ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

**ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KỲ**

Môn học: Phân tích và thiết kế thuật toán

Mã lớp: CS112.L12.KHCL

Học kỳ I (2020-2021)

**CHỦ ĐỀ 7:**

**GIỚI THIỆU PHƯƠNG PHÁP THIẾT KẾ THUẬT TOÁN**

**PHƯƠNG PHÁP NHÁNH VÀ CẬN (BRANCH AND BOUND)**

**GVHD**: Nguyễn Thanh Sơn

**Các thành viên**

Ngô Đức Tuấn – 18520186

Vũ Gia Bảo - 18520504

Lê Hoàng Long - 18521027

**MỤC LỤC**

[**PHẦN I:**](#_Toc60357980)

[**ĐẶT VẤN ĐỀ** 3](#_Toc60357981)

[**PHẦN II:**](#_Toc60357982)

[**NỘI DUNG** 4](#_Toc60357983)

[1. Khái niệm 4](#_Toc60357984)

[2. Đặc điểm của thuật toán 4](#_Toc60357985)

[3. Khi nào nên sử dụng thuật toán Nhánh-cận 5](#_Toc60357986)

[4. Mô hình chung của thuật toán 5](#_Toc60357987)

[5. Ví dụ minh họa 7](#_Toc60357988)

[*5.1 Bài toán thực tế: Bài toán người đi du lịch (Travelling Salesman Problem)* 7](#_Toc60357989)

[*5.2 Bài toán thực tế: Bài toán cái túi xách (Knapsack Problem)* 11](#_Toc60357990)

[*5.3 Bài toán thực tế: Bài toán phân công công việc (Job Assignment Problem)* 17](#_Toc60357991)

[6. Ưu nhược điểm 21](#_Toc60357992)

[*6.1 Ưu điểm* 21](#_Toc60357993)

[*6.2 Nhược điểm* 21](#_Toc60357994)

[7. So sánh sự khác nhau giữa kĩ thuật Backtracking và Brand–N–Bound 21](#_Toc60357995)

[**PHẦN III:**](#_Toc60357996)

[**TÀI LIỆU THAM KHẢO** 23](#_Toc60357997)

# **PHẦN I:**

# **ĐẶT VẤN ĐỀ**

Có một số bài toán trên thực tế yêu cầu chỉ rõ: trong một tập các đối tượng cho trước có bao nhiêu đối tượng thỏa mãn những điều kiện nhất định. Bài toán đó gọi là **bài toán đếm**.

Trong lớp các bài toán đếm, có những bài toán còn yêu cầu chỉ rõ những cấu hình tìm được thỏa mãn điều kiện đã cho là những cấu hình nào. Bài toán yêu cầu đưa ra danh sách các cấu hình có thể có gọi là **bài toán liệt kê**.

Để giải bài toán liệt kê, cần phải xác định được một **thuật toán**để có thể theo đó lần lượt xây dựng được tất cả các cấu hình đang quan tâm. Có nhiều phương pháp liệt kê, nhưng chúng cần phải đáp ứng được hai yêu cầu dưới đây:

* Không được lặp lại một cấu hình
* Không được bỏ sót một cấu hình

Có thể nói rằng, **phương pháp liệt kê** là **phương án cuối cùng** để giải được một số **bài toán tổ hợp** hiện nay. Khó khăn chính của phương pháp này chính là sự bùng nổ tổ hợp dẫn tới sự đòi hỏi lớn về không gian và thời gian thực hiện chương trình. Tuy nhiên cùng với sự phát triển của máy tính điện tử, bằng phương pháp liệt kê, nhiều bài toán tổ hợp đã tìm thấy lời giải. Qua đó, ta cũng nên biết rằng **chỉ nên dùng phương pháp liệt kê khi không còn một phương pháp nào khác**tìm ra lời giải.

Với các bài toán tìm phương án tối ưu, nếu chúng ta xét hết tất cả các phương án thì mất rất nhiều thời gian, nhưng nếu sử dụng phương pháp tham lam thì phương án tìm được chưa hẳn đã là phương án tối ưu. Để có thể **xây dựng** được **phương án tối ưu**, người ta đã nêu ra một kĩ thuật gọi là **kĩ thuật nhánh-cận**.

# **PHẦN II:**

# **NỘI DUNG**

## 1. Khái niệm

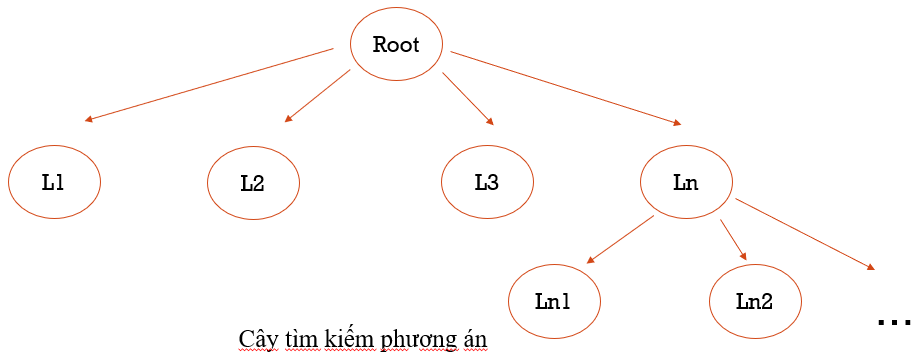
Nhánh-cận (Brach and Bound) là là một mô hình thiết kế thuật toán cho các bài toán **tối ưu hóa tổ hợp** và **rời rạc**, là kĩ thuật **xây dựng cây** tìm kiếm **phương án tối ưu**, nhưng không xây dựng toàn bộ cây mà sử dụng **giá trị cận** để **hạn chế** bớt các **nhánh**.

Thuật toán nhánh-cận thực chất là thuật toán quay lui.

* là một **kĩ thuật đánh giá** việc **tiếp tục đào sâu** có tạo ra **cấu hình tốt hơn cấu hình tốt nhất** mà ta lưu trữ hay không.
* nhờ có Nhánh-cận mà ta có thể đưa ra **quyết định Quay lui sớm hơn** thuật toán backtracking cổ điển.

Từ thuật toán backtracking cổ điển, khi **xác định điều kiện P** (điều kiện xác định cấu hình đề cử), ta **sử dụng thêm một hàm đánh giá** f(v1, v2,…,vk-1) để xác định việc đi tiếp có hy vọng tìm ra lời giải hay không.

## 2. Đặc điểm của thuật toán

****

*Hình 1: Cây tìm kiếm phương án*

Cây tìm kiếm phương án có **nút gốc** biểu diễn cho **tập tất cả các phương án có thể có**, mỗi **nút lá** biểu diễn cho **một phương án nào đó**. Nút n có các nút con tương ứng với các khả năng có thể lựa chọn tập phương án xuất phát từ n. Kĩ thuật này gọi là phân nhánh. Ý tưởng chính của nó là:

Trong quá trình duyệt ta luôn giữ lại một **phương án mẫu** (có thể xem là lời giải tối ưu cục bộ - chẳng hạn có giá trị nhỏ nhất tại thời điểm đó). 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 lựa chọn theo hướng khác.

Với mỗi nút trên cây ta sẽ xác định một giá trị cận. **Giá trị cận** là một giá trị **gần** với giá trị của **các phương án**. Với bài toán tìm **min** ta sẽ xác định **cận dưới** còn với bài toán tìm **max** ta sẽ xác định **cận trên**. **Cận dưới** là giá trị **nhỏ hơn hoặc bằng** giá của phương án, ngược lại **cận trên** là giá trị **lớn hơn hoặc bằng** giá của phương án.

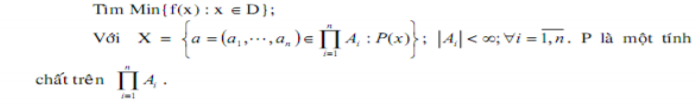
## 3. Khi nào nên sử dụng thuật toán Nhánh-cận

Đối với các bài toán tìm phương án tối ưu, ta sử dụng Nhánh-cận khi ta giải quyết bằng các thuật toán khác phải xét cả những phương án không khả thi, điều này gây ra bùng nổ tổ hợp khi dữ liệu đầu vào n lớn.

Giải quyết các bài tối ưu hóa tổ hợp. Những bài toán này thường theo cấp số nhân về độ phức tạp thời gian và có thể yêu cầu khám phá tất cả các hoán vị có thể có trong trường hợp xấu nhất. Nhánh cận giải quyết những vấn đề này khi nó đã có một giải pháp tối ưu tốt hơn mà giải pháp trước đó, nó sẽ loại bỏ giải pháp trước đó.

## 4. Mô hình chung của thuật toán

Giả sử bài toán tối ưu cho là:



Nghiệm của bài toán nếu có sẽ được biểu diễn dưới dạng: x = (x1,…,xn). Trong quá trình liệt kê theo phương pháp quay lui, ta xây dựng dần các thành phần của nghiệm.

Một bộ phận i thành phần (x1,…,xi) sẽ gọi là một lời giải (phương án) bộ phận cấp i. Ta gọi Xi là tập các lời giải bộ phận cấp i, mọi i = 1, n.

Đánh giá cận là tìm một hàm g xác định trên các Xi sao cho:

g (x1,…,xi) ≤ Min{f(a) : a = (a1,…,an) € X, xi = ai, mọi i = 1, n.

Thủ tục quay lui sửa lại thành thủ tục nhánh cận như sau:

Try (i)

for (j = 1 -> n)

if (Chấp nhận được)

{

Xác định xi theo j;

Ghi nhận trạng thái mới;

if (i == n)

Cập nhật lời giải (phương án) tối ưu;

else;

{

Xác định cận g (x1,…,xi)

if g (x1,…,xi) ≤ f\*

Try (i+1);

}

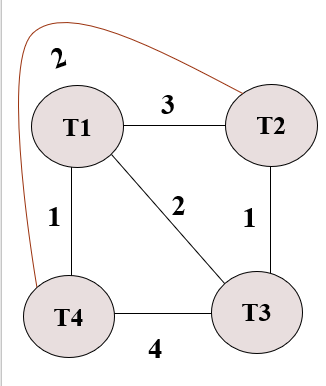
//Trả bài toán về trạng thái cũ

}

## 5. Ví dụ minh họa

### *5.1 Bài toán thực tế: Bài toán người đi du lịch (Travelling Salesman Problem)*

Bài toán người đi du lịch (Travelling Saleman Problem / TSP) là bài toán tối ưu liên quan đến chu trình Hamiltion được phát biểu như sau: Người du lịch muốn viếng thăm *n* thành phố và trở về lại nơi xuất phát. Biết chi phí đi lại giữa các thành phố, hãy tìm cách đi cho người du lịch sao cho có thể đến thăm mỗi thành phố đúng một lần, rồi quay lại thành phố xuất phát và tổng chi phí đi lại là bé nhất.



*Hình 2: Mô hình thành phố*

*Input:*

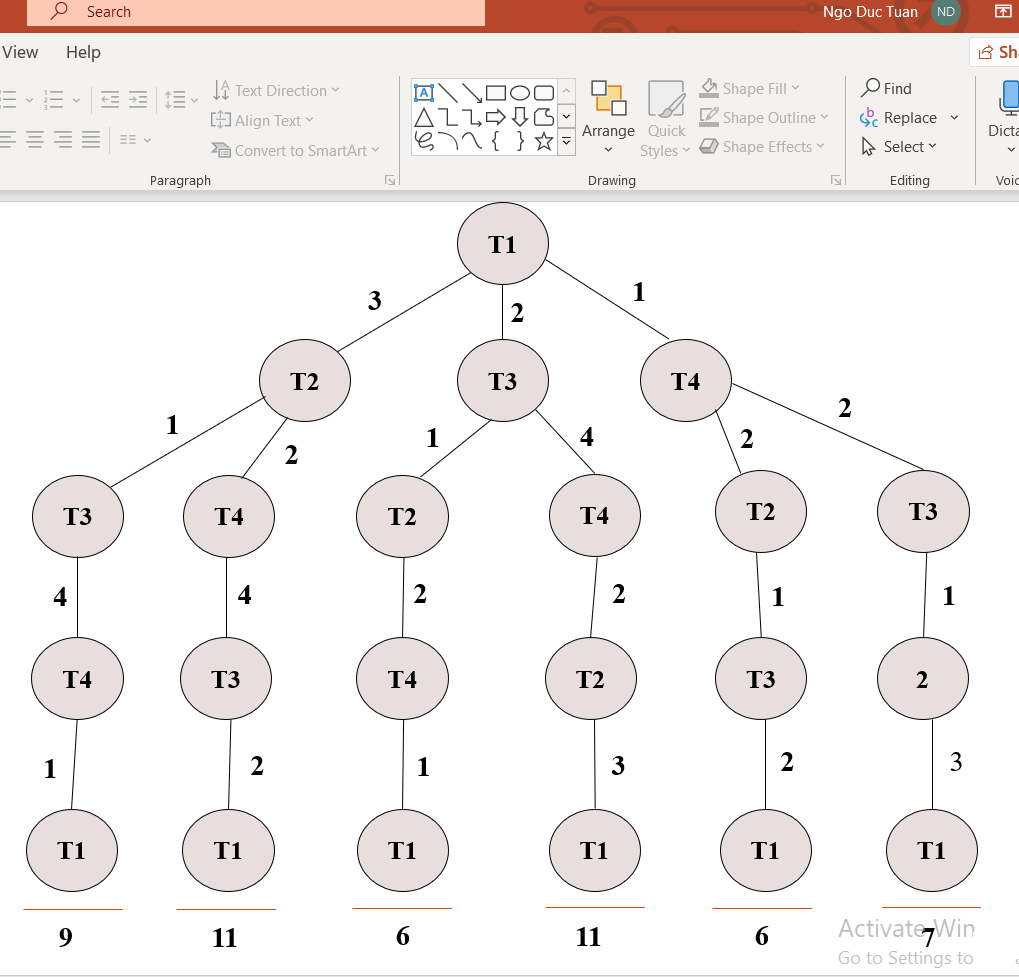
- Một người cần phải đi qua tại n thành phố T*1*, T*2*, …, T*n*

- C = (C*ij*): chi phí đi từ thành phố T*i* đến thành phố T*j* (*i* = 1, 2, … , *n*; *j* = 1, 2, …, *n*).

*Output:*

- Xác định hành trình của người đi du lịch, x\* = (x1,…,xn).

- Chi phí đi lại nhỏ nhất, f\* = f(x\*).



*Hình 3: Đường đi và chi phí khi xuất phát từ T1 đến các thành phố khác rồi quay về T1*

*Giải quyết bài toán:*

- Phương pháp vét cạn

- Phương pháp vét cạn quay lui

Tuy nhiên, nhược điểm khi dùng các phương pháp trên: Phải xét cả những phương án không khả thi (gây bùng nổ tổ hợp khi dữ liệu đầu vào n lớn).

**- Phương pháp Nhánh-cận \***

*Phân tích, thiết kế thuật toán:*

Gọi p là 1 hoán vị của {1,…,n} ta được hành trình Tp(1) -> Tp(2) -> … -> Tp(n).

Có n! hành trình.

Nếu cố định một thành phố xuất phát, chẳng hạn T1 thì có (n-1)! hành trình, bài toán trở thành:

* Tìm Min{f(a2,…,an): (a2,…,an) là hoán vị của {2,…,n}} với f(a1,…,an) = C1,a2 + Ca2,a3 + … + Can-1,an + Can,1.
* Ta sẽ kết hợp đánh giá nhánh cận trong quá trình liệt kê phương án của thuật toán quay lui.

*Hướng giải quyết:*

Cố định thành phố xuất phát là T1, duyệt vòng lặp từ j=2.

Đánh giá cận:

Đặt Cmin=Min{Cij: i,j = {1,…,n}}

Giả sử tại bước i ta tìm được lời giải bộ phận cấp i là (x1,…,xi), tức là đã đi đoạn đường T1->T2->…->Ti với chi phí: Si = C1,x2 + Cx2,x3 + … + Cxn-1,xn + Cxn,1.

Để phát triển hành trình bộ phận này thành một hành trình đầy đủ, ta còn phải đi qua n-i+1 đoạn đường nữa, gồm n-i thành phố còn lại và đoạn quay lại T1. 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:

Hàm cận: g(x1,…,xi) = Si + (n-i+1)Cmin

Điều kiện chấp nhận được của j là thành phố Tj chưa đi qua.

Ta dùng một mảng logic Daxet[] để biểu diễn trạng thái này.

Daxet[j] = 1 nếu T[j] đã qua

0 nếu T[j] chưa qua

Xác định xi theo j bằng câu lệnh gán xi = j, cập nhật trạng thái mới Daxet[j] = 1 và cập nhật lại chi phí sau khi tìm được xi S = S + Cxi-1,xi.

Cập nhật lời giải tối ưu:

Tính chi phí hành trình vừa tìm được:

Nếu i = n, Tong = S + Cxn,1;

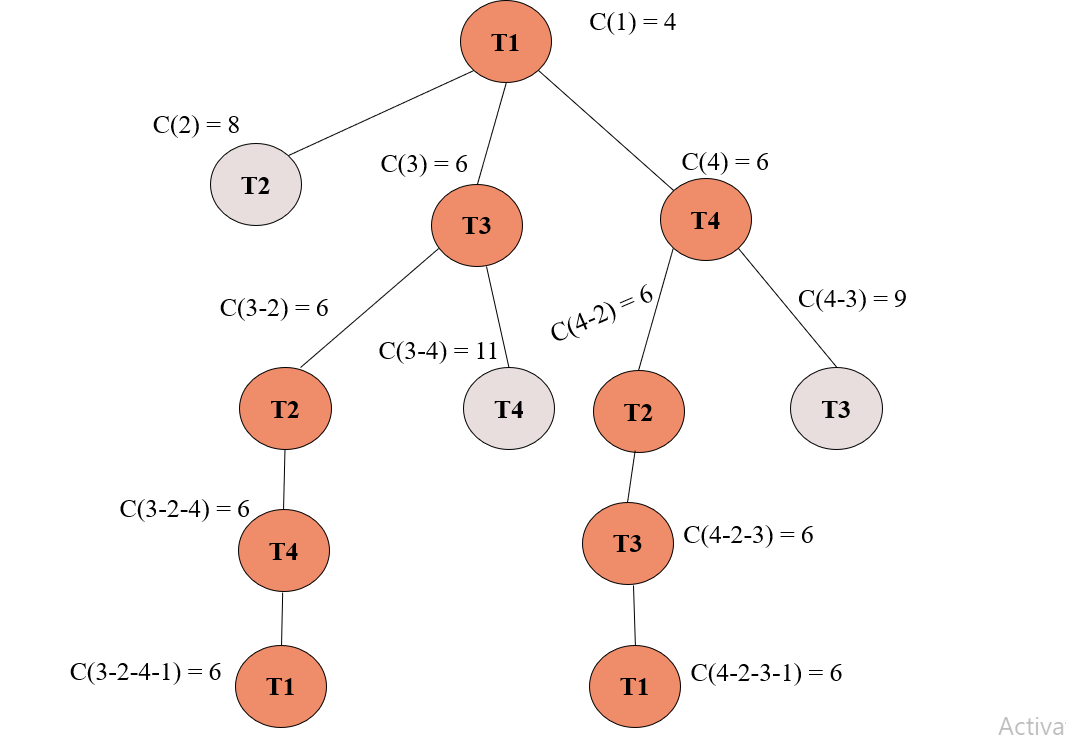
Nếu (Tong < f\*) thì lời giải tối ưu = x; f\* = Tong;

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

Trả lại chi phí cũ: S = S – Cxi-1,xi

Độ phức tạp tính toán của thuật toán trong trường hợp tổng quát vẫn là O(n!).

Độ phức tạp thời gian: Độ phức tạp trong trường hợp xấu nhất của thuật toán Nhánh cận vẫn giống như của Brute Force vì trong trường hợp xấu nhất, chúng ta có thể không bao giờ có cơ hội cắt bớt một nút. Trong khi đó, trên thực tế, nó hoạt động rất tốt tùy thuộc vào từng trường hợp khác nhau của TSP. Độ phức tạp cũng phụ thuộc vào sự lựa chọn của hàm cận (bounding function) vì chúng là hàm quyết định có bao nhiêu nút được cắt bỏ.

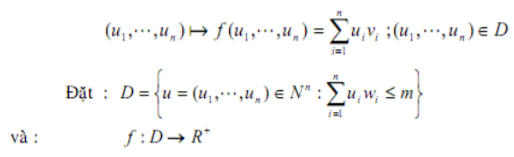


*Hình 4: Đường đi và tổng chi phí ngắn nhất của bài toán*

### *5.2 Bài toán thực tế: Bài toán cái túi xách (Knapsack Problem)*

Bài toán: Có *n* loại 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.

*Phân tích, thiết kế thuật toán:*



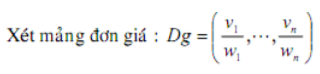
Bài toán cái túi chuyển về bài toán sau:



Cho nên ta sẽ kết hợp đánh giá nhánh-cận trong quá trình liệt kê các lời giải theo phương pháp quay lui.

*Hướng giải quyết:*

- Cách chọn vật:



+ 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.

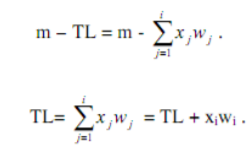
- Đánh giá cận trên:

+ Giả sử đã tìm được lời giải bộ phận : (x1,…,xi)



+ Khi đó :

* Giá trị của túi xách thu được :
* Tương ứng với trọng lượng các vật đã được xếp vào chiếc túi:



* Do đó, giới hạn trọng lượng của chiếc túi còn lại là :

Ta thấy đây là một bài toán tìm max.

Danh sách các đồ vật được sắp xếp theo thứ tự giảm của đơn giá để xét phân nhánh.

1. Nút gốc biểu diễn cho trạng thái ban đầu của ba lô, ở đó ta chưa chọn một vật nào. Tổng giá trị (TGT) được chọn TGT=0. Cận trên của nút gốc CT = W \* Đơn giá lớn nhất.

2. Nút gốc sẽ có các nút con tương ứng với các khả năng chọn đồ vật có đơn giá lớn nhất. Với mỗi nút con ta tính lại các thông số: · TGT = TGT (cũ) + số đồ vật được chọn \* giá trị mỗi vật. · W = W (cũ) – số đồ vật được chọn \* trọng lượng mỗi vật. · CT = TGT + W (mới) \* Đơn giá của vật sẽ xét kế tiếp.

3. Trong các nút con, ta sẽ ưu tiên phân nhánh cho nút con nào có cận trên lớn hơn trước. Các con của nút này tương ứng với các khả năng chọn đồ vật có đơn giá lớn tiếp theo. Với mỗi nút ta lại phải xác định lại các thông số TGT, W, CT theo công thức đã nói trong bước 2.

4. Lặp lại bước 3 với chú ý: đối với những nút có cận trên nhỏ hơn hoặc bằng giá lớn nhất tạm thời của một phương án đã được tìm thấy thì ta không cần phân nhánh cho nút đó nữa.

5. Nếu tất cả các nút đều đã được phân nhánh hoặc bị cắt bỏ thì phương án có giá lớn nhất là phương án cần tìm.

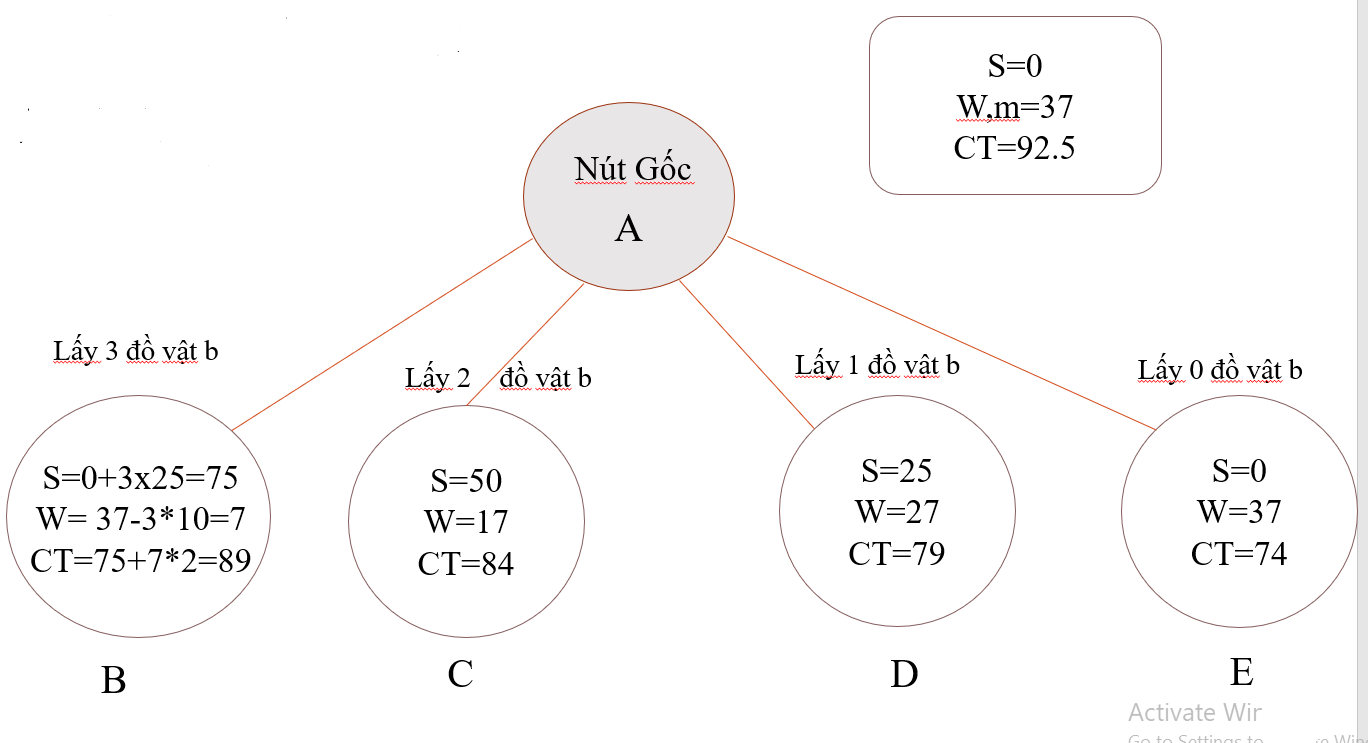
Ví dụ : Với bài toán cái túi đã cho, sau khi tính đơn giá cho các đồ vật và sắp xếp các đồ vật theo thứ tự giảm dần của đơn giá ta được bảng sau.

|  |  |  |  |
| --- | --- | --- | --- |
| Loại đồ vật | Trọng lượng | Giá trị | Đơn giá |
| b | 10 | 25 | 2.5 |
| a | 15 | 30 | 2.0 |
| d | 4 | 6 | 1.5 |
| c | 2 | 2 | 1 |

*Bảng 1*

Gọi x1, x2, x3, x4 là số lượng cần chọn tương ứng của các đồ vật b, a, d, c.

- Với ***nút gốc A***, biểu diễn cho trạng thái ta chưa chọn bất cứ một đồ vật nào.



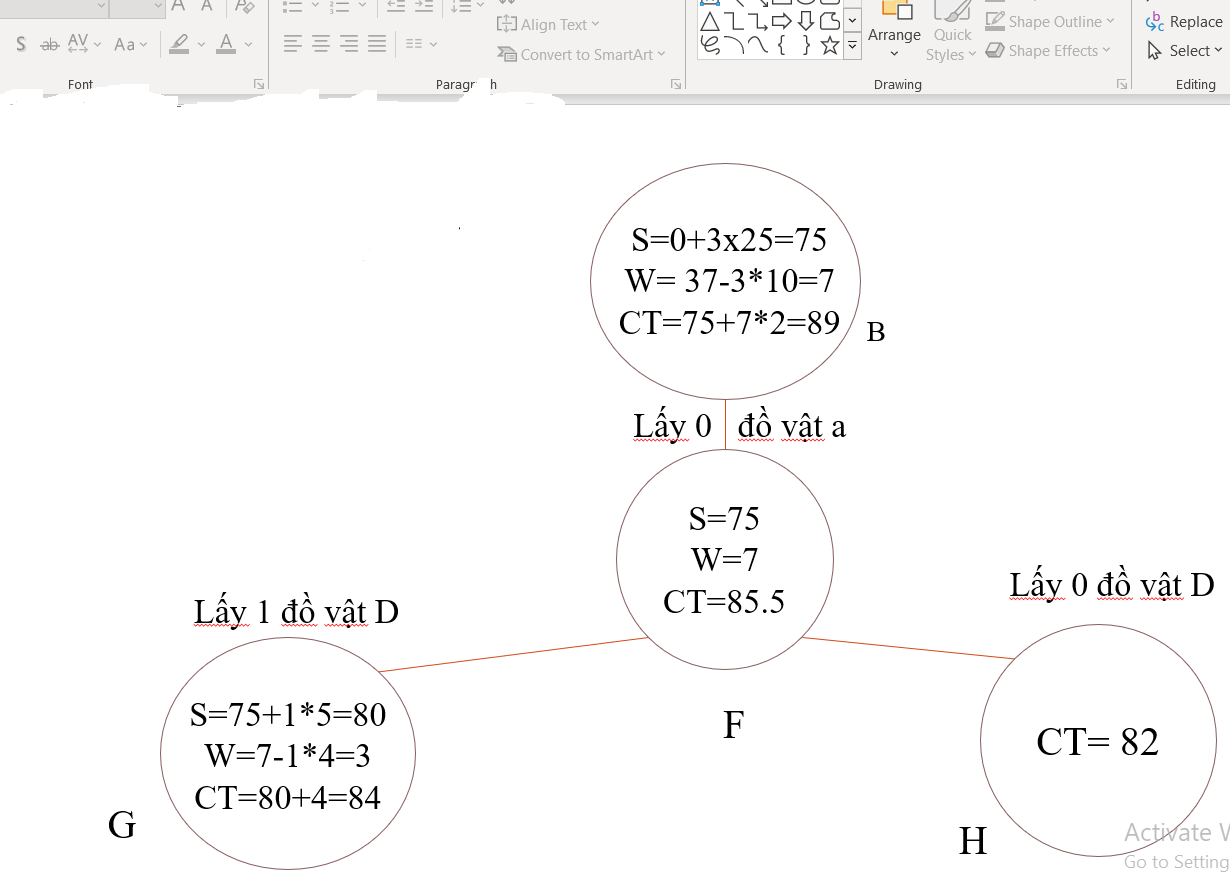
*Hình 5: Nút gốc A*

Khi đó tổng giá trị TGT =0, trọng lượng của ba lô W=37 (theo đề ra) và cận trên CT = 37\*2.5 = 92.5, trong đó 37 là W, 2.5 là đơn giá của đồ vật b.

Với đồ vật b, ta có 4 khả năng: chọn 3 đồ vật b (X1=3), chọn 2 đồ vật b (X1=2), chọn 1 đồ vật b (X1=1) và không chọn đồ vật b (X1=0).

Ứng với 4 khả năng này, ta phân nhánh cho nút gốc A thành 4 con B, C, D và E.

- Với ***nút con B***, ta có TGT = 0+ 3\*25 = 75, trong đó 3 là số vật b được chọn, 25 là giá trị của mỗi đồ vật b.



*Hình 6*

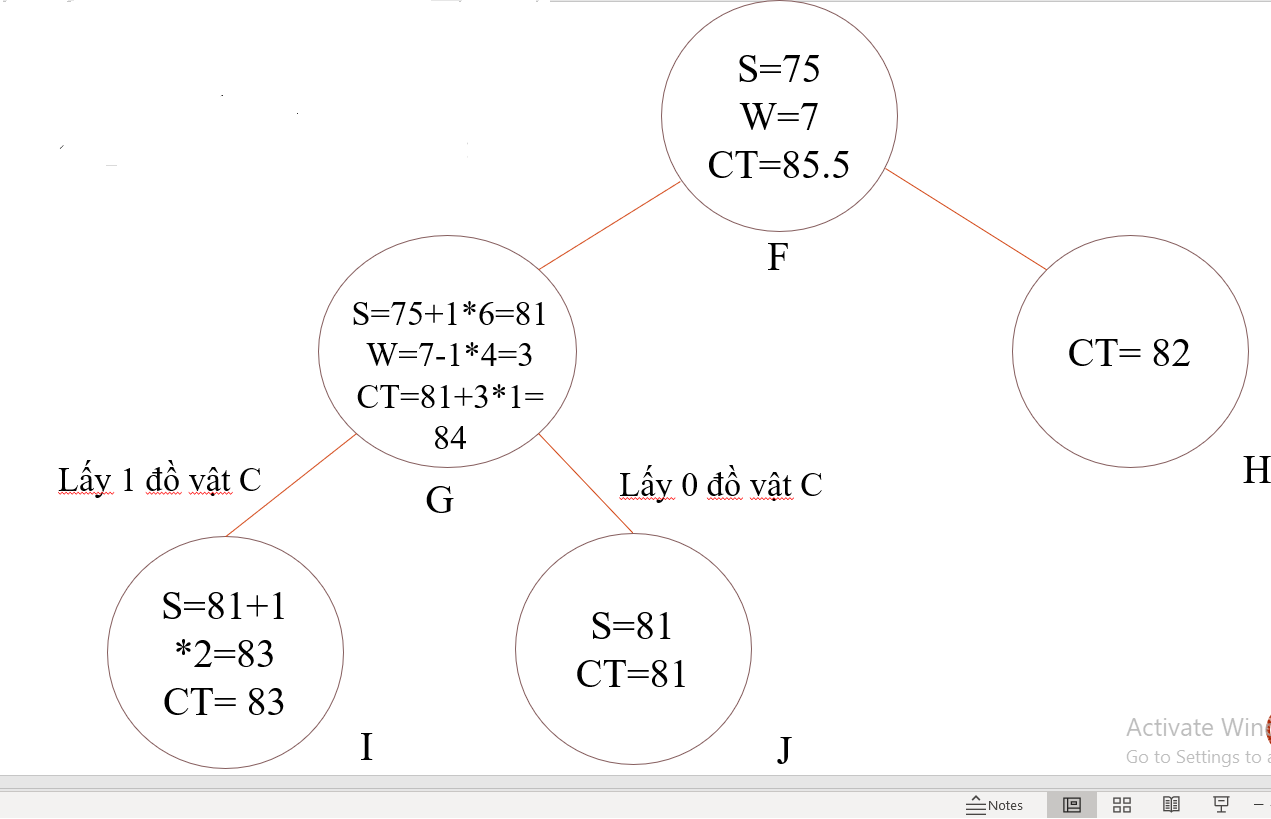
W = 37- 3\*10 = 7, trong đó 37 là trọng lượng ban đầu của ba lô, 3 là số vật b được, 10 là trọng lượng mõi đồ vật b. CT = 75 + 7\*2 = 89, trong đó 75 là TGT, 7 là trọng lượng còn lại của ba lô và 2 là đơn giá của đồ vật a.

Tương tự ta tính được các thông số cho các nút C, D và E, trong đó cận trên tương ứng là 84, 79 và 74. Trong các nút B, C, D và E thì nút B có cận trên lớn nhất nên ta sẽ phân nhánh cho nút B trước với hy vọng sẽ có được phương án tốt từ hướng này.

Từ nút B ta chỉ có một nút con F duy nhất ứng với X2=0 (do trọng lượng còn lại của ba lô là 7, trong khi trọng lượng của mỗi đồ vật a là 15).

Sau khi xác định các thông số cho nút F ta có cận trên của F là 85.5. Ta tiếp tục phân nhánh cho nút F. Nút F có 2 con G và H tương ứng với X3=1 và X3=0.

Sau khi xác định các thông số cho hai nút này ta thấy cận trên của G là 84 và của H là 82 nên ta tiếp tục phân nhánh cho nút G.



*Hình 7*

Nút G có hai con là I và J tương ứng với X4=1 và X4=0. Đây là hai nút lá (biểu diễn cho phương án) vì với mỗi nút thì số các đồ vật đã được chọn xong.

Trong đó nút I biểu diễn cho phương án chọn X1=3, X2=0, X3=1 và X4=1 với giá 83, trong khi nút J biểu diễn cho phương án chọn X1=3, X2=0, X3=1 và X4=01 với giá 81.

Như vậy giá lớn nhất tạm thời ở đây là 83. Quay lui lên nút H, ta thấy cận trên của H là 8283 nên tiếp tục phân nhánh cho nút C.

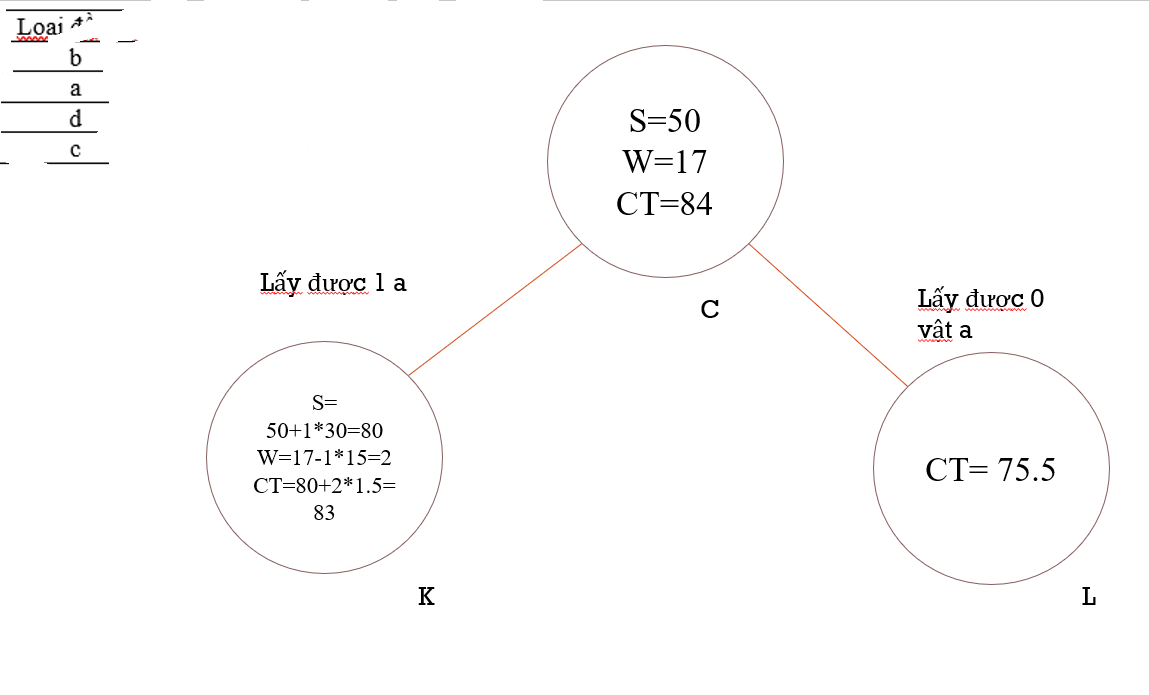
Nút C có hai con là K và L ứng với X2=1 và X2=0.

Sau khi tính các thông số cho K và L ta thấy ận trên của K là 83 và của L là 75.25.

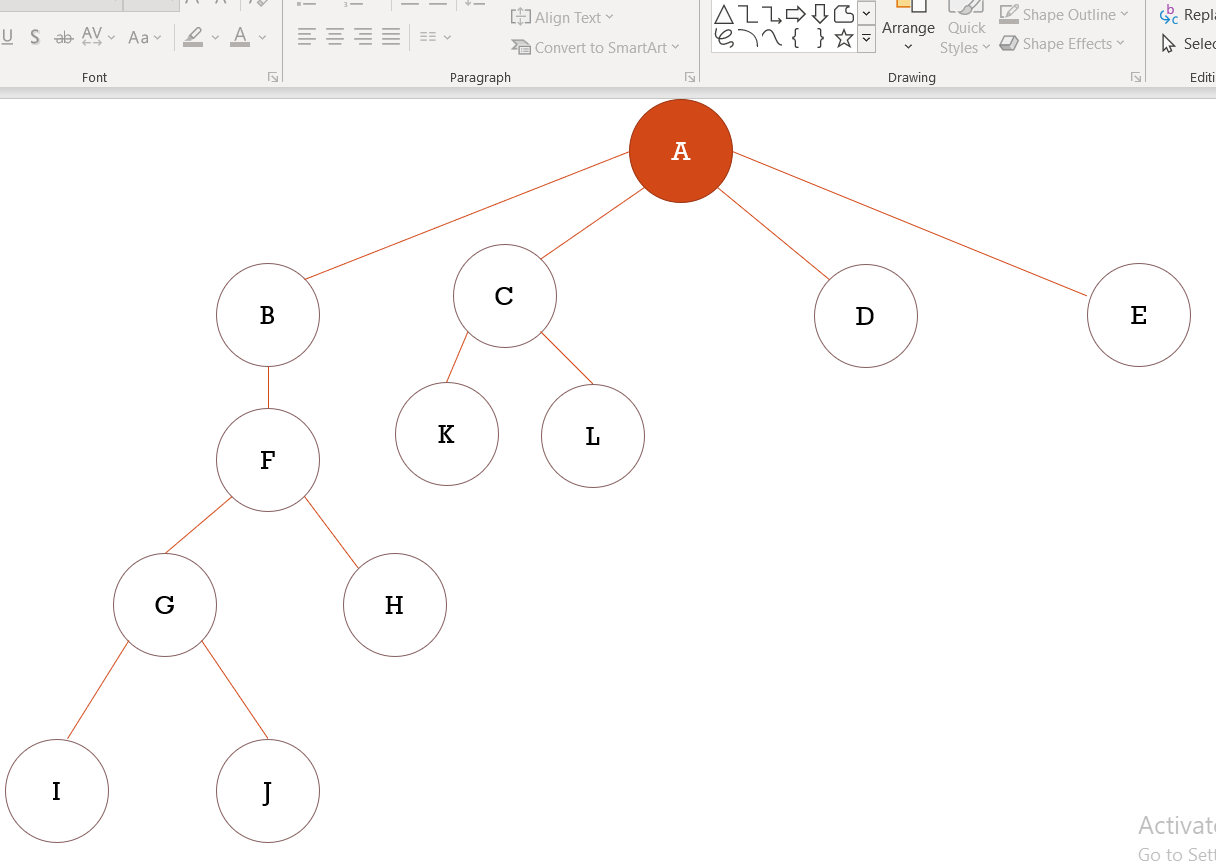
Cả hai giá trị này đều không lớn hơn 83 nên cả hai nút này đều bị cắt tỉa.

Cuối cùng các nút D và E cũng bị cắt tỉa. Như vậy tất cả các nút trên cây đều đã được phân nhánh hoặc bị cắt tỉa nên phương án tốt nhất tạm thời là phương án cần tìm.

Theo đó ta cần chọn 3 đồ vật loại b, 1 đồ vật loại d và một đồ vật loại c với tổng giá trị là 83, tổng trọng lượng là 36.



*Hình 8*



*Hình 9*

### *5.3 Bài toán thực tế: Bài toán phân công công việc (Job Assignment Problem)*

Input: Có *N* công nhân và *N* công việc. Bất kì công nhân nào cũng có thể được chỉ định thực hiện bất kì công việc nào, chịu một số chi phí có thể thay đổi tùy theo công việc được giao.

Output: Cần phải thực hiện tất cả các công việc bằng cách phân công chính xác một công nhân cho mỗi công việc và chính xác một công việc cho mỗi đại lý (agent) theo cách sao cho tổng chi phí của công việc được tối thiểu.

A picture containing table

Description automatically generated

*Hình 10*

Các hướng tiếp cận:

Giải pháp 1: Brute Force: Chúng ta tạo ra n! các công việc có thể được phân công và đối với mỗi phân công như vậy, chúng ta tính tổng chi phí của nó và trả lại cái phân công mà có chi phí ít tống kém hơn. Vì giải pháp là một hoán vị của n công việc nên độ phức tạp của nó là O(n!).

Giải pháp 2: Thuật toán Hungarian: Phân công tối ưu được tìm thấy bằng cách sử dụng thuật toán Hungarian. Thuật toán có độ phức tạp thời gian chạy trong trường hợp xấu nhất la 0 (n^3).

**Giải pháp 3: Branch-N-Bound**

Quy tắc lựa chọn cho nút tiếp theo trong BFS và DFS là “mù”, tức là quy tắc lựa chọn không đưa ra bất kì ưu tiên nào cho một nút có cơ hội rất tốt để tìm kiếm nhanh chóng đến một nút trả lời (answer node). Việc tìm kiếm giải pháp tối ưu thường có thể được tăng tốc bằng cách sử dụng hàm xếp hạng “thông minh” (“intelligent” ranking function), còn gọi là hàm chi phí gần đúng (approximate cost function) để tránh tìm kiếm trong các cây con không chứa giải pháp tối ưu. Nó tương tự như tìm kiếm BFS nhưng với một tối ưu hóa chính. Thay vì tuân theo thứ tự FIFO, chúng tôi chọn một nút trực tiếp (live node) với chi phí thấp nhất. Chúng ta có thể không nhận được giải pháp tối ưu bằng cách theo dõi nút với chi phí ít hứa hẹn nhất, nhưng nó sẽ cung cấp cơ hội rất tốt để tìm kiếm nhanh chóng đến một nút trả lời.

Có hai cách tiếp cận để tính toán hàm chi phí:

* Với mỗi công nhân, chúng ta chọn công việc với chi phí tối thiểu từ danh sách các công việc chưa được giao (lấy mục nhập tối thiểu từ mỗi hàng).
* Với mỗi công việc, chúng ta chọn một công nhân có chi phí thấp nhấp cho công việc đó từ danh sách công nhân chưa được phân công (lấy từ mục nhập tối thiểu từ mỗi cột).

Trong bài toán này, nhóm chúng mình sẽ tiếp cận với cách thứ nhất.

Lấy ví dụ dưới đây và cố gắng tính toán chi phí hứa hẹn khi công việc 2 (Job 2) được giao cho công nhân A (worker A).

Calendar

Description automatically generated

*Hình 11*

Vì Job 2 được phân công cho worker A (được đánh dấu màu xanh lá), chi phí là 2 và Job 2 và worker A trở nên không khả dụng (được dánh dấu màu đỏ).

Calendar

Description automatically generated

*Hình 12*

Bây giờ chúng ta phân công Job 3 cho worker B vì nó có chi phí tối thiểu từ danh sách các công việc chưa phân được phân công. Chi phí lúc này là 2+3 = 5 và Job 3 và worker B cũng trở nên không khả dụng.

Calendar

Description automatically generated

*Hình 13*

Cuối cùng, Job 1 được phân công cho worker C vì nó có chi phí tối thiểu trong số các công việc chưa được phân công và Job 4 tương tự được giao cho worker D. Tổng chi phí lúc này là 2+3+5+4 = 14.

Calendar

Description automatically generated

*Hình 14*

Sơ đồ bên dưới hiển thị sơ đồ không gian tìm kiếm hoàn chỉnh hiển thị đường dẫn giải pháp tối ưu màu xanh lá.

Diagram

Description automatically generated

*Hình 15*

## 6. Ưu nhược điểm

### *6.1 Ưu điểm*

Ưu điểm của thuật toán nhánh-cận là hạn chế được nhiều tính toán trong quá trình xây dựng phương án và được xem là thuật toán tốt nhất hiện nay cho TSP.

Nhánh cận là một mô hình thiết kế thuật toán thường được sử dụng để giải các bài toán tối ưu hóa tổ hợp. Những bài toán này thường theo cấp số nhân về độ phức tạp thời gian và có thể yêu cầu khám phá tất cả các hoán vị có thể có trong trường hợp xấu nhất. Nhánh cận giải quyết những vấn đề này tương đối nhanh chóng.

### *6.2 Nhược điểm*

Tuy là thuật toán tốt nhưng kết quả tìm được chưa có tối ưu nhất có thể.

## 7. So sánh sự khác nhau giữa kĩ thuật Backtracking và Brand–N–Bound

Backtracking (Quay lui): Backtracking là một thuật toán chung để tìm tất cả các giải pháp cho một số vấn đề về tính toán, đặc biệt là các bài toán về thỏa mãn tính ràng buộc (constraint satisfaction problems), từng bước xây dựng các ứng viên khả thi, cho các giải pháp và loại bỏ một ứng viên khi xác định ứng viên đó không thể hoàn thành để cuối cùng trở thành một giải pháp hợp lệ. Đây là một kỹ thuật để giải quyết các bài toán về đệ quy bằng cách cố gắng xây dựng một giải pháp tăng dần, từng phần một, loại bỏ đi những giải pháp không thỏa mãn các ràng buộc của bài toán tại bất kì thời điểm nào.

Branch and Bound (Nhánh và cận): Branch and Bound là một mô hình thiết kế thuật toán cho các bài toán tối ưu hóa tổ hợp và rời rạc (discrete and combinatoric optimisation problems), cũng như tối ưu hóa toán học (mathematical optimisation). Một thuật toán Branch và Bound bao gồm một liệt kê có hệ thống các giải pháp ứng viên. Có nghĩa là, tập hợp các giải pháp ứng viên được coi như là hình thành một cây gốc với tập hợp đầy đủ các gốc. Thuật toán khám phá các nhánh của cây này, đại diện cho các tập con của tập giải pháp. Trước khi liệt kê các giải pháp ứng viên của một nhánh, nhánh được kiểm tra dựa trên các cận ước tính trên và dưới của giải pháp tối ưu và bị loại bỏ nếu nó không thể tạo ra giải pháp tốt hơn giải pháp tốt nhất được tìm thấy.

Sự khác nhau giữa hai thuật toán:

|  |  |  |
| --- | --- | --- |
| **Tham số** | **Backtracking** | **Branch-N-Bound** |
| Tiếp cận | Được dùng để tìm tất cả các giải pháp khả thi có sẵn cho một bài toán. Khi nó đã đưa ra một lựa chọn tồi, nó sẽ hoàn tác lựa chọn cuối cùng bằng cách backing nó. Nó tìm kiếm cây không gian trạng thái (state-space tree) cho đến khi tìm ra giải pháp cho bài toán. | Được dùng để giải quyết các bài toán tối ưu hóa. Khi nó đã có một giải pháp tối ưu tốt hơn mà giải pháp trước đó, nó sẽ loại bỏ giải pháp trước đó. Nó hoàn toàn tìm kiếm cây không gian trạng thái (state-space tree) để có được giải pháp tối ưu. |
| Duyệt cây | Duyệt qua cây theo cách DFS. | Duyệt qua cây theo bất kì cách nào, DFS hoặc BFS. |
| Bài toán | Được dùng để giải quyết bài toán quyết định (Decision Problem). | Được dùng để giải quyết bài toán tối ưu hóa (Optimisation Problem). |
| Tìm kiếm | Cây không gian trạng thái được tìm kiếm cho đến khi thu được giải pháp. | Giải pháp tối ưu có ở bất kì vị trí nào trong cây không gian trạng thái, vì vậy cây cần được tìm kiếm toàn bộ. |
| Hiệu quả | Hiệu quả hơn. | Kém hiệu quả hơn. |
| Ứng dụng | Giải quyết bài toán N-Queen, Tổng của các tập con. | Giải quyết bài toán Knapsack, bài toán Travelling Salesman. |

*Bảng 2*

# **PHẦN III:**

# **TÀI LIỆU THAM KHẢO**

1. *Anany Levitin* - Introduction to the Design and Analysis of Algorithms

2. Difference between Bactracking and Branch-N-Bound technique

<https://www.geeksforgeeks.org/difference-between-backtracking-and-branch-n-bound-technique/?ref=rp>

3. Travelling Salesman Problem using Branch and Bound

<https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/>

4. Job Assignment Problem using Branch and Bound

<https://www.geeksforgeeks.org/job-assignment-problem-using-branch-and-bound/?ref=rp>

5. Implementation of 0/1 Knapsack using Branch and Bound

<https://www.geeksforgeeks.org/implementation-of-0-1-knapsack-using-branch-and-bound/>

6. Giải thuật và lập trình: Kĩ thuật nhánh cận

<https://v1study.com/giai-thuat-va-lap-trinh-ky-thuat-nhanh-can.html>

7. Thiết kế và đánh giá thuật toán (Khoa CNTT ĐHSP KT Hưng Yên)

<https://voer.edu.vn/c/thiet-ke-va-danh-gia-thuat-toan/018b828c>