**Học viện Kỹ thuật Quân sự**

**Ảnh có chứa văn bản, ký hiệu

Mô tả được tạo tự động**

**Báo cáo môn bài tập lớn môn:**

**Lý thuyết hệ điều hành**

**Đề bài**: Viết chương trình mô phỏng các giải thuật thay thế trang: OPT, FIFO, LRU (LFU, NUR,...) Second Chance (Clock). Thực hiện theo từng bước và thực hiện toàn bộ. Input: số khung trang, chuỗi tham khảo trang, yêu cầu thay thế trang…được sinh ngẫu nhiên. Output: sơ đồ các bước thay thế trang, số PAGE FAULTS. Sử dụng Gridbox hoặc đồ họa. Đánh giá độ hiệu quả (so sánh số page fault) của từng giải thuật

|  |  |
| --- | --- |
| Họ và tên | Lê Minh Thành |
| Khoa | KHMT-17 |
| Mã sinh viên | 18150035 |

Tháng 11/2021

Mục lục

[I. Cơ sở lý thuyết 2](#_Toc87877619)

[II. Phát triển chương trình 3](#_Toc87877620)

[2.1 Hàm tính toán và mô phỏng giải thuật OPT: 3](#_Toc87877621)

[2.2 Sử dụng hàm tính toán và mô phỏng giải thuật LRU: 4](#_Toc87877622)

[2.3 Sử dụng hàm tính toán và mô phỏng giải thuật FIFO: 4](#_Toc87877623)

[2.4 Sử dụng hàm tính toán và mô phỏng giải thuật clock: 5](#_Toc87877624)

[2. 5 Quy trình hoạt động 6](#_Toc87877625)

[2.5.1 Mô tả giải thuật OPT 7](#_Toc87877626)

[2.5.2 Mô phỏng giải thuật LRU 8](#_Toc87877627)

[2.5.3 Mô phỏng giải thuật FIFO 9](#_Toc87877628)

[2.5.4 Mô phỏng giải thuật clock 10](#_Toc87877629)

[III. Giao diện người dùng 11](#_Toc87877630)

[3.1 Nhập dữ liệu đầu vào 11](#_Toc87877631)

[3.2. Kết quả 11](#_Toc87877632)

[IV Đánh giá và kết luận 12](#_Toc87877633)

[4.1 Số page-fault 13](#_Toc87877634)

[4.2. Thời gian tính toán 14](#_Toc87877635)

# I. Cơ sở lý thuyết

Thay thế trang nhớ:

1. Xác định vị trí trên đĩa của trang dang cần
2. Tìm 1 frame trống
   1. Nếu có frame trống thì dùng nó
   2. Nếu không thì dùng giải thuật(OPT, LRU, FIFO, clock,…) để chon ra victim page
   3. Ghi victim page lên đĩa; cập nhật page table và frame table tương ứng
3. Đọc trang đang cần vào frame trống(ở bước 2); cập nhật page table và frame tương ứng.

Thuật toán thay thế trang được xây dựng với mục đích:

* Chọn frame của process sẽ được đưa thay thế trang nhớ.
* Đạt được số lượng page-fault nhỏ nhất
* Đánh giá bằng cách thực thi giải thuật đối với một chuỗi tham chiếu bộ nhớ (memory reference string) và xác định số lần xảy ra page-fault

Số lượng frame cấp cho progcess

Trong một số trường hợp nhất định, các ứng dụng truy cập dữ liệu thông qua bộ nhớ ảo của hệ điều hành hoạt động kém hơn nếu hệ điều hành không cung cấp bộ đệm nào cả. Một ví dụ điển hình là cơ sở dữ liệu, cung cấp khả năng quản lý bộ nhớ và đệm I / O của riêng nó. Các ứng dụng như thế này hiểu việc sử dụng bộ nhớ và sử dụng đĩa của chúng tốt hơn hệ điều hành đang triển khai các thuật toán cho mục đích chung. Tuy nhiên, nếu hệ điều hành đang đệm I / O và ứng dụng cũng đang làm như vậy, thì bộ nhớ gấp đôi đang được sử dụng cho một tập hợp I / O.

Kho dữ liệu thường xuyên thực hiện các lần đọc đĩa tuần tự lớn, sau đó là tính toán và ghi. Thuật toán LRU sẽ xóa các trang cũ và giữ lại các trang mới, trong khi ứng dụng có nhiều khả năng đọc các trang cũ hơn các trang mới hơn (vì nó bắt đầu đọc lại tuần tự)

Giải Thuật OPT:Thay thế trang nhớ được tham chiếu trễ nhất trong tương lai

Giải thuật LRU(Least Recently Used):Thay thế trang nhớ không được tham chiếu lâu nhất

Giải thuật FIFO:Khi bộ đệm đầy, trang nhớ cũ nhất sẽ bị thay thế. Dùng con trỏ xoay vòng các frame của process

Giải thuật CLOCK: Khi 1 trang được thay, con trỏ chỉ đến frame kế tiếp trong bộ đệm xoay vòng, mỗi frame có 1 use bit, bit này được set =1 khi trang chứa trong frame đã được tham chiếu từ trước

# II. Phát triển chương trình

Được xây dựng trên python với GUI từ thư viện Tkinter có sẵn của Python, package Random để mô phỏng các thuật toán như OPT, LRU, FIFO, clock,…

Với s là chuỗi trang cần thay thế, capacity là số khung trang

## 2.1 Hàm tính toán và mô phỏng giải thuật OPT:

print("\nString|Frame →\t", end='')

for i in range(capacity):

    print(i, end='  ')

print("Fault\n   ↓\n")

st = [None for i in range(capacity)]

for i in range(len(s)):

    if s[i] not in f:

        if len(f) < capacity:

            f.append(s[i])

        else:

            for x in range(len(f)):

                if f[x] not in s[i+1:]:

                    f[x] = s[i]

                    break

                else:

                    st[x] = s[i+1:].index(f[x])

            else:

                f[st.index(max(st))] = s[i]

            # f[st.index(max(st))] = s[i]

        fault += 1

        pf = 'Yes'

    else:

        pf = 'No'

    print("   %d\t\t" % s[i], end='')

    if(pf == 'No'):

        for x in f:

            print(x, end='  ')

    else:

        for y in f:

            if(y != s[i]):

                print(str(y)+'\*', end=' ')

            else:

                print(y, end='  ')

    for x in range(capacity-len(f)):

        print('  ', end=' ')

    print(" %s" % pf)

print('Total Page Faults:', fault)

## 2.2 Sử dụng hàm tính toán và mô phỏng giải thuật LRU:

print("\nString|Frame →\t", end='')

for i in range(capacity):

    print(i, end=' ')

print("Fault\n   ↓\n")

for i in s:

    if i not in f:

        if len(f) < capacity:

            f.append(i)

            st.append(len(f)-1)

        else:

            ind = st.pop(0)

            f[ind] = i

            st.append(ind)

        pf = 'Yes'

        fault += 1

    else:

        st.append(st.pop(st.index(f.index(i))))

        pf = 'No'

    print("   %d\t\t" % i, end='')

    for x in f:

        print(x, end=' ')

    for x in range(capacity-len(f)):

        print(' ', end=' ')

    print(" %s" % pf)

print("\nTotal Page Faults: %d" %

      (fault))

## 2.3 Sử dụng hàm tính toán và mô phỏng giải thuật FIFO:

print("\nString|Frame →\t", end='')

for i in range(capacity):

    print(i, end=' ')

print("Fault Pointer\n   ↓\n")

for i in s:

    if i not in f:

        if len(f) < capacity:

            f.append(i)

        else:

            f[top] = i

            top = (top+1) % capacity

        fault += 1

        pf = 'T'

    else:

        pf = 'F'

    print("   %d\t\t" % i, end='')

    for x in f:

        print(x, end=' ')

    for x in range(capacity-len(f)):

        print(' ', end=' ')

    print(pf, end=' '\*6)

    print(top)

print("\Total Page Faults: %d" %

      (fault))

## 2.4 Sử dụng hàm tính toán và mô phỏng giải thuật clock:

pf = ''

fault, top, f, status = 0, 0, [], [0]\*capacity

for i in s:

    if i not in f:

        if len(f) < capacity:

            f.append(i)

            top = (top+1) % capacity

        else:

            while status[top] == 1:

                status[top] = 0

                top = (top+1) % capacity

            f[top] = i

            top = (top+1) % capacity

        pf = '|Yes|'

        fault += 1

    else:

        status[f.index(i)] = 1

        pf = '| No|'

    output = ' '+str(i)+' '\*2

#    print("   %d\t\t" % i, end='')

    for x in f:

        output = output+str(x)+' '

    for x in range(capacity-len(f)):

        output = output+' '\*2

    output = output + pf + ' '

    for y in status:

        output = output + str(y) + ' '

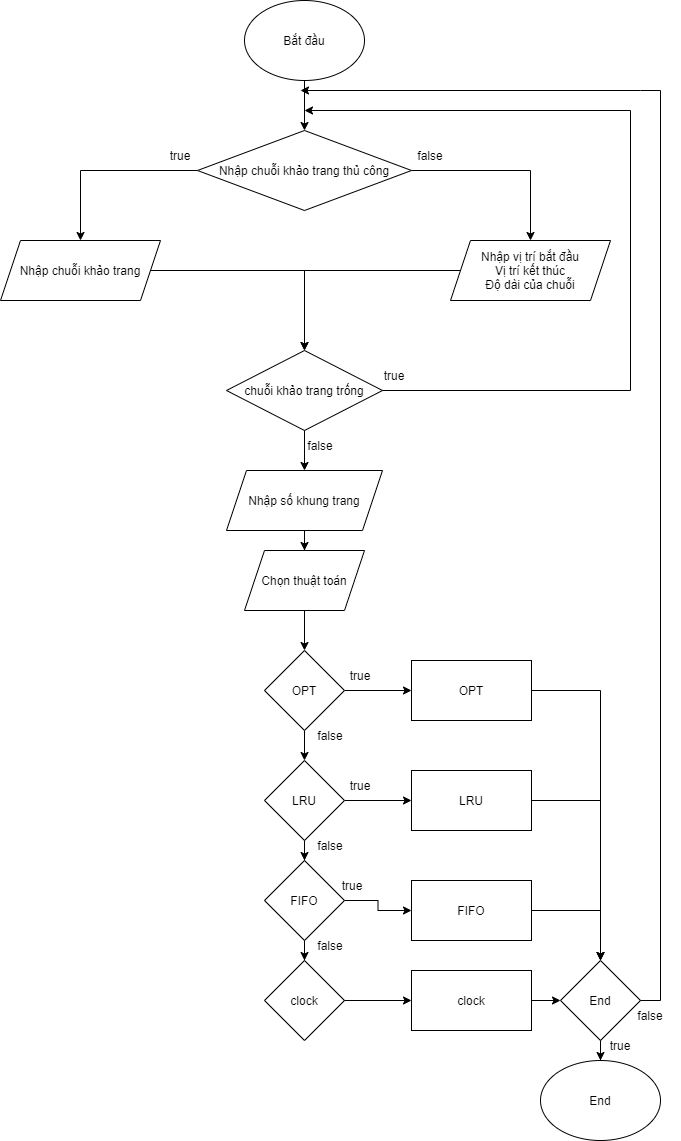
    output = output + '|' + str(top)

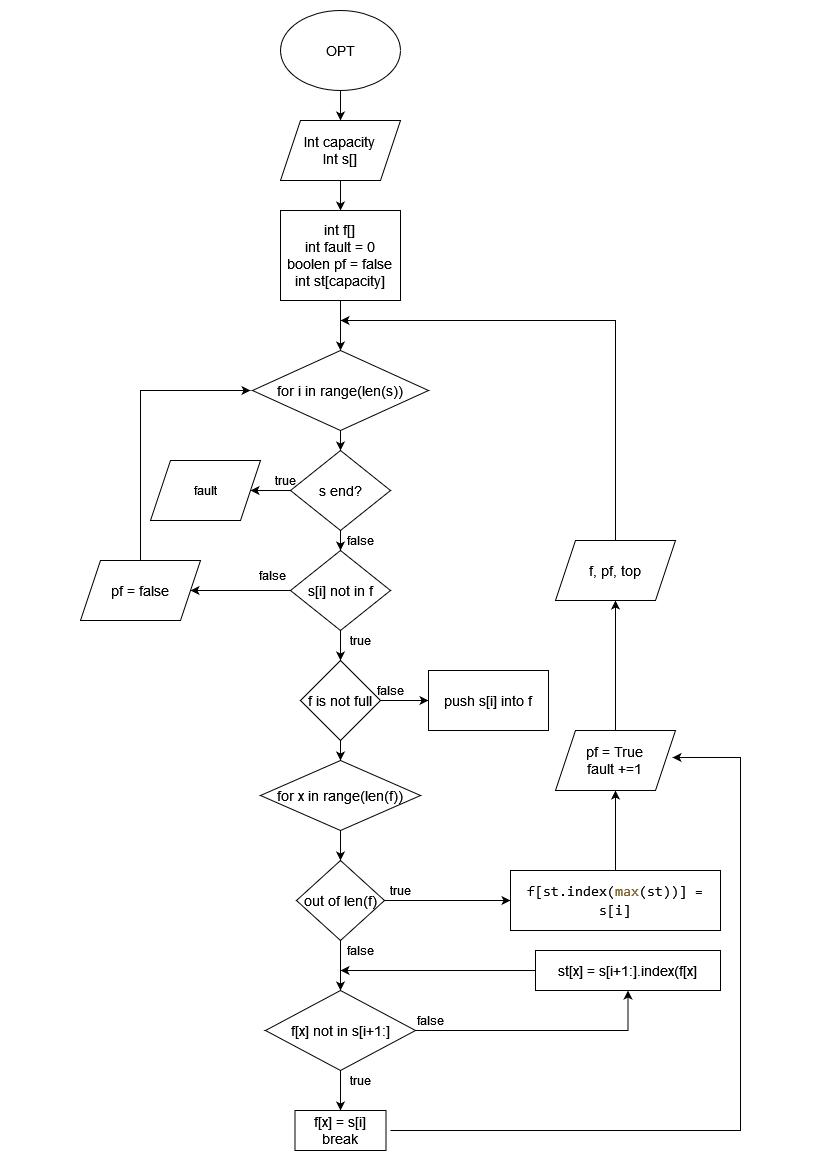
    print(output)

print(fault)

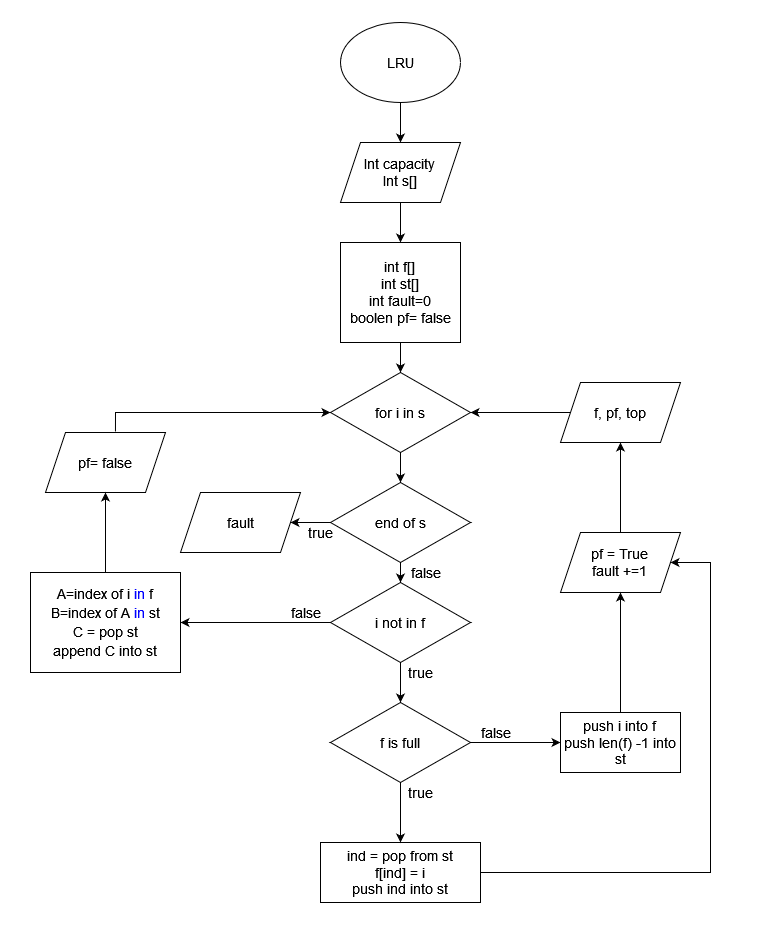
## 2. 5 Quy trình hoạt động

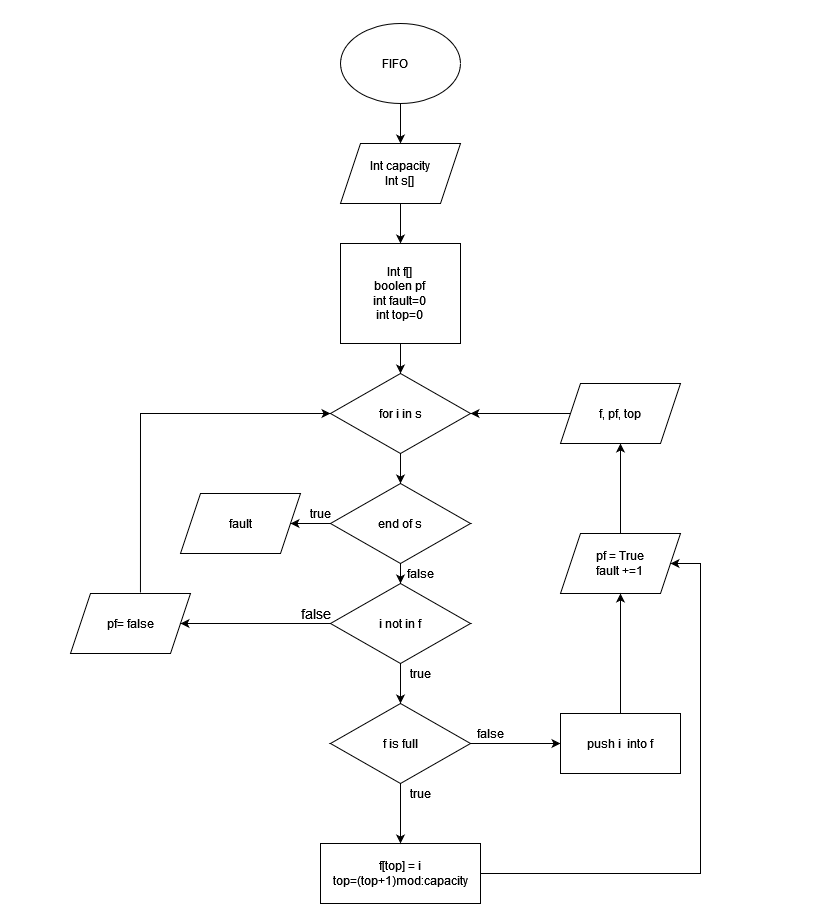
Với đầu vào là chuỗi khảo trang và số khung trang, ta xây dựng chương trình với quy trình hoạt động như sau:

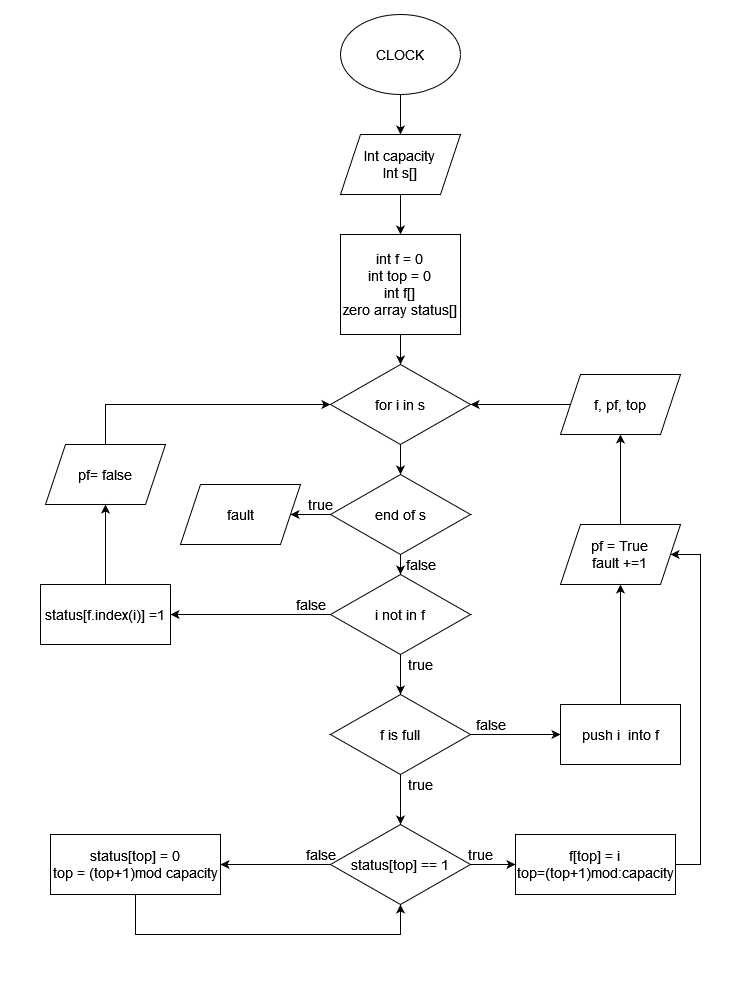


* + 1. Mô phỏng giải thuật OPT

### Mô phỏng giải thuật LRU



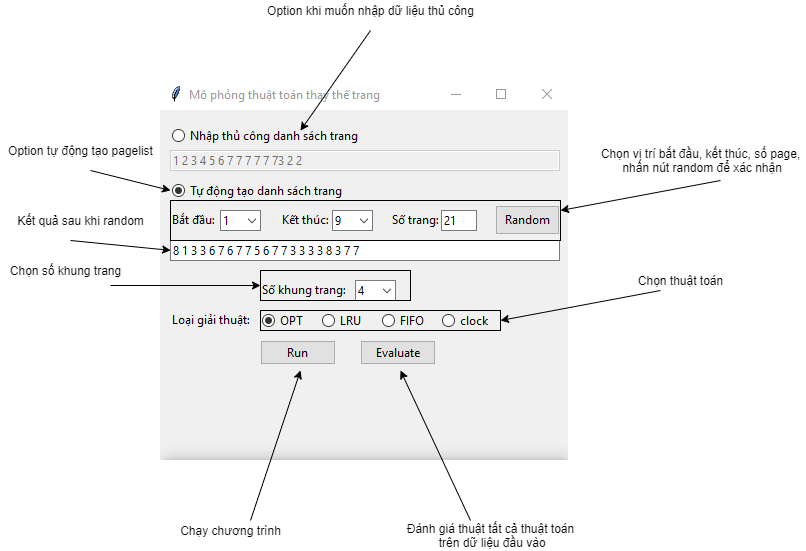
* + 1. Mô phỏng giải thuật FIFO

2.5.4 Mô phỏng giải thuật clock

# III. Giao diện người dùng

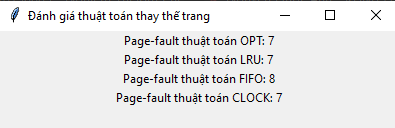
## 3.1 Nhập dữ liệu đầu vào

Ta chọn các thông số đầu vào được hướng dẫn ở hình bên dưới

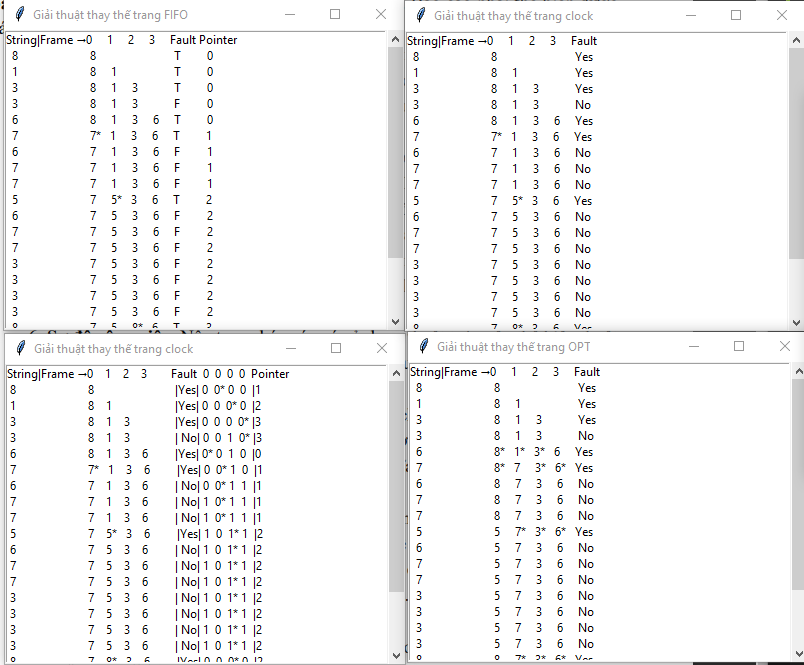


## 3.2. Kết quả

Đánh giá các thuật toán trên dữ liệu đầu vào, t nhấn nút “Evaluate”:



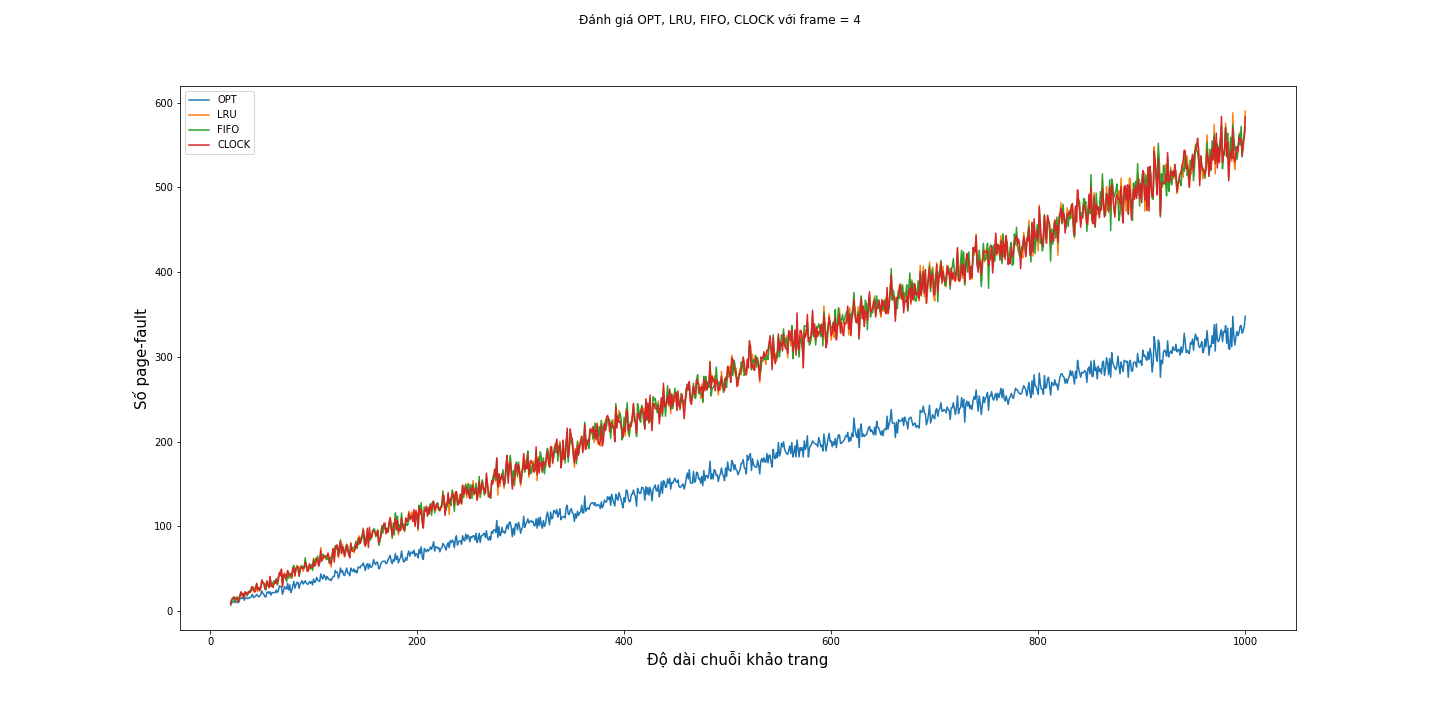
Để chạy chương trình, nhấn nút “Run” ta thu được kết quả:

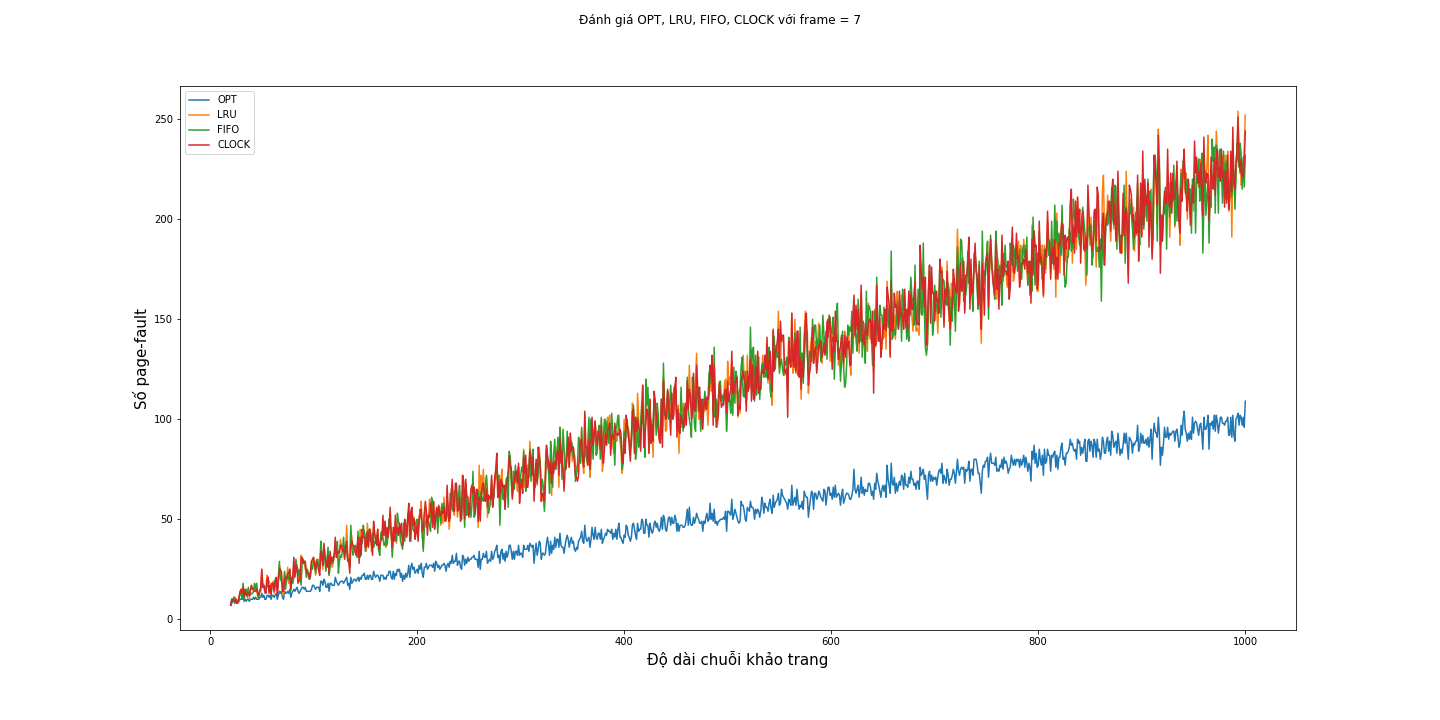


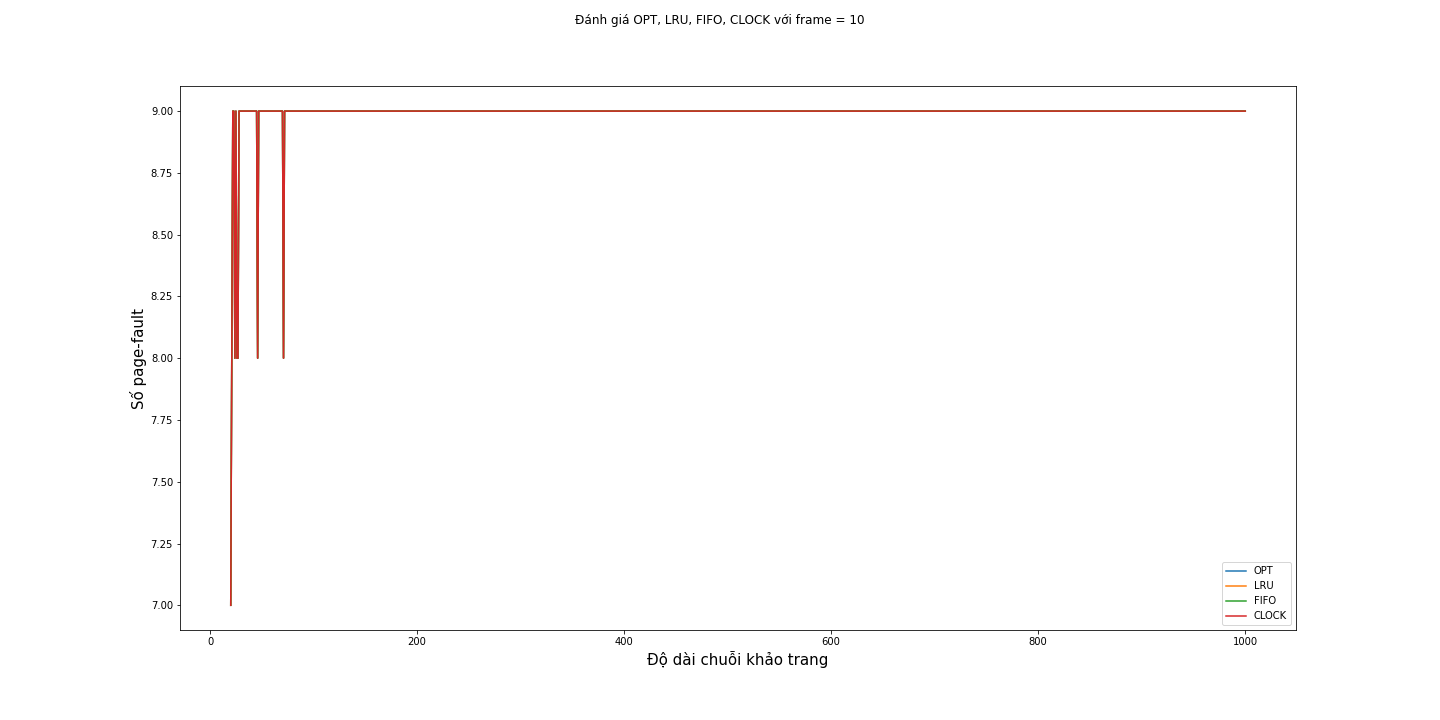
# IV Đánh giá và kết luận

Ta thực hiện đánh giá 4 thuật toán OPT, LRU, FIFO, CLOCK với lần lượt từ các page-list có độ dài từ 20 tới 1000 trang, với số frame trong khoảng 4, 7, 10 ta được kết quả như sau:

## 4.1 Số page-fault





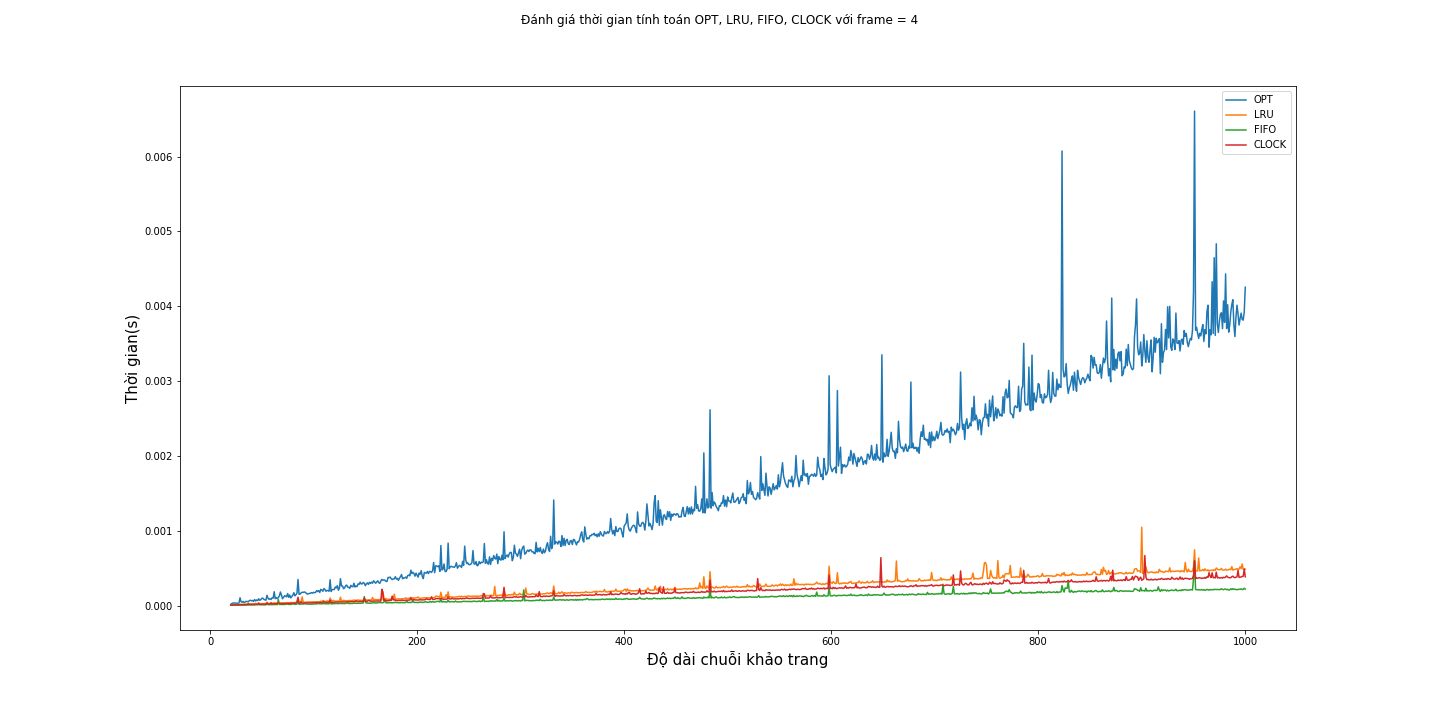


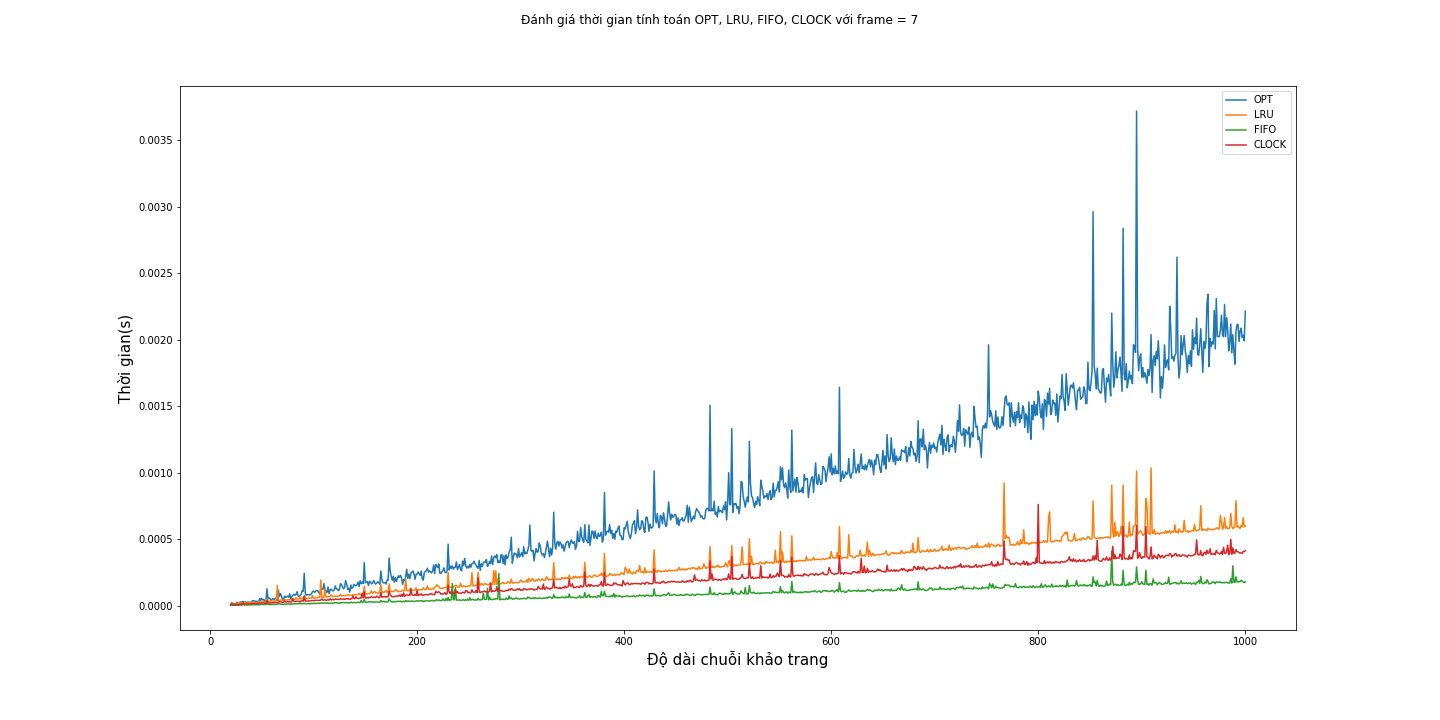
Từ biểu đồ ta thấy với số frame nhỏ với các thuật toán như LRU, FIFO, CLOCK ta được số page fault gần bằng nhau, đặc biệt OPT có kết quả tốt nhất khi luôn cho ra giá trị page fault nhỏ nhất

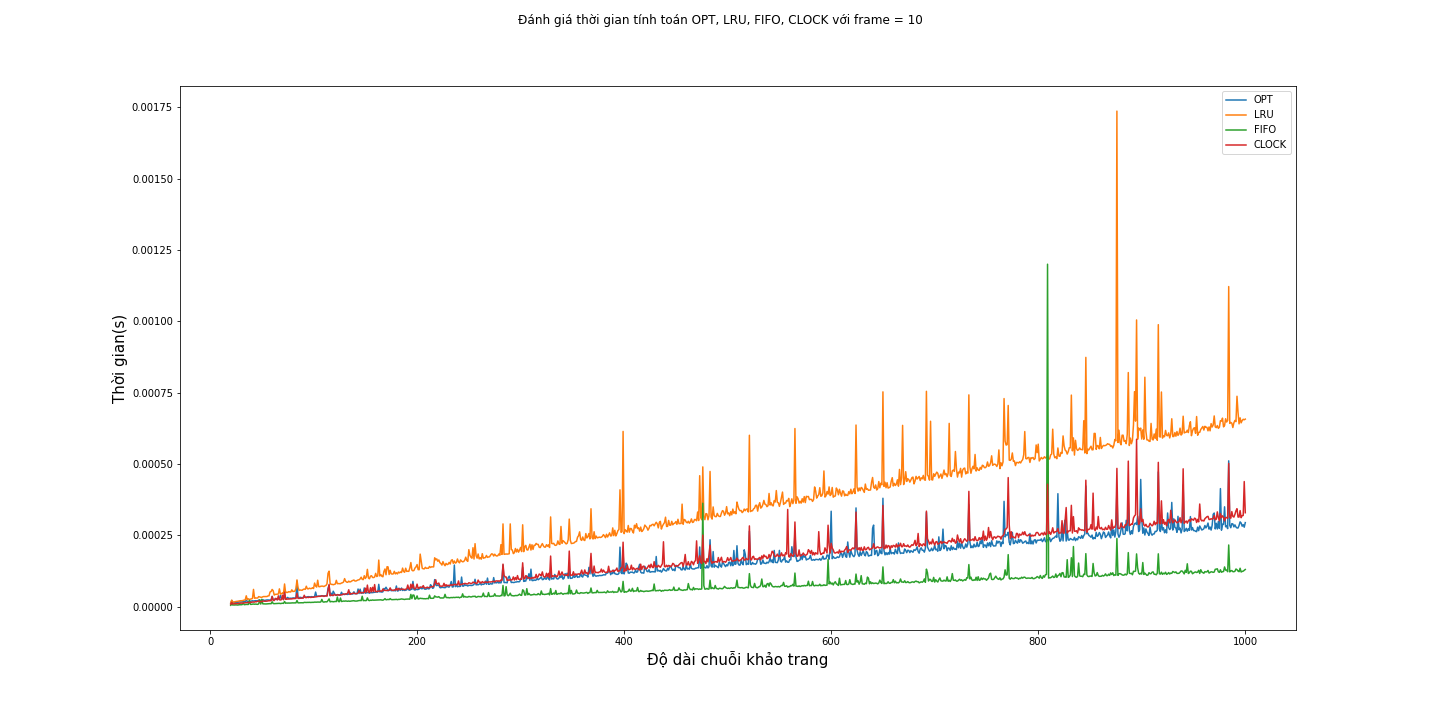
Khi frame lớn =10 ta thấy page-fault của các thuật toán trả về kết quả giống nhau

## 4.2. Thời gian tính toán

Cũng với số pagelist ta xây dựng từ phần trước, ta xét thời gian của thuật toán được đồ thị như sau:

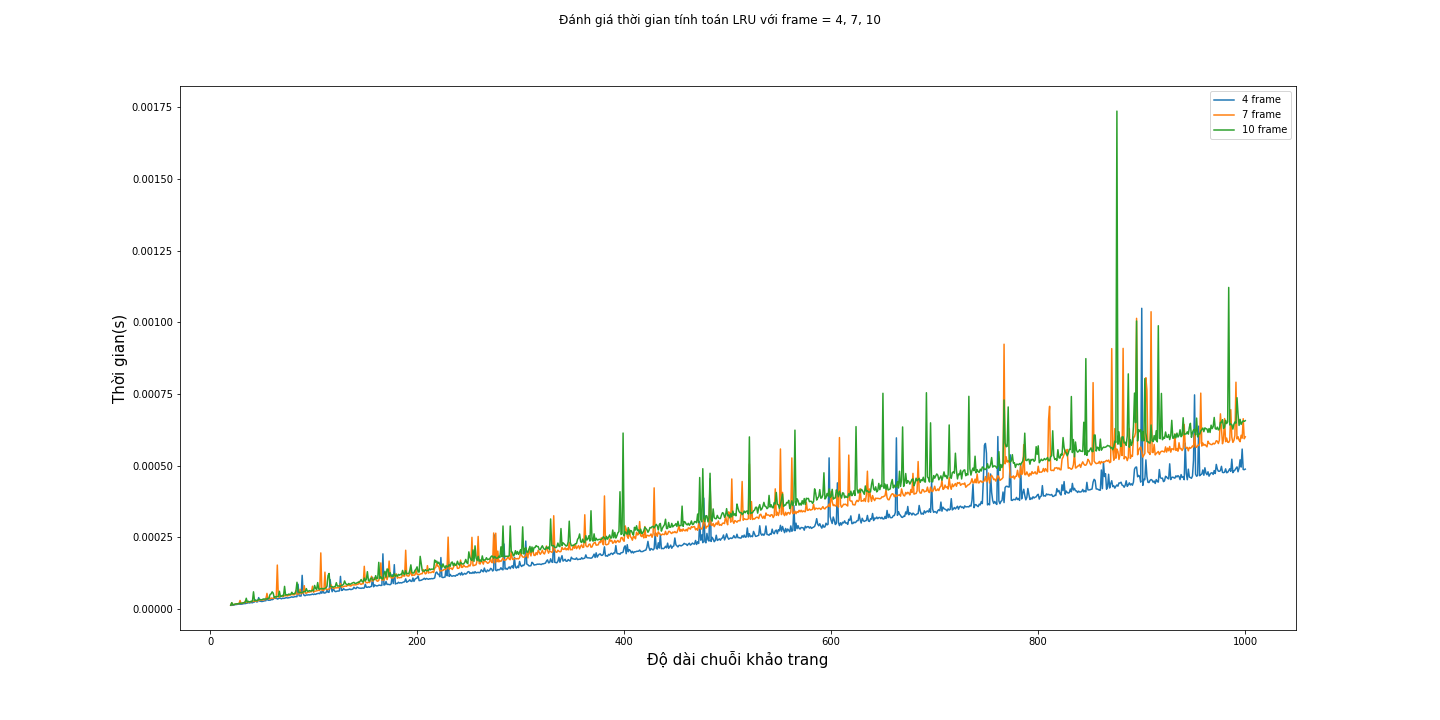
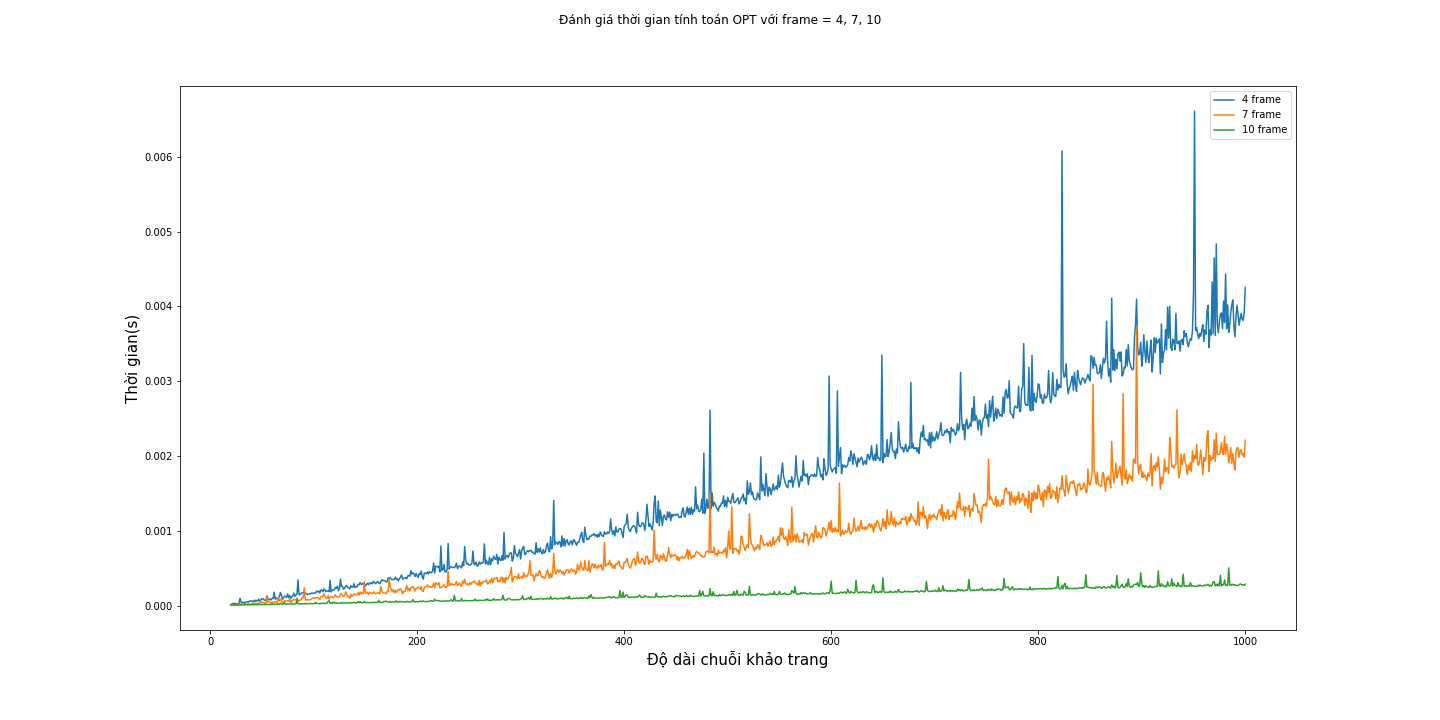


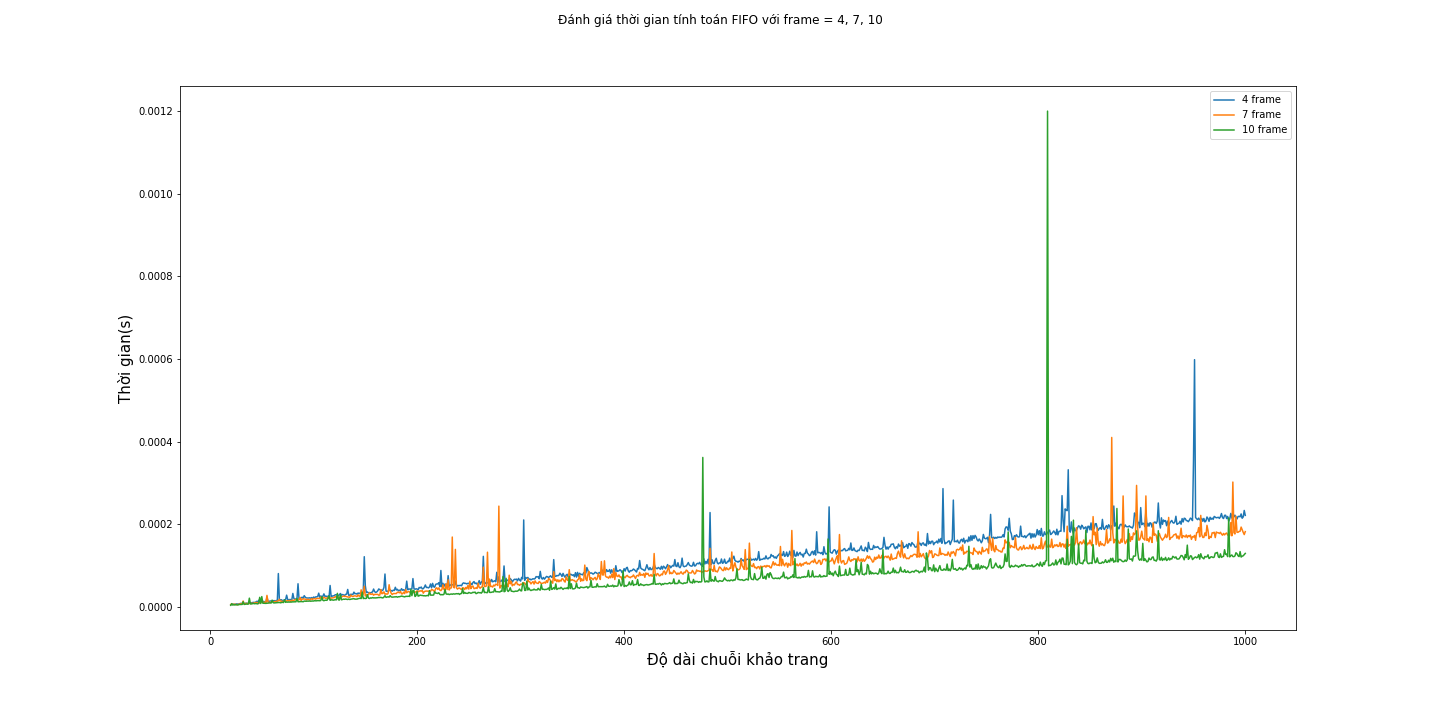


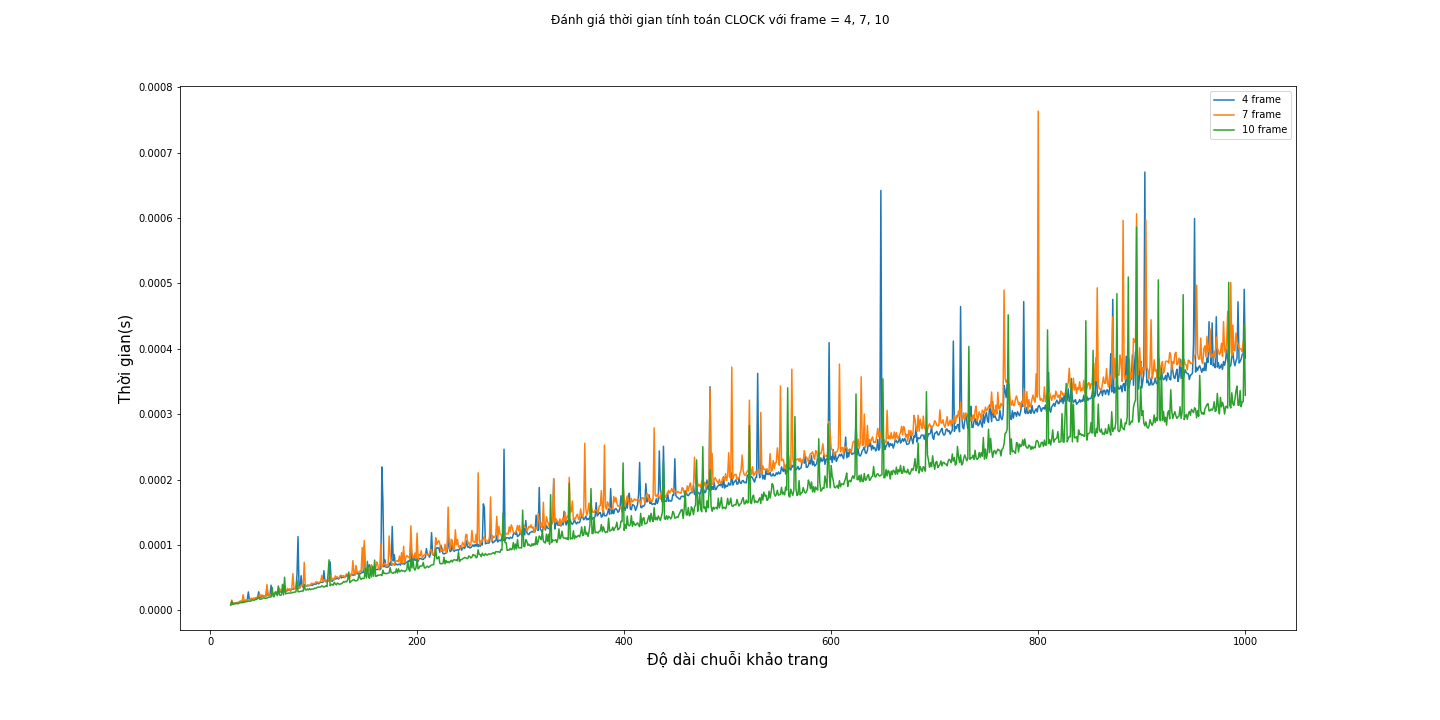


Ta thấy thuật toán FIFO luôn có thời gian hoàn thành xuất sắc nhất, đồng thời OPT khi sử dụng frame nhỏ luôn tốn thời gian hơn các thuật toán khác.

Tiếp tục xây dựng biểu đồ thời gian của từng thuật toán ta được







Với thuật toán CLOCK, FIFO ta thấy với số frame càng nhiều, thời gian chạy cảng giảm

Ngược lại với thuật toán LRU, OPT frame càng nhiều, thời gian chạy càng tăng

**Hết**