Lý thuyết 6: Tham lam

Bùi Khánh Duy - 20001898

Exercise 1.

Cho mảng A có kích thước n. Giả sử, giá trị A[i] cho biết độ dài của tệp thứ i và người ta cần ghép (merge) tất cả các file đó thành 1 file duy nhất. Hãy kiểm tra xem thuật toán dưới đây có đưa ra lời giải tốt nhất cho bài toán hay không? Vì sao?
Thuật toán: Ghép các file một cách liên tục (tức là chọn 2 file đầu tiên và ghép chúng lại. Sau đó, lấy file kết quả với ghép rồi ghép với file thứ 3 và cứ tiếp tục như thế, ...
Lưu ý: Với 2 file X và Y có kích thước là x và y, độ phức tạp của việc ghép làm $O(x+y)$.

Bài làm:

Thuật toán ghép các file một cách liên tục không đưa ra lời giải tốt nhất cho bài toán ghép các file thành một file duy nhất. Lý do là khi ghép các file có kích thước gần bằng nhau, thì độ phức tạp của việc ghép các file này sẽ tăng lên theo cấp số nhân. Ví dụ, khi ghép 4 file có kích thước lần lượt là 1, 2, 4, 8 thì độ phức tạp của việc ghép chúng sẽ là 1+2+(1+2+4)+(1+2+4+8) = 25. Trong khi đó, giả sử 4 file có kích thước lần lượt là 8, 4, 2, 1 thì việc ghép chúng sẽ là 8+4+(8+4+2)+(8+4+2+1) = 41 ⇒ lớn hơn so với phía trên rất nhiều.

```
def cach_1(data):
    # Merge file without sort
    ans = 0
    cur = 0
    for i in range(len(data)):
        cur = cur + data[i]
        ans = ans + cur
    return ans
```

Exercise 2.

Tương tự của bài 1, thuật toán dưới đây có đưa ra lời giải tối ưu hay không?
Thuật toán: Ghép các file theo cặp (tức là, sau bước đầu tiên thuật toán tạo ra n/2 file trung gian).

Bài làm:

Việc chia đôi rồi ghép các file, ta có kết quả là số lần gộp file như sau:

$$S(n) = \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^k} + ... + \frac{n}{2^{\log_2 n}} = n$$

(tính theo công thức tổng của cấp số cộng). ⇒ Không tốt hơn việc cộng liên tục như ở thuật toán trên.

```
def cach_2(data):
    # Merge file with 2 pair for each time
    ans = 0
    x = data.copy()
    table = []
    while len(x) > 1:
        cur = []
        if len(x) % 2 == 1:
            cur.append(x[len(x)//2])

        for i in range(len(x)//2):
            cur.append(x[i] + x[len(x)-i-1])
            i += 1

        table.append(cur)
        x = cur
    ans = sum(sum(x) for x in table)
    return ans
```

Exercise 3.

Dựa vào kết quả phân tích bài 1 và bài 2 thì cách tốt nhất để ghép tất cả các file thành 1 file duy nhất là gì? Hãy nêu rõ cách đó.

Bài làm:

Ở câu 1 và câu 2, số bước ghép file là như nhau, nhưng như đã phân tích ở câu 1, vì độ phức tạp của việc ghép 2 file có kích thước lần lượt x, y là $O(x + y)$ nên thứ tự ghép các file là quan trọng:

Do đó, để đưa ra lời giải tốt nhất cho bài toán ghép các file thành một file duy nhất, ta có thể được thực hiện bằng cách sử dụng hàng đợi ưu tiên (priority queue) để lưu trữ các file theo thứ tự tăng dần của kích thước. Mỗi lần lấy ra hai file có kích thước nhỏ nhất trong hàng đợi ưu tiên và ghép chúng lại thành một file mới, sau đó đưa file mới này vào hàng đợi ưu tiên. Việc này được lặp lại cho đến khi chỉ còn một file duy nhất trong hàng đợi ưu tiên, tức là đã ghép được tất cả các file thành một file duy nhất. Độ phức tạp của thuật toán này là $O(n \log n)$. Ngoài ra ta có thể sắp xếp danh sách của các phần tử theo kích thước tăng dần, với độ phức tạp cũng là $O(n \log n)$

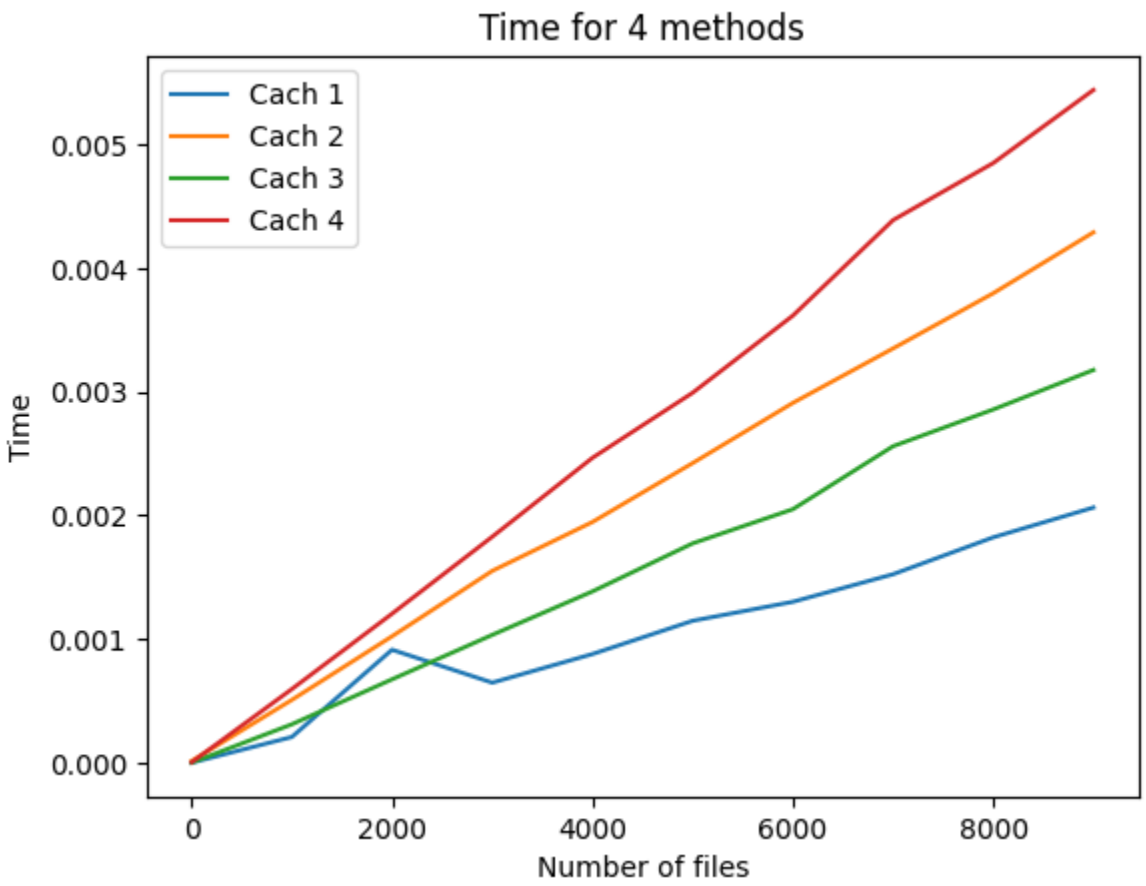
Tổng độ phức tạp của thuật toán sẽ là $O(n \log n + n^2 * X)$ với X là dung lượng của file có kích cỡ median trong tệp tin.

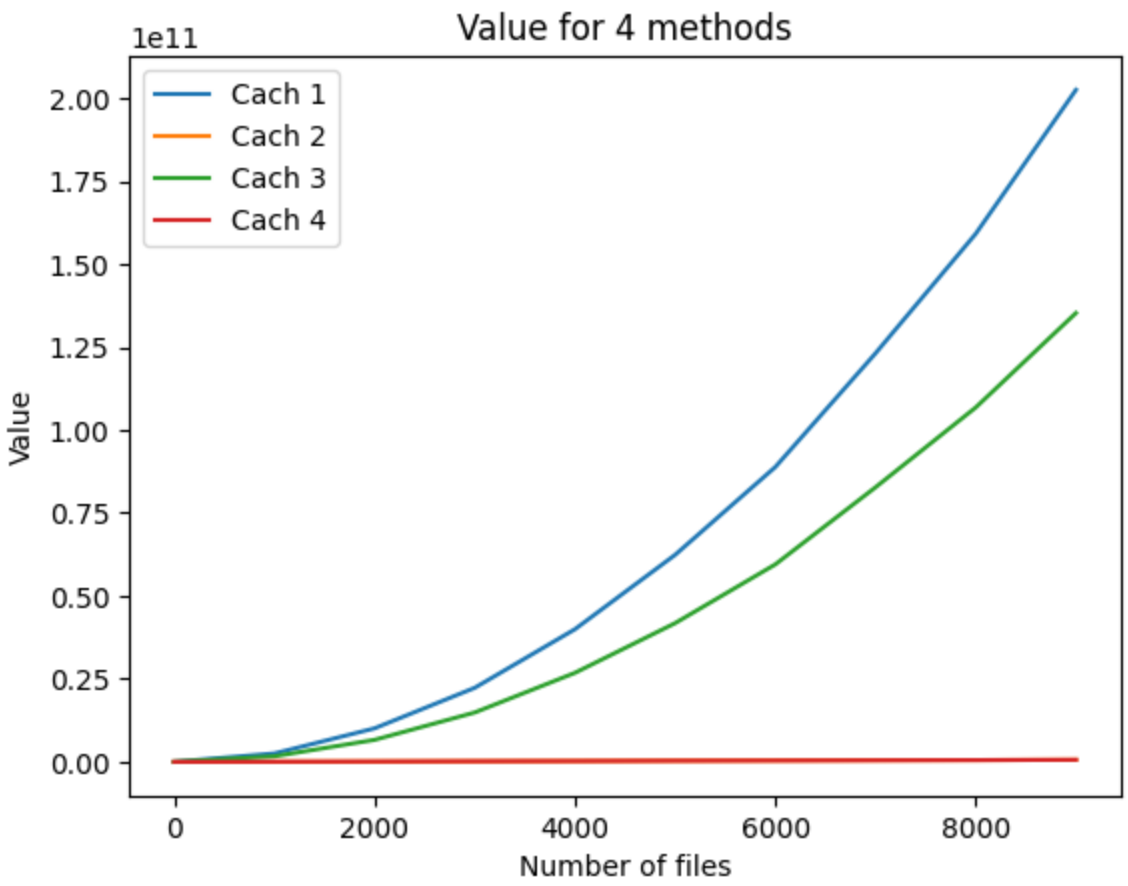
Ở đây sẽ thử sort rồi dùng code của cách 2 và 3.

```
def cach_3(data):
    x = sorted(data)
    return cach_1(x)
```

```
def cach_4(data):
    x = sorted(data)
    return cach_2(x)
```

Kết quả chạy của các cách như sau:


Time for 4 methods
Cach 1
Cach 2
Cach 3
Cach 4
Time
0.000
0.001
0.002
0.003
0.004
0.005
0
2000
4000
6000
8000
Number of files


Value for 4 methods
1e11
Cach 1
Cach 2
Cach 3
Cach 4
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Value
0
2000
4000
6000
8000
Number of files