



# 1



- 0001
- PAGE2
- Nginx
- 0000



1

page contents

# PAGE2

PAGES2

```
main() {  
  }
```

# Nginx

## Nginx

### 1

web

2

image-20211021081901761

Nginx

Nginx

web  
Nginx

### 2

Nginx 2

Ip hash

image-20211021082137339

Ip hash ip hash hash ip  
session

image-20211021082157521

### 3 web

Nginx FastCGI\_Cache FastCGI  
ngx\_cache\_purge URL

### 4

1.\$remote\_addr □ \$http\_x\_forwarded\_for □□□□□□□□ ip□□□

2.\$remote\_user □□□□□□□□□□

3.\$time\_local □ □□□□□□□□□

4.\$request □ □□□□□□□ url□ http□□□

5.\$status □ □□□□□□□□□ 200□

6.\$body\_bytes\_sent □□□□□□□□□□□□□□

7.\$http\_referer □□□□□□□□□□□□□□

8.\$http\_user\_agent □□□□□□□□□□□□□□



```
static boolean is(int x) {  
    return x%400==0 || (x%4==0 && x%100!=0);  
}
```



```
static boolean is(int n) {  
    if(n==1)  
        return false;  
    for(int i=2;i*i<=Math.sqrt(n);i++)  
        if(n%i==0)  
            return false;  
    return true;  
}
```



```
public class 测试 {  
  
    public static void main(String[] args) {  
        boolean[] is = new boolean[1000005];  
        is[1] = true; //true  
        for(int i=2;i<1000005;i++)  
            if(!is[2])  
                for(int j=2*i;j<1000005;j+=i)  
                    is[j] = true;  
        System.out.println(is[2004]);  
    }  
  
}
```

# 01

```
public class _01 {  
  
    public static void main(String[] args) {  
        Scanner in = new Scanner(System.in);  
        int m = in.nextInt();  
        int n = in.nextInt();  
        int[] dp = new int[m+5];  
        for(int i=1;i<=n;i++) {  
            int v = in.nextInt();  
            int w = in.nextInt();  
            for(int j=m;j>=v;j--)  
                dp[j] = Math.max(dp[j], dp[j-v]+w);  
        }  
        System.out.println(dp[m]);  
    }  
}
```



```
import java.util.Scanner;  
  
public class _ {  
  
    public static void main(String[] args) {  
        Scanner in = new Scanner(System.in);  
        int m = in.nextInt();  
        int n = in.nextInt();  
        int[] dp = new int[m+5];  
        for(int i=1;i<=n;i++) {  
            int v = in.nextInt();  
            int w = in.nextInt();  
            for(int j=v;j<=m;j++)  
                dp[j] = Math.max(dp[j], dp[j-v]+w);  
        }  
        System.out.println(dp[m]);  
    }  
}
```

```
}

```

```
}
```



## gcd

```
static int gcd(int a,int b) {
    return b==0?a:gcd(b,a%b);
}
```



## lcm

```
static int lcm(int a,int b) {
    return a*b/gcd(a,b);
}
```



```
static void f(int[] a) {
    int n = a.length;
    int l=0,r=n-1,ans=0;
    while(l<=r) {
        int mid = l + (r-l)/2;
        if(ok(a[mid])) {
            r = mid-1;
            ans = mid;
        }else
            l = mid+1;
    }
    System.out.println(ans);
}

static boolean ok(int x) {
    return false;
}
```





```
import java.util.Scanner;

public class [ ] [ ] [ ] [ ] {

    [ ] public static void main(String[] args) {
        [ ] Scanner in = new Scanner(System.in);
        [ ] int n = in.nextInt();
        [ ] int[] sum = new int[n+1];
        [ ] for(int i=1;i<=n;i++)
            [ ] sum[i] = sum[i-1] + in.nextInt(); // [ ] [ ] [ ] [ ]

    [ ] }

}
```



( [ ] [ ] [ ] [ ] )

```
[ ] static int n,m,cnt=0;
[ ] static int[] f = new int[10005]; // [ ] [ ] [ ] [ ] [ ] f[i]=i
[ ]
[ ] static int find(int x) {
    [ ] if(f[x]==x)
        [ ] return x;
    [ ] return f[x] = find(f[x]);
    [ ] }
[ ]
[ ] static void union(int x,int y) {
    [ ] int a = find(x);
    [ ] int b = find(y);
    [ ] if(a!=b) {
        [ ] f[a] = b;
        [ ] cnt++;
    [ ] }
    [ ] }
```



# BigInteger & BigDecimal

```
import java.math.BigInteger;

public class Main {

    public static void main(String[] args) {

        //Addition
        BigInteger add1 = new BigInteger("10");
        System.out.println(add1.add(new BigInteger("20")));

        //Subtraction
        BigInteger sub1 = new BigInteger("10");
        System.out.println(sub1.subtract(new BigInteger("20")));

        //Division
        BigInteger div1 = new BigInteger("10");
        System.out.println(div1.divide(new BigInteger("20")));

        //Multiplication
        BigInteger mul1 = new BigInteger("10");
        System.out.println(mul1.multiply(new BigInteger("20")));

        //Remainder
        BigInteger remain1 = new BigInteger("10");
        System.out.println(remain1.remainder(new BigInteger("8")));

        //ModPow
        BigInteger mod = new BigInteger ("10");
        BigInteger pow = new BigInteger ("20");
        System.out.println(pow.modPow(pow,mod));

        //Compare
        //10, -1, 1, 0, 0
        BigInteger comp1 = new BigInteger("10");
        System.out.println(comp1.compareTo(new BigInteger("18")));

        //Power
        BigInteger power1 = new BigInteger("2");
        System.out.println(power1.pow(10));

        //Factorial
```

```
BigInteger min1 = new BigInteger("2");
System.out.println(min1.min(new BigInteger("-23")));
```

```
//[ ] [ ] [ ] [ ]
```

```
BigInteger max1 = new BigInteger("2");
System.out.println(max1.max(new BigInteger("-23")));
```

```
//[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
```

```
BigInteger val = new BigInteger("123");
System.out.println(val.intValue());
```

```
//[ ] [ ] [ ] [ ] [ ] [ ]
```

```
BigInteger gcd1 = new BigInteger("12");
System.out.println(gcd1.gcd(new BigInteger("6")));
```

```
//[ ] [ ]
```

```
BigInteger neg1 = new BigInteger("12");
System.out.println(neg1.negate());
```

```
//[ ] [ ] [ ] [ ]
```

```
BigInteger and1 = new BigInteger("10");
System.out.println(and1.and(new BigInteger("1")));
```

```
//[ ] [ ] [ ] [ ]
```

```
BigInteger or1 = new BigInteger("10");
System.out.println(or1.or(new BigInteger("10")));
```

```
//[ ] [ ]
```

```
BigInteger xor1 = new BigInteger("10");
System.out.println(xor1.xor(new BigInteger("10")));
```

```
//[ ] [ ] n[ ] [ ] [ ] [ ] [ ] ( [ ] [ ] [ ] [ ] )
```

```
BigInteger decimal1 = new BigInteger("12");
System.out.println(decimal1.toString(2));
```

```
//[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
```

```
BigInteger abs1 = new BigInteger("-12");
System.out.println(abs1.abs());
```

```
//[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
```

```

BigInteger testBit1 = new BigInteger("4");
System.out.println(testBit1.testBit(2));

// 1
BigInteger moveLeftBit1 = new BigInteger("4");
System.out.println(moveLeftBit1.shiftLeft(1));

// 1
BigInteger moveRightBit1 = new BigInteger("4");
System.out.println(moveRightBit1.shiftLeft(-1));

//
BigInteger not = new BigInteger ("10");
System.out.println(not.not());

//valueOf()
//
//
BigInteger negate = new BigInteger ("10");
System.out.println(negate.negate());

//
//
BigInteger prime = new BigInteger ("10");
System.out.println(prime.probablePrime());
System.out.println(prime.nextprobablePrime());
}
}

```

```

package com.vivo.ars.util;
import java.math.BigDecimal;

/**
 *
 */
public class ArithmeticUtils {
    //
    private static final int DEF_DIV_SCALE = 10;

    /**
     *
     *

```

```

* @param v1 []
* @param v2 []
* @return []
*/

```

```

public static double add(double v1, double v2) {
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));
    return b1.add(b2).doubleValue();
}

```

```

/**
 * []
 *
 * @param v1 []
 * @param v2 []
 * @return []
 */

```

```

public static BigDecimal add(String v1, String v2) {
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.add(b2);
}

```

```

/**
 * []
 *
 * @param v1 []
 * @param v2 []
 * @param scale [] scale []
 * @return []
 */

```

```

public static String add(String v1, String v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException(
            "The scale must be a positive integer or zero");
    }
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.add(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
}

```

```

/**
 * [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
 *
 * @param v1 [ ] [ ]
 * @param v2 [ ] [ ]
 * @return [ ] [ ] [ ] [ ] [ ]
 */
public static double sub(double v1, double v2) {
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));
    return b1.subtract(b2).doubleValue();
}

```

```

/**
 * [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
 *
 * @param v1 [ ] [ ]
 * @param v2 [ ] [ ]
 * @return [ ] [ ] [ ] [ ] [ ]
 */
public static BigDecimal sub(String v1, String v2) {
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.subtract(b2);
}

```

```

/**
 * [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
 *
 * @param v1 [ ] [ ] [ ]
 * @param v2 [ ] [ ]
 * @param scale [ ] [ ] scale [ ] [ ]
 * @return [ ] [ ] [ ] [ ] [ ]
 */
public static String sub(String v1, String v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException(
            "The scale must be a positive integer or zero");
    }
    BigDecimal b1 = new BigDecimal(v1);

```

```

        BigDecimal b2 = new BigDecimal(v2);
        return b1.subtract(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
    }

```

```

/**
 * [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
 *
 * @param v1 [ ] [ ]
 * @param v2 [ ] [ ]
 * @return [ ] [ ] [ ] [ ] [ ]
 */
public static double mul(double v1, double v2) {
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));
    return b1.multiply(b2).doubleValue();
}

```

```

/**
 * [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
 *
 * @param v1 [ ] [ ]
 * @param v2 [ ] [ ]
 * @return [ ] [ ] [ ] [ ] [ ]
 */
public static BigDecimal mul(String v1, String v2) {
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.multiply(b2);
}

```

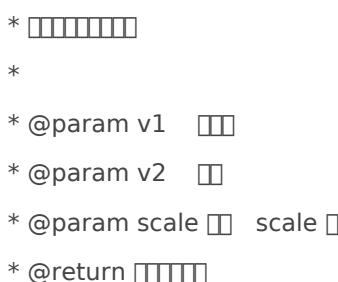





```

/**
 * [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
 *
 * @param v1 [ ] [ ]
 * @param v2 [ ] [ ]
 * @param scale [ ] [ ] scale [ ] [ ]
 * @return [ ] [ ] [ ] [ ] [ ]
 */
public static double mul(double v1, double v2, int scale) {
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));

```






```

        return round(b1.multiply(b2).doubleValue(), scale);
    }

    /**
     * 
     *
     * @param v1 
     * @param v2 
     * @param scale  scale 
     * @return 
     */
    public static String mul(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                "The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.multiply(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
    }

```

```

    /**
     * 
     * 
     *
     * @param v1 
     * @param v2 
     * @return 
     */

    public static double div(double v1, double v2) {
        return div(v1, v2, DEF_DIV_SCALE);
    }

```

```

    /**
     *  scale
     * 
     *
     * @param v1 
     * @param v2 
     * @param scale 

```



```

* @return double
*/
public static double div(double v1, double v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException("The scale must be a positive integer or zero");
    }
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));
    return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).doubleValue();
}

```

```

/**
 * Divides the value of the first operand by the value of the second operand
 * and returns the result. The scale of the result is the same as the scale
 * of the first operand.
 *
 * @param v1 the first operand
 * @param v2 the second operand
 * @param scale the scale of the result
 * @return double
 */
public static String div(String v1, String v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException("The scale must be a positive integer or zero");
    }
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).toString();
}

```

```

/**
 * Divides the value of the first operand by the value of the second operand
 * and returns the result. The scale of the result is the same as the scale
 * of the first operand.
 *
 * @param v the first operand
 * @param scale the scale of the result
 * @return double
 */
public static double round(double v, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException("The scale must be a positive integer or zero");
    }
    BigDecimal b = new BigDecimal(Double.toString(v));

```

```

        return b.setScale(scale, BigDecimal.ROUND_HALF_UP).doubleValue();
    }

```

```

/**

```

```

 *         

```

```

 *


```

```

 * @param v      






```

```

 * @param scale      

```

```

 * @return      

```

```

 */

```

```

public static String round(String v, int scale) {

```

```

    if (scale < 0) {

```

```

        throw new IllegalArgumentException(

```

```

            "The scale must be a positive integer or zero");

```

```

    }

```

```

    BigDecimal b = new BigDecimal(v);

```

```

    return b.setScale(scale, BigDecimal.ROUND_HALF_UP).toString();

```

```

}

```

```

/**

```

```

 *   




```

```

 *



```

```

 * @param v1   

```

```

 * @param v2  

```

```

 * @param scale      

```

```

 * @return  

```

```

 */

```

```

public static String remainder(String v1, String v2, int scale) {

```

```

    if (scale < 0) {

```

```

        throw new IllegalArgumentException(

```

```

            "The scale must be a positive integer or zero");

```

```

    }

```

```

    BigDecimal b1 = new BigDecimal(v1);

```

```

    BigDecimal b2 = new BigDecimal(v2);

```

```

    return b1.remainder(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();

```

```

}

```

```

/**

```

```

 *    BigDecimal




```

```

 *

```

```

 * @param v1   

```

```

* @param v2  []
* @param scale  []
* @return []
*/
public static BigDecimal remainder(BigDecimal v1, BigDecimal v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException(
            "The scale must be a positive integer or zero");
    }
    return v1.remainder(v2).setScale(scale, BigDecimal.ROUND_HALF_UP);
}

/**
 * []
 *
 * @param v1 []
 * @param v2 []
 * @return [] v1 [] v2 [] [] true [] false
 */
public static boolean compare(String v1, String v2) {
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    int bj = b1.compareTo(b2);
    boolean res;
    if (bj > 0)
        res = true;
    else
        res = false;
    return res;
}
}

```