ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

THIẾT KẾ VÀ ĐÁNH GIÁ THUẬT TOÁN

Bài 4

Lớp phương pháp thử sai

(Trial and Error Methods)

Nguyễn Thị Hồng Minh
minhnth@gmail.com

Nội dung

1. Phương pháp vét cạn

Ý tưởng, mô hình, lược đồ, ví dụ

2. Phương pháp quay lui

Ý tưởng, mô hình, lược đồ, ví dụ

3. Phương pháp nhánh cận

Ý tưởng, Nguyên lí đánh giá cận

Mô hình, lược đồ, ví dụ

Phương pháp vét cạn

- **Ý tưởng**
  - Vét cạn (Exhautive): Duyệt tất cả các phương án tồn tại nghiệm của bài toán để xác định nghiệm đúng
  - Nguyên lí Edison "Tìm kim trong đống rơm!"

- **Mô hình**

Không gian nghiệm của bài toán (tập khả năng) $D=\{(x_1,x_2,\ldots,x_n)\}$

$x_i \in D_i$, với $D_i = \{d_{ij} \mid j = 1\ldots m_i\}$ – tập hữu hạn có m_i phần tử

Quy tắc xác định lời giải bài toán :

$f : D \rightarrow \{true, false\}$

Nghiệm của bài toán: $x= (x_1,x_2,\ldots,x_n)$ sao cho $f(x)= true$

Phương pháp vét cạn

- **Lược đồ phương pháp**

```
VetCan(D)≡
|   For (x1 = d11 .. d1m1)
|      For (x2 = d21 .. d2m2)
|         ……
|         For (xn = dn1 .. dnmn)
|            if (f(x1,x2,…xn))
|              <Chấp nhận nghiệm x=(x1,x2,…xn)>;
End.
```

Phương pháp vét cạn

- **Ví dụ**
  - ***Bài toán gà ,chó***

    Vừa gà vừa chó; Bó lại cho tròn;

    Ba mươi sáu con; Một trăm chân chẵn

    Hỏi số gà, số chó?
  - Phân tích:

    Gọi x,y là số gà, chó: $x \in D_1 = \{1,2,\ldots,36\}$, $y \in D_2 = \{1,2,\ldots,36\}$

    Không gian nghiệm của bài toán: $D = \{(x, y)\}$

    Quy tắc xác định lời giải: $f(x,y) = (x+y = 36)$ and $(2x + 4y = 100)$

    Nghiệm của bài toán $x=(x,y)$ sao cho $f(x,y) = true$

Phương pháp vét cạn

- **Ví dụ:** ***Bài toán gà chó***
  - Lược đồ thuật bài toán:

```
GaCho ≡
    For (x=0 .. 36)
        For(y=0 .. 36)
            if((x+y)= 36 &(2x + 4y)= 100 )
                <In ra kết quả (x,y)>;
End.
```

  - Độ phức tạp thuật toán: O(n^2), n=36 – phụ thuộc không gian bài toán
  - Cải tiến:
    - Giảm không gian nghiệm bài toán: D={(x)}, $x \in D_1$ ={1,2,…,25}
    - Quy tắc xác định nghiệm: f(x)= (4x+2(36-x) = 100)
    - Độ phức tạp: O(n), n=25

Phương pháp vét cạn

- **Các bài toán ứng dụng:**
  - Các thuật toán sắp xếp: nổi bọt, chèn,…
  - Các bài toán tìm kiếm:
    - Tim phần tử trong dãy
    - Tìm chuỗi kí tự trong xâu (string matching)
    - Tìm cặp điểm gần nhất (closest pair)
    - Tìm bao lồi (convex hull)
    - …
  - Một số bài toán xác định phương án nghiệm từ nhiều phương án khả thi
    - Xây dựng chu trình Euler, Hamilton của đồ thị
    - Bài toán người bán hàng (travelling salesman)
    - Bài toán xếp balo (knapsack)
    - …

      => khả thi với lượng dữ liệu nhỏ

Phương pháp quay lui

- **Ý tưởng**
  - Quay lui (Back tracking): Theo nguyên tắc vét cạn, nhưng chỉ xét những trường hợp khả quan.
  - Dùng để giải bài toán liệt kê các cấu hình:
    - Mỗi cấu hình được xây dựng bằng cách xác định từng phần tử
    - Mỗi phần tử được chọn bằng cách thử các khả năng ***có thể***.
  - Tại mỗi bước, nếu có một lựa chọn được chấp thuận thì ghi nhận lại lựa chọn này và tiến hành các bước thử tiếp theo. Còn ngược lại không có lựa chọn nào thích hợp thì quay lại bước trước $\Rightarrow$ ***quay lui***.

Phương pháp quay lui

- **Mô hình**

  Không gian nghiệm của bài toán (tập khả năng) D={(x_1,x_2,…,x_n)} gồm các cấu hình liệt kê có dạng (x_1, $x_{2, ...,}$ x_n) cần được xây dựng:

  - Cho x_1 nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x_1 thì:
  - Cho x_2 nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x_2 thì xét khả năng chọn x_3,…, x_n, => cấu hình tìm được (x_1, $x_{2, ...,}$ x_n).
  - Tại mỗi bước i: Xây dựng thành phần x_i
    - Xác định x_i theo khả năng *v*.
    - Nếu *i* = *n* thì ta có được một lời giải, ngược lại thì tiến hành bước i+1 để xác định x_{i+1}.
    - Nếu không có một khả năng nào chấp nhận được cho x_i thì lại lùi lại bước trước để xác định lại thành phần x_{i-1}.

Phương pháp quay lui

- **Lược đồ phương pháp**

```
Try(i) ≡ //Sinh thành phần thứ i của cấu hình
    for (v thuộc tập khả năng thành phần nghiệm x_i)
        if ( x_i chấp nhận được giá trị v)
            x_i = v;
            <Ghi nhận trạng thái chấp nhận v>;
            if(i = n) //đủ n thành phần của cấu hình đã xác định
                <ghi nhận nghiệm>;
            else       //lời gọi sinh thành phần tiếp theo của cấu hình
                Try (i + 1)
            <Khôi phục trạng thái chưa chấp nhận v>;
        endif;
    endfor
End.
```

Phương pháp quay lui

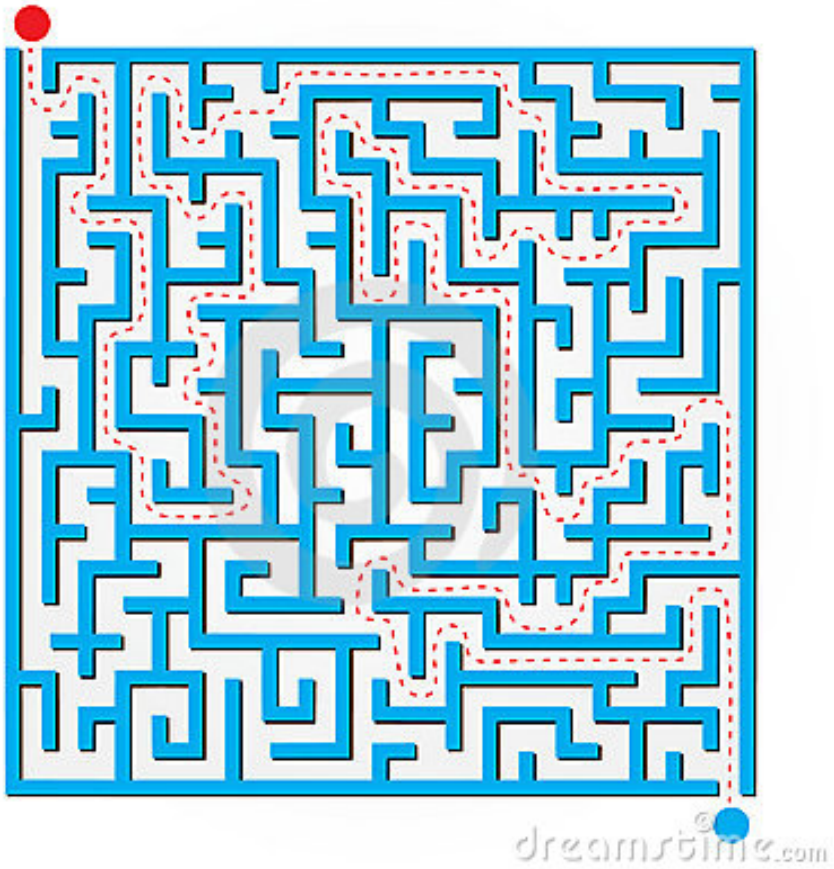
- **Ví dụ 1:**
  - Bài toán: Liệt kê dãy nhị phân độ dài n
  - Phân tích:
    - Dãy nhị phân ($x_1 x_2 \cdots x_{n)}$ trong đó x_i =0,1
    - Dùng giải thuật Try(*i*) để sinh giá trị x_i
    - Nếu *i*=*n* thì in giá trị nghiệm, ngược lại sinh tiếp x_{i+1} bằng Try(*i*+1)
  - Lược đồ:

```
Try(i) ≡
    for (v=0..1)   //v nhận giá trị 0 hoặc 1
        x_i = v;
        if (i= n) printResult (x);
        else Try(i+1) ;
    endfor;
End.
```

Lời gọi ban đầu **Try(1)**

Phương pháp quay lui

- **Ví dụ 2:**
  - Bài toán tìm đường đi trong mê cung (maze):

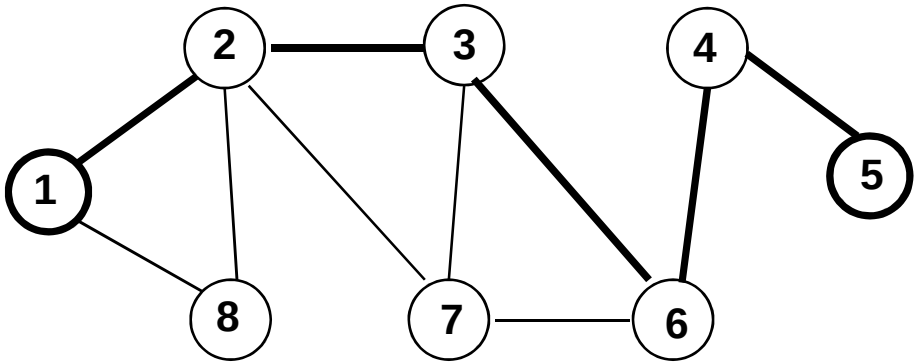


Phương pháp quay lui

- **Ví dụ 2: Tìm đường trong mê cung**
  - Phân tích: Xét mê cung như một đồ thị vô hướng $G=(V,E)$
    - Mỗi phòng của mê cung là một nút của đồ thị
    - Mỗi hành lang giữa các phòng là một cạnh nối giữa các nút của đồ thị
    - Đỉnh xuất phát: $S \in V$, đỉnh kết thúc $F \in V$
    - Đường đi tìm được là dãy $S=x_1,x_2,\ldots,x_k=F$ với $x_i \in V$, $(x_i,x_{i+1}) \in E$
    - Dùng giải thuật Try(i) để sinh giá trị x_i
      - Đỉnh v được chấp nhận cho x_i nếu v chưa được đi qua, có cạnh (x_{i-1},v)
      - Nếu $x_i=F$ thì kết thúc, in đường đi tìm được
      - Ngược lại sinh tiếp x_{i+1} bằng Try(i+1)
    - Lời gọi ban đầu
      - $x_1=S$
      - Try(2)

Phương pháp quay lui

- **Ví dụ 2: Tìm đường trong mê cung**



  - Đỉnh xuất phát: $S=1 \in V$, đỉnh kết thúc $F=5 \in V$
  - Đường đi tìm được là dãy S=1-2-3-6-4-5=F

Phương pháp quay lui

- **Ví dụ 2: Tìm đường trong mê cung**
  - Mô tả dữ liệu
    - Đánh chỉ số các đỉnh của đồ thị từ 1..*n*
    - Biểu diễn đồ thị G bằng ma trận kề $M = (m_{ij})$ cỡ n x n

      m_{ij}= 1 nếu có cạnh nối đỉnh *i* với đỉnh *j*

      = 0 nếu ngược lại
    - Mảng: Daqua[1..*n*] đánh dấu đỉnh *i* đã được đi qua trên đường đi hay chưa?

      Daqua[*i*]= *true* nếu đỉnh *i* đã có trên đường đi

      = *false* nếu ngược lại, đỉnh *i* chưa có trên đường đi

      Khởi tạo: Daqua[1..*n*] = *false*

Phương pháp quay lui

- **Ví dụ 2: Tìm đường trong mê cung**
  - Lược đồ:

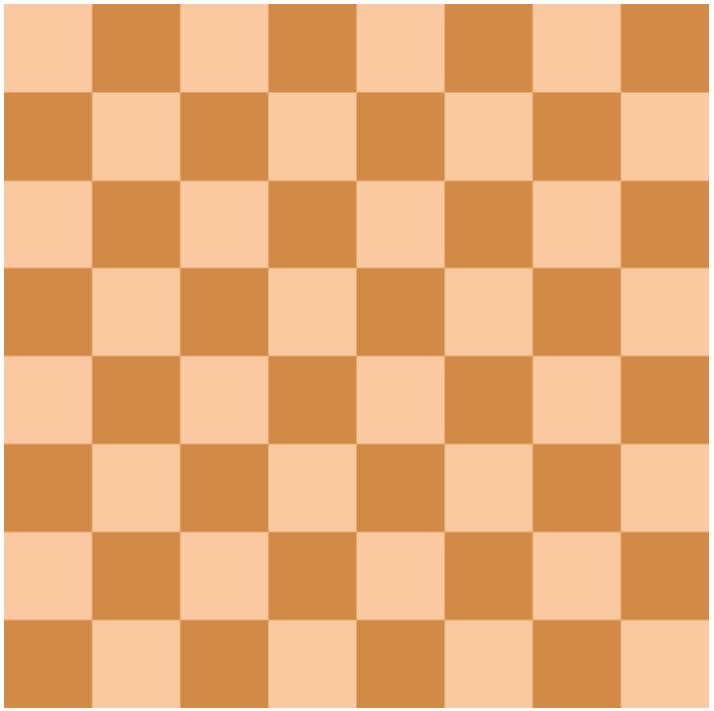
```
Try(i) ≡
    for (v=1..n)   //duyệt qua các đỉnh
        if ((m[x_{i-1},v]=1)&(not Daqua[v])) //v chấp nhận được
            x_i = v;
            Daqua[v] = true; //ghi nhận trạng thái đã chọn v
            if (x_i= F) printResult (x_1,x_2,…,x_i);
            else Try(i+1) ;
            Daqua[v] = false; //khôi phục trạng thái chưa chọn v
        endif;
    endfor;
End.
```

Lời gọi ban đầu:

```
x_1=S;
Try(2);
```

Phương pháp quay lui

- **Bài toán xếp 8 hậu (Eight queens puzzle)**



(http://en.wikipedia.org/wiki/Eight_queens_puzzle)

Phương pháp quay lui

- **Các bài toán khác**
  - Tìm chu trình Hamilton của đồ thị (Hamiltonian Path Problem)
  - Bài toán người đưa hàng (Traverling Salesman Problem)
  - Bài toán xếp balo (Knapsack Problem)
  - Bài toán tô màu bản đồ (Map Coloring Problem)
  - Bài toán xếp quân cờ: Xếp *n* quân hậu/mã trên bàn cờ *n* x *n* sao cho không quân nào khống chế quân nào.
  - ….

Phương pháp nhánh cận

- **Bài toán tối ưu**
  - Bài toán yêu cầu tìm ra một phương án tốt nhất thỏa mãn một số yêu cầu ràng buộc nào đó – nghiệm của bài toán đạt giá trị max/min trong không gian nghiệm.
  - Thuộc lĩnh vực Tối ưu toán học hoặc Quy hoạch toán học. Lời giải toán có thể khó => Sự vào cuộc của Tin học
  - Hai hướng tiếp cận tìm lời giải tối ưu cho bài toán:
    - Tìm từng lời giải, khi hoàn tất một lời giải thì so sánh chi phí của nó với chi phí tốt nhất hiện có. Nếu tốt hơn thì cập nhật chi phí tốt nhất mới.
    - Với mỗi lời giải, khi xây dựng các thành phần nghiệm luôn kiểm tra điều kiện nếu đi tiếp theo hướng này thì có khả năng nhận được lời giải tốt hơn lời giải hiện có không? Nếu không thì thôi không đi theo hướng này nữa. => Nguyên lí ***nhánh cận*** (*Branch and Bound*)

Phương pháp nhánh cận

- **Ý tưởng**
  - Nhánh cận (Branch and Bound): Thuật toán tìm lời giải cho các bài toán tối ưu dạng liệt kê cấu hình dựa trên nguyên lí đánh giá nhánh cận.
  - ***Nguyên lí đánh giá nhánh cận***: Sử dụng các thông tin đã tìm được trong lời giải của bài toán để loại bỏ sớm phương án không dẫn tới lời giải tối ưu
  - Bản chất:
    - Sử dụng phương pháp quay lui nhưng tại mỗi bước đưa thêm thao tác đánh giá giá trị phương án hiện có.
    - Nếu đó là phương án tối ưu hoặc có hy vọng trở thành phương án tối ưu (tức là tốt hơn phương án hiện có) thì cập nhật lại phương án tối ưu hoặc đi tiếp theo hướng đó.
    - Trong trường hợp ngược lại thì bỏ qua hướng đang xét.

Phương pháp nhánh cận

- **Mô hình**

  Không gian của bài toán (tập khả năng) $D=\{(x_1,x_2,\ldots,x_n)\}$ gồm các cấu hình liệt kê có dạng $(x_1, x_{2, \ldots\ldots,} x_n)$.

  Mỗi cấu hình x sẽ xác định một giá trị hàm chi phí $f(x)$

  $$f : D \rightarrow Z \quad f(x) = C \ \ (C \in Z)$$

  Nghiệm của bài toán: $x= (x_1,x_2,\ldots,x_n)$ sao cho $f(x)$= *giá trị tối ưu (max/min)*

  - Cho x_i nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x_i xét khả năng chọn x_{i+1}, $x_{i+2}\ldots$
  - Tại mỗi bước i: Xây dựng thành phần x_i
    - Xác định x_i theo khả năng *v*.
    - Tính chi phí lời giải nhận được. Nếu "*tốt hơn*" lời giải hiện thời thì chấp nhận x_i theo khả năng *v*. Tiếp tục xác định $x_{i+1},\ldots$đến khi gặp nghiệm
    - Nếu không có một khả năng nào chấp nhận được cho x_i hoặc lời giải xấu hơn thì lùi lại bước trước để xác định lại thành phần x_{i-1}.

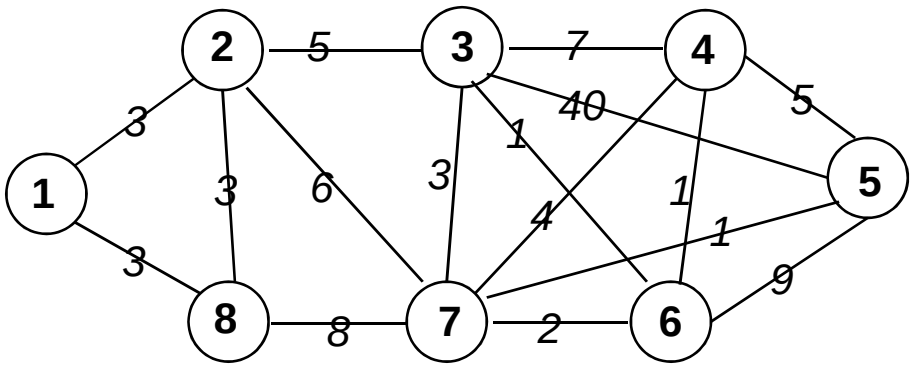
Phương pháp nhánh cận

- **Lược đồ**

```
Try(i,S )≡ //Sinh thành phần thứ i của cấu hình với chi phí hiện thời S
  for (v thuộc tập khả năng thành phần nghiệm x_i)
    if( v là chấp nhận được)
      T = S + Chi phí nghiệm khi có thêm thành phần v;
      if (T tốt hơn Toptimize) //Tốt hơn chi phí tốt nhất hiện có
        x_i  = v ;
        <Ghi nhận trạng thái chấp nhận v>;
        if( x_i là trạng thái kết thúc)
          <Ghi nhận nghiệm>;
          Toptimize = T; //Cập nhật chi phí tốt nhất
        else
          Try(i+1,T); //sinh thành phần tiếp theo với chi phí hiện thời T
        endif;
        <Khôi phục trạng thái chưa chấp nhận v>;
      endif;
    endif;
End.
```

Phương pháp nhánh cận

- **Ví dụ:**
  - Bài toán người đưa hàng (*Traveling Salesman*): Một người bán hàng trên hệ thống *n* thành phố. Giữa các thành phố có thể có hoặc không các đường nối, mỗi đường nối có chi phí xác định từ trước. Người bán hàng xuất phát từ một thành phố, đi tới tất cả các thành phố khác và mỗi thành phố đi qua một lần và quay trở lại thành phố ban đầu. Hãy xác định một hành trình sao cho tổng chi phí trên đường đi là nhỏ nhất.



Phương pháp nhánh cận

- **Ví dụ: Người bán hàng**
  - Phân tích: Biểu diễn mạng lưới giao thông giữa các thành phố như một đồ thị có trọng số $G=(V,E)$
    - Mỗi thành phố là một nút của đồ thị (đánh số 1,..,n)
    - Mỗi đường đi giữa các thành phố là một cạnh nối giữa các nút của đồ thị, có thể có hướng hoặc vô hướng, trên đó có ghi trọng số là chi phí đường đi

      Các cặp cạnh không có đường đi trọng số là ∞
    - Đỉnh xuất phát ≡ kết thúc: $S \in V$
    - Đường đi tìm được là dãy $S=x_1,x_2,\ldots,x_n,x_1=S$ với $x_i \in V$, $(x_i,x_{i+1}) \in E$, có tổng chi phí nhỏ nhất

    $\Rightarrow$Sinh các dãy hoán vị 1..n và tính dãy có chi phí nhỏ nhất:

    - Quay lui
    - Nhánh cận

Phương pháp nhánh cận

- **Ví dụ: Người bán hàng**
  - Phân tích: Tiếp cận theo phương pháp nhánh cận
    - Chi phí tốt nhất đã tìm được (*BestCost*). Ban đầu *BestCost*= +∞
    - Tại mỗi bước chọn x_i : Chi phí đường đi từ x_1 đến x_{i-1} là C
      - Với mỗi khả năng v, tính chi phí $C_1 = C$ + chi phí từ x_{i-1} tới v
      - Nếu C_1 xấu hơn (lớn hơn) *BestCost* hoặc không có khả năng nào chấp nhận được cho x_i thì lùi lại bước trước để xác định lại thành phần x_{i-1}.
      - Nếu C_1 tốt hơn (nhỏ hơn) *BestCost* thì chấp nhận x_i theo khả năng v. Tiếp tục xác định x_{i+1},…
      - Đến khi gặp nghiệm ($i=n+1$ & $x_i=S$):
        - Cập nhật đường đi tốt nhất hiện tại đã tìm được
        - Cập nhật giá trị *BestCost* mới: *BestCost* = C_1
    - Kết thúc tìm kiếm nếu *BestCost*= +∞ => không có đường đi

Phương pháp nhánh cận

- **Ví dụ 2: Người bán hàng**
  - Mô tả dữ liệu
    - Đánh chỉ số các đỉnh của đồ thị từ 1..*n*
    - Biểu diễn đồ thị G bằng ma trận kề $M = (c_{ij})$ cỡ n x n

      c_{ij} = *cost* nếu có cạnh nối đỉnh *i* với đỉnh *j* với chi phí *cost*

      = ∞ nếu không có đường đi
    - Mảng: Daqua[1..*n*] đánh dấu đỉnh *i* đã được đi qua trên đường đi hay chưa?

      Daqua[*i*]= *true* nếu đỉnh *i* đã có trên đường đi

      = *false* nếu ngược lại, đỉnh *i* chưa có trên đường đi

      Khởi tạo: Daqua[1..*n*] = *false*

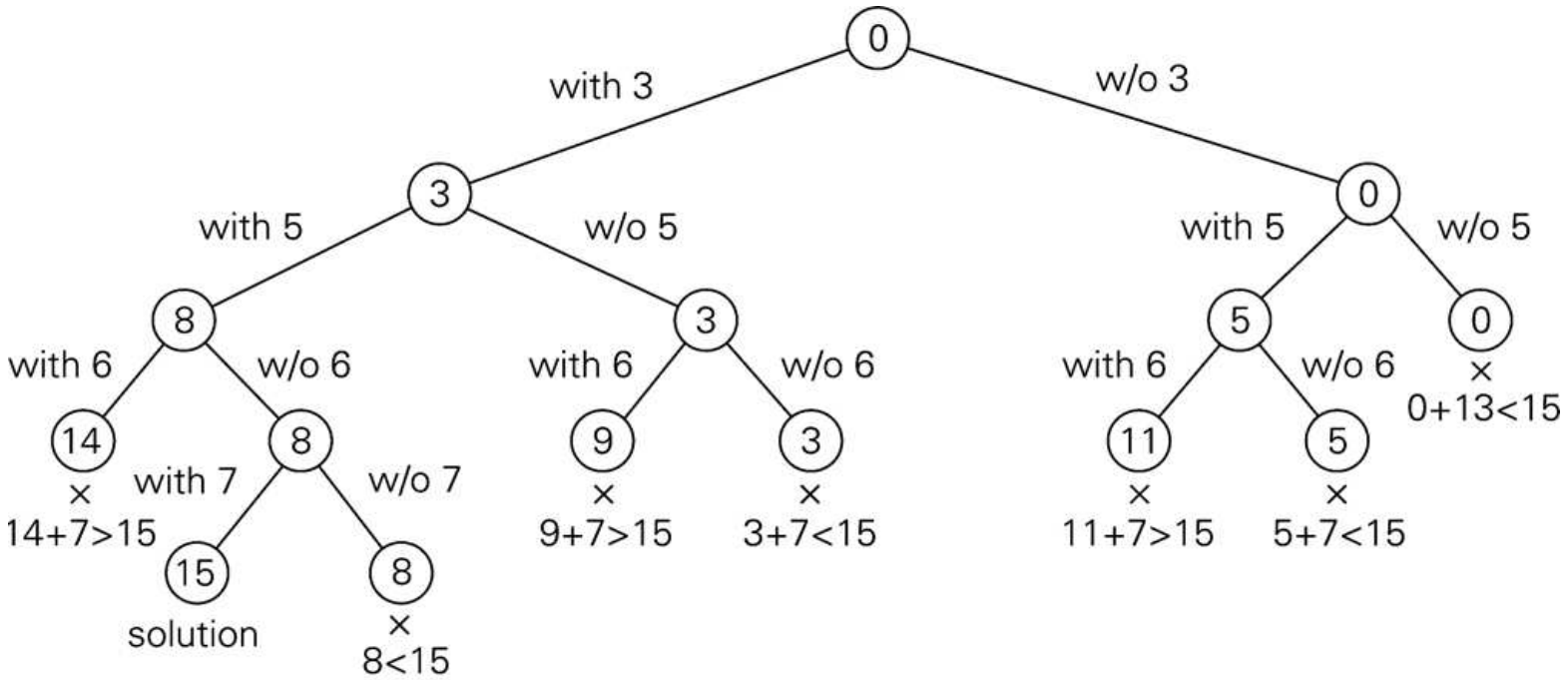
Phương pháp nhánh cận

- **Ví dụ: Người bán hàng**
  - Thuật toán

```
Try(i,C)≡ //Sinh thành phần thứ i của cấu hình với chi phí hiện thời C
    for (v = 1..n)
      if (c[x_{i-1},v]<∞)&(not Daqua[v])) )
        C1 = C + c[x_{i-1},v];
        if (C1 < BestCost) //Tốt hơn chi phí tốt nhất hiện có
           x_i = v ;
           Daqua[v] = true;
           if (i = n+1)&(x_i = S)
             <Ghi nhận nghiệm x_1,x_2…x_{n+1}>;
             BestCost = C1; //Cập nhật chi phí tốt nhất
           else if (i<=n)
              Try(i+1,C1); //sinh thành phần tiếp theo với chi phí hiện thời C1
           endif;
           Daqua[v] = false;
        endif;
      endif;
End.
```

Phương pháp nhánh cận

- **Bài toán tổng tập con (Subset-Sum Problem):** tìm tổng một tập con của tập S = {s1 , s2 , . . . , sn } n số nguyên dương có tổng bằng một giá trị d cho trước (VD: S = {3, 5, 6, 7, 8, 9}, d = 15.)



Phương pháp nhánh cận

- **Các bài toán khác**
  - Xếp balo (Knapsack problem)
  - Quy hoạch nguyên (Integer programming)
  - Quy hoạch phi tuyến (Nonlinear programming)
  - Người bán hàng (Traveling salesman problem - TSP)
  - Bài toán thỏa được cực đại (Maximum satisfiability problem - MAX-SAT)
  - Tìm kiếm láng giềng gần nhất (Nearest neighbor search - NNS)
  - Cutting stock problem
  - False noise analysis (FNA)
  - …

  Xem thêm http://en.wikipedia.org/wiki/Branch_and_bound