Trần Tấn Phát
2274802010644

Lab3

Bài làm

1. Sử dụng hàm sắp xếp QuickSort

```python
import random
import threading
import time

# 1. Tạo file "numbers.txt" với 25 triệu số ngẫu nhiên
Codeium: Refactor | Explain | ×
def generate_numbers_file(filename="numbers.txt", count=25_000_000, max_value=25_000_000, chunk_size=1_000_000):
    """Tạo file chứa số ngẫu nhiên theo từng khối để tránh quá tải bộ nhớ."""
    start_time = time.time()  # Bắt đầu đo thời gian

    with open(filename, "w") as f:
        for _ in range(count // chunk_size):
            numbers = [str(random.randint(1, max_value)) for _ in range(chunk_size)]
            f.write("\n".join(numbers) + "\n")

    end_time = time.time()  # Kết thúc đo thời gian
    print(f"File {filename} đã được tạo với {count} số ngẫu nhiên.")
    print(f"Thời gian tạo file: {end_time - start_time:.2f} giây\n")

# 2. Thuật toán QuickSort song song
Codeium: Refactor | Explain | ×
def quicksort(arr):
    """Hàm sắp xếp QuickSort tiêu chuẩn."""
```

```python
def threaded_quicksort(arr):
    """Hàm QuickSort có sử dụng đa luồng."""
    if len(arr) <= 10_000:  # Khi nhỏ, dùng QuickSort bình thường
        return quicksort(arr)

    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]

    left_sorted = []
    right_sorted = []

    t1 = threading.Thread(target=lambda: left_sorted.extend(quicksort(left)))
    t2 = threading.Thread(target=lambda: right_sorted.extend(quicksort(right)))

    t1.start()
    t2.start()
    t1.join()
    t2.join()

    return left_sorted + middle + right_sorted
```

```python

# 3. Đọc dữ liệu từ file theo từng khối
Codeium: Refactor | Explain | ×
def read_numbers_in_chunks(filename, chunk_size=1_000_000):
    """Đọc dữ liệu từ file theo từng khối để tiết kiệm bộ nhớ."""
    with open(filename, "r") as f:
        while True:
            chunk = f.readlines(chunk_size)
            if not chunk:
                break
            yield list(map(int, chunk))

# 4. Sắp xếp song song và ghi ra file
Codeium: Refactor | Explain | ×
def sort_numbers_from_file(input_file="numbers.txt", output_file="sorted_numbers.txt"):
    """Đọc dữ liệu từ file theo từng khối, sắp xếp song song và ghi ra file."""
    start_time = time.time()  # Bắt đầu đo thời gian sắp xếp

    sorted_chunks = []
    threads = []

    for chunk in read_numbers_in_chunks(input_file):
        thread = threading.Thread(target=lambda c: sorted_chunks.append(threaded_quicksort(c)), args=(chunk,))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()
```

```
    # Hợp nhất danh sách đã sắp xếp
    sorted_numbers = []
    for chunk in sorted_chunks:
        sorted_numbers.extend(chunk)
    sorted_numbers.sort()  # Đảm bảo toàn bộ danh sách được sắp xếp đúng

    # Ghi kết quả vào file
    with open(output_file, "w") as f:
        for num in sorted_numbers:
            f.write(f"{num}\n")

    end_time = time.time()  # Kết thúc đo thời gian sắp xếp
    print(f"Dữ liệu đã được sắp xếp và lưu vào {output_file}.")
    print(f"Thời gian sắp xếp và ghi file: {end_time - start_time:.2f} giây\n")

# Chạy chương trình
if __name__ == "__main__":
    total_start_time = time.time()  # Bắt đầu đo tổng thời gian

    generate_numbers_file()
    sort_numbers_from_file()

    total_end_time = time.time()  # Kết thúc đo tổng thời gian
    print(f"Tổng thời gian thực thi: {total_end_time - total_start_time:.2f} giây")
```

```
Tổng thời gian thực thi: 122.83 giây
(base) trantanphat@Macintosh-5 TSS % /Users/trantanphat/anaconda3/bin/python /Users/trantanphat/Documents/Python/TSS/Lab3/bt1.py
File numbers.txt đã được tạo với 25000000 số ngẫu nhiên.
Thời gian tạo file: 11.39 giây

Dữ liệu đã được sắp xếp và lưu vào sorted_numbers.txt.
Thời gian sắp xếp và ghi file: 56.05 giây

Tổng thời gian thực thi: 68.03 giây
(base) trantanphat@Macintosh-5 TSS %
```

2. Sử dụng hàm MergeSort

```
Codeium: Refactor | Explain | ×
def merge(left, right):
    """Hợp nhất hai danh sách đã sắp xếp."""
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])
    return result

Codeium: Refactor | Explain | ×
def threaded_merge_sort(arr):
    """Hàm Merge Sort có sử dụng đa luồng."""
    if len(arr) <= 10_000:  # Khi nhỏ, dùng Merge Sort bình thường
        return merge_sort(arr)

    mid = len(arr) // 2
    left_sorted = []
    right_sorted = []

    t1 = threading.Thread(target=lambda: left_sorted.extend(merge_sort(arr[:mid])))
    t2 = threading.Thread(target=lambda: right_sorted.extend(merge_sort(arr[mid:])))

    t1.start()
```

```
Lab3 > bt2.py > ...
import random
import threading
import time

# 1. Tạo file "numbers.txt" với 25 triệu số ngẫu nhiên
Codeium: Refactor | Explain | ×
def generate_numbers_file(filename="numbers.txt", count=25_000_000, max_value=25_000_000, chunk_size=1_000_000):
    """Tạo file chứa số ngẫu nhiên theo từng khối để tránh quá tải bộ nhớ."""
    start_time = time.time()

    with open(filename, "w") as f:
        for _ in range(count // chunk_size):
            numbers = [str(random.randint(1, max_value)) for _ in range(chunk_size)]
            f.write("\n".join(numbers) + "\n")

    end_time = time.time()
    print(f"File {filename} đã được tạo với {count} số ngẫu nhiên.")
    print(f"Thời gian tạo file: {end_time - start_time:.2f} giây\n")

# 2. Merge Sort và Merge Sort song song
Codeium: Refactor | Explain | ×
def merge_sort(arr):
    """Thuật toán Merge Sort tiêu chuẩn."""
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)
```

```
    t1.start()
    t2.start()
    t1.join()
    t2.join()

    return merge(left_sorted, right_sorted)

# 3. Đọc dữ liệu từ file theo từng khối
Codeium: Refactor | Explain | ×
def read_numbers_in_chunks(filename, chunk_size=1_000_000):
    """Đọc dữ liệu từ file theo từng khối để tiết kiệm bộ nhớ."""
    with open(filename, "r") as f:
        while True:
            chunk = f.readlines(chunk_size)
            if not chunk:
                break
            yield list(map(int, chunk))

# 4. Sắp xếp song song và ghi ra file
Codeium: Refactor | Explain | ×
def sort_numbers_from_file(input_file="numbers.txt", output_file="sorted_numbers.txt"):
    """Đọc dữ liệu từ file theo từng khối, sắp xếp song song và ghi ra file."""
    start_time = time.time()

    sorted_chunks = []
    threads = []

    for chunk in read_numbers_in_chunks(input_file):
        thread = threading.Thread(target=lambda c: sorted_chunks.append(threaded_merge_sort(c)), args=(chunk,))
        threads.append(thread)
```

```
    for thread in threads:
        thread.join()

    # Hợp nhất danh sách đã sắp xếp
    while len(sorted_chunks) > 1:
        chunk1 = sorted_chunks.pop(0)
        chunk2 = sorted_chunks.pop(0)
        sorted_chunks.append(merge(chunk1, chunk2))  # Dùng merge để hợp nhất các khối đã sắp xếp

    sorted_numbers = sorted_chunks[0] if sorted_chunks else []

    # Ghi kết quả vào file
    with open(output_file, "w") as f:
        for num in sorted_numbers:
            f.write(f"{num}\n")

    end_time = time.time()
    print(f"Dữ liệu đã được sắp xếp và lưu vào {output_file}.")
    print(f"Thời gian sắp xếp và ghi file: {end_time - start_time:.2f} giây\n")

# Chạy chương trình
if __name__ == "__main__":
    total_start_time = time.time()

    generate_numbers_file()
    sort_numbers_from_file()

    total_end_time = time.time()
    print(f"Tổng thời gian thực thi: {total_end_time - total_start_time:.2f} giây")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  COMMENTS                                   Python

/Users/trantanphat/anaconda3/bin/python /Users/trantanphat/Documents/Python/TSS/Lab3/bt2.py
(base) trantanphat@Macintosh-5 TSS % /Users/trantanphat/anaconda3/bin/python /Users/trantanphat/Documents/Python/TSS/Lab3/bt2.py
File numbers.txt đã được tạo với 25000000 số ngẫu nhiên.
Thời gian tạo file: 11.76 giây

Dữ liệu đã được sắp xếp và lưu vào sorted_numbers.txt.
Thời gian sắp xếp và ghi file: 109.88 giây

Tổng thời gian thực thi: 122.83 giây
```

3. Sử dụng hàm SelectionSort

```
Lab3 > bt3.py > sort_numbers_from_file
import random
import threading
import time

# 1. Tạo file "numbers.txt" với 25 triệu số ngẫu nhiên
Codeium: Refactor | Explain | ×
def generate_numbers_file(filename="numbers.txt", count=25_000_000, max_value=25_000_000, chunk_size=1_000_000):
    """Tạo file chứa số ngẫu nhiên theo từng khối để tránh quá tải bộ nhớ."""
    start_time = time.time()

    with open(filename, "w") as f:
        for _ in range(count // chunk_size):
            numbers = [str(random.randint(1, max_value)) for _ in range(chunk_size)]
            f.write("\n".join(numbers) + "\n")

    end_time = time.time()
    print(f"File {filename} đã được tạo với {count} số ngẫu nhiên.")
    print(f"Thời gian tạo file: {end_time - start_time:.2f} giây\n")
```

```python
Codeium: Refactor | Explain | ×
def threaded_selection_sort(arr):
    """Selection Sort có sử dụng đa luồng."""
    if len(arr) <= 10_000:  # Khi nhỏ, dùng Selection Sort bình thường
        return selection_sort(arr)

    mid = len(arr) // 2
    left_sorted = []
    right_sorted = []

    t1 = threading.Thread(target=lambda: left_sorted.extend(selection_sort(arr[:mid])))
    t2 = threading.Thread(target=lambda: right_sorted.extend(selection_sort(arr[mid:])))

    t1.start()
    t2.start()
    t1.join()
    t2.join()

    return left_sorted + right_sorted

# 3. Đọc dữ liệu từ file theo từng khối
Codeium: Refactor | Explain | ×
def read_numbers_in_chunks(filename, chunk_size=1_000_000):
    """Đọc dữ liệu từ file theo từng khối để tiết kiệm bộ nhớ."""
    with open(filename, "r") as f:
        while True:
            chunk = f.readlines(chunk_size)
            if not chunk:
                break
            yield list(map(int, chunk))

```

```python
# 4. Sắp xếp song song và ghi ra file
Codeium: Refactor | Explain | ×
def sort_numbers_from_file(input_file="numbers.txt", output_file="sorted_numbers.txt"):
    """Đọc dữ liệu từ file theo từng khối, sắp xếp song song và ghi ra file."""
    start_time = time.time()

    sorted_chunks = []
    threads = []

    for chunk in read_numbers_in_chunks(input_file):
        thread = threading.Thread(target=lambda c: sorted_chunks.append(threaded_selection_sort(c)), args=(chunk,))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    # Hợp nhất danh sách đã sắp xếp
    sorted_numbers = []
    for chunk in sorted_chunks:
        sorted_numbers.extend(chunk)
    sorted_numbers.sort()  # Dùng sort() để đảm bảo danh sách đã đúng thứ tự

    # Ghi kết quả vào file
    with open(output_file, "w") as f:
        for num in sorted_numbers:
            f.write(f"{num}\n")

    end_time = time.time()
    print(f"Dữ liệu đã được sắp xếp và lưu vào {output_file}.")
    print(f"Thời gian sắp xếp và ghi file: {end_time - start_time:.2f} giây\n")
```

```python

    end_time = time.time()
    print(f"Dữ liệu đã được sắp xếp và lưu vào {output_file}.")
    print(f"Thời gian sắp xếp và ghi file: {end_time - start_time:.2f} giây\n")

# Chạy chương trình
if __name__ == "__main__":
    total_start_time = time.time()

    generate_numbers_file()
    sort_numbers_from_file()

    total_end_time = time.time()
    print(f"Tổng thời gian thực thi: {total_end_time - total_start_time:.2f} giây")

```

4. Sử dụng hàm CountingSort

Lab3 > bt4.py > ...

```python
import random
import threading
import time

# 1. Tạo file "numbers.txt" với 25 triệu số ngẫu nhiên
Codeium: Refactor | Explain | ×
def generate_numbers_file(filename="numbers.txt", count=25_000_000, max_value=25_000_000, chunk_size=1_000_000):
    """Tạo file chứa số ngẫu nhiên theo từng khối để tránh quá tải bộ nhớ."""
    start_time = time.time()

    with open(filename, "w") as f:
        for _ in range(count // chunk_size):
            numbers = [str(random.randint(1, max_value)) for _ in range(chunk_size)]
            f.write("\n".join(numbers) + "\n")

    end_time = time.time()
    print(f"File {filename} đã được tạo với {count} số ngẫu nhiên.")
    print(f"Thời gian tạo file: {end_time - start_time:.2f} giây\n")

# 2. Counting Sort và Counting Sort song song
Codeium: Refactor | Explain | ×
def counting_sort(arr, max_value):
    """Thuật toán Counting Sort tiêu chuẩn."""
    count = [0] * (max_value + 1)
    output = []

    for num in arr:
        count[num] += 1

    for num, freq in enumerate(count):
        output.extend([num] * freq)

    return output

Codeium: Refactor | Explain | ×
def threaded_counting_sort(arr, max_value):
    """Counting Sort có sử dụng đa luồng."""
    if len(arr) <= 100_000:  # Nếu mảng nhỏ, dùng Counting Sort trực tiếp
        return counting_sort(arr, max_value)

    mid = len(arr) // 2
    left_sorted = []
    right_sorted = []

    t1 = threading.Thread(target=lambda: left_sorted.extend(counting_sort(arr[:mid], max_value)))
    t2 = threading.Thread(target=lambda: right_sorted.extend(counting_sort(arr[mid:], max_value)))

    t1.start()
    t2.start()
    t1.join()
    t2.join()

    return left_sorted + right_sorted

# 3. Đọc dữ liệu từ file theo từng khối
Codeium: Refactor | Explain | ×
def read_numbers_in_chunks(filename, chunk_size=1_000_000):
    """Đọc dữ liệu từ file theo từng khối để tiết kiệm bộ nhớ."""
    with open(filename, "r") as f:
        while True:
            chunk = f.readlines(chunk_size)
            if not chunk:
                break
            yield list(map(int, chunk))

# 4. Sắp xếp song song và ghi ra file
Codeium: Refactor | Explain | ×
def sort_numbers_from_file(input_file="numbers.txt", output_file="sorted_numbers.txt", max_value=25_000_000):
    """Đọc dữ liệu từ file theo từng khối, sắp xếp song song và ghi ra file."""
    start_time = time.time()

    sorted_chunks = []
    threads = []

    for chunk in read_numbers_in_chunks(input_file):
        thread = threading.Thread(target=lambda c: sorted_chunks.append(threaded_counting_sort(c, max_value)), args=(chunk,))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    # Hợp nhất danh sách đã sắp xếp
    sorted_numbers = []
    for chunk in sorted_chunks:
        sorted_numbers.extend(chunk)

    # Ghi kết quả vào file
    with open(output_file, "w") as f:
        for num in sorted_numbers:
            f.write(f"{num}\n")
```

```
            f.write(f"{num}\n")

    end_time = time.time()
    print(f"Dữ liệu đã được sắp xếp và lưu vào {output_file}.")
    print(f"Thời gian sắp xếp và ghi file: {end_time - start_time:.2f} giây\n")

# Chạy chương trình
if __name__ == "__main__":
    total_start_time = time.time()

    generate_numbers_file()
    sort_numbers_from_file()

    total_end_time = time.time()
    print(f"Tổng thời gian thực thi: {total_end_time - total_start_time:.2f} giây")
```