

python / cpython Public

<> Code Issues 5k+ Pull requests 2.2k Actions Projects Security and q

# Commit bca11ae



miss-islington and serhiy-storchaka authored on Oct 7, 2025 · ✖ 52 / 65 · Verified

[3.10] [gh-139700](#): Check consistency of the zip64 end of central directory record ([GH-139702](#)) ([GH-139708](#)) ([#139714](#))

Support records with "zip64 extensible data" if there are no bytes prepended to the ZIP file.

(cherry picked from commit [333d4a6](#))  
(cherry picked from commit [162997b](#))

Co-authored-by: Serhiy Storchaka <storchaka@gmail.com>

3.10 ([#139714](#)) · v3.10.20 v3.10.19

1 parent [7317e0b](#) commit bca11ae

**3 files changed** +113 -23 ●●●●● ●

Top

- ✓ Lib
  - ✓ test
    - test\_zipfile.py
    - zipfile.py
- ✓ Misc/NEWS.d/next/Security
  - 2025-10-07-19-31-34.gh-issue-139700.vNHU1O.rst

```

Lib/test/test_zipfile.py
↑
@@ -862,6 +862,8 @@ def make_zip64_file(

```

```

862 862         self, file_size_64_set=False, file_size_extra=False,
863 863         compress_size_64_set=False, compress_size_extra=False,
864 864         header_offset_64_set=False, header_offset_extra=False,
865 865         +         extensible_data=b'',
866 866         +         end_of_central_dir_size=None, offset_to_end_of_central_dir=None,
865 867     ):
866 868         """Generate bytes sequence for a zip with (incomplete) zip64 data.
867 869
@@ -915,6 +917,12 @@ def make_zip64_file(
915 917
916 918         central_dir_size = struct.pack('<Q', 58 + 8 *
len(central_zip64_fields))
917 919         offset_to_central_dir = struct.pack('<Q', 50 + 8 *
len(local_zip64_fields))
920 920         +         if end_of_central_dir_size is None:
921 921         +             end_of_central_dir_size = 44 + len(extensible_data)
922 922         +         if offset_to_end_of_central_dir is None:
923 923         +             offset_to_end_of_central_dir = (108
924 924         +                 + 8 * len(local_zip64_fields)
925 925         +                 + 8 * len(central_zip64_fields))
918 926
919 927         local_extra_length = struct.pack("<H", 4 + 8 *
len(local_zip64_fields))
920 928         central_extra_length = struct.pack("<H", 4 + 8 *
len(central_zip64_fields))
@@ -943,14 +951,17 @@ def make_zip64_file(
943 951         + filename
944 952         + central_extra
945 953         # Zip64 end of central directory
946 946         -         + b"PK\x06\x06,\x00\x00\x00\x00\x00\x00\x00-\x00-"
947 947         -         + b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00"
954 954         +         + b"PK\x06\x06"
955 955         +         + struct.pack('<Q', end_of_central_dir_size)
956 956         +         + b"- \x00-
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00"
948 957         + b"\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00"
949 958         + central_dir_size
950 959         + offset_to_central_dir

```

960	+	+ extensible_data
951	961	# Zip64 end of central directory locator
952	-	+ b"PK\x06\x07\x00\x00\x00\x001\x00\x00\x00\x00\x00\x01"
953	-	+ b"\x00\x00\x00"
962	+	+ b"PK\x06\x07\x00\x00\x00\x00"
963	+	+ struct.pack('<Q', offset_to_end_of_central_dir)
964	+	+ b"\x01\x00\x00\x00"
954	965	# end of central directory
955	966	+ b"PK\x05\x06\x00\x00\x00\x00\x01\x00\x01\x00:\x00\x00\x002\x00"
956	967	+ b"\x00\x00\x00\x00"
⋮ ↓ ↑ ⋮		@@ -981,6 +992,7 @@ def test_bad_zip64_extra(self):
981	992	with self.assertRaises(zipfile.BadZipFile) as e:
982	993	zipfile.ZipFile(io.BytesIO(missing_file_size_extra))
983	994	self.assertIn('file size', str(e.exception).lower())
995	+	self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_file_size_extra)))
984	996	
985	997	# zip64 file size present, zip64 compress size present, one field in
986	998	# extra, expecting two, equals missing compress size.
⋮ ↕ ⋮		@@ -992,6 +1004,7 @@ def test_bad_zip64_extra(self):
992	1004	with self.assertRaises(zipfile.BadZipFile) as e:
993	1005	zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
994	1006	self.assertIn('compress size', str(e.exception).lower())
1007	+	self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
995	1008	
996	1009	# zip64 compress size present, no fields in extra, expecting one,
997	1010	# equals missing compress size.
⋮ ↕ ⋮		@@ -1001,6 +1014,7 @@ def test_bad_zip64_extra(self):
1001	1014	with self.assertRaises(zipfile.BadZipFile) as e:
1002	1015	zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
1003	1016	self.assertIn('compress size', str(e.exception).lower())
1017	+	self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
1004	1018	
1005	1019	# zip64 file size present, zip64 compress size present, zip64 header
1006	1020	# offset present, two fields in extra, expecting three, equals
		missing
⋮ ↕ ⋮		@@ -1015,6 +1029,7 @@ def test_bad_zip64_extra(self):

```

1015 1029         with self.assertRaises(zipfile.BadZipFile) as e:
1016 1030             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1017 1031         self.assertIn('header offset', str(e.exception).lower())
1032 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1018 1033
1019 1034         # zip64 compress size present, zip64 header offset present, one field
1020 1035         # in extra, expecting two, equals missing header offset
@@ -1027,6 +1042,7 @@ def test_bad_zip64_extra(self):
1027 1042         with self.assertRaises(zipfile.BadZipFile) as e:
1028 1043             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1029 1044         self.assertIn('header offset', str(e.exception).lower())
1045 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1030 1046
1031 1047         # zip64 file size present, zip64 header offset present, one field in
1032 1048         # extra, expecting two, equals missing header offset
@@ -1039,6 +1055,7 @@ def test_bad_zip64_extra(self):
1039 1055         with self.assertRaises(zipfile.BadZipFile) as e:
1040 1056             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1041 1057         self.assertIn('header offset', str(e.exception).lower())
1058 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1042 1059
1043 1060         # zip64 header offset present, no fields in extra, expecting one,
1044 1061         # equals missing header offset
@@ -1050,6 +1067,63 @@ def test_bad_zip64_extra(self):
1050 1067         with self.assertRaises(zipfile.BadZipFile) as e:
1051 1068             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1052 1069         self.assertIn('header offset', str(e.exception).lower())
1070 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1071 +
1072 +         def test_bad_zip64_end_of_central_dir(self):
1073 +             zipdata = self.make_zip64_file(end_of_central_dir_size=0)
1074 +             with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1075 +                 zipfile.ZipFile(io.BytesIO(zipdata))
1076 +             self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1077 +
1078 +             zipdata = self.make_zip64_file(end_of_central_dir_size=100)

```

```
1079 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1080 +             zipfile.ZipFile(io.BytesIO(zipdata))
1081 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1082 +
1083 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=0)
1084 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1085 +             zipfile.ZipFile(io.BytesIO(zipdata))
1086 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1087 +
1088 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=1000)
1089 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*locator'):
1090 +             zipfile.ZipFile(io.BytesIO(zipdata))
1091 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1092 +
1093 +     def test_zip64_end_of_central_dir_record_not_found(self):
1094 +         zipdata = self.make_zip64_file()
1095 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1096 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1097 +             zipfile.ZipFile(io.BytesIO(zipdata))
1098 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1099 +
1100 +         zipdata = self.make_zip64_file(
1101 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1102 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1103 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1104 +             zipfile.ZipFile(io.BytesIO(zipdata))
1105 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1106 +
1107 +     def test_zip64_extensible_data(self):
1108 +         # These values are what is set in the make_zip64_file method.
1109 +         expected_file_size = 8
1110 +         expected_compress_size = 8
1111 +         expected_header_offset = 0
1112 +         expected_content = b"test1234"
1113 +
1114 +         zipdata = self.make_zip64_file(
1115 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1116 +         with zipfile.ZipFile(io.BytesIO(zipdata)) as zf:
1117 +             zinfo = zf.infolist()[0]
1118 +             self.assertEqual(zinfo.file_size, expected_file_size)
```

```

1119 +         self.assertEqual(zinfo.compress_size, expected_compress_size)
1120 +         self.assertEqual(zinfo.header_offset, expected_header_offset)
1121 +         self.assertEqual(zf.read(zinfo), expected_content)
1122 +         self.assertTrue(zipfile.is_zipfile(io.BytesIO(zipdata)))
1123 +
1124 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1125 +             zipfile.ZipFile(io.BytesIO(b'prepended' + zipdata))
1126 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(b'prepended' +
zipdata)))

```

1053 1127

1054 1128 def test\_generated\_valid\_zip64\_extra(self):

1055 1129 # These values are what is set in the make\_zip64\_file method.



Lib/zipfile.py



@@ -210,41 +210,57 @@ def is\_zipfile(filename):

```

210 210         else:
211 211             with open(filename, "rb") as fp:
212 212                 result = _check_zipfile(fp)
213 213 -         except OSError:
213 213 +         except (OSError, BadZipFile):
214 214             pass
215 215             return result
216 216
217 217     def _EndRecData64(fpin, offset, endrec):
218 218         """
219 219         Read the ZIP64 end-of-archive records and use that to update endrec
220 220         """
221 221 -         try:
222 222 -             fpin.seek(offset - sizeEndCentDir64Locator, 2)
223 223 -         except OSError:
224 224 -             # If the seek fails, the file is not large enough to contain a ZIP64
221 221 +         offset -= sizeEndCentDir64Locator
222 222 +         if offset < 0:
223 223 +             # The file is not large enough to contain a ZIP64
225 224             # end-of-archive record, so just return the end record we were given.
226 225             return endrec
227 227 -
226 226 +         fpin.seek(offset)
228 227         data = fpin.read(sizeEndCentDir64Locator)

```

```

229 228         if len(data) != sizeEndCentDir64Locator:
230 -         return endrec
229 +         raise OSError("Unknown I/O error")
231 230         sig, diskno, reloff, disks = struct.unpack(structEndArchive64Locator,
data)
232 231         if sig != stringEndArchive64Locator:
233 232             return endrec
234 233
235 234         if diskno != 0 or disks > 1:
236 235             raise BadZipFile("zipfiles that span multiple disks are not
supported")
237 236
238 - # Assume no 'zip64 extensible data'
239 - fpin.seek(offset - sizeEndCentDir64Locator - sizeEndCentDir64, 2)
237 + offset -= sizeEndCentDir64
238 + if reloff > offset:
239 +     raise BadZipFile("Corrupt zip64 end of central directory locator")
240 + # First, check the assumption that there is no prepended data.
241 + fpin.seek(reloff)
242 + extrasz = offset - reloff
240 243 data = fpin.read(sizeEndCentDir64)
241 244 if len(data) != sizeEndCentDir64:
242 - return endrec
245 + raise OSError("Unknown I/O error")
246 + if not data.startswith(stringEndArchive64) and reloff != offset:
247 +     # Since we already have seen the Zip64 EOCD Locator, it's
248 +     # possible we got here because there is prepended data.
249 +     # Assume no 'zip64 extensible data'
250 +     fpin.seek(offset)
251 +     extrasz = 0
252 +     data = fpin.read(sizeEndCentDir64)
253 +     if len(data) != sizeEndCentDir64:
254 +         raise OSError("Unknown I/O error")
255 +     if not data.startswith(stringEndArchive64):
256 +         raise BadZipFile("Zip64 end of central directory record not found")
257 +
243 258 sig, sz, create_version, read_version, disk_num, disk_dir, \
244 259     dircount, dircount2, dirsiz, diroffset = \
245 260     struct.unpack(structEndArchive64, data)
246 - if sig != stringEndArchive64:

```

247	-	<code>return endrec</code>
261	+	<code>if (diroffset + dirsize != reloff or</code>
262	+	<code>sz + 12 != sizeEndCentDir64 + extrasz):</code>
263	+	<code>raise BadZipFile("Corrupt zip64 end of central directory record")</code>
248	264	
249	265	<code># Update the original endrec using data from the ZIP64 record</code>
250	266	<code>endrec[_ECD_SIGNATURE] = sig</code>
↕		<code>@@ -254,6 +270,7 @@ def _EndRecData64(fpin, offset, endrec):</code>
254	270	<code>endrec[_ECD_ENTRIES_TOTAL] = dircount2</code>
255	271	<code>endrec[_ECD_SIZE] = dirsize</code>
256	272	<code>endrec[_ECD_OFFSET] = diroffset</code>
273	+	<code>endrec[_ECD_LOCATION] = offset - extrasz</code>
257	274	<code>return endrec</code>
258	275	
259	276	
↓		<code>@@ -287,7 +304,7 @@ def _EndRecData(fpin):</code>
↑		
287	304	<code>endrec.append(filesize - sizeEndCentDir)</code>
288	305	
289	306	<code># Try to read the "Zip64 end of central directory" structure</code>
290	-	<code>return _EndRecData64(fpin, -sizeEndCentDir, endrec)</code>
307	+	<code>return _EndRecData64(fpin, filesize - sizeEndCentDir, endrec)</code>
291	308	
292	309	<code># Either this is not a ZIP file, or it is a ZIP file with an archive</code>
293	310	<code># comment. Search the end of the file for the "end of central directory"</code>
↕		<code>@@ -311,8 +328,7 @@ def _EndRecData(fpin):</code>
311	328	<code>endrec.append(maxCommentStart + start)</code>
312	329	
313	330	<code># Try to read the "Zip64 end of central directory" structure</code>
314	-	<code>return _EndRecData64(fpin, maxCommentStart + start - filesize,</code>
315	-	<code>endrec)</code>
331	+	<code>return _EndRecData64(fpin, maxCommentStart + start, endrec)</code>
316	332	
317	333	<code># Unable to find a valid end of central directory structure</code>
318	334	<code>return None</code>
↓		<code>@@ -1345,9 +1361,6 @@ def _RealGetContents(self):</code>
↑		
1345	1361	
1346	1362	<code># "concat" is zero, unless zip was concatenated to another file</code>
1347	1363	<code>concat = endrec[_ECD_LOCATION] - size_cd - offset_cd</code>

```

1348 -         if endrec[_ECD_SIGNATURE] == stringEndArchive64:
1349 -             # If Zip64 extension structures are present, account for them
1350 -             concat -= (sizeEndCentDir64 + sizeEndCentDir64Locator)
1351 1364
1352 1365         if self.debug > 2:
1353 1366             inferred = concat + offset_cd
1354 1367
1355 1368     @@ -1928,7 +1941,7 @@ def _write_end_record(self):
1356 1369         " would require ZIP64 extensions")
1357 1370         zip64endrec = struct.pack(
1358 1371             structEndArchive64, stringEndArchive64,
1359 1372             44, 45, 45, 0, 0, centDirCount, centDirCount,
1360 1373             sizeEndCentDir64 - 12, 45, 45, 0, 0, centDirCount,
1361 1374             centDirCount,
1362 1375             centDirSize, centDirOffset)
1363 1376         self.fp.write(zip64endrec)
1364 1377
1365 1378
1366 1379
1367 1380
1368 1381
1369 1382
1370 1383
1371 1384
1372 1385
1373 1386
1374 1387
1375 1388
1376 1389
1377 1390
1378 1391
1379 1392
1380 1393
1381 1394
1382 1395
1383 1396
1384 1397
1385 1398
1386 1399
1387 1400
1388 1401
1389 1402
1390 1403
1391 1404
1392 1405
1393 1406
1394 1407
1395 1408
1396 1409
1397 1410
1398 1411
1399 1412
1400 1413
1401 1414
1402 1415
1403 1416
1404 1417
1405 1418
1406 1419
1407 1420
1408 1421
1409 1422
1410 1423
1411 1424
1412 1425
1413 1426
1414 1427
1415 1428
1416 1429
1417 1430
1418 1431
1419 1432
1420 1433
1421 1434
1422 1435
1423 1436
1424 1437
1425 1438
1426 1439
1427 1440
1428 1441
1429 1442
1430 1443
1431 1444
1432 1445
1433 1446
1434 1447
1435 1448
1436 1449
1437 1450
1438 1451
1439 1452
1440 1453
1441 1454
1442 1455
1443 1456
1444 1457
1445 1458
1446 1459
1447 1460
1448 1461
1449 1462
1450 1463
1451 1464
1452 1465
1453 1466
1454 1467
1455 1468
1456 1469
1457 1470
1458 1471
1459 1472
1460 1473
1461 1474
1462 1475
1463 1476
1464 1477
1465 1478
1466 1479
1467 1480
1468 1481
1469 1482
1470 1483
1471 1484
1472 1485
1473 1486
1474 1487
1475 1488
1476 1489
1477 1490
1478 1491
1479 1492
1480 1493
1481 1494
1482 1495
1483 1496
1484 1497
1485 1498
1486 1499
1487 1500
1488 1501
1489 1502
1490 1503
1491 1504
1492 1505
1493 1506
1494 1507
1495 1508
1496 1509
1497 1510
1498 1511
1499 1512
1500 1513
1501 1514
1502 1515
1503 1516
1504 1517
1505 1518
1506 1519
1507 1520
1508 1521
1509 1522
1510 1523
1511 1524
1512 1525
1513 1526
1514 1527
1515 1528
1516 1529
1517 1530
1518 1531
1519 1532
1520 1533
1521 1534
1522 1535
1523 1536
1524 1537
1525 1538
1526 1539
1527 1540
1528 1541
1529 1542
1530 1543
1531 1544
1532 1545
1533 1546
1534 1547
1535 1548
1536 1549
1537 1550
1538 1551
1539 1552
1540 1553
1541 1554
1542 1555
1543 1556
1544 1557
1545 1558
1546 1559
1547 1560
1548 1561
1549 1562
1550 1563
1551 1564
1552 1565
1553 1566
1554 1567
1555 1568
1556 1569
1557 1570
1558 1571
1559 1572
1560 1573
1561 1574
1562 1575
1563 1576
1564 1577
1565 1578
1566 1579
1567 1580
1568 1581
1569 1582
1570 1583
1571 1584
1572 1585
1573 1586
1574 1587
1575 1588
1576 1589
1577 1590
1578 1591
1579 1592
1580 1593
1581 1594
1582 1595
1583 1596
1584 1597
1585 1598
1586 1599
1587 1600
1588 1601
1589 1602
1590 1603
1591 1604
1592 1605
1593 1606
1594 1607
1595 1608
1596 1609
1597 1610
1598 1611
1599 1612
1600 1613
1601 1614
1602 1615
1603 1616
1604 1617
1605 1618
1606 1619
1607 1620
1608 1621
1609 1622
1610 1623
1611 1624
1612 1625
1613 1626
1614 1627
1615 1628
1616 1629
1617 1630
1618 1631
1619 1632
1620 1633
1621 1634
1622 1635
1623 1636
1624 1637
1625 1638
1626 1639
1627 1640
1628 1641
1629 1642
1630 1643
1631 1644
1632 1645
1633 1646
1634 1647
1635 1648
1636 1649
1637 1650
1638 1651
1639 1652
1640 1653
1641 1654
1642 1655
1643 1656
1644 1657
1645 1658
1646 1659
1647 1660
1648 1661
1649 1662
1650 1663
1651 1664
1652 1665
1653 1666
1654 1667
1655 1668
1656 1669
1657 1670
1658 1671
1659 1672
1660 1673
1661 1674
1662 1675
1663 1676
1664 1677
1665 1678
1666 1679
1667 1680
1668 1681
1669 1682
1670 1683
1671 1684
1672 1685
1673 1686
1674 1687
1675 1688
1676 1689
1677 1690
1678 1691
1679 1692
1680 1693
1681 1694
1682 1695
1683 1696
1684 1697
1685 1698
1686 1699
1687 1700
1688 1701
1689 1702
1690 1703
1691 1704
1692 1705
1693 1706
1694 1707
1695 1708
1696 1709
1697 1710
1698 1711
1699 1712
1700 1713
1701 1714
1702 1715
1703 1716
1704 1717
1705 1718
1706 1719
1707 1720
1708 1721
1709 1722
1710 1723
1711 1724
1712 1725
1713 1726
1714 1727
1715 1728
1716 1729
1717 1730
1718 1731
1719 1732
1720 1733
1721 1734
1722 1735
1723 1736
1724 1737
1725 1738
1726 1739
1727 1740
1728 1741
1729 1742
1730 1743
1731 1744
1732 1745
1733 1746
1734 1747
1735 1748
1736 1749
1737 1750
1738 1751
1739 1752
1740 1753
1741 1754
1742 1755
1743 1756
1744 1757
1745 1758
1746 1759
1747 1760
1748 1761
1749 1762
1750 1763
1751 1764
1752 1765
1753 1766
1754 1767
1755 1768
1756 1769
1757 1770
1758 1771
1759 1772
1760 1773
1761 1774
1762 1775
1763 1776
1764 1777
1765 1778
1766 1779
1767 1780
1768 1781
1769 1782
1770 1783
1771 1784
1772 1785
1773 1786
1774 1787
1775 1788
1776 1789
1777 1790
1778 1791
1779 1792
1780 1793
1781 1794
1782 1795
1783 1796
1784 1797
1785 1798
1786 1799
1787 1800
1788 1801
1789 1802
1790 1803
1791 1804
1792 1805
1793 1806
1794 1807
1795 1808
1796 1809
1797 1810
1798 1811
1799 1812
1800 1813
1801 1814
1802 1815
1803 1816
1804 1817
1805 1818
1806 1819
1807 1820
1808 1821
1809 1822
1810 1823
1811 1824
1812 1825
1813 1826
1814 1827
1815 1828
1816 1829
1817 1830
1818 1831
1819 1832
1820 1833
1821 1834
1822 1835
1823 1836
1824 1837
1825 1838
1826 1839
1827 1840
1828 1841
1829 1842
1830 1843
1831 1844
1832 1845
1833 1846
1834 1847
1835 1848
1836 1849
1837 1850
1838 1851
1839 1852
1840 1853
1841 1854
1842 1855
1843 1856
1844 1857
1845 1858
1846 1859
1847 1860
1848 1861
1849 1862
1850 1863
1851 1864
1852 1865
1853 1866
1854 1867
1855 1868
1856 1869
1857 1870
1858 1871
1859 1872
1860 1873
1861 1874
1862 1875
1863 1876
1864 1877
1865 1878
1866 1879
1867 1880
1868 1881
1869 1882
1870 1883
1871 1884
1872 1885
1873 1886
1874 1887
1875 1888
1876 1889
1877 1890
1878 1891
1879 1892
1880 1893
1881 1894
1882 1895
1883 1896
1884 1897
1885 1898
1886 1899
1887 1900
1888 1901
1889 1902
1890 1903
1891 1904
1892 1905
1893 1906
1894 1907
1895 1908
1896 1909
1897 1910
1898 1911
1899 1912
1900 1913
1901 1914
1902 1915
1903 1916
1904 1917
1905 1918
1906 1919
1907 1920
1908 1921
1909 1922
1910 1923
1911 1924
1912 1925
1913 1926
1914 1927
1915 1928
1916 1929
1917 1930
1918 1931
1919 1932
1920 1933
1921 1934
1922 1935
1923 1936
1924 1937
1925 1938
1926 1939
1927 1940
1928 1941
1929 1942
1930 1943
1931 1944
1932 1945
1933 1946
1934 1947
1935 1948
1936 1949
1937 1950
1938 1951
1939 1952
1940 1953
1941 1954
1942 1955
1943 1956
1944 1957
1945 1958
1946 1959
1947 1960
1948 1961
1949 1962
1950 1963
1951 1964
1952 1965
1953 1966
1954 1967
1955 1968
1956 1969
1957 1970
1958 1971
1959 1972
1960 1973
1961 1974
1962 1975
1963 1976
1964 1977
1965 1978
1966 1979
1967 1980
1968 1981
1969 1982
1970 1983
1971 1984
1972 1985
1973 1986
1974 1987
1975 1988
1976 1989
1977 1990
1978 1991
1979 1992
1980 1993
1981 1994
1982 1995
1983 1996
1984 1997
1985 1998
1986 1999
1987 2000
1988 2001
1989 2002
1990 2003
1991 2004
1992 2005
1993 2006
1994 2007
1995 2008
1996 2009
1997 2010
1998 2011
1999 2012
2000 2013
2001 2014
2002 2015
2003 2016
2004 2017
2005 2018
2006 2019
2007 2020
2008 2021
2009 2022
2010 2023
2011 2024
2012 2025
2013 2026
2014 2027
2015 2028
2016 2029
2017 2030
2018 2031
2019 2032
2020 2033
2021 2034
2022 2035
2023 2036
2024 2037
2025 2038
2026 2039
2027 2040
2028 2041
2029 2042
2030 2043
2031 2044
2032 2045
2033 2046
2034 2047
2035 2048
2036 2049
2037 2050
2038 2051
2039 2052
2040 2053
2041 2054
2042 2055
2043 2056
2044 2057
2045 2058
2046 2059
2047 2060
2048 2061
2049 2062
2050 2063
2051 2064
2052 2065
2053 2066
2054 2067
2055 2068
2056 2069
2057 2070
2058 2071
2059 2072
2060 2073
2061 2074
2062 2075
2063 2076
2064 2077
2065 2078
2066 2079
2067 2080
2068 2081
2069 2082
2070 2083
2071 2084
2072 2085
2073 2086
2074 2087
2075 2088
2076 2089
2077 2090
2078 2091
2079 2092
2080 2093
2081 2094
2082 2095
2083 2096
2084 2097
2085 2098
2086 2099
2087 2100
2088 2101
2089 2102
2090 2103
2091 2104
2092 2105
2093 2106
2094 2107
2095 2108
2096 2109
2097 2110
2098 2111
2099 2112
2100 2113
2101 2114
2102 2115
2103 2116
2104 2117
2105 2118
2106 2119
2107 2120
2108 2121
2109 2122
2110 2123
2111 2124
2112 2125
2113 2126
2114 2127
2115 2128
2116 2129
2117 2130
2118 2131
2119 2132
2120 2133
2121 2134
2122 2135
2123 2136
2124 2137
2125 2138
2126 2139
2127 2140
2128 2141
2129 2142
2130 2143
2131 2144
2132 2145
2133 2146
2134 2147
2135 2148
2136 2149
2137 2150
2138 2151
2139 2152
2140 2153
2141 2154
2142 2155
2143 2156
2144 2157
2145 2158
2146 2159
2147 2160
2148 2161
2149 2162
2150 2163
2151 2164
2152 2165
2153 2166
2154 2167
2155 2168
2156 2169
2157 2170
2158 2171
2159 2172
2160 2173
2161 2174
2162 2175
2163 2176
2164 2177
2165 2178
2166 2179
2167 2180
2168 2181
2169 2182
2170 2183
2171 2184
2172 2185
2173 2186
2174 2187
2175 2188
2176 2189
2177 2190
2178 2191
2179 2192
2180 2193
2181 2194
2182 2195
2183 2196
2184 2197
2185 2198
2186 2199
2187 2200
2188 2201
2189 2202
2190 2203
2191 2204
2192 2205
2193 2206
2194 2207
2195 2208
2196 2209
2197 2210
2198 2211
2199 2212
2200 2213
2201 2214
2202 2215
2203 2216
2204 2217
2205 2218
2206 2219
2207 2220
2208 2221
2209 2222
2210 2223
2211 2224
2212 2225
2213 2226
2214 2227
2215 2228
2216 2229
2217 2230
2218 2231
2219 2232
2220 2233
2221 2234
2222 2235
2223 2236
2224 2237
2225 2238
2226 2239
2227 2240
2228 2241
2229 2242
2230 2243
2231 2244
2232 2245
2233 2246
2234 2247
2235 2248
2236 2249
2237 2250
2238 2251
2239 2252
2240 2253
2241 2254
2242 2255
2243 2256
2244 2257
2245 2258
2246 2259
2247 2260
2248 2261
2249 2262
2250 2263
2251 2264
2252 2265
2253 2266
2254 2267
2255 2268
2256 2269
2257 2270
2258 2271
2259 2272
2260 2273
2261 2274
2262 2275
2263 2276
2264 2277
2265 2278
2266 2279
2267 2280
2268 2281
2269 2282
2270 2283
2271 2284
2272 2285
2273 2286
2274 2287
2275 2288
2276 2289
2277 2290
2278 2291
2279 2292
2280 2293
2281 2294
2282 2295
2283 2296
2284 2297
2285 2298
2286 2299
2287 2300
2288 2301
2289 2302
2290 2303
2291 2304
2292 2305
2293 2306
2294 2307
2295 2308
2296 2309
2297 2310
2298 2311
2299 2312
2300 2313
2301 2314
2302 2315
2303 2316
2304 2317
2305 2318
2306 2319
2307 2320
2308 2321
2309 2322
2310 2323
2311 2324
2312 2325
2313 2326
2314 2327
2315 2328
2316 2329
2317 2330
2318 2331
2319 2332
2320 2333
2321 2334
2322 2335
2323 2336
2324 2337
2325 2338
2326 2339
2327 2340
2328 2341
2329 2342
2330 2343
2331 2344
2332 2345
2333 2346
2334 2347
2335 2348
2336 2349
2337 2350
2338 2351
2339 2352
2340 2353
2341 2354
2342 2355
2343 2356
2344 2357
2345 2358
2346 2359
2347 2360
2348 2361
2349 2362
2350 2363
2351 2364
2352 2365
2353 2366
2354 2367
2355 2368
2356 2369
2357 2370
2358 2371
2359 2372
2360 2373
2361 2374
2362 2375
2363 2376
2364 2377
2365 2378
2366 2379
2367 2380
2368 2381
2369 2382
2370 2383
2371 2384
2372 2385
2373 2386
2374 2387
2375 2388
2376 2389
2377 2390
2378 2391
2379 2392
2380 2393
2381 2394
2382 2395
2383 2396
2384 2397
2385 2398
2386 2399
2387 2400
2388 2401
2389 2402
2390 2403
2391 2404
2392 2405
2393 2406
2394 2407
2395 2408
2396 2409
2397 2410
2398 2411
2399 2412
2400 2413
2401 2414
2402 2415
2403 2416
2404 2417
2405 2418
2406 2419
2407 2420
2408 2421
2409 2422
2410 2423
2411 2424
2412 2425
2413 2426
2414 2427
2415 2428
2416 2429
2417 2430
2418 2431
2419 2432
2420 2433
2421 2434
2422 2435
2423 2436
2424 2437
2425 2438
2426 2439
2427 2440
2428 2441
2429 2442
2430 2443
2431 2444
2432 2445
2433 2446
2434 2447
2435 2448
2436 2449
2437 2450
2438 2451
2439 2452
2440 2453
2441 2454
2442 2455
2443 2456
2444 2457
2445 2458
2446 2459
2447 2460
2448 2461
2449 2462
2450 2463
2451 2464
2452 2465
2453 2466
2454 2467
2455 2468
2456 2469
2457 2470
2458 2471
2459 2472
2460 2473
2461 2474
2462 2475
2463 2476
2464 2477
2465 2478
2466 2479
2467 2480
2468 2481
2469 2482
2470 2483
2471 2484
2472 2485
2473 2486
2474 2487
2475 2488
2476 2489
2477 2490
2478 2491
2479 2492
2480 2493
2481 2494
2482 2495
2483 2496
2484 2497
2485 2498
2486 2499
2487 2500
2488 2501
2489 2502
2490 2503
2491 2504
2492 2505
2493 2506
2494 2507
2495 2508
2496 2509
2497 2510
2498 2511
2499 2512
2500 2513
2501 2514
2502 2515
2503 2516
2504 2517
2505 2518
2506 2519
2507 2520
2508 2521
2509 2522
2510 2523
2511 2524
2512 2525
2513 2526
2514 2527
2515 2528
2516 2529
2517 2530
2518 2531
2519 2532
2520 2533
2521 2534
2522 2535
2523 2536
2524 2537
2525 2538
2526 2539
2527 2540
2528 2541
2529 2542
2530 2543
2531 2544
2532 2545
2533 2546
2534 2547
2535 2548
2536 2549
2537 2550
2538 2551
2539 2552
2540 2553
2541 2554
2542 2555
2543 2556
2544 2557
2545 2558
2546 2559
2547 2560
2548 2561
2549 2562
2550 2563
2551 2564
2552 25
```