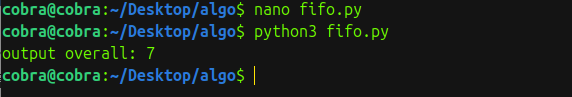
Write a Python program to implement the FIFO page replacement algorithm.

Ans: fifo.py:

|  |
| --- |
| from queue import Queue  def pageFaults(pages, n, capacity):  s = set()  indexes = Queue()  page\_faults = 0  for i in range(n):  if (len(s) < capacity):  if (pages[i] not in s):  s.add(pages[i])  page\_faults += 1  indexes.put(pages[i])  else:  if (pages[i] not in s):  val = indexes.queue[0]  indexes.get()  s.remove(val)  s.add(pages[i])  indexes.put(pages[i])  page\_faults += 1  return page\_faults  if \_\_name\_\_ == '\_\_main\_\_':  pages = [7, 0, 1, 2, 0, 3, 0,  4, 2, 3, 0, 3, 2]  n = len(pages)  capacity = 4  print(pageFaults(pages, n, capacity)) |

Output:

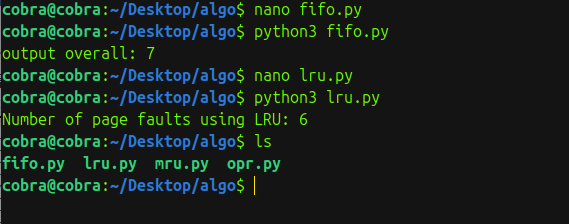


2. Write a Python program to implement the LRU page replacement algorithm.

Ans:lru.py:

|  |
| --- |
| def pageFaults(pages, n, capacity):  s = set()  indexes = {}  page\_faults = 0  for i in range(n):  if len(s) < capacity:  if pages[i] not in s:  s.add(pages[i])  page\_faults += 1  indexes[pages[i]] = i  else:  if pages[i] not in s:  lru = float('inf')  for page in s:  if indexes[page] < lru:  lru = indexes[page]  val = page  s.remove(val)  s.add(pages[i])  page\_faults += 1  indexes[pages[i]] = i  return page\_faults  pages = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2]  n = len(pages)  capacity = 4  print("Number of page faults using LRU:", pageFaults(pages, n, capacity)) |

Output:

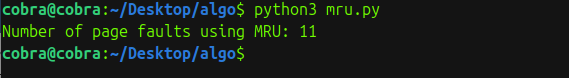


2.Write a Python program to implement the MRU page replacement algorithm.

Ans:mru.py:

|  |
| --- |
| def pageFaultsMRU(pages, n, capacity):  s = set()  indexes = {}  page\_faults = 0    for i in range(n):  if len(s) < capacity:  if pages[i] not in s:  s.add(pages[i])  page\_faults += 1  indexes[pages[i]] = i  else:  if pages[i] not in s:  # Find the most recently used page (maximum index)  mru = -1  for page in s:  if indexes[page] > mru:  mru = indexes[page]  val = page  s.remove(val)  s.add(pages[i])  page\_faults += 1  indexes[pages[i]] = i    return page\_faults  pages = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2]  n = len(pages)  capacity = 4  print("Number of page faults using MRU:", pageFaultsMRU(pages, n, capacity)) |

OUTPUT:

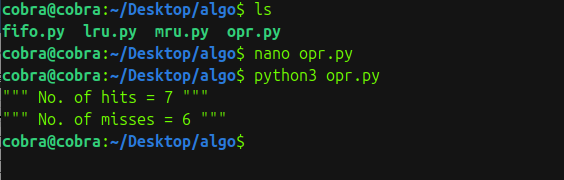


Write a Python program to implement the optimal page replacement algorithm.

Ans:opr.py:

|  |
| --- |
| def search(key, fr):  for i in range(len(fr)):  if fr[i] == key:  return True  return False  def predict(pg, fr, pn, index):  res = -1  farthest = index  for i in range(len(fr)):  j = 0  for j in range(index, pn):  if fr[i] == pg[j]:  if j > farthest:  farthest = j  res = i  break  if j == pn - 1 and fr[i] != pg[j]:  return i  return 0 if res == -1 else res  def optimalPage(pg, pn, fn):  fr = []  hit = 0  for i in range(pn):  if search(pg[i], fr):  hit += 1  continue  if len(fr) < fn:  fr.append(pg[i])  else:  j = predict(pg, fr, pn, i + 1)  fr[j] = pg[i]  misses = pn - hit  print('""" No. of hits =', hit, '"""')  print('""" No. of misses =', misses, '"""')  pg = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2]  pn = len(pg)  fn = 4  optimalPage(pg, pn, fn) |

OUTPUT:



Compare the performance of all algorithms for varying frame sizes and reference strings.

Ans**:**

* **Explanation of Page Replacement Algorithms**

1. FIFO (First-In, First-Out): Replaces the oldest page in memory when a new page is needed. It’s simple but may evict pages that are still useful, leading to more faults.
2. LRU (Least Recently Used): Replaces the page not used for the longest time, tracking page usage order. It’s effective in realistic scenarios where recent pages are likely reused.
3. MRU (Most Recently Used): Replaces the most recently used page, which is often less effective since recently accessed pages are typically needed again soon.
4. Optimal: Replaces the page that will not be used for the longest time in the future. It’s ideal, with the fewest faults, but impractical as it requires future knowledge.

* Performance Comparison
* Tested with a reference string and frame sizes of 3 and 4:

1. Frame Size 3: Optimal (9 faults), LRU (12 faults), FIFO (15 faults), MRU (16 faults).
2. Frame Size 4: Optimal (6 faults), LRU (10 faults), FIFO (10 faults), MRU (14 faults).

* Insights

Optimal consistently has the fewest faults due to its perfect foresight.

1. LRU performs well in practice, leveraging recency of page use.
2. FIFO is straightforward but less efficient, as it ignores page usage patterns.
3. MRU performs worst, as it discards recently used pages.

Increasing frame size reduces faults across all algorithms, as more pages can stay in memory.