```java
/*
 * Ada Trabalho 1 - Game of beans
 *
 * @author Joana Soares Faria n55754
 * @author Goncalo Martins Lourenco n55780
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;


public class Main {

    public static void main(String[] args) throws IOException {

        GameOfBeans game;

        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

        //Tests, Piles, Depth
        int numTests = Integer.parseInt(in.readLine());

        for (int i = 0; i < numTests; i++) {
            //Get P and D
            String[] P_D = in.readLine().split(" ");
            int numPiles = Integer.parseInt(P_D[0]);
            int depth = Integer.parseInt(P_D[1]);

            String[] piles = in.readLine().split(" ");
            int[] aux = new int[numPiles];
            for (int j = 0; j < numPiles; j++) {
                aux[j] = Integer.parseInt(piles[j]);
            }
            String player = in.readLine();
            game = new GameOfBeans(depth, aux, player);
            int solution = game.bestJabaScore();
            System.out.println(solution);

        }

    }

}
```

```java
/**
 * Ada Trabalho 1 - Game of beans
 *
 * @author Joana Soares Faria n55754
 * @author Goncalo Martins Lourenco n55780
 */

public class GameOfBeans {

    private static final int JABA = 0;
    private static final int PIETON = 1;
    private static final int LEFT = 0;
    private static final int RIGHT = 1;
    private final int[][][] bestScores;
    private final int depth;
    private final int[] piles;
    private final String firstPlayer;


    public GameOfBeans(int depth, int[] piles, String firstPlayer) {
        this.depth = depth;
        this.piles = piles;
        this.firstPlayer = firstPlayer;
        bestScores = new int[piles.length + 1][piles.length + 1][2];
    }

    /**
     * Computes  Pieton's play.
     *
     * @param i left index(-1) of the pile
     * @param j right index(-1) of the pile
     * @return the number of piles removed from which side
     */
    private int[] pietonPlay(int i, int j) {
        int maxScore = Integer.MIN_VALUE;
        int[] answer = {0, 0};

        //left play
        for (int k = 1; k <= depth && (i + k <= j + 1); k++) { // k is the number of piles to remove
            int sum = 0;

            for (int c = 0; c < k; c++) { // c is a counter to sum all the piles to remove
                sum += piles[i + c - 1];
            }

            if (sum > maxScore) {
                maxScore = sum;
                answer = new int[]{k, 0};
            }
        }

```

```java
        //right play
        for (int k = 1; k <= depth && (j - k + 1 >= i); k++) {
            int sum = 0;

            for (int c = 0; c < k; c++) {
                sum += piles[j - c - 1];
            }

            if (sum > maxScore) {
                maxScore = sum;
                answer = new int[]{0, k};
            }
        }

        return answer;
    }


    /**
     * Computes the Jabas's score from removing k piles from the piles i to j
     * k>0 means we should augment i, k<0 means we should move j backwards
     *
     * @param k piles to remove
     * @param i left index(-1) of the pile
     * @param j right index(-1) of the pile
     * @return the score from removing k piles
     */
    private int score(int k, int i, int j) {
        int score = 0;

        if (k < 0) {
            for (int counter = 0; counter < -k; counter++) {
                score += piles[j - 1 - counter];
            }
        } else {
            for (int counter = 0; counter < k; counter++) {
                score += piles[i - 1 + counter];
            }
        }

        return score;
    }

    /**
     * Computes the max score for Jaba, solves the problem
     *
     * @return the maximum Jaba score
     */
    public int bestJabaScore() {
        int player = firstPlayer.equalsIgnoreCase("jaba") ? JABA : PIETON;
```

```
102         //base cases = only one pile left (i=j)
103         for (int i = 1; i <= piles.length; i++) {
104             bestScores[i][i][PIETON] = 0;
105             bestScores[i][i][JABA] = piles[i - 1];
106         }
107 
108         //general case. P is the difference of indices of piles
109         for (int p = 1; p < piles.length; p++) {
110             for (int i = 1; i <= piles.length - p; i++) { //i is the left
     index
111                 int j = i + p; //j is the right index
112                 int maxScoreJaba = Integer.MIN_VALUE;
113                 int maxToRemove = Integer.min(depth, p + 1);
114 
115                 //it is Jaba 's turn to play
116                 for (int k = 1; k <= maxToRemove; k++) { //k is the
    number of piles to remove
117                     //remove from left
118                     int scoreLeft = score(k, i, j);
119                     if (k + i <= piles.length)// does not empty the piles
120                         scoreLeft += bestScores[i + k][j][PIETON];
121                     maxScoreJaba = Integer.max(scoreLeft, maxScoreJaba);
122                     //remove from right
123                     int scoreRight = score(-k, i, j);
124                     if (j - k > 0)// does not empty the piles
125                         scoreRight += bestScores[i][j - k][PIETON];
126                     maxScoreJaba = Integer.max(scoreRight, maxScoreJaba);
127                 }
128 
129                 bestScores[i][j][JABA] = maxScoreJaba;
130 
131                 //it is Pieton 's turn to play
132                 int[] play = pietonPlay(i, j);
133                 int maxScorePieton = 0; //Jaba's best score if is Pieton'
    s play
134                 //does not empty the piles
135                 if (play[LEFT] + i <= piles.length && j - play[RIGHT] > 0
    ) {
136                     maxScorePieton = bestScores[i + play[LEFT]][j - play[
    RIGHT]][JABA];
137                 }
138                 bestScores[i][j][PIETON] = maxScorePieton;
139 
140             }
141 
142         }
143         //solution is the best possible way for Jaba to score with the
    piles from 1 to the last
144         return bestScores[1][piles.length][player];
145     }
146 
147 }
```