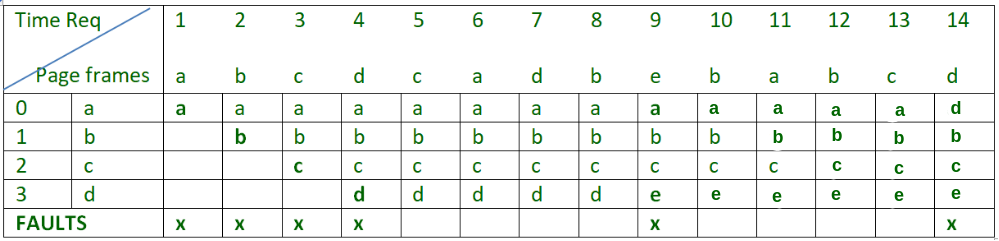
**Page Replacement Algorithms:**

* **Optimal Page replacement –**  
  In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Let us consider page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 and 4 page slots.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>**4 Page faults**  
0 is already there so —> **0 Page fault.**  
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**  
0 is already there so —>**0 Page fault.**.  
4 will takes place of 1 —>**1 Page Fault.**  
Now for the further page reference string —>**0 Page fault** because they are already available in the memory.

Example-2, Let’s have a reference string: a, b, c, d, c, a, d, b, e, b, a, b, c, d and the size of the frame be 4.



There are 6 page faults using optimal algorithm.

Optimal page replacement is perfect, but not possible in practice as operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  main()  {      int fr[5],i,j,k,t[5],p=1,flag=0,page[25],psz,nf,t1,u[5];      clrscr();      printf("enter the number of frames:");      scanf("%d",&nf);      printf("\n enter the page size");      scanf("%d",&psz);        printf("\nenter the page sequence:");      for(i=1; i<=psz; i++)          scanf("%d",&page[i]);        for(i=1; i<=nf; i++)          fr[i]=-1;      for(i=1; i<=psz; i++)      {          if(full(fr,nf)==1)             break;          else          {              flag=0;              for(j=1; j<=nf; j++)              {                 if(page[i]==fr[j])                  {                      flag=1;                      printf("          \t%d:\t",page[i]);                      break;                  }              }              if(flag==0)              {                  fr[p]=page[i];                  printf("          \t%d:\t",page[i]);                  p++;              }                for(j=1; j<=nf; j++)                  printf(" %d  ",fr[j]);              printf("\n");          }      }      p=0;      for(i=0; i<=psz; i++)      {          flag=0;          for(j=1; j<=nf; j++)          {              if(page[i]==fr[j])              {                  flag=1;                  break;              }          }         if(flag==0)          {              p++;              for(j=1; j<=nf; j++)              {                  for(k=i+1; k<=psz; k++)                  {                      if(fr[j]==page[k])                      {                          u[j]=k;                          break;                      }                      else                          u[j]=21;                  }              }              for(j=1; j<=nf; j++)                  t[j]=u[j];              for(j=1; j<=nf; j++)              {                  for(k=j+1; k<=nf; k++)                  {                      if(t[j]<t[k])                      {                          t1=t[j];                          t[j]=t[k];                          t[k]=t1;                      }                  }              }              for(j=1; j<=nf; j++)              {                  if(t[1]==u[j])                  {                      fr[j]=page[i];                      u[j]=i;                  }              }              printf("page fault\t");          }          else              printf("          \t");          printf("%d:\t",page[i]);          for(j=1; j<=nf; j++)              printf(" %d  ",fr[j]);          printf("\n");      }      printf("\ntotal page faults:  %d",p+3);  // getch();  }  int full(int a[],int n)  {      int k;      for(k=1; k<=n; k++)      {          if(a[k]==-1)              return 0;      }      return 1;  } |

 OUTPUT:

enter the number of frames:5

enter the page size2

enter the page sequence:1

2

1: 1 -1 -1 -1 -1

2: 1 2 -1 -1 -1

total page faults: 3

#include<stdio.h>

#define MAX\_PAGE 25 // Define a constant, program can support only this MAX\_PAGE number of pages.

#define MAX\_FRAME 25 // Define a constant, program can support only this MAX\_FRAME number of frames.

/\*\*

\* Function main returns 0 on success.

\*/

int main()

{

// Declare arrays for pages and frames and distance\_from\_future\_occurance

int pages[MAX\_PAGE], frames[MAX\_FRAME], distances[MAX\_PAGE];

// Number of pages, frames, page faults variables

int numberPages, numberFrames, numberPageFaults = 0;

// Other variables

int i, j, temp, flag, found, lastFilledFrame, index, highestDistance;

// Get number of frames from user

printf("Enter number of frames (max limit is %d): ", MAX\_FRAME);

scanf("%d", &numberFrames);

printf("You provided number of frames: %d\n", numberFrames);

// Initialize frames array with -1 values and set lastFilledFrame value to -1

for(i = 0; i < numberFrames; i++) frames[i] = -1;

lastFilledFrame = -1;

// Get pages from user

printf("Enter pages (enter -999 to exit, max number of pages limit is %d):\n", MAX\_PAGE);

numberPages = 0;

for(i = 0; i < MAX\_PAGE; i++)

{

scanf("%d", &temp);

if(temp == -999 || numberPages == MAX\_PAGE) break;

pages[i] = temp;

numberPages++;

}

printf("You provided number of pages: %d\n", numberPages);

// Traverse the sequence of pages according Optimal Page Replacement Algorithm

for(i = 0; i < numberPages; i++) // For every page

{

flag = 0; // Flag to show availability of page in frame

// Get the availability of required page in frame

for(j = 0; j < numberFrames; j++) // For every frame

{

if(frames[j] == pages[i]) // If page found in frame

{

flag = 1; // Set flag to 1, showing that page is available in frame

printf("\t"); // Print tab space instead "FAULT" word

break; // Break the loop

}

}

// If page is not available in frame, replace some page by required page and print "FAULT" word

if(flag == 0)

{

if (lastFilledFrame == numberFrames-1) // If frames fully filled

{

// For every frame containing page, calculate distances to future occurance

for(j = 0; j < numberFrames; j++)

{

for(temp = i + 1; temp < numberPages; temp++) // For every future page

{

distances[j] = 0; // Set this page\_occurances\_in\_future to zero, showing no occurance

if (frames[j] == pages[temp]) // If frame containing page matches to future occuring page

{

distances[j] = temp - i; // Store distance

break;

}

} // For every future page loop ends here

} // Loop for calculating distances ends here

// Choose best candidate index for page replacement in frame, this best candidate is not occuring in future

found = 0;

for(j = 0; j < numberFrames; j++) // For every frame

{

if(distances[j] == 0) // If page's distance value is 0, means if no occurance in future

{ // Or if frame value is -1, means empty frame

index = j; // Set this frame index to index variable

found = 1; // Set to 1, means a page not occuring in future is found

break;

}

}

}

else // If frames has not fully filled, best candidate is blank frame

{

index = ++lastFilledFrame; // Set blank frame's index to index variable

found = 1; // Set to 1, means a suitable frame index found

}

// If still not choosed best candidate, get best candidate that is having highest distance in future

if(found == 0)

{

highestDistance = distances[0];

index = 0;

for(j = 1; j<numberFrames; j++) // For every frame

{

if(highestDistance < distances[j])

{

highestDistance = distances[j];

index = j; // In last, index will contain farest distanced element's index in current frames

}

}

}

// Do the page replacement

frames[index] = pages[i]; // Replace the identified index located page by required page

printf("FAULT\t"); // Print "FAULT"

numberPageFaults++; // Increment number of page faults

}

// Print the pages that are present in current frames

for(j = 0; j < numberFrames; j++)

{

if(frames[j] != -1) printf("%d\t", frames[j]);

}

printf("\n");

}

// Print number of page faults.

printf("Number of page faults = %d\n", numberPageFaults);

return 0;

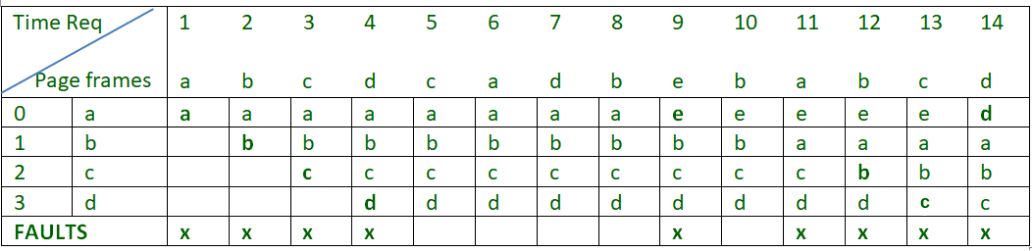
}

* **First In First Out (FIFO) –**  
  This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

For example-1, consider page reference string 1, 3, 0, 3, 5, 6 and 3 page slots.

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**  
when 3 comes, it is already in  memory so —> **0 Page Faults.**  
Then 5 comes, it is not available in  memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**  
Finally 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**

Example-2, Let’s have a reference string: a, b, c, d, c, a, d, b, e, b, a, b, c, d and the size of the frame be 4.



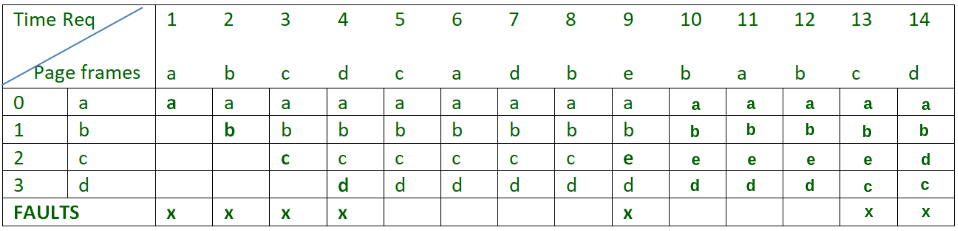
There are 9 page faults using FIFO algorithm.

**Belady’s anomaly –** Belady’s anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm.  For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

* **Least Recently Used –**  
  In this algorithm page will be replaced which is least recently used.

Let say the page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 . Initially we have 4 page slots empty.  
Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>**4 Page faults**  
0 is already their so —> **0 Page fault.**  
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**  
0 is already in memory so —>**0 Page fault**.  
4 will takes place of 1 —>**1 Page Fault**  
Now for the further page reference string —>**0 Page fault** because they are already available in the memory.

Example-2, Let’s have a reference string: a, b, c, d, c, a, d, b, e, b, a, b, c, d and the size of the frame be 4.



There are 7 page faults using LRU algorithm.