

```

1 # Authored By: Jayant Sogikar
2 import ast
3 from nis import cat
4 import geocoder
5 from datetime import datetime
6 import json
7 from math import asin, cos, pi, sqrt
8 import os
9 from flask import Flask, jsonify, url_for, redirect, request, Blueprint
10 import pymongo
11 from bson.json import dumps
12 from bson import ObjectId
13 from flask_caching import Cache
14 from dataGenerator.barcodeLookup import getBarcodeInfo
15 from dataGenerator.closestSupplierInfo import getSupplierInfo
16 from dataGenerator.distanceEmission import Emission
17 from dataGenerator.geoRun import calcTotalDistance
18 from predictor.predict import predictCategory
19 import redis
20 from flask_cors import CORS
21 from google_images_search import GoogleImagesSearch
22 from hellosign.sdk import ApiClient, ApiException, Configuration, apis, models
23 import requests
24
25 subscription_key = "8e2959e6bd2247c28b7c9059fe237b60"
26 endpoint = "https://sih-vision.cognitiveservices.azure.com/vision/v3.2/analyze?visualFeatures=Tags"
27
28
29
30 API_KEY = "AizaSyAVhuJbVRBxIjerR-TuR97vY_ubUeKEfp0"
31 CX = "145dcb7386f4248c6"
32 gis = GoogleImagesSearch(API_KEY, CX)
33
34
35 config = {
36     "CACHE_TYPE": "RedisCache",
37     "CACHE_REDIS_HOST": "localhost",
38     "CACHE_REDIS_PORT": "6379",
39     "CACHE_REDIS_URL": "redis://localhost:6379"
40 }
41
42 mongoClnet = pymongo.MongoClient('mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.3.1')
43 redisclient = redis.StrictRedis('localhost',port=6379,db=0,charset="utf-8", decode_responses=True)
44
45 helloSignConfig = Configuration(
46     usernames="096f2e433b09835cb775448649e17799bb699a49a682ec660218bbc059c8d625",
47 )
48
49
50 db = mongoClnet['new']
51
52 app = Flask(__name__)
53 CORS(app)
54
55 cache = Cache(app, config = config)
56
57 def getImage(name):
58     API_KEY = "AizaSyAs0NhZ3xEGNMxXfHuvrHU_3_kkPHYgMM"
59     CX = "f76f632382fc4e4c"
60
61     res = requests.get("https://www.googleapis.com/customsearch/v1", params={"key" : API_KEY, "cx" : CX, "q" : name})
62
63
64     for x in json.loads(res.content)["items"]:
65
66         if "pagemap" in x and "cse_thumbnail" in x["pagemap"] and len(x["pagemap"]["cse_thumbnail"]) > 0 and int(x["pagemap"]["cse_thumbnail"][0]["width"]) < 420:
67             return x["pagemap"]["cse_thumbnail"][0]["src"]
68
69
70     return "https://upload.wikimedia.org/wikipedia/commons/thumb/a/ac/No_image_available.svg/1024px-No_image_available.svg.png"
71
72
73 @app.route('/')
74 def home_page():
75     return 'Hello World'
76
77
78 @app.route('/getProduct')
79 def get_product():
80     """
81     GET request | Returns product details
82     Query Params : barcode
83     Returns : json containing all the details
84     """
85     details = request.args
86     barcode = details["barcode"]
87
88     mongoCollection = db['products']
89
90     return dumps(mongoCollection.find_one({'_id' : barcode}))
91
92 @app.route('/getProductsByCategory')
93 def get_product_by_category():
94     """
95     GET request | Returns product details
96     Query Params : categories
97     Returns : json containing all the details
98     """
99     details = request.args
100     cats = []
101     for x in details["categories"].split(','):
102         cats.append(x.strip())
103
104     mongoCollection = db['products']
105
106     return dumps(list(mongoCollection.find({"category" : {"$in" : cats}})))
107
108 # Generate a key for product name caching
109 def productNameKey():
110     data = request.args
111
112     return str(data["searchTerm"]).lower()
113
114 @app.route('/getProductByName')
115 @cache.cached(timeout=100, key_prefix=productNameKey)
116 def get_product_by_name():
117     """
118     GET request | Returns list of product details
119     Query Params : searchTerm
120     Returns : json containing all the details
121     """
122     details = request.args
123     searchTerm = details["searchTerm"]
124
125     mongoCollection = db['products']
126
127     res = mongoCollection.find(
128         { '$text': { '$search': searchTerm } },
129         { '$score': { '$meta': "textScore" } }
130     ).sort([( '$score', { '$meta': "textScore" })])
131
132     return dumps(list(res))
133
134 @app.route('/getProductFromManufacturer')
135 def get_product_from_manufacturer():
136     """
137     GET request | Returns list of product details
138     Query Params : mid
139     Returns : json containing all the details
140     """
141     details = request.args
142     mid = details["mid"]

```

```

143     mongoCollection = db['products']
144
145     return dumps(list(mongoCollection.find({"manufacturer" : {"$in" : [mid]}])))
146 # Generate a key for supplier caching
147 def supplierKey():
148     data = request.args
149     return str(data["searchTerm"]).lower()
150
151 @app.route('/getSuppliers')
152 @cache.cached(timeout=50, key_prefix=supplierKey)
153 def get_supplier():
154     """
155     GET request | Returns supplier details
156     Query Params : searchTerm, latitude, longitude
157     Returns : json containing all the details
158     """
159     details = request.args
160
161     term = details["searchTerm"]
162     lat = float(details["latitude"])
163     long = float(details["longitude"])
164
165     res = getSupplierInfo(term,lat,long)
166
167     return json.dumps(res)
168
169 # Generate a key for barcode caching
170 def barcodeKey():
171     data = request.args
172     return str(data["barcode"]).lower()
173
174 @app.route('/getProductNameByBarcode')
175 @cache.cached(timeout=0, key_prefix=barcodeKey)
176 def get_product_name_by_barcode():
177     """
178     GET request | Returns barcode product
179     Query Params : barcode
180     Returns : json containing all the details
181     """
182     details = request.args
183
184     barcode = details["barcode"]
185
186     return getBarcodeInfo(barcode)
187
188 @app.route('/getProductDetailsByBarcode')
189 def get_product_details_by_barcode():
190     """
191     GET request | Returns barcode product
192     Query Params : barcode
193     Returns : json containing all the details
194     """
195     details = request.args
196
197     barcode = details["barcode"]
198
199     mongoCollection = db['products']
200
201     res = mongoCollection.find_one({'_id' : barcode})
202
203     if res:
204         return dumps(res)
205
206     redisRes = redisclient.get("flask_cache_" + barcode)
207
208     if redisRes:
209         name = json.loads(redisRes)["productName"]
210     else:
211         name = getBarcodeInfo(barcode)["productName"]
212
213     redisclient.set("flask_cache_" + barcode, json.dumps({"productName":name,"status":"Passed"}))
214
215     predisRed = redisclient.get("flask_cache_cat" + name.lower())
216
217     imgUrl = getImage(name)
218
219     # t1 = ThreadWithReturnValue(target = getImage, args = (name,))
220
221     # t1.start()
222
223     if predisRed:
224         cats = json.loads(predisRed)
225     else:
226         cats = predictCategory(name)[:10]
227
228     redisclient.set("flask_cache_cat" + name.lower(), json.dumps(cats))
229
230
231     catId = '+'.join(cats[:5])
232
233     res = db['categoryEmission'].find_one({'_id' : catId})
234
235     emission = 150
236
237     rating = 2.5
238
239     if res:
240         emission = res['totalEmission'] / res['totalManufacturers']
241
242     # imgUrl = t1.join()
243
244     return dumps({"_id" : "NA", "weight" : -1, "price" : -1, "category" : cats, "categoryID" : catId, "image_url" : imgUrl, "manufacturer" : [], "name" : name, "rating" : 2.5, "emission" : emission})
245
246 @app.route('/getAllRoutes')
247 def get_all_routes():
248     """
249     GET request | Returns all routes from A to B and its emission
250     Query Params : fromAddress, toAddress
251     Returns : json containing all the details
252     """
253     details = request.args
254
255     return calcTotalDistance(details["fromAddress"], details["toAddress"])
256
257
258 @app.route('/getManufacturers')
259 def get_manufacturers():
260     """
261     Deprecated
262     GET request | Returns all Manufacturers
263     Query Params : searchTerm
264     Returns : json containing all the details
265     """
266     details = request.args
267
268     searchTerm = details["searchTerm"]
269
270     mongoCollection = db['manufacturers']
271
272     res = mongoCollection.find(
273         { '$text': { '$search': searchTerm } },
274         { 'score': { '$meta': "textScore" } } )
275

```

```

285     ).sort([('score', {'$meta': 'textScore'})])
286
287     return dumps(list(res))
288
289 @app.route('/getManufacturer')
290 def get_manufacturer():
291     """
292     GET request | Returns the Manufacturer details
293     Query Params : mid
294     Returns : json containing all the details
295     """
296
297     details = request.args
298
299     mid = details["mid"]
300
301     mongoCollection = db['manufacturers']
302
303     res = mongoCollection.find_one({"_id" : mid})
304
305     return dumps(res)
306
307 # Generate a key for category caching
308 def categoryKey():
309     data = request.args
310
311     return str("cat" + data["searchTerm"]).lower()
312
313 @app.route('/getCategories')
314 @cache.cached(timeout=0,key_prefix=categoryKey)
315 def get_categories():
316     """
317     GET request | Returns category through name
318     Query Params : searchTerm
319     Returns : json array having top 10 categories
320     """
321
322     details = request.args
323
324     searchTerm = details["searchTerm"]
325
326     res = predictCategory(searchTerm)
327
328     return dumps(res[:10])
329
330
331 @app.route('/addManufacturer',methods=['POST'])
332 def add_manufacturer():
333     """
334     POST request | Adds a Manufacturer
335     Json params : [_id,companyName,lat,long,address,phone]
336     """
337
338     details = request.json
339
340     mongoCollection = db['manufacturers']
341
342     res = mongoCollection.insert_one(details)
343
344     return jsonify({'_id' : str(res.inserted_id)}) if res.acknowledged else 'Failed'
345
346
347 @app.route('/addProduct',methods=['POST'])
348 def add_product():
349     """
350     POST request | Adds a Product
351     Json params : [name, weight, price, category, emission, manufacturer, barcode, rawMaterials, components]
352     """
353
354     details = request.json
355     mongoCollection = db['products']
356
357     # Details Init
358     name = details["name"]
359     cats = details["category"]
360     emission = details["emission"]
361     mid = details["manufacturer"]
362     barcode = details["barcode"]
363     rawMaterials = details["rawMaterials"]
364     weight = details["weight"]
365     price = details["price"]
366     components = details["components"]
367
368     imgUrl = getImage(name)
369
370     # Create a category ID
371     catId = str(hash("+".join(cats).replace(" ", "")))
372
373     # Increment for that category using upsert
374     db['categoryEmission'].update_one({'_id' : catId}, {'$inc' : {'totalEmission' : emission, 'totalManufacturers' : 1, }}, upsert=True)
375
376     # Get total value
377     query = db['categoryEmission'].find_one({'_id' : catId})
378
379     tot = query['totalEmission']
380     ct = query["totalManufacturers"]
381
382     # Get rating by using a stat formula
383     rating = 5 * (tot) / (tot + (emission * (ct - 1)))
384
385     res2 = db['manpro'].insert_one({'category' : cats, 'categoryID': catId, 'barcode' : barcode, 'name' : name, 'emission' : emission , 'manufacturer' : mid, 'rawMaterials' : rawMaterials, 'components' : components})
386
387     # Push the product details
388     f = mongoCollection.find_one({'_id' : barcode})
389
390     if f is None:
391         mongoCollection.insert_one({'_id' : barcode, 'category' : cats, 'name' : name, 'categoryID': catId, 'image_url' : imgUrl, 'weight' : weight, "price" : price, 'rating' : rating})
392
393         for m in rawMaterials:
394
395             mongoCollection.update_one({'_id' : barcode}, {'$push': {'rawMaterials': m}})
396
397
398         for m in components:
399
400             mongoCollection.update_one({'_id' : barcode}, {'$push': {'components': m}})
401
402
403     res = mongoCollection.update_one({'_id' : barcode},
404     {
405         '$set' : {'rating' : rating},
406         '$push' : {'manufacturer' : mid},
407         '$inc' : {'totalManufacturers' : 1, 'totalEmission' : emission},
408     }, upsert= True)
409
410     # Insert into a collection that has mid and pid present linking each person with their product
411     res2 = db['manpro'].insert_one({'category' : cats, 'categoryID': catId, 'barcode' : barcode, 'name' : name, 'emission' : emission , 'manufacturer' : mid, 'rawMaterials' : rawMaterials, 'components' : components})
412
413     # Update each product in the category to maintain dynamic scoring
414     for row in mongoCollection.find({'categoryID' : catId}):
415
416         bc = row['_id']
417
418         avgEmission = row['totalEmission'] / row['totalManufacturers']
419
420         rowRating = 5 * (tot) / (tot + (avgEmission * (ct - 1)))
421
422
423         mongoCollection.update_one({'_id' : bc}, {'$set' : {'rating' : rowRating}})
424
425     # Append product into the manufacturer thingy
426

```

```

428 db['manufacturers'].update_one({'_id' : mid}, {'$push' : {'products' : {product}}})
429
430 details = db["manufacturers"].find_one({'_id' : mid})
431
432 JWT = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJlZlZlZWxsbldvcmtzIiwiaXNjaXoxNjYyMjMxODM4LCJpYXQ0IjE2NjIyMzAwMzg5ImVycyI6IkhkbGxvV29ya3MiLCJqdGkiOiJmNmNjUwZmZlcyYwZDU5LTQzOTEtODI0SiJmJzczZTRmNTF9"
433
434 headers = {
435     'Authorization': f"Bearer {JWT}",
436     'Content-Type': 'application/x-www-form-urlencoded',
437 }
438
439 data = f'workflow_id=JTvkdP6QUlFRyQs8&participants[participant1_g2xB3p][type]=email&participants[participant1_g2xB3p][value]=lms19is051@gmail.com&participants[participant1_g2xB3p][full_name]=(details["name"])&shipment_id={shipment_id}&status={status}'
440
441 response = requests.post('https://api.helloworks.com/v3/workflow_instances', headers=headers, data=data)
442 print(data)
443 print(response.text)
444 return jsonify({'status' : str(res.acknowledged and res2.acknowledged)})
445
446
447 # returns the shipment statuses to the manufacturer
448 @app.route('/getShipments')
449 def get_shipments():
450     """
451     GET request | Returns shipment details for a manufacturer
452     Query Params : manufacturer
453     Returns : json containing all the details
454     """
455
456     details = request.args
457
458     manufacturer = details["manufacturer"]
459
460     mongoCollection = db['shipments']
461
462     res = mongoCollection.find({'manufacturer' : manufacturer})
463
464     return dumps(list(res))
465
466
467
468 @app.route('/addShipment',methods=['POST'])
469 def add_shipment():
470     """
471     POST request | Adds a shipment
472     Json params : [manufacturer, startLocation, pid, totalWeight, currentLat, currentLong]
473     dlong and dlong are destination lat and long
474     currentLat and currentLong are the lat and long of where the product is added
475     journey shall be ["Bangalore", "Delhi", "Kolkata"] basically the location names where all it reaches
476     """
477
478     details = request.json
479
480     manufacturer = details['manufacturer']
481
482     details["status"] = "PROCESSING"
483     details["timestamp"] = datetime.now()
484     details["journey"] = [details["startLocation"]]
485     details["transportMode"] = "-"
486     details["enroute_to"] = "-"
487     details["emission"] = 0 # just the carbon so far used. in update we will update this
488
489     mongoCollection = db['shipments']
490     res = mongoCollection.insert_one(details)
491
492     db['manufacturers'].update_one({'_id': manufacturer}, {'$push':{'currentShipments': res.inserted_id}},upsert=True)
493
494     return dumps({'id' : res.inserted_id})
495
496 def distance(lat1, lon1, lat2, lon2):
497
498     p = pi/180
499     a = 0.5 - cos((lat2-lat1)*p)/2 + cos(lat1*p) * cos(lat2*p) * (1-cos((lon2-lon1)*p))/2
500
501     return 12742 * asin(sqrt(a))
502
503 @app.route('/updateShipment',methods=['POST'])
504 def update_shipment():
505     """
506     POST request | Update a shipment
507     Json params : [shipmentID, location, transportMode, currentLat, currentLong, enroute_to, status]
508     """
509
510     details = request.json
511
512     status = details["status"]
513
514     shipmentID = ObjectId(details["shipmentID"])
515
516     res = db['shipments'].find_one({'_id' : shipmentID})
517
518     prevMode = res["transportMode"]
519
520     emission = 0
521
522     if res["status"] == "TRAVEL":
523
524         prevLat, prevLong = res["currentLat"], res["currentLong"]
525         curLat, curLong = details["currentLat"], details["currentLong"]
526
527         if prevMode == "AIR": emission += Emission.airEmission(distance(curLat, curLong,prevLat, prevLong))
528
529         elif prevMode == "WATER" : emission += Emission.waterEmission(distance(curLat, curLong,prevLat, prevLong))
530
531         elif prevMode == "RAIL" : emission += Emission.railEmission(distance(curLat, curLong,prevLat, prevLong))
532
533         elif prevMode == "ROAD" : emission += Emission.roadEmission(distance(curLat, curLong,prevLat, prevLong))
534
535
536
537     db['shipments'].update_one(
538         {'_id' : shipmentID},
539         {
540             "$set" : {
541                 "transportMode" : details["transportMode"],
542                 "status" : status,
543                 "currentLat" : details["currentLat"],
544                 "currentLong" : details["currentLong"],
545                 "enroute_to" : details["enroute_to"]
546             },
547             '$push' : {'journey' : details["location"]},
548             '$inc' : {'emission' : emission },
549         }
550     )
551
552 else:
553
554     db['shipments'].update_one(
555         {'_id' : shipmentID},
556         {
557             "$set" : {
558                 "transportMode" : details["transportMode"],
559                 "status" : status,
560                 "currentLat" : details["currentLat"],
561                 "currentLong" : details["currentLong"],
562                 "enroute_to" : details["enroute_to"]
563             },
564             '$inc' : {'emission' : emission },
565         }
566     )
567
568 if (status == "DELIVERED"):

```

```

569         state = geocoder.osm([details["currentLat"], details["currentLong"]], method='reverse').state
570
571         ship = db["shipments"].find_one({"_id":ObjectId(details["shipmentID"])}))
572
573         prod = db["products"].find_one({"_id" : ship["pid"]})
574
575         db["categoryEmission"].update_one({"_id" : prod["categoryID"]}, {"$inc" : {"states." + state.lower() : 1}}, upsert = True)
576
577
578     return jsonify({'status' : True})
579
580
581
582
583 @app.route('/detectImage',methods=['POST'])
584 def detect_image():
585     direc = os.getcwd()
586     path = os.path.join(direc,'test.jpg')
587
588     if 'Picture' not in request.files:
589         return "something went wrong 1"
590
591     user_file = request.files['Picture']
592
593
594
595     user_file.save(path)
596
597     with open(path, 'rb') as f:
598         data = f.read()
599         headers = {
600             'Ocp-Apim-Subscription-Key': subscription_key,
601             'Content-type': 'application/octet-stream'
602         }
603         response = requests.post(
604             endpoint, headers=headers, data=data)
605
606         # Print results with confidence score
607         # print("Tags in the remote image: ")
608         # if (len(tags_result_remote.tags) == 0):
609         #     print("No tags detected.")
610         # else:
611         #     for tag in tags_result_remote.tags:
612         #         print("{}' with confidence {:.2f}%".format(tag.name, tag.confidence * 100))
613         # print()
614
615         return jsonify({"object" : response.json()[0]["tags"]}[0]["name"]})
616
617 @app.route('/addInspectionForm',methods=['POST'])
618 def add_inspection_form():
619     """
620     POST request | Add an inspection form
621     json params : ["manufacturer", "inputFields", "inputTypes", "formName", "makerName"]
622     """
623
624     details = request.json
625
626     mname = details["manufacturer"]
627
628     mid = db["manufacturers"].find(
629         { 'text': { '$search': mname } },
630         { 'score': { '$meta': "textScore" } }
631     ).sort([(('score', {'$meta': 'textScore'})[0])["_id"]])
632
633     formName = details["formName"]
634
635     makerName = details["makerName"]
636
637     inputFields = details["inputFields"]
638
639     inputTypes = details["inputTypes"]
640
641     cats = set()
642
643     pids = set(db["manufacturers"].find_one({"_id" : mid})["products"])
644
645     for i in pids:
646
647         pcats = db["products"].find_one({"_id" : i})["category"]
648
649         for c in pcats:
650
651             cats.add(c)
652
653
654     res = db["forms"].insert_one({"manufacturer" : mid, "inputFields" : inputFields, "inputTypes" : inputTypes, "formName" : formName, "makerName" : makerName, "targetCategories" : list(cats)})
655
656     d = {
657         "template_ids": [
658             "28b41595c417b0023384b0c04f23e38f169d4fb3"
659         ],
660         "subject": "E-Signature for audit confirmation",
661         "message": "Please sign this form so that we can confirm your submission of the audit format.\nThis shall help us take a step towards a better tomorrow.\nThanks & Regards,\nTeam EcoTag",
662         "signers": [
663             {
664                 "role": "Auditor",
665                 "name": makerName,
666                 "email_address": "lms19is05l@gmail.com"
667             }
668         ],
669         "custom_fields": [
670             {
671                 "name": "Factory",
672                 "value": mname,
673                 "editor": "Auditor",
674                 "required": True
675             },
676             {
677                 "name": "Name1",
678                 "value": makerName,
679                 "editor": "Auditor",
680                 "required": True
681             },
682             {
683                 "name": "Name2",
684                 "value": makerName,
685                 "editor": "Auditor",
686                 "required": True
687             }
688         ],
689         "signing_options": {
690             "draw": True,
691             "type": True,
692             "upload": True,
693             "phone": False,
694             "default_type": "draw"
695         },
696         "test_mode": True
697     }
698     headers = {'Content-type': 'application/json'}
699
700     resp = requests.post("https://096f2e433b09835cb775448649e17799bb699a49a682ec660218bbc059c8d625@api.hellosign.com/v3/signature_request/send_with_template", data = json.dumps(d), headers=headers)
701
702
703     return json.dumps({"status" : res.acknowledged })
704
705
706
707 @app.route('/getInspectionForms')
708 def get_inspection_forms():
709
710     return dumps(list(db["forms"].find()))

```

```

711
712 @app.route('/getStates')
713 def get_states():
714     details = request.args
715
716     return dumps(db["categoryEmission"].find_one({"_id" : details["categoryID"]}))
717
718
719
720
721 @app.route('/reverseLoc')
722 def reverse_loc():
723
724     details = request.args
725
726     return dumps({"state" : geocoder.osm([details["lat"], details["long"]], method='reverse').state))
727
728
729 if __name__ == "__main__":
730     app.run("0.0.0.0",80,True)

```

```

1 // Authored By: Jayant Sogikar
2 #include <omp.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(){
7
8     // Init a parallel section
9     #pragma omp parallel
10    {
11        // Finding thread number
12        int t = omp_get_thread_num();
13
14        // output
15        printf("Hello world from thread number %d \n",t);
16    }
17
18    return 0;
19 }
20
21 // gcc hw.c -fopenmp
22 // ./a.out

```

```

1 // Authored By: Jayant Sogikar
2 from collections import defaultdict
3
4
5 def solution(map):
6
7     fromA = defaultdict(lambda : float('inf'))
8     fromB = defaultdict(lambda : float('inf'))
9
10    r, c = len(map) , len(map[0])
11    fromA[(0,0)] = 1
12
13    q = [(0,0,1)]
14
15    while q:
16        x, y, score = q.pop(0)
17
18
19
20        for (i,j) in [(0,1), (1,0),(-1,0), (0, -1)]:
21            if 0 <= x + i < r and 0 <= y + j < c and fromA[(x + i,y + j)] == float('inf'):
22
23                fromA[(x + i,y + j)] = score + 1
24
25                if map[x + i][y + j] == 0 : q.append((x + i, y + j, score + 1))
26
27
28    q = [(r - 1, c - 1, 1)]
29    fromB[(r-1,c-1)] = 1
30
31    while q:
32        x, y, score = q.pop(0)
33
34
35
36        for (i,j) in [(0,1), (1,0),(-1,0), (0, -1)]:
37            if 0 <= x + i < r and 0 <= y + j < c and fromB[(x + i,y + j)] == float('inf'):
38
39                fromB[(x + i,y + j)] = score + 1
40
41                if map[x + i][y + j] == 0 : q.append((x + i, y + j, score + 1))
42
43
44    minVal = float('inf')
45
46
47    for i in range(r):
48        for j in range(c):
49            # print(fromA[(i,j)], fromB[(i,j)], fromA[(i,j)] + fromB[(i,j)])
50            minVal = min(minVal, fromA[(i,j)] + fromB[(i,j)])
51
52
53    return minVal - 1
54
55
56
57
58
59 print(solution([[0, 1, 1, 0], [0, 0, 0, 1], [1, 1, 0, 0], [1, 1, 1, 0]]))

```