Ex3:

Bloom2.py
(With using files)

```
import math

def read_elements_from_file(filename):
    with open(filename, 'r') as file:
        content = file.read()
        elements = [int(x) for x in content.split() if x.strip()]
    return elements
def write_output_to_file(filename, bit_array, collisions, ones_count, error_rates):
    with open(filename, 'w') as file:
        file.write("Bit Array (first 100 bits shown):\n")
        file.write(' '.join(map(str, bit_array[:100])) + '\n\n')
        file.write("Number of collisions: %s\n" % collisions)
        file.write("Number of 1's in bit array: %s\n" % ones_count)
        file.write("Fill percentage: %.2f%%\n\n" % (ones_count/len(bit_array)*100))
        file.write("Error rates analysis:\n")
        for k,rate in error_rates.items():
            file.write("With %s hash functions: %.6f\n" % (k, rate))
def hash1(x, size):
    return (2*x + 10) % size

def hash2(x, size):
    return (10 * ((5*x) + 30)) % size

def hash3(x, size):
    return (x + 4) % size

def bloom_filter(elements, bit_array_size):
    bit_array = [0] * bit_array_size
    collisions = 0

    for x in elements:
        h1 = hash1(x, bit_array_size)
        h2 = hash2(x, bit_array_size)
        h3 = hash3(x, bit_array_size)

        for h in [h1, h2, h3]:
            if bit_array[h] == 0:
                bit_array[h] = 1
            else:
```

```
            collisions += 1
    ones_count = sum(bit_array)

    error_rates = {}
    n = len(elements)
    m = bit_array_size

    for k in [1, 2, 3]:
        exponent = -k * m / n
        error_rate = 1 - math.exp(exponent) ** k
        error_rates[k] = error_rate
    return bit_array, collisions, ones_count, error_rates
def main():
    input_filename = 'input.txt'
    output_filename = 'output.txt'
    bit_array_size = 300  # Using 300-bit array as requested

    elements = read_elements_from_file(input_filename)
    if len(elements) != 500:
        print("Warning: Expected 500 elements, found %s" % len(elements))
    bit_array, collisions, ones_count, error_rates = bloom_filter(elements, bit_array_size)

    write_output_to_file(output_filename, bit_array, collisions, ones_count, error_rates)
    print("Processing complete. Results written to %s" % output_filename)
    print("\nFor %s elements and %s-bit array:" % (len(elements), bit_array_size))
    print("- Optimal hash functions (k): %s" % round((bit_array_size/len(elements)) * math.log(2)))
    print("- Theoretical false positive rate: %.6f" % ((1 - math.exp(-3 *
bit_array_size/len(elements))) ** 3))
if __name__ == "__main__":
    main()
```

input.txt

101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166
167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188
189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232
233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254
255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276
277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298
299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320
321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342

343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364
365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386
387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408
409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430
431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452
453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474
475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496
497 498 499 500

bloomfilter.py

(This is what mam explained in class without using files)

```
arr=[0] * 13
collisions=0
def hash1(x):
    global collisions
    output=(2*x+10)%13
    if arr[output] ==0:
        arr[output]=1
    else:
        collisions+=1
def hash2(x):
    global collisions
    output =(10*((5*x) + 30)) %13
    if arr[output]==0:
        arr[output]=1
    else:
        collisions +=1
def hash3(x):
    global collisions
    output=(x+4)%13
    if arr[output]==0:
        arr[output]=1
    else:
        collisions +=1
hash1(153)
hash2(153)
hash3(153)
hash1(210)
hash2(210)
hash3(210)
hash1(145)
hash2(145)
hash3(145)
```

```
hash1(201)
hash2(201)
hash3(201)
print(arr)
print("no. of collisions:", collisions)
one=0
for i in arr:
    if arr[i]==1:
        one+=1
print("no. of one's in the bit array:", one)
#analysis
y=13
x=4
import math
def error_calc(k):
    error=1-(math.exp(-k*y/x))**k
    print(error)
print("Case 1: no. of hash func=1")
error_calc(1)
print("Case 2: no. of hash func=2")
error_calc(2)
print("Case 3: no. of hash func=3")
error_calc(3)
```

Ex2

gedit mapper_csv.py
Gedit reducer_csv.py
Gedit weather.txt (year,temp)

Cat weather.txt | python mapper_csv.py
Cat weather.txt | python mapper_csv.py | sort -k1,1 |python reducer_csv.py

Now create a sample jar1.txt to get path

hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.6.0-mr1-cdh5.4.2.jar -file /home/cloudera/mapper_csv.py -file /home/cloudera/reducer_csv.py -mapper "python mapper_csv.py" -reducer "python reducer_csv.py" -input weather.txt -output /home/cloudera/nn

```
hadoop fs -ls /nn

 hadoop fs -cat /nn/part-00000 | sort -t$' ' -k2,2n | head -n 1
```

Mapper_csv.py

```
import sys

sys.stdin.readline()

for line in sys.stdin:
    field=line.strip().split(',')
    try:
        year,temperature=field[0],int(field[1])
        print year, temperature
    except ValueError:
        Continue
```

Reducer_csv.py

```
c

    try:
        year,temp_str=line.split(' ')
        temperature=int(temp_str)
    except ValueError:
        continue
    if current_year==year:
        if temperature<current_min_temp:
            current_min_temp=temperature
    else:
        if current_year:
            print current_year,current_min_temp
        current_min_temp=temperature
        current_year=year
if current_year:
    print current_year,current_min_temp
```

Weather.txt

```
2010,25
2012,30
```

2013,35
2015,40
2024,47
2022,10

Ex1

hadoop fs -cat /nn/part-00000

mapper.py

```
import sys

for line in sys.stdin:
	line=line.strip()
	words = line.split()
	for word in words:
		print word,"	",1
```

reducer.py

```
import sys
def main():
	current_word =None
	current_count=0
	for line in sys.stdin:
		line = line.strip()
		word,count = line.split(' ',1)
		try:
			count=int(count)
		except ValueError:
			continue
		if current_word==word:
			current_count+=count
		else:
			if current_word:
				print current_word,"	",current_count
			current_word=word
			current_count=count
		if current_word:
			print current_word,"	",current_count
if __name__=="__main__":
  main()
```

Sample.txt
Sastra university sastra good

Ex 5 (page rank)

Page_input.txt

A B C D
B D E
C A F
D B E
E F
F C
G H
H G

job_mapper:

```
#!/usr/bin/env python
import sys

def main():
    """
    Mapper for Job 1: Graph Parsing.
    Input: A line like "A B C D"
    Output: A tab-separated key-value pair: "A\tB,C,D"
    """
    for line in sys.stdin:
        parts = line.strip().split()
        if len(parts) < 1:
            continue

        source_page = parts[0]
        out_links = parts[1:]

        # Emit the source page and its adjacency list as a comma-separated string
        print(f"{source_page}\t{','.join(out_links)}")

if __name__ == "__main__":
    main()
```

job_reducer:

```
#!/usr/bin/env python
import sys

# Total number of pages in our graph.
TOTAL_PAGES = 8.0
INITIAL_RANK = 1.0 / TOTAL_PAGES

def main():
    """
    Reducer for Job 1: Initial Rank Assignment.
    Input: A line like "A\tB,C,D"
    Output: A line with the initial rank: "A\t0.125,B,C,D"
    """
    for line in sys.stdin:
        source_page, out_links_str = line.strip().split('\t', 1)

        # Prepend the initial rank to the out-links string
        print(f"{source_page}\t{INITIAL_RANK},{out_links_str}")

if __name__ == "__main__":
    main()
```

```
hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.6.0-mr1-cdh5.4.2.jar -file /home/cloudera/job_mapper.py -file /home/cloudera/job_reducer.py -mapper "python job_mapper.py" -reducer "python job_reducer.py" -input page_input.txt -output /home/cloudera/page_rank
```

mapper_page :

```
import sys

def main():
    """
    Mapper for the iterative PageRank calculation (Job 2).
    Input: <PageID>\t<Rank>,<Outlink1>,<Outlink2>...
    """
    for line in sys.stdin:
        page_id, data = line.strip().split('\t', 1)

        # --- FIX IS HERE ---
        # Split the 'data' part by the first comma to correctly separate rank from links.
        try:
            rank_str, out_links_str = data.split(',', 1)
```

```
        except ValueError:
            # This handles lines that might not have out-links (e.g., "E\t0.125,")
            rank_str = data.rstrip(',')
            out_links_str = ""
        current_rank = float(rank_str)
        out_links = out_links_str.split(',') if out_links_str else []
            out = ','.join(out_links)
        # --- END OF FIX ---

        # 1. Pass along the graph structure
        # Re-join the original out_links to preserve the structure
        print page_id + '\t' + '|' + out

        # 2. Distribute rank to out-links
        if out_links:
            num_out_links = len(out_links)
            if num_out_links > 0:
                rank_contribution = current_rank / num_out_links
                for link in out_links:
                    if link: # Ensure the link is not an empty string
                        print link + '\t' + str(rank_contribution)

if __name__ == "__main__":
    main()
```

reducer_page :

```
import sys

DAMPING_FACTOR = 0.85

def main():
    current_page = None
    total_rank_contribution = 0.0
    adjacency_list = ""

    for line in sys.stdin:
        page_id, value = line.strip().split('\t', 1)

        if current_page and current_page != page_id:
            # Process the completed page
            new_rank = (1 - DAMPING_FACTOR) + (DAMPING_FACTOR * total_rank_contribution)
            print current_page + '\t' + str(new_rank) + ',' + adjacency_list

            # Reset for the next page
```

```
            current_page = page_id
            total_rank_contribution = 0.0
            adjacency_list = ""

        if not current_page:
            current_page = page_id

        # Process the value
        if value.startswith('|'):
            adjacency_list = value[1:]
        else:
            total_rank_contribution += float(value)

    # Output the last page
    if current_page:
        new_rank = (1 - DAMPING_FACTOR) + (DAMPING_FACTOR * total_rank_contribution)
        print current_page + '\t' + str(new_rank) + ',' +adjacency_list

if __name__ == "__main__":
    main()
```

```
hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.6.0-mr1-cdh5.4.2.jar -file /home/cloudera/mapper_p.py -file /home/cloudera/reducer_p.py -mapper "python mapper_p.py" -reducer "python reducer_p.py" -input /home/cloudera/page_rank/part-00000 -output /home/cloudera/page_rank_output

hadoop fs -cat /home/cloudera/page_rank/part-00000
```

Ex 4

```
import string
import math

def read_emails_from_file(filepath):
    with open(filepath,'r')as f:
        emails=[line.strip() for line in f]
    return emails

def decode_emails_to_decimals(data):
    decimal_values=[]
```

```
    for email in data:
        decimal_sum=sum(ord(char)for char in email)
        decimal_values.append(decimal_sum)
    print("decimal value %s " %(decimal_values))
    return decimal_values

def hash_func1(value):
    return (4*value+5)%13

def hash_func2(value):
    return(value/3)%5

def count_trailing_zeros(n):
    if n==0:
        return 0
    binary_rep=bin(n)
    return len(binary_rep)-len(binary_rep.rstrip('0'))

def find_max_trailing_zeros(decimal_values,hash_func):
    max_r=0
    for value in decimal_values:
        hashed_value=hash_func(value)
        trailing_zeros=count_trailing_zeros(hashed_value)
        if trailing_zeros>max_r:
            max_r=trailing_zeros
    return max_r
def error_rate(r,m):
    val1=2**(-r)
    val2=-m*val1
    val3=math.exp(val2)
    val4=1-val3
    return val4

def main():
    input_file="email.txt"
    emails=read_emails_from_file(input_file)
    if not emails:
        print("Error: Input file is empty or could not be read!")
        return
    actual_distinct_count=len(set(emails))
    print("Total E-mails Processes : %s" %(len(emails)))
    print("Actual unique emails : %s" %(actual_distinct_count))
    decimal_inputs=decode_emails_to_decimals(emails)
    R1=find_max_trailing_zeros(decimal_inputs,hash_func1)
```

```
    print("maximum trailing zeros (R-value1) : %s" %(R1))
    estimated_distinct_count1=2**R1
    print("Estimated unique emails (hash func1) : %s" %(estimated_distinct_count1))
    R2=find_max_trailing_zeros(decimal_inputs,hash_func2)
    print("maximum trailing zeros (R-value2) : %s" %(R2))
    estimated_distinct_count2=2**R2
    print("Estimated unique emails (hash func2) : %s" %(estimated_distinct_count2))
    m=len(emails)
    err1=error_rate(R1,m)
    print("error rate for hash function 1 : %s" %(err1))
    err2=error_rate(R2,m)
    print("error rate for hash function 2 : %s" %(err2))
    if err1<err2:
        print("Hash function 1 is efficient")
    else:
        print("hash function 2 is efficient")

if __name__ == "__main__":
    main()
```