1. Thuật toán:
   1. **Định nghĩa**

* Thuật toán là tập hợp các chỉ dẫn đầu vào để giải quyết bài toán

+ Có đầu vào hợp lệ

+ Tạo ra đầu ra

+ Kết thúc trong một thời gian có hạn

* Một thuật toán là một bản liệt kê các chỉ dẫn, các quy tắc cần thực hiện theo từng bước xác định nhằm giải quyết một bài toán đã cho trong một khoảng thời gian hữu hạn.
  1. **Tính chất**
* Tính xác định: Các bước của thuật toán phải được xác định một cách chính xác, các chỉ dẫn phải rõ ràng, có thể thực hiện được
* Tính hữu hạn: Thuật toán phải kết thúc sau một số hữu hạn các bước.
* Tính đúng đắn: Thuật toán phải cho kết quả đúng theo yêu cầu của bài toán đặt ra.
* Tính tổng quát: Thuật toán phải áp dụng được cho mọi bài toán cùng loại, với mọi dữ liệu đầu vào như đã được mô tả.
  1. **Cách biểu diễn.**
* Mô tả các bước thực hiện: có thể sử dụng ngôn ngữ tự nhiên hoặc ngôn ngữ tự nhiên kết hợp với ngôn ngữ lập trình, tức là giả mã
* Sử dụng lưu đồ( sơ đồ) giải thuật: sử dụng các ký hiệu hình khối cơ bản để tạo thành một mô tả mang tính hình thức của thuật toán

1. Phân tích thuật toán, hiệu quả và độ phức tạp của thuật toán:
   1. **Hướng tiếp cận và cách đánh giá:**

* **Hướng tiếp cận**:

Bước 1: Xác định bài toán

* Làm rõ yêu cầu đặt ra của bài toán

Bước 2: Phân tích bài toán và lựa chọn cách giải quyết

* Với một bài toán đặt ra thường có nhiều cách giải quyết
* Mỗi cách giải quyết đều có thể khác nhau về thời gian thực hiện, dung lượng bộ nhớ, độ chính xác có thể đạt được.
* Tùy theo yêu cầu và điều kiện lựa chọn cách tiếp cận cho thích hợp

Bước 3: Xây dựng thuật toán

Trên cơ sở mô hình đã xây dựng được từ hai bước trước, chi tiết hóa các bước thực hiện hoặc lựa chọn một trong các thuật toán đã biết phù hợp với bài toán

Bước 4: Viết chương trình theo thuật toán đã lựa chọn

* Dựa vào thuật toán đã được xây dựng, lựa chọn ngôn ngữ lập trình phù hợp với thuật toán và dữ liệu để thiết kế chương trình

Bước 5: Xây dựng chương trình, thử nghiệm, triển khai

* **Đánh giá**:
* Phân tích trực tiếp để tính số thao tác:

Phân tích trực tiếp đoạn mã và sử dụng các kỹ thuật: phép đếm, tính tổng hữu hạn. xét dấu hàm,…phép toán cơ bản trong các đoạn mã giả thường là gán và so sánh, tuy nhiên đánh giá theo phương pháp này lại không giải quyết được tất cả các trường hợp

* Phương pháp đệ quy:

Đối với đoạn chương trình có gọi chương trình con: thành lập phương trình đệ quy, sau đó giải phương trình đệ quy. Nghiệm của phương trình này là thời gian thực hiện chương trình

* Đánh giá bằng thực nghiệm:

Cho thêm lệnh đếm trong đoạn mã, phát sinh dữ liệu thực thi đoạn mã, ghi xuống file, dùng excel vẽ đồ thị tính phương sai, độ lệch chuẩn -> ước lượng độ phức tạp

* 1. **Đánh giá O lớn, các lớp độ phức tạp của thuật toán**

Cho hai hàm số f và g, f: R→R, g: R→R, khi bàn đến sự so sánh độ tăng của 2 hàm f(x) và g(x) khi x →+∞

* Ta nói rằng f(x) là O-lớn của g(x) khi x →+∞.
* Kí hiệu f(x) = O(g(x)) hoặc đôi khi viết f(x) là O(g(x)) nếu như tồn tại hai hằng số C >0 và N >0 sao cho với mọi x > N thì |f(x) | ≤ C.|g(x)|
* Ví dụ:
* f= x^2+2x+3

=> f=O(x^2) vì f<=x^2+2x^2+3x^2=6x^2

* Một số độ phức tạp:
* O(1) Độ phức tạp hằng số.
* O(log n) Độ phức tạp logarit.
* O( n) Độ phức tạp tuyến tính.
* O( ) Độ phức tạp đa thức.
* O( nlog n) Độ phức tạp nlogn.
* O(),b>1 Độ phức tạp hàm mũ
* O(n!) Độ phức tạp giai thừa
  1. **Các kỹ thuật đánh giá độ phức tạp thuật toán**
* Tính hiệu quả của thuật toán

Một giải thuật được gọi là hiệu quả nếu nó tiết kiệm được không gian và thời gian

* Thời gian thực hiện thuật toán

+ Xem xét kích thước đầu vào

+ Phép tính cơ bản được sử dụng: các phép tính số học, logic, đổi chỗ, gán…

+ Giá trị C(n)

+ Giá trị của O

1. **Thiết kế thuật toán theo phương pháp trực tiếp (Brute Force & Exhaustive Search)  
   a. Modul hóa và phân tích top-down**

* Chia bài toán thành các bài toán nhỏ hơn: coi bài toán cần giải quyết là module chính thì ta chia thành các module nhỏ hơn, tức là các bài toán con. Đây chính là quá trình module hóa.
* Phương pháp phân tích top-down là quá trình phân tích bài toán được thực hiện từ trên xuống dưới. Từ mức tổng quát là các ý tưởng giải quyết, các bước để giải quyết bài toán cho đến mức chi tiết là các câu lệnh trong chương trình. Quá trình phân rã được thực hiên theo từng mức khác nhau
* Mức có chỉ số thấp nhất (đầu tiên) được gọi là mức tổng quan, ở mức tổng quan có thể xem xét tổng thể lời giải của bài toán thống qua các nhiệm vụ chính
* Tiếp tục cho đến khi thành các hàm đơn thể, trong đó mọi công việc được hình dung khá rõ ràng và xác định.

**b. Chiến lược tiếp cận và hướng giải quyết**

* Từ bài toán đã cho xem xét những tính chất đặc trưng của nó -> xác định mối liên hệ giữa đầu vào và đầu ra

+ Xác định đầu vào là gì?, đầu ra là gì?

+ Xác định mối liên hệ gồm: các công thức, khái niệm, quy luật…

* Sử dụng các công cụ biểu diễn để mô tả chương trình từ đơn giản đến phức tạp
* Từng bước chi tiết cho đến chương trình trên ngôn ngữ lập trình

1. **Phương pháp giảm và trị (decrease-and-conquer)  
   a. Lược đồ chung**

* Kỹ thuật thiết kế giải thuật giảm để trị lợi dụng mối liên hệ giữa lời giải cho một thể hiện của một bài toán và lời giải cho một thể hiện nhỏ hơn của cùng một bài toán.
* **Có 3 phương pháp giảm để trị:**

+ Giảm bởi một hằng số

Kích thước của một đối tượng được giảm xuống theo cùng một hằng số trên mỗi lần lặp lại thuật toán

+ Giảm bởi một hệ số

+ Giảm kích thước của biến: ứng dụng cho bài toán tìm ước chung lớn nhất gcd(m,n) = gcd(n, m mod n)

**b. Các bài toán áp dụng**

**1. Bài toán Sắp xếp chèn( Insertion Sort)**

- Sắp xếp một mảng a[0….n-1]

- Ta giả sử rằng mảng a[0…n-2] đã được sắp xếp

-> Chèn phần tử a[n-1] vào mảng con đã có thứ tự a[0…n-2], bằng cách:

Duyệt mảng con đã có thứ tự từ phải sang trái hoặc từ trái sang phải cho đến khi tìm thấy được phần tử đầu tiên nhỏ hơn/lớn hơn với phần tử a[n-1] => Tiến hành chèn phần tử a[n-1] vào bên phải/bên trái phần tử này.

* ***Cworst*(*n*) = *n*(*n*-1)/2 ∈ Θ(*n*2)**
* **Cavg(*n*) ≈ *n*2/4 ∈ Θ(*n*2)**
* ***Cbest*(*n*) = *n* - 1 ∈ Θ(*n*)** **(also fast on almost sorted arrays)**

**2. Duyệt đồ thị theo chiều rộng (BFS), chiều sâu (DFS)**

**2.1. Duyệt đồ thị theo chiều sâu (DFS)**

**-** Từ một đỉnh nào đó chưa thăm, thăm , rồi tìm đỉnh (chưa thăm) kề với , thăm …Thuật toán lặp lại việc thăm cho tới khi tất cả các đỉnh đều được thăm

- Nếu tại một đỉnh nào đó, không còn đỉnh nào kề với là chưa thăm thì quay trở lại tiếp tục tìm đỉnh kề chưa thăm khác của

- Thuật toán:

Procedure DFS(v);

(\*Tìm kiếm theo chiều sâu bắt đầu từ đỉnh v;

Các biến Chuaxet, Ke là biến toàn cục\*)

Begin

Tham\_dinh(v);

Chuaxet[v]:=false;

For u Є Ke(v) do

If Chuaxet[u] then DFS(u);

End; (\*đỉnh v đã duyệt xong\*)

Khi đó, tìm kiếm theo chiều sâu trên đồ thị được thực hiện nhờ thuật toán sau:

Begin (\*Khởi tạo tất cả các đỉnh của đồ thị\*)

for v Є V do

Chuaxet[v]:=true;

for v Є V do

If Chuaxet[v] then DFS(v);

End.

**2.2. Duyệt đồ thị theo chiều rộng( BFS)**

**-** Từ đỉnh v nào đó chưa thăm, thăm v, cất tất cả các đỉnh u (chưa thăm) kề với v vào hàng đợi. Lấy từ hàng đợi một đỉnh u, thăm u rồi lại cất tất cả các đỉnh t (chưa thăm) kề với u vào hàng đợi…

- Thuật toán lặp lại việc thăm cho tới khi hàng đợi rỗng

- Nếu tại một đỉnh x nào đó, không còn đỉnh nào kề với x là chưa thăm thì quay trở lại tiếp tục tìm đỉnh kề chưa thăm khác của y (y là đỉnh trước khi đến x)

- Thuật toán:

Procedure BFS(v);

(\*BFS bat dau tu dinh v, cac bien Chuaxet, Ke la bien cuc bo\*)

Begin

QUEUE:=Ø;

QUEUE⇐ v; (\*kết quả nạp vào hàng đợi QUEUE\*)

Chuaxet[v]:=false;

While QUEUE<> Ø do

Begin

p ⇐ QUEUE; (\*lấy p từ hàng đợi QUEUE:\*)

Tham\_dinh(p);

For u Є Ke(v) do

If Chuaxet[u] then

Begin

QUEUE ⇐ u;

Chuaxet[u]:=false;

End;

End;

End;

Khi đó, tìm kiếm theo chiều rộng trên đồ thị được thực hiện nhờ thuật toán sau:

Begin

for f Є V do

Chuaxet[v]:=true; (\*Khởi tạo các đỉnh của đồ thị là chưa xét\*)

for v Є V do

if Chuaxet[v] then BFS(v);

End.

**3. Sinh các hoán vị**

* Sinh hoán vị của tập n phần tử, với mỗi hoán vị từ tập con n-1 phần tử (gồm tất cả (n-1)! các hoán vị này), ta đưa phần tử còn lại vào n vị trí khả hữu.
* Như vậy, tổng cộng có n.(n-1)! thao tác chèn phần tử còn lại vào một hoán vị đã có
  1. Phương pháp biến đổi và trị (**transform-and-conquer)**

1. **Lược đồ chung**

* Phương pháp biến đổi và trị là một bước biến đổi, biến thể của bài toán được biến đổi để chuyển sang một dạng dễ dẫn đến lời giải
* Sau đó, tìm ra lời giải cho bài toán
* **Một số dạng biến đổi:**

+ Biến đổi để đưa đến một dạng thể hiện đơn giản hơn của bài toán đã cho (instance simplification)

+ Biến đổi để đưa đến một biểu diễn khác của cùng bài toán (representation change)

+ Biến đổi để đưa đến một thể hiện của một bái toán khác mà đã tổn tại giải thuật (problem reduction)

1. **Các bài toán áp dụng**
2. Bài toán cây AVL

Cây cân bằng là cây nhị phân tìm kiếm, trong đó sự khác biệt về chiều cao giữa cây con bên phải và cây con bên trái không quá 1

1. Giải thuật Gauss: giải hệ phương trình tuyến tính

* Biến đổi hệ thống n phương trình tuyến tính với n biến thành một hệ thống tương đương, tức là có cùng lời giải như hệ phương trình ban đầu với một ma trận tam giác trên, bằng các cách sau:

+ Hoán vị hai phương trình trong hệ thống

+ Thay một phương trình bằng phương trình đó nhân với một hệ số

+ Thay một phương trình với tổng hay hiệu phương trình đó với một phương trình khác được nhân với một hệ số

* Ma trận tam giác trên là ma trận có tất cả các phần tử nằm trên đường chéo chính bằng 0

1. **Giải thuật Horner**

Sử dụng kỹ thuật Biến đổi và trị (representation change) thu được một đa thức mới từ đa thức đã biết, qua đó định trị đa thức

1. HeapSort

* Hàng đợi có độ ưu tiên khác với hàng đợi thông thường ở điển khi lấy phần tử ra khỏi hàng đợi thì đó không phải là phần tử cũ nhất trong hàng đợi mà là phần tử có độ ưu tiên lớn nhất trong hàng đợi
* Thực thi các thao tác với hàng đợi ưu tiên:

+ Thêm một phần tử mới vào cấu trúc

+ Tìm phần tử có độ ưu tiên lớn nhất

+ Xóa bỏ phẩn tử có độ ưu tiên lớn nhất

1. Giải thuật Rabin-Karp

So trùng dòng ký tự: tìm tất cả sự xuất hiện của một khuôn mẫu trong một văn bản

1. Phương pháp chia để trị (divide-and-conquer)  
   **a. Lược đồ chung**

Khi cần giải quyết một bài toán, ta sẽ tiến hành chia bài toán đó thành các bài toán nhỏ hơn, giải các bài toán nhỏ hơn đó, sau đó kết hợp nghiệm của các bài toán nhỏ hơn đó lại thành nghiệm của bài toán ban đầu.

* Chia: bằng cách nào đó chia tập hợp các đối tượng của bài toán thành các bài toán con độc lập. Tiếp tục chia bài toán con cho đến khi có thể giải trực tiếp (không cần, không thể chia nhỏ nữa)
* Trị: trên các bài toán con thực hiện cùng một cách thức, đó là chia nhỏ nếu cần hoặc giải trực tiếp
* Tổng hợp: khi mỗi bài toán con được giải, ta tổng hợp kết quả của các bài toán con để có kết quả của bài toán ban đầu.

**b. Các bài toán áp dụng**

* Sắp xếp trộn: merge sort

+ Chia mảng cần sắp xếp thành 2 nửa

+ Sắp xếp 2 nửa đó một cách đệ quy bằng cách gọi tới thủ tục thực hiện chính merge sort

+ Trộn 2 nửa đã sắp xếp để nhận được một mảng được sắp xếp

* Sắp xếp nhanh: quick sort

+ Chọn 1 phần tử được gọi là phần tử quay (pivot)

+ Phân hoạch: đặt tất cả các phần tử của mảng nhỏ hơn phần tử pivot sang bên trái và tất cả các phần tử lớn hơn phần tử pivot sang bên phải phần tử pivot

+ Đệ quy: gọi tới chính thủ tục sắp xếp nhanh đối với 2 nửa mảng nằm 2 bên phần tử pivot

* Tìm kiếm nhị phân:

+ Xét đoạn mảng arr[left…right] cần tìm kiếm phần tử x. Ta so sánh x với phần tử ở vị trí giữa của mảng(mid = (left + right)/2). Nếu:

+ Nếu phần tử arr[mid] = x. Kết luận và thoát chương trình.

+ Nếu arr[mid] < x. Chỉ thực hiện tìm kiếm trên đoạn arr[mid+1…right].

+ Nếu arr[mid] > x. Chỉ thực hiện tìm kiếm trên đoạn arr[left…mid-1].

+ Tiếp tục thực hiện chia đôi các khoảng tìm kiếm tới khi nào tìm thấy được vị trí của phần tử trong mảng hoặc khi đã duyệt hết mảng

1. Phương pháp tham lam (greedy technique)
2. **Lược đồ chung**

* Lựa chọn giải pháp tốt nhất ở thời điểm hiện tại và sau đó giải bài toán con nảy sinh từ việc thực hiện lựa chọn đó.
* Lựa chọn của giải thuật tham lam có thể phụ thuộc vào các lựa chọn trước đó
* **Tham lam khác với quy hoạch động**:

+ Quy hoạch động duyệt hết và luôn đảm bảo tìm thấy lời giải

+ Tại mỗi bước của thuật toán, quy hoạch động đưa ra quyết định dựa trên các quyết định của bước trước đó và có thể xét lại đường đi của bước trước

+ Giải thuật tham lam không bao giờ xét lại các lựa chọn đã xét trước đó.

* **Tại mỗi lựa chọn đều phải đảm bảo**:

+ Tính khả thi (feasiable): thỏa mãn được các ràng buộc của bài toán

+ Tính tối ưu cục bộ (local optimal): là lựa chọn tốt nhất trong số các lựa chọn khả thi ở bước đó

+ Tính không thể thay đổi (irrevocable): không thay đổi ở bước tiếp theo

* Giải thuật tham lam gồm 5 thành phần:

+ Một tập hợp các ứng viên (candidate), để từ đó tạo ra lời giải

+ Một hàm lựa chọn để theo đó lựa chọn ứng viên tốt nhất để bổ sung vào lời giải

+ Một hàm khả thi (feasiblity) dùng để quyết định nếu một ứng viên có thể được dùng để xây dựng lời giải hay không?

+ Một hàm mục tiêu: ấn định giá trị của lời giải hoặc một lời giải chưa hoàn chỉnh

+ Một hàm đánh giá: chỉ ra khi nào ta tìm ra một lời giải hoàn chỉnh

1. **Bài toán áp dụng:**

A, Prim: tìm cây bao trùm nhỏ nhất của đồ thị

* Nạp dần các đỉnh vào cây khung. Mỗi lần chọn một đỉnh chưa nạp sao cho đỉnh đó kề và gần nhất với các đỉnh đã nạp.
* Prim phụ thuộc vào đỉnh xuất phát của quá trình tìm kiếm
* Prim hiệu quả trong trường hợp đồ thị dày
* Quá trình tìm kiếm đỉnh kế tiếp không được tạo ra một chu trình so với các đỉnh đã chọn
* Tiếp tục cho đến khi thu được cây gồm n đỉnh và n-1 cạnh chính là cây khung nhỏ nhất cần tìm.

B, Kruskal

* Thuật toán sẽ xây dựng tập cạnh T của cây khung nhỏ nhất H=(V,T) theo từng bước.
* Trước hết sắp xếp các cạnh của đồ thị G theo thứ tự không giảm của độ dài. Bắt đầu từ tập T=Æ , ở mỗi bước ta sẽ lần lượt duyệt trong danh sách cạnh đã sắp xếp, từ cạnh có độ dài nhỏ đến cạnh có độ dài lớn hơn, để tìm ra cạnh mà việc bổ sung nó vào tập T gồm n-1 cạnh.
* Kém hiệu quả với những đồ thị dày

C, Dijkstra

\*DFS: cho ta tìm đường đi đến đỉnh ta muốn, có thể chưa phải là ngắn nhất

\*BFS: cũng là tìm đường đi đến đỉnh ta muốn nhưng là đường ngắn nhất có thể

\*Dijkstra: tìm đường đi ngắn nhất từ một đỉnh s đến các đỉnh còn lại của đồ thị và chiều dài (trọng số tương ứng)

- Dijkstra là một thuật toán Greedy, chúng ta sẽ đi tìm con đường ngắn nhất từ đỉnh này đến đỉnh kia

D, Huffman

* Được sử dụng để nén dữ liệu

1. Phương pháp quy hoạch động (dynamic programming)  
   **a. Lược đồ chung**

* Quy hoạch động (Dynamic Programing) là một kỹ thuật thiết kế thuật toán theo kiểu chia bài toán lớn thành các bài toán con, sử dụng lời giải của các bài toán con để tìm lời giải cho bài toán ban đầu.
* **Khác với chia để trị**, quy hoạch động, thay vì gọi đệ quy thì sẽ tìm lời giải của các bài toán con và **lưu vào bộ nhớ** (thường là một mảng), và sau đó lấy lời giải của các bài toán con ở trong mảng đã tính trước để giải bài toán lớn.
* Việc lưu lại lời giải các bài toán con vào bộ nhớ khiến cho ta không phải tính lại lời giải của các bài toán con mỗi khi cần, do đó, tiết kiệm được thời gian tính toán.
* **Gồm các bước**:

+ Phân rã: chia bài toán cần giải thành các bài toán con nhỏ hơn đến mức có thể giải trực tiếp

+ Giải các bài toán con và ghi nhận lời giải: lưu trữ kết quả của các bài toán con vào một mảng để sử dụng về sau

+ Tổng hợp lời giải: tổng hợp lời giải của các bài toán con thành lời giải cho bài toán lớn hơn

+ Tiếp tục cho đến khi thu được lời giải của bài toán lớn

* Mấu chốt của việc giải một bài toán quy hoạch động chính là việc tìm ra công thức truy hồi, hoặc dạng đệ quy của bài toán ( thường là thể hiện tính chất của bài toán từ phương án tổng quát đến cụ thể - top-down), sau đó, ta chỉ việc implement lại công thức đó theo thứ tự ngược lại ( từ trường hợp cụ thể đến tồng quát -bottom-up)  
  **b. Các bài toán áp dụng**

1. **Bài toán Fibonaci**

trong đó

* **Ta có thủ tục đệ quy:**

Recurse(n):

if n<2

return n

else

return Recurse(n-1) + Recurse(n-2)

Giả sử để tính , thủ tục đệ quy sẽ tính lại 3 lần và tính lại 2 lần. Nguyên nhân tính lặp đi lặp lại là sau khi tính bằng hàm Recurse(2), thuật toán không lưu lại giá trị đã tính.

* **Một số ưu điểm của quy hoạch động:**

+ Lưu lại những gì đã tính:

Dùng một mảng F[1,2,….,n] để lưu lại giá trị đã tính, F[i] = . Khi gọi đệ quy, nếu đã được tính trước đó rồi, ta chỉ việc trả lại .

Mem(n):

if n<2

Return n

else

if F[n] is undefined

F[n] ← Mem(n-1) + Mem(n-2)

return F[n]

+ Quy hoạch động:

Thời gian tính được quy về thời gian cần thiết để cập nhật mảng F[1,2,….,n]. Ta có thể cập nhật mảng bằng cách lặp. Thay vì dùng đệ quy có nhớ, ta dùng vòng lặp để cập nhật lời giải các bài toán con và lưu vào bộ nhớ (một mảng).

. Giả mã:

Dynamic(n):

F[0] ← 0; F[1] ← 1

for I ← 2 to n

F[i] ← F[i-1] + F[i-2]

return F[n]

+ Cải tiến, tiết kiệm bộ nhớ:

* **Nhận xét:**
* Quy hoạch động là một kỹ thuật tính toán đệ quy hiệu quả bằng cách lưu trữ kết quả cục bộ
* Trong quy hoạch động, kết quả của các bài toán con thường được lưu vào một mảng

1. **Phương pháp quay lui (backtracking)  
   a. Lược đồ chung**

Thuật toán quay lui (Backtracking) là một kĩ thuật thiết kế giải thuật dựa trên [đệ quy](https://o2.edu.vn/de-quy-la-gi/). Ý tưởng của quay lui là tìm lời giải từng bước, mỗi bước chọn một trong số các lựa chọn khả dĩ và đệ quy.

Bản chất của quay lui là một quá trình tìm kiếm theo chiều sâu(Depth-First Search).

* Ưu điểm: Việc quay lui là thử tất cả các tổ hợp để tìm được một lời giải. Thế mạnh của phương pháp này là nhiều cài đặt tránh được việc phải thử nhiều trường hợp chưa hoàn chỉnh, nhờ đó giảm thời gian chạy.
* Nhược điểm: Trong trường hợp xấu nhất độ phức tạp của quay lui vẫn là cấp số mũ. Vì nó mắc phải các nhược điểm sau:
* Rơi vào tình trạng “thrashing”: quá trình tìm kiếm cứ gặp phải bế tắc với cùng một nguyên nhân.
  + Thực hiện các công việc dư thừa: Mỗi lần chúng ta quay lui, chúng ta cần phải đánh giá lại lời giải trong khi đôi lúc điều đó không cần thiết.
  + Không sớm phát hiện được các khả năng bị bế tắc trong tương lai. Quay lui chuẩn, không có cơ chế nhìn về tương lai để nhận biết được nhánh tìm kiếm sẽ đi vào bế tắc.

**b. Các bài toán áp dụng**

1. **Liệt kê các cấu hình (Sinh xâu nhị phân)**

Mỗi cấu hình được xây dựng bằng cách xác định từng phần tử. Mỗi phần tử lại được chọn bằng cách thử tất cả các khả năng.

Các bước trong việc liệt kê cấu hình dạng X[1…n]:

* Xét tất cả các giá trị X[1] có thể nhận, thử X[1] nhận các giá trị đó. Với mỗi giá trị của X[1] ta sẽ:
* Xét tất cả giá trị X[2] có thể nhận, lại thử X[2] cho các giá trị đó. Với mỗi giá trị X[2] lại xét khả năng giá trị của X[3]…tiếp tục như vậy cho tới bước:
* …
* Xét tất cả giá trị X[n] có thể nhận, thử cho X[n] nhận lần lượt giá trị đó.
* Thông báo cấu hình tìm được.

1. **Bài toán xếp hậu**

Mấu chốt của thuật toán rõ ràng là xét xem có thể đặt quân hậu tiếp theo như thế nào. Theo luật cờ vua, một quân hậu có thế ăn các quẩn khác nếu nằm trên cùng 1 đường, đường này có thể là:

- Hàng

- Cột

- Các đường chéo (đi qua tọa độ vị trí của hậu).

Suy ra rằng mỗi hàng chỉ có thể chứa 1 và chỉ 1 quân hậu. Nên việc chọn vị trí cho quân hậu thứ i có thể giới hạn được ở hàng thứ i. Như thế tham số i trở thành chỉ hàng, và quá trình chọn vị trí cho quân hậu tiến hành trên toàn giá trị có thể có của các cột j.

1. Phương pháp nhánh cận (branch-and-bound)  
   **a. Lược đồ chung**

* Phương pháp quay lui, vét cạn có thể giải các bài toán tối ưu, bằng cách lựa chọn phương án tối ưu trong tất cả các lời giải tìm được. Nhưng nhiều bài toán không gian các lời giải là quá lớn, nên áp dụng phương pháp quay lui khó đảm bảo về thời gian cũng như kỹ thuật. Cho nên ta cần phải cải tiến thuật toán quay lui để hạn chế bớt việc duyệt các phương án. Có nhiều cách cải tiến, trong đó có phương pháp nhánh cận
* Phương pháp nhánh cận là một cải tiến của phương pháp quay lui, dùng để tìm lời giải tối ưu của bài toán.
* Ý tưởng:

+ Trong quá trình duyệt, ta luôn giữ lại một phương án mẫu ( có thể xem là lời giải tối ưu cục bộ)

+ Đánh giá nhánh cận là phương pháp tính giá của phương án ngay trong quá trình xây dựng các thành phần của phương án theo hướng đang xây dựng có thể tốt hơn phương án mẫu hay không. Nếu không ta chọn theo hướng khác.

* Thủ tục:

Try(i) =

for(j = 1 → n)

if ( chấp nhận được)

{

Xác định theo j;

Ghi nhận trạng thái mới;

if ( i== n)

Cập nhật lời giải tối ưu;

else

{

Xác định cận g(,…..,);

if ( g(,…..,) ≤ )

Try ( I + 1)

}

// Trả bài toán về trạng thái cũ

}

* Thực chất của phương pháp nhánh cận là tìm kiếm được theo chiều sâu trên cây liệt kê lời giải như phương pháp quay lui, chỉ khác có một điều là khi tìm được mà đánh giá cận g(,…..,) > thì ta cắt bỏ các nhánh con từ đi xuống, mà quay lên ngay cha của nó là   
  **b. Các bài toán áp dụng**
  + 1. **Bài toán người đi du lịch**
* Một người đi du lịch muốn tham quan n thành phố ,…, *.*  Xuất phát từ một thành phố nào đó, người đi du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua đúng một lần rồi quay trở lại thành phố xuất phát
* Gọi là chi phí đi từ thành phố đến . Hãy tìm một hành trình sao cho chi phí là nhỏ nhất
* Nếu ta cố định xuất phát từ , ta duyệt vòng lặp từ j=2
* **Đánh giá nhánh cận:**

Đặt : Cmin = Min{:i,j {1,…,n}}

Giả sử vào các bước i ta tìm được lời giải bộ phận cấp i là (x1,…,xi), tức là đã đi qua đoạn đường , tương ứng với chi phí:

Để phát triển hành trình bộ phận này thành một hành trình đầy đủ, ta còn phải đi qua n-i+1 đoạn nữa, gồm n-I thành phố còn lại và đoạn quay lại .

Do chi phí mỗi một trong n-i+1 đoạn còn lại không nhỏ hơn Cmin, nên hàm đánh giá cận có thể xác định như sau:

g( ) =

* Điều kiện chấp nhận được của j là thành phố chưa đi qua.
* Xác định theo j bằng câu lệnh gán : = j

Cập nhật trạng thái mới: Daxet[j] = 1.

Cập nhật lại chi phí sau khi tìm tìm được : S=S+

* **Cập nhật lời giải tối ưu:**

Tính chi phí hành trình vừa tìm được:

Tong = S+;

Nếu (Tong <) thì

Lgtu = x;

= Tong;

Thao tác hủy bỏ trạng thái: Daxet[j] = 0

Trả lại chi phí cũ : S=S-

1. **Bài toán cái túi**

Có n đồ vật, mỗi loại có số lượng không hạn chế. Đồ vật loại i, đặc trưng bởi trọng lượng Wi và giá trị sử dụng Vi, với mọi I € { 1,…,n }.

Cần chọn các vật này đặt vào một chiếc túi xách có giới hạn trọng lượng m, sao cho tổng giá trị sử dụng các vật được chọn là lớn nhất.

* **Cách chọn vật**:

Xét mảng đơn giá:

Ta chọn vật theo đơn giá giảm dần.

Không mất tính tổng quát, ta giả sử các loại vật cho theo thứ tự giảm dần của đơn giá.

* **Đánh giá cận trên**:

Giả sử đã tìm được lời giải bộ phận: () .Khi đó:

Cận trên cho các lời giải bộ phận cấp i có thể xác định bởi:

g() =

* Theo biểu thức xác định cận trên g, các giá trị có thể chấp nhận được cho là:

t=0

* Thao tác ghi nhận trạng thái mới khi xác định được chẳng qua là cập nhật lại giá trị thu được và giới hạn trọng lượng mới của chiếc túi:

S = S +

T = T +

* Vì vậy thao tác trả lại trạng thái cũ cho bài toán:

S = S -

T = T -

* Cập nhật lời giải tối ưu:

Khi tìm được một lời giải, ta so sánh lời giải này với lời giải mà ta coi là tốt nhất vào thời điểm hiện tại để chọn lời giải tối ưu.