[PHẦN I 2](#_Toc88035523)

[Chương 1: Cài đặt một số thuật toán Sắp xếp 2](#_Toc88035524)

[**1. Mô tả chương trình** 2](#_Toc88035525)

[**2. Các thuật toán Sắp xếp** 2](#_Toc88035526)

[**2.1. Sắp xếp Nổi bọt (Bubble Sort)** 2](#_Toc88035527)

[**2.2. Sắp xếp Lựa chọn (Selection Sort)** 3](#_Toc88035528)

[**2.3. Sắp xếp Vun đống (Heap Sort)** 3](#_Toc88035529)

[**2.4. Sắp xếp Nhanh (Quick Sort)** 4](#_Toc88035530)

[**2.5. Sắp xếp Trộn (Merge Sort)** 4](#_Toc88035531)

[**3. Kết quả chạy chương trình** 5](#_Toc88035532)

[**4. Đánh giá thuật toán** 5](#_Toc88035533)

[Chương 2: Cài đặt thuật toán giải một số bài toán 7](#_Toc88035534)

[**1. Giải bài toán Mã đi tuần bằng thuật toán Quay lui** 7](#_Toc88035535)

[**1.1. Phát biểu bài toán** 7](#_Toc88035536)

[**1.2. Mô tả chương trình** 8](#_Toc88035537)

[**1.3. Thuật toán sử dụng** 8](#_Toc88035538)

[**1.4. Kết quả chạy chương trình** 9](#_Toc88035539)

[**2. Giải bài toán Người du lịch bằng thuật toán Vét cạn và Quay lui nhánh cận.** 10](#_Toc88035540)

[**2.1. Phát biểu bài toán** 10](#_Toc88035541)

[**2.2. Mô tả chương trình** 11](#_Toc88035542)

[**2.3. Thuật toán sử dụng** 11](#_Toc88035543)

[**2.4. Kết quả chạy chương trình** 12](#_Toc88035544)

[**2.5. Đánh giá thuật toán** 13](#_Toc88035545)

[Chương 3: Lập bảng chỉ mục cho file văn bản 15](#_Toc88035546)

[**1. Mô tả chương trình** 15](#_Toc88035547)

[**2. Các cấu trúc dữ liệu sử dụng** 15](#_Toc88035548)

[**2.1. Danh sách liên kết** 15](#_Toc88035549)

[**2.2. Mảng băm và danh sách liên kết** 15](#_Toc88035550)

[**2.3. Cây nhị phân tìm kiếm** 15](#_Toc88035551)

[**3. Kết quả chạy chương trình** 15](#_Toc88035552)

[**4. Đánh giá các cấu trúc dữ liệu** 15](#_Toc88035553)

# **PHẦN I**

## **Chương 1: Cài đặt một số thuật toán Sắp xếp**

### **1. Mô tả chương trình**

Từ dữ liệu đầu vào là một mảng số nguyên bất kỳ với n phần tử (n xác định), người dùng lựa chọn thuật toán và chương trình xử lý sắp xếp các phần tử trong mảng theo thứ tự tăng dần. Cuối cùng, chương trình đưa ra đầu ra.

Yêu cầu chương trình:

* Đầu vào dữ liệu:
  + Nhập vào từ bàn phím
  + Đọc từ tệp văn bản (.txt)
  + Sinh ngẫu nhiên
* Các thuật toán sử dụng:
  + Sắp xếp Nổi bọt (Bubble Sort)
  + Sắp xếp Lựa chọn (Selection Sort)
  + Sắp xếp Vun đống (Heap Sort)
  + Sắp xếp Nhanh (Quick Sort)
  + Sắp xếp Trộn (Merge Sort)
* Đầu ra dữ liệu:
  + In ra màn hình
  + Ghi ra tệp văn bản (.txt)

### **2. Các thuật toán Sắp xếp**

#### **2.1. Sắp xếp Nổi bọt (Bubble Sort)**

Mô tả thuật toán:

* Trong mỗi vòng lặp, duyệt lần lượt các cặp phần tử trong nhóm các phần tử chưa đúng thứ tự. Nếu cặp này ở thứ tự sai (phần tử đứng trước lớn hơn) thì đổi chỗ cho nhau.
* Cuối mỗi vòng lặp, phần tử lớn nhất trong nhóm các phần tử vừa xét sẽ đưa về cuối nhóm, phần tử này đã ở vị trí chính xác. Sau đó tiếp tục vòng lặp tiếp theo cho đến khi tất cả phần tử đã đúng vị trí.

Độ phức tạp thời gian: O(n2).

Đánh giá:

* Thuật toán ngắn gọn, đơn giản, dễ cài đặt.
* Kém hiệu quả về mặt thời gian.

#### **2.2. Sắp xếp Lựa chọn (Selection Sort)**

Mô tả thuật toán:

* Trong mỗi vòng lặp, duyệt lần lượt các phần tử trong nhóm các phần tử chưa đúng thứ tự, tìm ra phần tử có giá trị nhỏ nhất trong nhóm này.
* Cuối vòng lặp, ta đổi chỗ phần tử này cho phần tử đầu tiên trong nhóm, khi đó phần tử này đã ở vị trí chính xác. Sau đó tiếp tục vòng lặp tiếp theo cho đến khi tất cả phần tử đã đúng vị trí.

Độ phức tạp thời gian: O(n2).

Đánh giá:

* Thuật toán ngắn gọn, đơn giản, dễ cài đặt.
* Giảm số cặp phải đổi vị trí so với sắp xếp Nổi bọt.
* Kém hiệu quả về mặt thời gian.

#### **2.3. Sắp xếp Vun đống (Heap Sort)**

Mô tả thuật toán:

* Thuật toán này dựa trên đặc điểm của đống cực đại (max-heap) trong cây nhị phân hoàn thiện (Complete Binary Tree).
  + Max-heap là cây nhị phân mà mỗi nút cha sẽ luôn không nhỏ hơn 2 con của nó. Khi đó giá trị nút lớn nhất sẽ là nút gốc của cây.
  + Với cây nhị phân hoàn thiện, ta đánh số thứ tự từ 0 với các nút tính từ gốc tăng dần theo độ cao, từ trái sang phải. Ở một nút thứ tự i bất kỳ, thứ tự nút con trái của nó là (2i+1), nút con phải của nó là (2i+2). Khi lần lượt lưu trữ các nút vào mảng, chỉ số sẽ tương ứng giá trị trên.
  + Khi một nút có 2 nhánh con là các max-heap, để cây từ nút này trở đi trở thành max-heap, ta lần lượt so sánh nút cha này với 2 con của nó, nếu nút cha có giá trị nhỏ hơn nút con lớn nhất thì đổi chỗ 2 nút cho nhau, sau đó lặp lại việc sắp xếp theo nhánh đó, đến khi nút cha ban đầu thỏa mãn max-heap. Quá trình này sẽ là hàm heapify().
* Đầu tiên, ta biến đổi cây nhị phân thành một max-heap bằng việc lần lượt heapify() các nhánh cây nhị phân có độ cao lớn, giảm dần cho tới nút gốc. Giá trị nút lớn nhất sẽ ở gốc.
* Sau đó, lặp lại việc chuyển nút gốc xuống cuối, loại ra khỏi cây nhị phân trong lần lặp tiến theo và heapify() từ nút gốc mới.

Độ phức tạp thời gian: O(n.*log*n).

Đánh giá:

* Hiệu quả cao về mặt thời gian.
* Để cài đặt đòi hỏi kiến thức về thuật toán với Cây nhị phân.

#### **2.4. Sắp xếp Nhanh (Quick Sort)**

Mô tả thuật toán

* Đây là một thuật toán chia để trị, chia đôi nhóm cần sắp xếp qua một điểm (pivot), bên trái là các giá trị nhỏ hơn pivot, bên phải là các giá trị lớn hơn pivot, sau đó tiếp tục lặp lại với 2 nhóm nhỏ hơn đó.
* Khi kích thước mỗi phía của nhóm cần sắp xếp không lớn hơn 1, nhóm này đã được sắp xếp đúng thứ tự, ta không cần chia đôi nữa.
* Trong cách giải này, quá trình chia đôi các nhóm cần sắp xếp thể hiện trong hàm partition(). Chọn pivot là phần tử đầu tiên trong nhóm, lần lượt thực hiện các vòng lặp sau:
  + Với các phần tử bên trái (tính từ sau pivot), lần lượt dịch một con trỏ (left) sang phải cho đến khi gặp phần tử lớn hơn pivot.
  + Với các phần tử bên phải, lần lượt dịch một con trỏ khác (right) sang trái cho đến khi gặp phần tử nhỏ hơn pivot.
  + Phần tử nằm ở hai con trỏ hiện tại nằm nhầm phía, nên ta đổi chỗ hai phần tử này cho nhau.
  + Tiếp tục vòng lặp đến khi 2 con trỏ vượt qua nhau.
  + Sau khi dừng, ta chuyển pivot vào vị trí giữa hai phía (đổi chỗ cho phần tử nằm ở con trỏ right).

Độ phức tạp thời gian: O(n.*log*n).

Đánh giá:

* Hiệu quả cao về mặt thời gian.
* Để cài đặt đòi hỏi kiến thức về thuật toán.

#### **2.5. Sắp xếp Trộn (Merge Sort)**

Mô tả thuật toán

* Đây là thuật toán chia để trị, chia đôi nhóm cần sắp xếp thành 2 nhóm con, sắp xếp riêng 2 nhóm con này theo thuật toán, sau đó trộn 2 nhóm con theo thứ tự.
* Khi nhóm con cần sắp xếp có kích thước là 1, nhóm đã được sắp xếp đúng thứ tự, ta không cần chia đôi nữa.
* Trong cách giải này, quá trình trộn 2 nhóm con đã được sắp xếp thể hiện trong hàm merge():
  + Lần lượt ghi giá trị nhỏ hơn trong 2 phần tử đầu của 2 nhóm con vào cha, loại phần tử này khỏi nhóm con và lặp lại bước trên.
  + Khi một trong 2 nhóm con đã được ghi hết, ta ghi toàn bộ phần còn lại của nhóm con kia vào cha.

Độ phức tạp thời gian: O(n.*log*n).

Đánh giá:

* Hiệu quả cao về mặt thời gian.
* Để cài đặt đòi hỏi kiến thức về thuật toán.
* Khi thực hiện trộn phải phát sinh thêm bộ nhớ

### **3. Kết quả chạy chương trình**

Đây là minh họa cho chương trình khi chạy thử:

Text

Description automatically generated

### **4. Đánh giá thuật toán**

Ta sẽ so sánh các thuật toán sắp xếp qua các đồ thị dưới đây:

* Trục hoành thể hiện kích thước đầu vào, được đo bằng logarit cơ số 10 của kích thước dữ liệu vào.
* Trục tung thể hiện thời gian thực hiện, đơn vị giây.

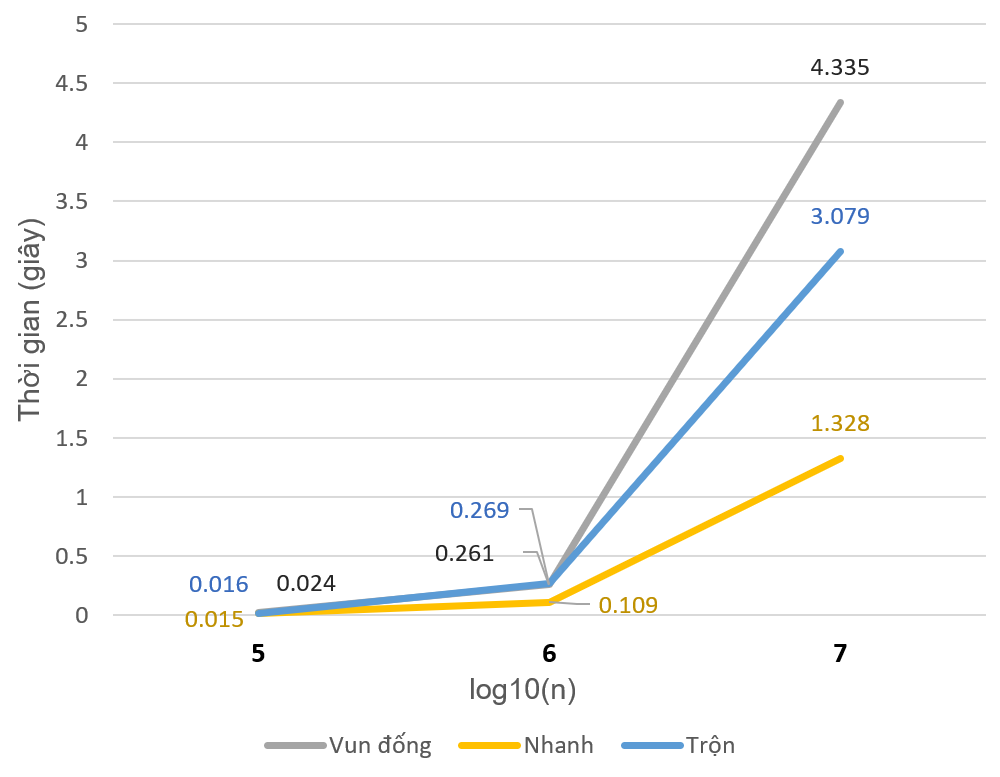
Các loại thuật toán có độ phức tạp khác nhau nên thời gian xử lý sẽ có sự chênh lệch rất lớn, vì vậy chia ra làm hai đồ thị mà mỗi đồ thì thuật toán có cùng độ phức tạp. Các thuật toán trong cùng một đồ thị sẽ sử dụng cùng một dữ liệu đầu vào.

Ở đây, ta sẽ chỉ đo duy nhất thời gian thuật toán thực hiện sắp xếp, không đo thời gian nhập xuất dữ liệu. Các kết quả ở đây có tính chính xác phụ thuộc vào máy tính đo.

Chart, line chart

Description automatically generated

So sánh giữa sắp xếp Nổi bọt và sắp xếp Lựa chọn với cùng độ phức tạp thời gian O(n2). Có thể thấy sắp xếp Lựa chọn giảm bớt quá trình đổi chỗ giữa các thành phần so với sắp xếp Nổi bọt nên hiệu quả thời gian tăng dần khi dữ liệu càng lớn.



So sánh giữa 3 thuật toán có cùng độ phức tạp thời gian O(n.*log*n): sắp xếp Vun đống, sắp xếp Nhanh, sắp xếp Trộn. Có thể thấy dựa trên bảng là sắp xếp Nhanh có hiệu quả thời gian cao nhất. Điều này có thể đến do trong các mỗi lần đệ quy, việc xử lý (sử dụng qua hàm phụ trợ) của sắp xếp Nhanh là đơn giản nhất.

## **Chương 2: Cài đặt thuật toán giải một số bài toán**

### **1. Giải bài toán Mã đi tuần bằng thuật toán Quay lui**

#### **1.1. Phát biểu bài toán**

Mã đi tuần là bài toán về việc di chuyển một quân mã trên bàn cờ vua (8×8). Quân mã được đặt ở một ô trên một bàn cờ trống nó phải di chuyển theo quy tắc của cờ vua để đi qua mỗi ô trên bàn cờ đúng một lần.

#### **1.2. Mô tả chương trình**

Khi bắt đầu, chương trình sẽ yêu cầu người dùng nhập vào một vị trí xuất phát của quân mã. Từ vị trí xuất phát này, thuật toán Quay lui sẽ tìm đến khi ra lời giải đường đi thỏa mãn yêu cầu. Khi đó, chương trình sẽ in ra bàn cờ 8×8, lần lượt hiển thị từng bước đi trong lời giải.

#### **1.3. Thuật toán sử dụng**

Bài toán Mã đi tuần sẽ được giải bằng thuật toán Quay lui.

Quay lui là một kĩ thuật thiết kế giải thuật dựa trên đệ quy. Ý tưởng của quay lui là giải quyết vấn đề bắt đầu từ lời giải rỗng và xây dựng dần lời giải bộ phận (partial solution) để ngày càng tiến gần tới lời giải bài toán. Nếu một lời giải bộ phận không thể tiếp tục phát triển, ta sẽ bỏ nó và quay sang xét tiếp các ứng cử viên khác. Bản chất của quay lui là một quá trình tìm kiếm theo chiều sâu (Depth-First Search).

Thuật toán có mã giả tổng quát như sau:



Cụ thể, áp dụng với cách giải bài toán Mã đi tuần:

* Đầu tiên, ta kiểm tra điều kiện thành công (điều kiện dừng) của bài toán là k = 64, vì để đi hết mọi ô trong bàn cờ ta chỉ cần dùng 63 nước đi. Khi xét tới bước thứ 64 tức là đã hoàn thành 63 nước trước đó, thuật toán dừng lại và in ra kết quả. Ở đây điều kiện dừng được đưa ra ngoài vòng lặp vì chỉ phụ thuộc vào k chứ không phụ thuộc vào phương án đi.
* Với một bước đi bất kỳ (có số thứ tự k < 64), ta sẽ tạo vòng lặp cho 8 phương án di chuyển có thể của quân mã theo luật cờ vua.
* Điều kiện chấp nhận phương án là nước đi tiếp theo nằm trong phạm vi bàn cờ và vị trí này chưa từng đi qua.
* Khi điều kiện thỏa mãn, ta sẽ ghi lại vị trí nước đi này và đánh dấu đã đi qua trên bàn cờ, và tiến tới đệ quy cho bước đi tiếp theo.
* Sau khi thực hiện đệ quy xong mà không đạt được điều kiện dừng, ta xóa nước đi này, trả lại vị trí đã đánh dấu và chuyển sang phương án khác.

#### **1.4. Kết quả chạy chương trình**

Đây là minh họa cho chương trình khi chạy thử:

Text

Description automatically generated

Calendar

Description automatically generated

### **2. Giải bài toán Người du lịch bằng thuật toán Vét cạn và Quay lui nhánh cận.**

#### **2.1. Phát biểu bài toán**

Một người du lịch muốn đi tham quan n thành phố T1,T2,…, Tn . Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua duy nhất 1 lần rồi quay trở lại thành phố xuất phát.

Gọi Cij là chi phí đi từ thành phố Ti đến Tj . Hãy tìm một hành trình thỏa yêu cầu bài toán sao cho chi phí là nhỏ nhất.

#### **2.2. Mô tả chương trình**

Từ dữ liệu đầu vào là ma trận chi phí với n thành phố, thuật toán sẽ tìm lời giải là một cách di chuyển với chí phí nhỏ nhất. Cuối cùng, chương trình in ra màn hình chi phí nhỏ nhất và lời giải trên.

Yêu cầu chương trình:

* Đầu vào dữ liệu:
  + Nhập vào từ bàn phím
  + Đọc từ tệp văn bản (.txt)
  + Sinh ngẫu nhiên
* Thuật toán sử dụng:
  + Thuật toán Vét cạn (Brute Force)
  + Thuật toán Quay lui nhánh cận (Branch and Bound)

#### **2.3. Thuật toán sử dụng**

##### *2.3.1. Thuật toán Vét cạn*

Vét cạn là thuật toán dựa trên ý tưởng liệt kê ra mọi phương án có thể của lời giải và xét xem phương án nào thỏa mãn yêu cầu của đề bài.

Với bài toán Người đi du lịch, ta sẽ liệt kê ra mọi phương án hoán vị trong cách di chuyển qua tất cả các thành phố, tính chi phí di chuyển của phương án và so sánh với chi phí tối ưu tại thời điểm đó. Phương án nào cho ra chi phí nhỏ nhất sẽ là lời giải của bài toán.

Ở đây, ta cài đặt thuật toán tìm ra hoán vị tiếp theo của một phương án di chuyển. Xét với một mảng n phần tử chứa giá trị đại diện các thành phố, chỉ số của phần tử là số thứ tự của bước đi đến phần tử đó. Khởi điểm từ phương án giá trị các phần tử tăng dần, tại một phương án bất kỳ, ta có mã giả sau:



Các quá trình được mô tả như sau:

* Đầu tiên, ta tìm ra chỉ số lớn nhất trong mảng (đặt là i) mà giá trị phần tử tại nó nhỏ hơn giá trị phần tử sau đó.
* Nếu không tìm được chỉ số i, tức là giá trị các phần tử được sắp xếp theo thứ tự giảm dần, không thể tìm ra hoán vị tiếp theo. Ở đây, ta không tìm hoán vị với điểm xuất phát, nên khi i = 0, ta sẽ dừng tìm hoán vị.
* Ta tìm chỉ số lớn nhất (đặt là j) mà giá trị phần tử tại nó lớn hơn giá trị phần tử tại i. Sau đó ta đổi chỗ phần tử tại i với phần tử tại j.
* Từ sau vị trí i ở trên, ta đảo ngược lại giá trị phần tử của mảng. Ta đã sinh ra được hoán vị tiếp theo.

Độ phức tạp thời gian: O(n!).

##### *2.3.2. Thuật toán Quay lui nhánh cận*

Trong phương pháp Quay lui nhánh cận, về các cài đặt tương đối giống cài đặt thuật toán Quay lui thông thường. Tuy nhiên, trước đi thực hiện đệ quy, thuật toán Quay lui nhánh cận sẽ kiểm tra xem phương án này có khả năng cho ra kết quả tốt hơn kết quả hiện tại hay không. Nếu điều này không thể xảy ra, ta sẽ chuyển sang phương án khác.

Cụ thể, với bài toán Người đi du lịch:

* Ta lưu thêm một giá trị bằng với chi phí nhỏ nhất trong đồ thị di chuyển.
* Đầu tiên, ta kiểm tra điều kiện dừng là k = n, vì khi thực hiện bước đi thứ n, ta đã đi qua hết n thành phố và hiện tại quay trở về điểm xuất phát. Ta sẽ kiểm tra chi phí đã bỏ ra có thật sự nhỏ hơn chi phí tối ưu không. Nếu có, ta cập nhật lại chi phí tối ưu và ghi lại lịch trình.
* Khi k < n, ta sẽ lần lượt thử các phương án là các thành phố trong danh sách (trừ thành phố xuất phát), nếu thành phố đã từng đi qua thì chuyển sang phương án tiếp theo.
* Nếu thành phố chưa từng đi qua thì ta sẽ ghi lại, đánh dấu thành phố đã đi và cập nhật chi phí hiện tại.
* Sau đó, ta sẽ kiểm tra xem giả sử toàn bộ bước đi còn lại có chi phí nhỏ nhất, thì tổng chi phí dự kiến có nhỏ hơn chi phí tối ưu hay không, nếu có ta mới tiếp tục đệ quy sang bước di chuyển tiếp theo.
* Cuối cùng, sau quá trình trên, ta xóa đánh dấu, trả lại chi phí và chuyển sang phương án tiếp theo.

#### **2.4. Kết quả chạy chương trình**

Đây là minh họa cho chương trình khi chạy thử:

Text

Description automatically generated

#### **2.5. Đánh giá thuật toán**

Ta sẽ biểu diễn thời gian đo theo hai đồ thị dưới đây:

* Trục hoành thể hiện số thành phố (kích thước dữ liệu vào).
* Trục tung thể hiện thời gian thực hiện, đơn vị giây.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Từ hai đồ thị trên, có thể dễ dàng thấy rằng thuật toán Quay lui nhánh cận cho hiệu quả thời gian cao hơn rất nhiều so với thuật toán Vét cạn. Trong bài toán này, thuật toán Vét cạn luôn thực hiện với thời gian tồi nhất là O(n!). Trong khi đó thuật toán Quay lui nhánh cận dù lý thuyết tồi hơn nhưng khi đã cắt bỏ nhiều nhánh lời giải không cần thiết, việc tính toán giảm tải, hiệu quả hơn rất nhiều.

## **Chương 3: Lập bảng chỉ mục cho file văn bản**

### **1. Mô tả chương trình**

### **2. Các cấu trúc dữ liệu sử dụng**

#### **2.1. Danh sách liên kết**

#### **2.2. Mảng băm và danh sách liên kết**

#### **2.3. Cây nhị phân tìm kiếm**

### **3. Kết quả chạy chương trình**

### **4. Đánh giá các cấu trúc dữ liệu**