Họ và tên : Trần Đình Khang

MSSV: 18520072

Mã Lớp: IT007.K21.KHTN

Bài thực hành Lab06

6.3.3 Câu hỏi chuẩn bị

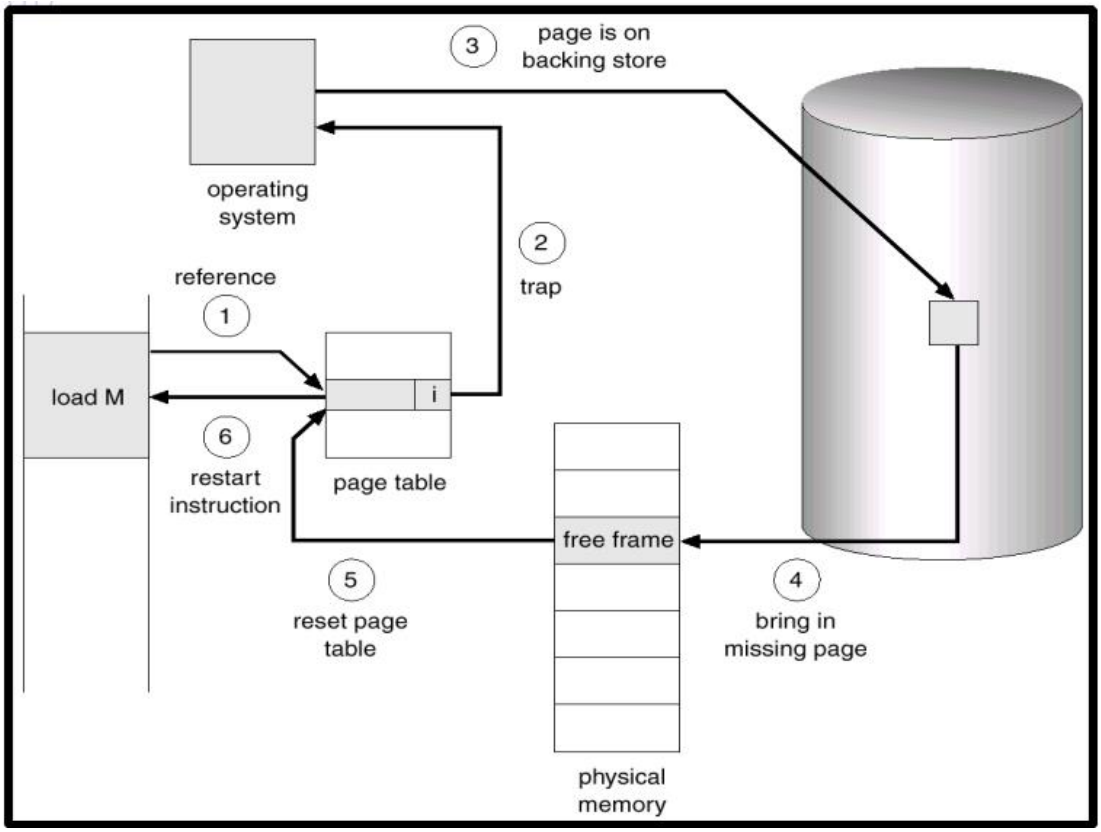
Câu 1:

- Lỗi trang là việc truy xuất không nằm trong bộ nhớ chính.

- Lỗi trang xảy ra khi người dùng truy cập tới một trang không hợp lệ hoặc truy cập tới trang hợp lệ nhưng ở bộ nhớ phụ.

- Cách cài đặt Demand paging:
  - Sử dụng kĩ thuật phân trang kết hợp kĩ thuật swapping
  - Cơ chế phân biệt trang ở bộ nhớ chính và trang trên đĩa (biến boolean)
  - Bảng trang: phản ánh tình trạng 1 trang ở bộ nhớ chính hay phụ.
  - Bộ nhớ phụ: lưu thông tin trang không được nạp vào bộ nhớ chính.

- Lưu đồ mô tả HĐH xử lí lỗi trang



3.
FIFO: 9 Faults, OPT 7 Faults, LRU 9 Faults
Giải thuật tốt nhất: OPT

Câu 2:

- Nếu đặt toàn bộ không gian địa chỉ vào bộ nhớ vật lý thì sẽ bị giới hạn bộ nhớ vật lý (dẫn tới tình trạng "out of memory")
- Chúng ta không cần thiết nạp toàn bộ chương trình vào bộ nhớ vật lý cùng lúc vì tại một thời điểm chỉ có một chỉ thị tiến trình được xử lý.

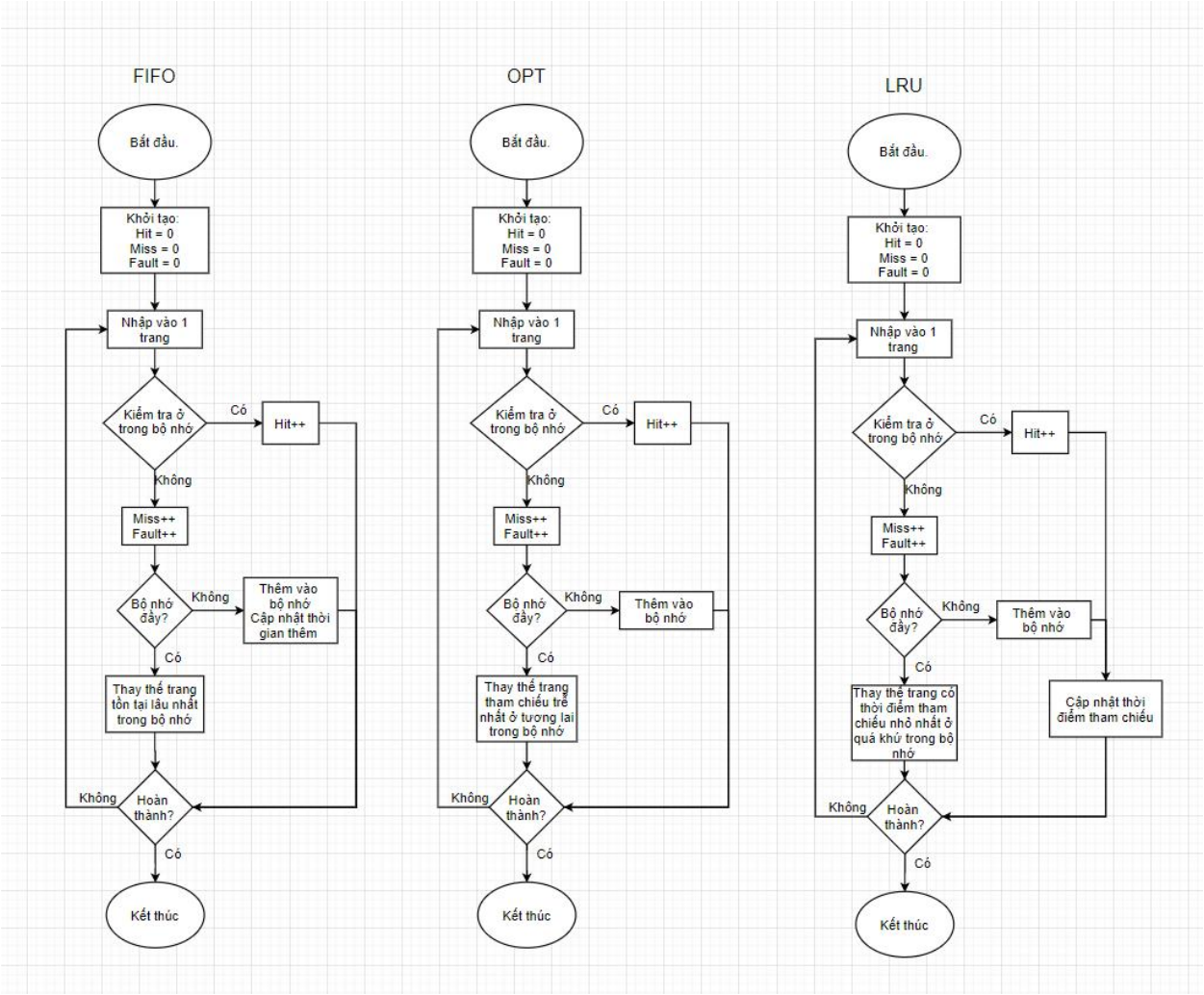
Vì vậy, Cần chiến lược thay thế trang để có thể cung cấp bộ nhớ ảo rất lớn so với bộ nhớ vật lý. (để đạt hiệu quả cao trong thực thi, chạy được chương trình yêu cầu bộ nhớ lớn hơn bộ nhớ vật lý).

Câu 3:

- FIFO: 9 Faults, OPT 7 Faults, LRU 9 Faults

Giải thuật tốt nhất: OPT

- Lưu đồ thuật toán:



6.4 Thực Hành

Câu 1:

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

void Print(int *seq, int **table, char* fault, int n,int f)
{
        int count = 0;
        printf("\n\t--- Page Replacement algorithm ---\n");
        printf("\t-----------------------------------\n\n\t");
        for (int i = 0; i < n; i++)
                printf("%2d ", seq[i]);
        printf("\n\n\t");
        for (int i = 0; i < f; i++)
        {
                for (int j = 0; j < n; j++)
                        if (table[i][j] != -1)
                                printf("%2d ", table[i][j]);
                        else
                                printf("   ");
                printf("\n\n\t");
        }
        for (int i = 0; i < n; i++)
        {
                printf(" %c ", fault[i]);
                if (fault[i] == '*')
                        count++;
        }
        printf("\n\n\tNumber of Page Fault : %d\n\n", count);
}
```

```
int in_MMR(int** table, int f, int i, int page)
{
        for (int j = 0; j < f; j++)
                if (table[j][i] == page || table[j][i]==-1)
                        return j;
        return -1;
}

void FIFO(int* seq, int **table, char *fault, int n, int f)
{
        int first = 0;
        table[0][0] = seq[0];
        fault[0] = '*';
        for (int i = 1; i < n; i++)
        {
                for (int j = 0; j < f; j++)
                        table[j][i] = table[j][i - 1];
                int id = in_MMR(table, f, i, seq[i]);
                if (id != -1)
                {
                        if (table[id][i] == -1) {
                                table[id][i] = seq[i];
                                fault[i] = '*';
                        } else  fault[i] = ' ';
                }
                else
                {
                        fault[i] = '*';
                        table[first][i] = seq[i];
                        first = (first + 1) % f;
                }
        }
}
```

```
void OPT(int* seq, int** table, char* fault, int n, int f)
{
        int* next,k;
        next = (int*)malloc(f * sizeof(int));

        table[0][0] = seq[0];
        fault[0] = '*';
        for (k = 1; k < n; k++)
                if (seq[k] == seq[0])
                        break;
        next[0] = k;
        for (int i = 1; i < n; i++)
        {
                int j;
                for (j = 0; j < f; j++)
                        table[j][i] = table[j][i - 1];

                int id = in_MMR(table, f, i, seq[i]);
                if (id == -1)
                {
                        int choose = 0;
                        for (j = 1; j < f; j++)
                                if (next[choose] < next[j])
                                        choose = j;
                        table[choose][i] = seq[i];
                        for (j = i + 1; j < n; j++)
                                if (seq[j] == seq[i])
                                        break;
                        next[choose] = j;
                        fault[i] = '*';
                }

                else
                {
                        if (table[id][i] == -1) {
                                table[id][i] = seq[i];
                                fault[i] = '*';
                        }
                        else            fault[i] = ' ';
                        j = i + 1;
                        for (j = i + 1; j < n; j++)
                                if (seq[j] == seq[i])
                                        break;
                        next[id] = j;
                }
        }
}
```

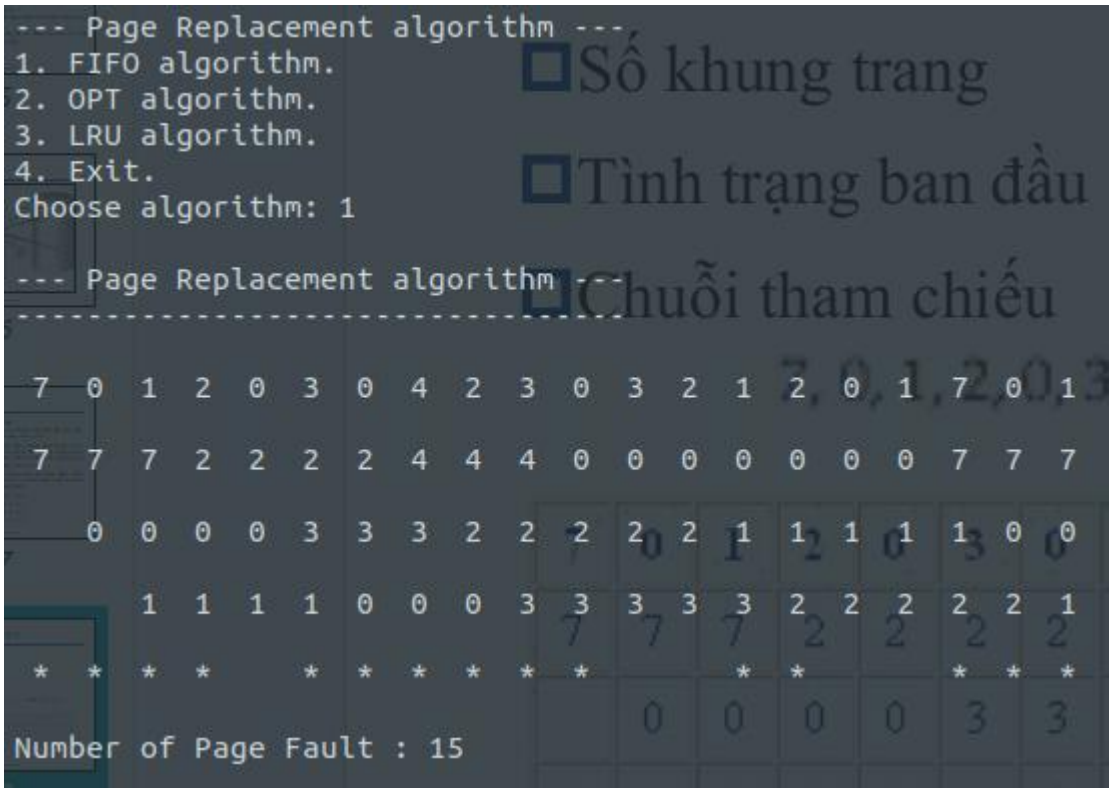
```
void LRU(int* seq, int** table, char* fault, int n, int f)
{
        int* last;
        last = (int*)malloc(f * sizeof(int));

        table[0][0] = seq[0];
        fault[0] = '*';
        last[0] = 0;

        for (int i = 1; i < n; i++)
        {
                for (int j = 0; j < f; j++)
                        table[j][i] = table[j][i - 1];

                int id = in_MMR(table, f, i, seq[i]);
                if (id == -1)
                {
                        fault[i] = '*';
                        int min_ref = 0;
                        for (int j = 1; j < f; j++)
                                if (last[j] < last[min_ref])
                                        min_ref = j;
                        last[min_ref] = i;
                        table[min_ref][i] = seq[i];
                }
                else
                {
                        if (table[id][i] == -1) {
                                table[id][i] = seq[i];
                                fault[i] = '*';
                        } else          fault[i] = ' ';
                        last[id] = i;
                }
        }
}
```

```
int main()
{
        int* DRS, *MIS, n,i, f, algo;
        printf("\t--- Page Replacement algorithm ---\n");
        printf("\t1. 1, 8, 5, 2, 0, 0, 7, 2, 0, 0, 7\n");
        printf("\t2. Manual input sequence:       \t\n");
        printf("\tInput length of sequences: ");
        scanf("%d", &n);
        MIS = (int*)malloc(n * sizeof(int));
        printf("\tInpute sequences: ");
        for (i = 0; i < n; i++)
                scanf("%d", &MIS[i]);
        printf("\t--- Page Replacement algorithm ---\n");
        printf("\tInpute page frames: ");
        scanf("%d", &f);
```

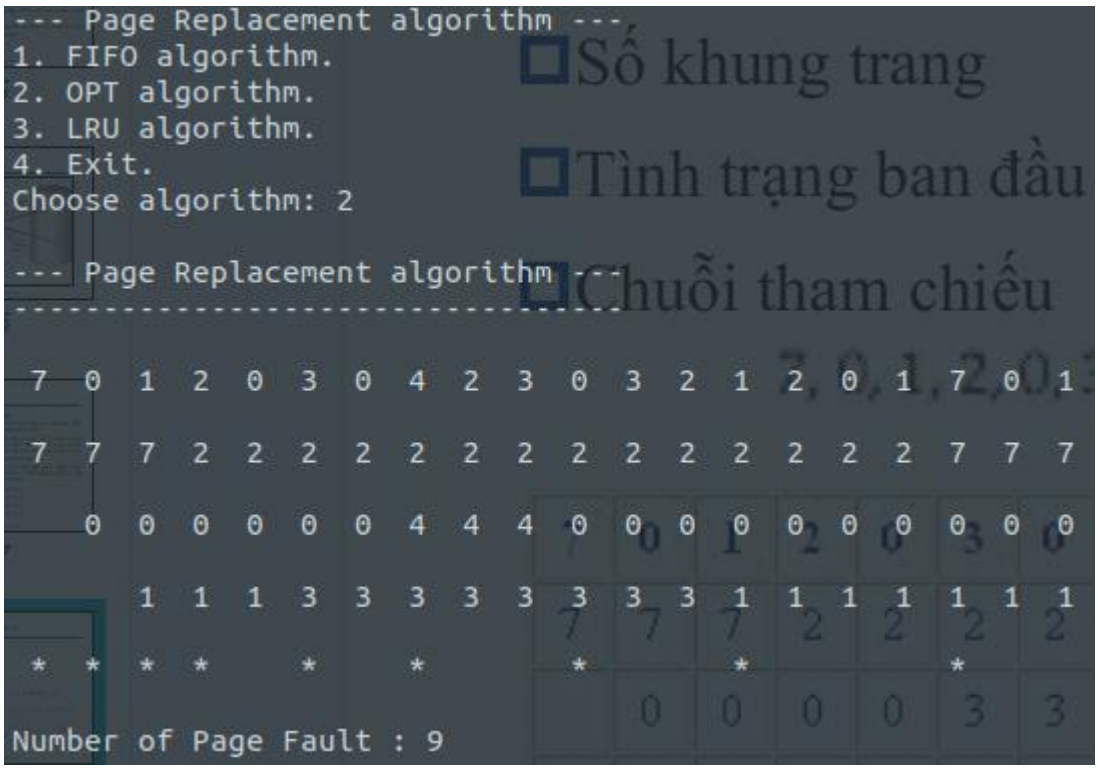
```
        while (1){
                char* fault = (char*)malloc((n + 1) * sizeof(char));
                int** table = (int**)malloc(f * sizeof(int*));
                for (i = 0; i < f; i++) {
                        table[i] = (int*)malloc(n * sizeof(int));
                        for (int j = 0; j < n; j++)
                                table[i][j] = -1;
                }
                printf("\n\t--- Page Replacement algorithm ---\n");
                printf("\t1. FIFO algorithm.\n");
                printf("\t2. OPT algorithm.\n");
                printf("\t3. LRU algorithm.\n");
                printf("\t4. Exit.\n");
                printf("\tChoose algorithm: ");
                scanf("%d", &algo);
                if (algo  == 4) break;
                switch (algo)
                {
                case 1:
                        FIFO(MIS, table, fault, n, f);
                        break;
                case 2:
                        OPT(MIS, table, fault, n, f);
                        break;
                case 3:
                        LRU(MIS, table, fault, n, f);
                        break;
                default:
                        return 0;
                }

                //system("cls");
                Print(MIS, table, fault, n, f);
        }
        return 0;
}
```

Demo:

- Giải thuật FIFO:



- Giải thuật OPT:



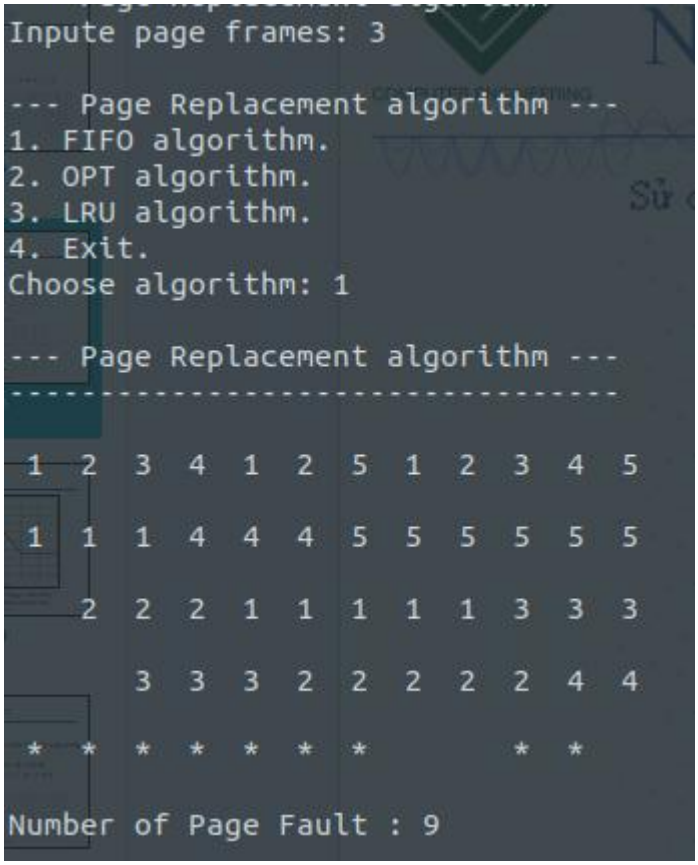
- Giải thuật LRU:

```
--- Page Replacement algorithm ---
1. FIFO algorithm.
2. OPT algorithm.
3. LRU algorithm.
4. Exit.
Choose algorithm: 3
--- Page Replacement algorithm ---
---------------------------------
 7  0  1  2  0  3  0  4  2  3  0  3  2  1  2  0  1  7  0  1
 7  7  7  2  2  2  2  4  4  4  0  0  0  1  1  1  1  1  1  1
    0  0  0  0  0  0  0  0  3  3  3  3  3  3  0  0  0  0  0
       1  1  1  3  3  3  2  2  2  2  2  2  2  2  2  7  7  7
 *  *  *  *     *     *  *  *  *        *     *     *
Number of Page Fault : 12
```

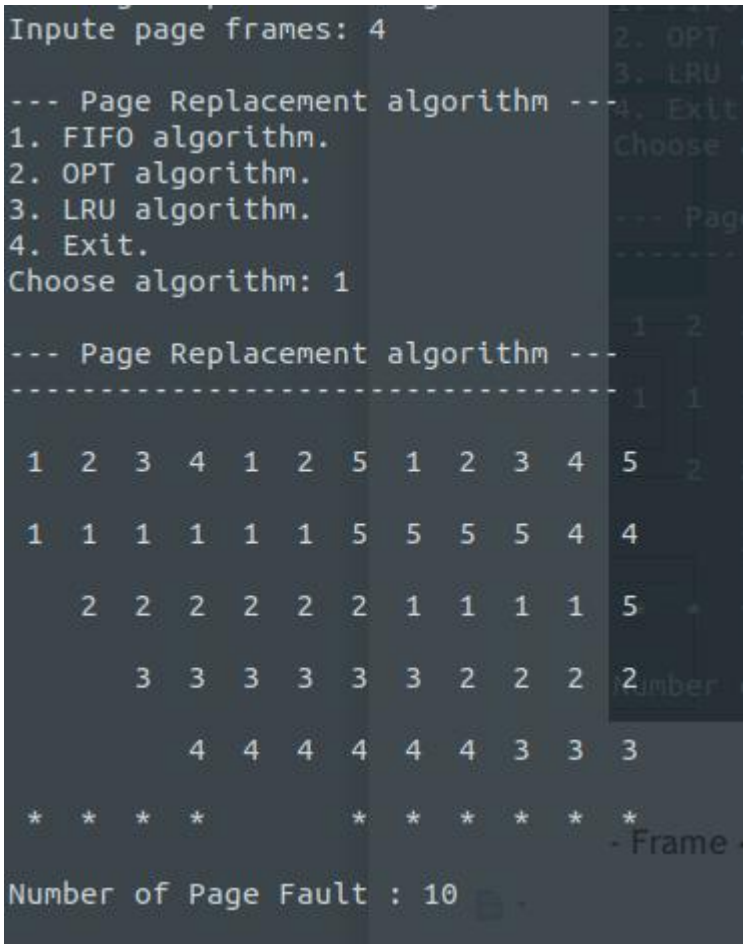
6.5 Bài Tập Ôn Tập

- Nghịch lý Belady là hiện tượng tăng số lượng frame nhưng số lỗi tăng chứ không giảm.

- Frame 3 có 9 lỗi:



- Frame 4 có 10 lỗi:



- Giải thuật tốt nhất: OPT (có 9 lỗi, FIFO có 15 lỗi, LRU có 12 lỗi)

Tuy nhiên giải thuật OPT khó thực hiện nhất bởi vì không thể biết được tương lai sẽ tham chiếu tới page nào.

- Giải thuật LRU thực hiện khá phức tạp do cần HĐH tìm kiếm chi phí trang nhớ RLU.