해 : 2.29E-4
8번째 문제에서 이 변수의 최적해 : 2.29E-4
9번째 문제에서 이 변수의 최적해 : 2.29E-4
10번째 문제에서 이 변수의 최적해 : 2.29E-4
11번째 문제에서 이 변수의 최적해 : 2.29E-4
12번째 문제에서 이 변수의 최적해 : 2.29E-4
13번째 문제에서 이 변수의 최적해 : 2.29E-4
14번째 문제에서 이 변수의 최적해 : 2.29E-4
15번째 문제에서 이 변수의 최적해 : 2.29E-4
16번째 문제에서 이 변수의 최적해 : 2.29E-4
17번째 문제에서 이 변수의 최적해 : 2.29E-4

82번째 문제 최적값 : 4950.0
83번째 문제 최적값 : 4930.0
84번째 문제 최적값 : 4890.0
85번째 문제 최적값 : 4960.0
86번째 문제 최적값 : 5570.0
87번째 문제 최적값 : 5270.0
88번째 문제 최적값 : 4910.0
89번째 문제 최적값 : 4690.0
90번째 문제 최적값 : 4830.0
91번째 문제 최적값 : 4660.0
92번째 문제 최적값 : 4600.0
93번째 문제 최적값 : 4980.0
94번째 문제 최적값 : 4480.0
95번째 문제 최적값 : 4940.0
96번째 문제 최적값 : 5750.0
97번째 문제 최적값 : 5320.0
98번째 문제 최적값 : 5300.0
99번째 문제 최적값 : 4860.0
100번째 문제 최적값 : 5020.0

최적값 평균 : 4985.263157894737 최적값 최대치 : 7330.0
```

```
무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
2
몇 번 변수(1~3000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
1
1번 변수 의 무엇에 대해 알고 싶습니까?
(1)풀린 문제들 번호 (2)풀린 문제들의 목적식 값들 (3)풀린 문제들 중 이 변수의 최적값 (4)특정 조건
4
문제에서 이 변수가 특정 값보다 작은지, 큰지를 알려주는 서비스를 제공하고 있습니다.
특정 값을 입력하세요.
1
이 값 이상인 값들을 원하면 >, 이하인 값들을 원하면 <를 입력하세요
<
조건에 맞는 변수의 최적해들은 다음과 같습니다.
1번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
2번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
3번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
4번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
5번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
6번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
7번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
8번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
9번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
10번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
11번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
12번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
13번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
14번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
15번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
16번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
17번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
18번째 문제에서 조건에 맞는 이 변수의 최적해 : 2.29E-4
```

```
무엇에 대해 알고 싶습니까?
(1)특정 문제에 대해서 (2)특정 변수에 대해서 (0)서비스 종료
2
몇 번 변수(1~3000)에 대해서 알고 싶습니까? 목적식 값에 대해 알고 싶으시면 0을 입력하세요.
0
목적값이 특정 값보다 작은지, 큰지를 알려주는 서비스를 제공하고 있습니다.
특정 값을 입력하세요.
5000
이 값 이상인 값들을 원하면 >, 이하인 값들을 원하면 <를 입력하세요
>
조건에 맞는 변수의 최적해들은 다음과 같습니다.
5번째 문제에서 조건에 맞는 이 변수의 최적해 : 5150.0
10번째 문제에서 조건에 맞는 이 변수의 최적해 : 5170.0
11번째 문제에서 조건에 맞는 이 변수의 최적해 : 5480.0
12번째 문제에서 조건에 맞는 이 변수의 최적해 : 5540.0
13번째 문제에서 조건에 맞는 이 변수의 최적해 : 7330.0
16번째 문제에서 조건에 맞는 이 변수의 최적해 : 5010.0
17번째 문제에서 조건에 맞는 이 변수의 최적해 : 5160.0
18번째 문제에서 조건에 맞는 이 변수의 최적해 : 5090.0
21번째 문제에서 조건에 맞는 이 변수의 최적해 : 5390.0
23번째 문제에서 조건에 맞는 이 변수의 최적해 : 5200.0
25번째 문제에서 조건에 맞는 이 변수의 최적해 : 5060.0
28번째 문제에서 조건에 맞는 이 변수의 최적해 : 5600.0
29번째 문제에서 조건에 맞는 이 변수의 최적해 : 5100.0
32번째 문제에서 조건에 맞는 이 변수의 최적해 : 5150.0
```

본 프로젝트에서 원래 LP의 개수는 80000개지만, 실행결과의 Example을 뽑을 때는 편의를 위해 100개의 Sample Result로 프로그래밍을 진행하였다. 구현한 모든 메소드에서 알맞은 결과값을 출력함을 확인할 수 있었다.

구조적인 문제점은 다음과 같다. 첫째, 문제의 개수가 매우 많은 상황에서는 Console 상에서 가독성이 다소 떨어지는 문제가 발생하는데, 이는 GUI 혹은 다른 프로그램을 이용하여 개선해야 할 것이다. 둘째, SparseMatrix에서 변수의 개수(3002), 문제의 개수(80000) 등 변수들이 하드코딩 되어있다. 따라서 변수의 개수나 문제의 개수가 달라지는 상황에선 사용자가 수동으로 숫자를 알맞게 수정해주어야 하는 문제가 있다.

본 팀에서 구현한 메소드는 모두 time-complexity가 O(n)을 가진다. (이때 n은 문제의 개수) ReadResult의 GetMatrix를 포함하여 InvestService의 메소드들이 모두 데이터를 하나하나 방문하는 작업을 거치기 때문이다. 본 조에서 사용자에게 가장 높은 값을 가지는 5개의 문제 혹은 변수를 제공하는 등의 메소드를 구현했다면 무조건 정렬하는 과정이 필요하다. 그러나 가장 빠른 정렬 알고리즘을 사용한다 해도 O(nlogn)의 time-complexity (Quick Sort 등) 가 요구된다. 하지만 SparseMatrix에서 HashMap의 Key를 MyIndex로 구현한 본 조에서는 이런 알고리즘을 구현하기는 어려운 데다가, 제공한다 하더라도 time-cost가 커지게 되는 상황이 발생하여 이런 서비스는 제공하지 않는 방식을 택했다. 또한, 특정 변수가 사용되는 문제들에서 기대 최적값과 평균 최적값을 알려주는 메소드는 존재하기 때문에 이러한 점을 다소 보완했다. 추가로 제공하는 모든 메소드들의 time-complexity가 O(n)으로 비교적 빠른 속도 진행됨을 기대할 수 있다.