TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN – ĐẠI HỌC TP.HCM
KHOA KHOA HỌC MÁY TÍNH

ĐỒ ÁN CTDL & GT NÂNG CAO

ĐỀ TÀI: TÌM HIỂU THUẬT TOÁN SẮP XẾP NGOẠI (EXTERNAL SORT) BẰNG HAI PHƯƠNG PHÁP TRỘN VÀ DEMO THUẬT TOÁN

Sinh viên thực hiện : TRẦN HOÀNG HẢI - 23520422
ĐẶNG VIỆT HOÀNG - 23520513
Lớp : CS523.P21

TP. Hồ Chí Minh, tháng 2 năm 2025

MỤC LỤC

1. GIỚI THIỆU

1.1. Tổng quan

Sắp xếp ngoại là thuật ngữ chỉ một lớp thuật toán sắp xếp có thể xử lý lượng dữ liệu lớn. Sắp xếp ngoại cần thiết khi dữ liệu được sắp xếp không vừa với bộ nhớ chính của thiết bị (thường là RAM) mà thay vào đó phải nằm trong bộ nhớ ngoài (thường là ổ cứng).

Thuật toán External Sort thường sử dụng chiến lược kết hợp giữa sắp xếp và trộn. Trong giai đoạn sắp xếp, các khối dữ liệu nhỏ đủ để vừa với bộ nhớ chính được đọc vào, sắp xếp và ghi ra tệp tạm thời. Trong giai đoạn trộn, các tệp con đã sắp xếp được kết hợp lại thành một tệp lớn duy nhất chứa dữ liệu đầu vào đã được sắp xếp.

Hai phương pháp trộn phổ biến trong External Sort

- Phương pháp trộn tự nhiên (Natural Merge)
- Phương pháp trộn nhiều đường cân bằng (Balance Multiway Merge)

1.2. Mục tiêu

Bản báo cáo nhằm trình bày chi tiết về các đặc điểm, cách hoạt động và cài đặt chương trình của thuật toán External Sort bằng hai phương pháp trộn tự nhiên và trộn bằng nhiều đường cân bằng.

2. CƠ SỞ LÝ THUYẾT

2.1. Thuật toán External Sort

2.1.1. Nguyên lý hoạt động

Thuật toán sắp xếp ngoại sắp xếp các khối mà mỗi khối vừa với RAM, sau đó hợp nhất các khối đã sắp xếp lại với nhau. Đầu tiên, chúng ta chia tệp thành các lần chạy sao cho kích thước của một lần chạy đủ nhỏ để vừa với bộ nhớ chính. Sau đó, sắp xếp từng lần chạy trong bộ nhớ chính bằng thuật toán sắp xếp (QuickSort, MergeSort, HeapSort,…). Cuối cùng, trộn các file đã sắp xếp lại với nhau thành các file đã sắp xếp lớn hơn cho đến khi chỉ còn một file đã được sắp xếp với hai phương pháp trộn: trộn tự nhiên và trộn bằng nhiều đường cân bằng.

2.1.2. Độ phức tạp

Các thuật toán sắp xếp bên ngoại có thể được phân tích trong mô hình bộ nhớ ngoài. Trong mô hình này, bộ nhớ trong có kích thước M và một bộ nhớ ngoài không giới hạn được chia thành các khối có kích thước B. Thời gian chạy của một thuật toán được xác định bởi số lần truyền dữ liệu giữa bộ nhớ trong và bộ nhớ ngoài. Giống như các thuật toán tương tự tối ưu hóa bộ nhớ đệm (cache-oblivious), các thuật toán sắp xếp bên ngoài tối ưu tiệm cận đạt được thời gian chạy (trong ký hiệu Big O) là:

$$O(\frac{N}{B} log_{M/B} \frac{N}{B})$$

2.2. Phương pháp trộn tự nhiên (Natural Merge)

2.2.1. Tổng quan

Đặc điểm của phương pháp trộn tự nhiên là tận dụng đường chạy tăng dần của các run ban đầu (dãy con liên tiếp đã sắp xếp sẵn trong dữ liệu đầu vào); nghĩa là thực hiện việc trộn các run đã được sắp xếp sẵn cho tới khi dãy chỉ còn 1 run duy nhất → dãy đã được sắp xếp.

2.2.2. Cách thực hiện

Bước 1: Xác định các dãy con có đường chạy tăng dần (runs)

- Duyệt qua mảng, cho các phần tử vào run i
- Nếu arr[i+1] > arr[i], thì đưa arr[i+1] vào run i+1

Bước 2: Trộn các runs

- Sử dụng Min-Heap (hàng đợi ưu tiên) để tìm phần tử nhỏ nhất từ k file.
- Ghi phần tử nhỏ nhất vào file kết quả.
- Tiếp tục đưa phần tử tiếp theo từ run tương ứng vào Min-Heap

Bước 3: Lặp lại cho tới khi không còn run nào thì file đã được sắp xếp

2.3. Phương pháp trộn nhiều đường cân bằng (Balanced Multiway Merge)

2.3.1. Tổng quan

Trộn nhiều đường cân bằng là một phương pháp chia mảng dữ liệu ban đầu thành các runs sao chi kích thước tối đa mỗi run bằng với MemorySize của thiết bị. Sau đó dùng thuật toán sắp xếp nội bộ để sắp xếp từng run và sắp xếp lại cấu trúc dữ liệu MinHeap.

2.3.2. Cách thực hiện

Bước 1: Chia dữ liệu thành các dãy con đã sắp xếp (Sorted Runs)

- Đọc từng khối dữ liệu có kích thước M (Memsize) từ bộ nhớ ngoài vào RAM.
- Sắp xếp khối đó bằng một thuật toán sắp xếp nội bộ (HeapSort, QuickSort, MergeSort,…).
- Ghi dãy con đã sắp xếp ra các file tạm.

Bước 2: Trộn các dãy con sử dụng k-Way Merge

- Sử dụng Min-Heap (hàng đợi ưu tiên) để tìm phần tử nhỏ nhất từ k file.
- Ghi phần tử nhỏ nhất vào file kết quả.
- Tiếp tục đưa phần tử tiếp theo từ file tương ứng vào Min-Heap

Bước 3: Lặp lại quá trình trộn cho đến khi hoàn tất

- Khi số lượng dãy con còn lại nhiều hơn k, tiếp tục lặp lại quá trình trộn k-way merge cho đến khi chỉ còn một file kết quả duy nhất.

3. So sánh

Phương pháp	Số lần trộn	Đặc điểm
Natural Merge	$log_2 N$	Hiệu quả với dữ liệu có các đoạn đã được sắp xếp sẵn
Balanced Multiway Merge	$log_k N$	Hiệu suất tốt với dữ liệu ngẫu nhiên, xử lý nhiều đường song song

3. CÀI ĐẶT THUẬT TOÁN

3.1. Natural External Sort

Cài đặt cấu trúc dữ liệu Min – Heap với các hàm

- Swap(): Hoán đổi vị trí giữa hai phần tử
- Push(): Thêm phần tử vào heap
- Pop(): Lấy phần tử nhỏ nhất (root của heap)
- HeapifyUp(): Duy trì Heap sau khi thêm phần tử
- HeapifyDown(): Duy trì Heap sau khi xóa phần tử
- isEmty(): Trả về 1 nếu Heap rỗng

```
class MinHeap {
    constructor()
    {
        this.heap = [];
    }

    getParentIndex(i) { return Math.floor((i - 1) / 2); }
    getLeftChildIndex(i) { return 2 * i + 1; }
    getRightChildIndex(i) { return 2 * i + 2; }

    swap(i, j)
    {
        [this.heap[i], this.heap[j]] = [this.heap[j], this.heap[i]];
    }

    push(value)
    {
        this.heap.push(value);
        this.heapifyUp();
    }

    pop()
    {
        if (this.heap.length === 0) return null;
        if (this.heap.length === 1) return this.heap.pop();
```

```
        const root = this.heap[0];
        this.heap[0] = this.heap.pop();
        this.heapifyDown();
        return root;
    }

    heapifyUp()
    {
        let index = this.heap.length - 1;
        while (index > 0 && this.heap[index].value <
this.heap[this.getParentIndex(index)].value) {
            this.swap(index, this.getParentIndex(index));
            index = this.getParentIndex(index);
        }
    }

    heapifyDown()
    {
        let index = 0;
        while (this.getLeftChildIndex(index) < this.heap.length) {
            let smallerChildIndex = this.getLeftChildIndex(index);
            if (this.getRightChildIndex(index) < this.heap.length &&
                this.heap[this.getRightChildIndex(index)].value <
this.heap[smallerChildIndex].value) {
                smallerChildIndex = this.getRightChildIndex(index);
            }

            if (this.heap[index].value <
this.heap[smallerChildIndex].value) break;
            this.swap(index, smallerChildIndex);
            index = smallerChildIndex;
        }
    }

    isEmpty()
    {
        return this.heap.length === 0;
    }
```

```
}
```

Tìm các dãy con có đường chạy tăng dần và lưu vào run

```
function findRuns(arr)
{
    let runs = [];
    let run = [arr[0]];
    for(let i = 0; i < arr.length; i++)
    {
        if(arr[i] < arr[i + 1])
        {
            run.push(arr[i + 1]);
        }
        else
        {
            runs.push(run);
            run = [arr[i + 1]];
        }
    }
    return runs;
}
```

Trộn các dãy con đã được sắp xếp

- Duyệt qua từng run và đưa phần tử đầu tiên của mỗi run vào MinHeap
- Lấy phần tử nhỏ nhất cảu MinHeap và lưu vào result
- Nếu run đó còn phần tử, chèn tiếp vào Heap

```
function mergeRuns(runs)
{
    let minHeap = new MinHeap();
    let result = [];

    // Đưa phần tử đầu tiên của mỗi run vào heap
```

```
    for (let i = 0; i < runs.length; i++)
    {
        if (runs[i].length > 0)
        {
            minHeap.push({ value: runs[i][0], runIndex: i,
pos: 0 });
        }
    }

    // Xử lý từng phần tử nhỏ nhất trong heap
    while (!minHeap.isEmpty())
    {

        let { value, runIndex, pos } = minHeap.pop();
        result.push(value);

        // Nếu run đó còn phần tử, chèn tiếp vào heap
        if (pos + 1 < runs[runIndex].length)
        {
            minHeap.push({ value: runs[runIndex][pos + 1],
runIndex, pos: pos + 1 });
        }
    }

    return result;
}
```

Hàm gọi:

```
function naturalMergeSort(arr)
{
    let runs = findRuns(arr);

    if (runs.length === 1) return runs[0];

    return mergeRuns(runs);
}
```

3.2. **Balance Multiway Merge**

Cài đặt cấu trúc dữ liệu MinHeap với các hàm:

- Swap(): Hoán đổi vị trí giữa hai phần tử
- Push(): Thêm phần tử vào heap
- Pop(): Lấy phần tử nhỏ nhất (root của heap)
- HeapifyUp(): Duy trì Heap sau khi thêm phần tử
- HeapifyDown(): Duy trì Heap sau khi xóa phần tử

** như trên **

Chia mảng dữ liệu thành các Runs sao cho mỗi kích thước tối đa mỗi Runs bằng với MemorySize của thiết bị

```
function divideRuns(arr, maxMemory)
{
  let runs = [];
  let count = 0;
  let run = [];
  for(let i = 0; i < arr.length; i++)
  {
      if(count === maxMemory)
      {
          runs.push(run);
          run = [];
          count = 0;
      }
      run.push(arr[i]);
      count++;
  }
  if(run.length > 0)
  {
      runs.push(run);
  }
  return runs;
}
```

Tạo một hàm QuickSort để sắp xếp mỗi runs

```
function quickSort(arr)
{
  if(arr.length <= 1)
  {
      return arr;
  }
  let pivot = arr[0];
  let left = [];
  let right = [];
  for(let i = 1; i < arr.length; i++)
  {
      if(arr[i] < pivot)
      {
          left.push(arr[i]);
      }
      else
      {
          right.push(arr[i]);
      }
  }
  return quickSort(left).concat(pivot,
quickSort(right));
}
```

Trộn các dãy con đã được sắp xếp

- Duyệt qua từng run và đưa phần tử đầu tiên của mỗi run vào MinHeap
- Lấy phần tử nhỏ nhất cảu MinHeap và lưu vào result
- Nếu run đó còn phần tử, chèn tiếp vào Heap

```
function mergeRuns(runs)
{
  let minHeap = new MinHeap();
  let result = [];
```

```
  // Đưa phần tử đầu tiên của mỗi run vào heap
  for (let i = 0; i < runs.length; i++)
  {
      if (runs[i].length > 0)
      {
          minHeap.push({ value: runs[i][0], runIndex: i, pos: 0
});
      }
  }

  // Xử lý từng phần tử nhỏ nhất trong heap
  while (!minHeap.isEmpty())
  {

      let { value, runIndex, pos } = minHeap.pop();
      result.push(value);

      // Nếu run đó còn phần tử, chèn tiếp vào heap
      if (pos + 1 < runs[runIndex].length)
      {
          minHeap.push({ value: runs[runIndex][pos + 1],
runIndex, pos: pos + 1 });
      }
  }

  return result;
}
```

Hàm gọi:

```
function multiwayBalanceMergeSort(arr, maxMemory)
{
  let runs = divideRuns(arr, maxMemory);

  for (let i = 0; i < runs.length; i++)
  {
      runs[i] = quickSort(runs[i]);
  }
```

```
  if (runs.length === 1) return runs[0];

  return mergeRuns(runs);
}
```

4. KẾT LUẬN

Bài báo cáo trên đã mô tả chi tiết về thuật toán, đặc biệt tập trung vào hai phương pháp chính:

- Trộn Tự Nhiên (Natural Merge Sort): Nhiên tận dụng các dãy con đã sắp xếp tự nhiên trong dữ liệu gốc
- Trộn Đường Cân Bằng (Balanced Multiway Merge Sort): chia dữ liệu thành nhiều phần cố định để trộn đồng thời, giúp tối ưu số lần đọc/ghi từ bộ nhớ ngoài.

Việc lựa chọn thuật toán External Sort phù hợp phụ thuộc vào cấu trúc dữ liệu đầu vào, dung lượng bộ nhớ khả dụng và yêu cầu về hiệu suất. Trong thực tế, các hệ thống xử lý dữ liệu lớn thường áp dụng kết hợp nhiều chiến lược trộn để tối ưu hóa tốc độ sắp xếp.

5. TÀI LIỆU THAM KHẢO

https://www.geeksforgeeks.org/external-sorting

https://www.happycoders.eu/algorithms/merge-sort/#:~:text=Natural%20Merge%20Sort%20is%20an,sorted%20in%20O(n)/

https://en.wikipedia.org/wiki/External_sorting

https://github.com/valeriodiste/ExternalMergeSortVisualizer