

Solution

Approach #1 Using HashMap [Accepted]

In this approach, we compare every string in `list1` and `list2` by traversing over the whole list for `list2` for every string chosen from `list1`. We make use of a `HashMap` map, which contains elements of the form `(sum : list2sum)`. Here, `sum` refers to the sum of indices of matching elements and `list2sum` refers to the list of existing strings whose index `sum` equals `sum`.

Thus, while doing the comparisons, whenever a match between strings at i^{th} index of `list1` and j^{th} index of `list2` is found, we make an entry in the map corresponding to the sum `sum = j`. This entry isn't already present. If an entry with this sum already exists, we need to keep a track of all the strings which lead to the same value sum. Thus, we append the current string to the list of strings corresponding to sum `sum = j`.

At the end, we traverse over the keys of the map and find out the list of strings corresponding to the key representing the minimum sum.

```
1 class Solution:
2     def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
3
4         d1, d2 = {}, {}
5         output = []
6
7         for i, v in enumerate(list1):
8             d1.update({v: i})
9
10        for i, v in enumerate(list2):
11            d2.update({v: i})
12
13        print(d1)
14        set1 = set(d1)
15        set2 = set(d2)
16
17        for key in set1.intersection(set2):
18            d1.update({key: (d1[key] + d2[key])})
19
20        for key, v in d1.items():
21            if v == min(d1.values()):
22                output.append(key)
23
24        return output
25
26
```

Complexity Analysis

- Time complexity: $O(l_1 \times l_2 + x)$. Every item of `list1` is compared with all the items of `list2`, l_1 and l_2 are the lengths of `list1` and `list2` respectively. And x refers to average string length.
- Space complexity: $O(l_2 + x)$ is worst case of items of `list1` and `list2` are same. In that case, `HashMap` also grows upto $l_2 + x$, where x refers to average string length.

Approach #2 Without Using HashMap [Accepted]

Algorithm

Another method could be to traverse over the vertical `sums`(index sum) values and determine if any such string exists in `list1` and `list2` such that the sum of its indices in the two lists equals `sum`.

Now, we know that the value of index `sum`, could range from 0 to `m + n - 1`. Here, `m` and `n` refer to the length of lists `list1` and `list2` respectively. Thus, we choose every value of `sum` in ascending order. For every `sum` chosen, we iterate over `list1`. Suppose, currently the string at i^{th} index in `list1` is being considered. Now, in order for the index `sum` to be the one corresponding to matching strings in `list1` and `list2`, the string at index `j` in `list2` should match the string at index `i` in `list1`, such that `sum = i + j`.

On a similar note, the string at index `j` in `list2` should be equal to the string at index `i` in `list1`, such that `j = sum - i`. Thus, for a particular `sum` and if (from `list1`), we can directly determine that we need to check the element at index `j = sum - i` in `list2`, instead of traversing over the whole `list2`.

Using hash character comparisons, iterate over all the indices of `list1` for every `sum` value chosen. Whenever a match occurs between `list1` and `list2`, we put the matching string in a list `res`.

We do the same process of checking the strings for the values of `sum` in ascending order. After completing every iteration over `list1` for a particular `sum`, we check if the list is empty or not. If it is empty, we need to continue the process with the next `sum` value considered. If not, the current `res` given is required list with minimum index sum. This is because we are already considering the index sum values in ascending order. So, the first list to be found is the required resultant list.

The following example depicts the process:

list1

Shogun

Taberna Express

Burger King

KFC

list2

KFC

Taberna Express

Shogun

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

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

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

Complexity Analysis

- Time complexity: $O(l_1(l_1 + l_2)^2 + x)$. There are two nested loops upto $l_1 + l_2$ and string comparison takes x time. Here, x refers to the average string length.
- Space complexity: $O(x + x)$. `res` list is used to store the result. Assuming x is the length of `res`.

Approach #3 Using Hash Map (linear) [Accepted]

We make use of a `HashMap` to save the `sum` process in a different way in this approach. Firstly, we traverse over the whole `list1` and create an entry for each element of `list1` in a `HashMap` map of the form `(list1[i] : i)`. Here, `i` refers to the index of the i^{th} element, and `list1[i]` is the i^{th} element in `list1`. Thus, we create a mapping from the elements of `list1` to their indices.

Now, we traverse over `list2` for every element `list2[i]`, if `list2` occurred, we check if the same element already exists as a key in the map. If so, it means that the element exists in both `list1` and `list2`. Thus, we find out the sum of indices corresponding to this element in the two lists, given by `sum = map.get(list2[i]) + i`. If this sum is lesser than the minimum sum obtained till now, we update the resultant list to be returned, `res`, with the current `list2[i]` as the new entry in `res`.

If the `sum` is equal to the minimum sum obtained till now, we put an extra entry corresponding to the element `list2[i]` in the `res` list.

Below code is inspired by [@tsutsui](#)

```
1 class Solution:
2     def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
3
4         posMap = {}
5         minSum = float('inf')
6         res = []
7
8         for i, v in enumerate(list1):
9             posMap[v] = i
10
11        for i, v in enumerate(list2):
12            if v in posMap:
13                sum = posMap[v] + i
14                if sum < minSum:
15                    minSum = sum
16                    res = [v]
17                elif sum == minSum:
18                    res.append(v)
19
20        return res
21
22
```

Complexity Analysis

- Time complexity: $O(l_1 + l_2)$. Every item of `list2` is checked in a map of `list1`, l_1 and l_2 are the lengths of `list1` and `list2` respectively.
- Space complexity: $O(l_1 + x)$. `HashMap` size grows upto $l_1 + x$, where x refers to average string length.

Report Action Issue

Comments: 40

Best

Most Votes

Oldest

Closest to Thread

Type comment here... (Markdown is supported)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

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</