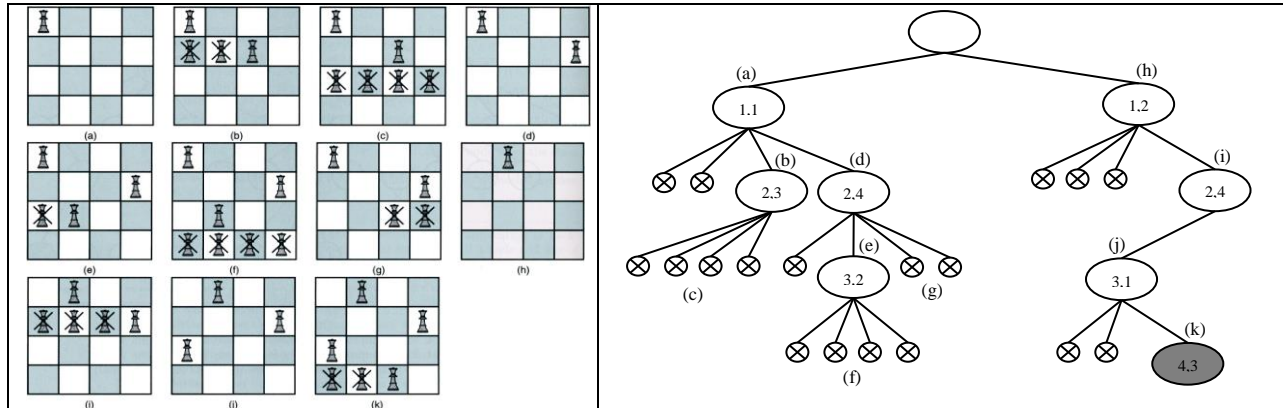


Chương 4: Kỹ thuật Quay lui và Nhánh cận

Quay lui: Giới thiệu chung

Phát biểu: Đặt n quân Hậu lên bàn cờ $n \times n$ sao cho chúng không thể tấn công lẫn nhau.

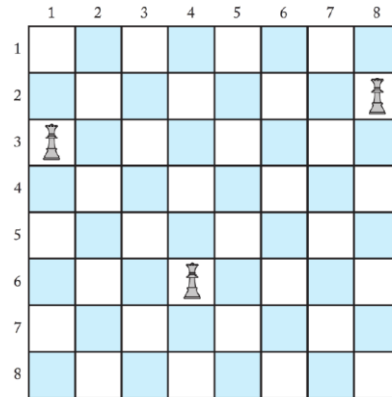


Dạng thức thứ nhất của giải thuật Quay lui:

```
Backtracking(u) {
    if (promising(u))
        if (Tồn tại một lời giải tại u)
            Xuất lời giải;
        else
            for (Mỗi nút con v của u)
                Backtracking(v);
}
```

Dạng thức thứ hai của giải thuật Quay lui:

```
Backtracking(u) {
    for (Mỗi nút con v của u)
        if (promising(v))
            if (Tồn tại một lời giải tại u)
                Xuất lời giải;
            else
                Backtracking(v);
}
```



Giải thuật (n-Hậu)

```

promising(i) {
    j = 1;
    flag = true;
    while (j < i && flag) {
        if (col[i] == col[j] || abs(col[i] - col[j]) == i - j)
            flag = false;
        j++;
    }
    return flag;
}

Dạng thức một
n_Queens(i) {
    if (promising(i))
        if (i == n)
            print(col[1 .. n]);
    else
        for (j = 1; j ≤ n; j++) {
            col[i + 1] = j;
            n_Queens(i + 1);
        }
}

n_Queens(0);

Dạng thức hai
n_Queens(i) {
    for (j = 1; j ≤ n; j++) {
        col[i] = j;
        if (promising(i))
            if (i == n)
                print(col[1 .. n]);
        else
            nQueens(i + 1);
    }
}

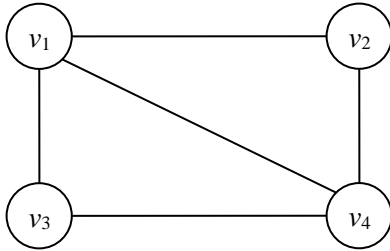
n_Queens(1);

```

Bài toán Tô màu

Phát biểu: Xác định tất cả các cách khi sử dụng m màu để tô n đỉnh của đồ thị vô hướng, sao cho hai đỉnh kề nhau không có cùng màu.

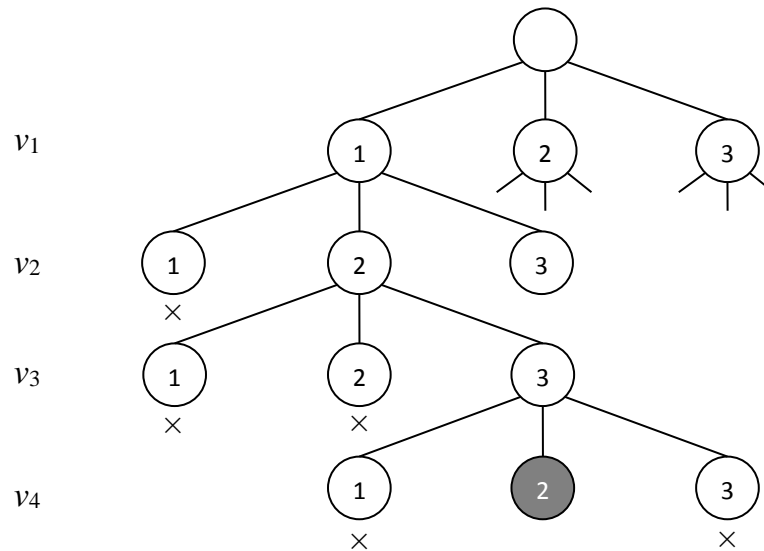
Ví dụ: Xét đồ thị có 4 đỉnh sau với $m = 3$.



Đỉnh	Màu	
v_1	1	2
v_2	2	3
v_3	3	1
v_4	2	3

Chú ý: Các màu được biểu diễn từ 1 đến m

Cây không gian các trạng thái



Giải thuật

```

M_coloring(i) {
    if (promising(i))
        if (i == n)
            print(color[1 .. n]);
        else
            for (j = 1; j ≤ m; j++) {
                color[i + 1] = j;
                M_coloring(i + 1);
            }
}

```


```

promising(i) {
    j = 1;
    flag = true;
    while (j < i && flag) {
        if (W[i][j] && color[i] == color[j])
            flag = false;
        j++;
    }
    return flag;
}
M_coloring(0);

```

Bài toán Ngựa đi tuần

Phát biểu: Đặt quân ngựa lên ô $\langle r_0, c_0 \rangle$ của bàn cờ $n \times n$. Hãy chỉ ra mọi hành trình (nếu có), đi qua tất cả các ô đúng một lần.

	4		3		-2
5				2	-1
					0
6				1	1
	7		0		2
-2	-1	0	1	2	

Giải thuật

```

KnightTour(int i, int r, int c) {
    for (k = 1; k ≤ 8; k++) {
        u = r + dong[k];
        v = c + cot[k];
        if ((1 ≤ u, v ≤ n) && (h[u][v] == 0)) {
            h[u][v] = i;
            if (i == n * n)
                Print(h);
            else
                KnightTour(i + 1, u, v);
            h[u][v] = 0;
        }
    }
}
h[r0][c0] = 1;
KnightTour(2, r0, c0);

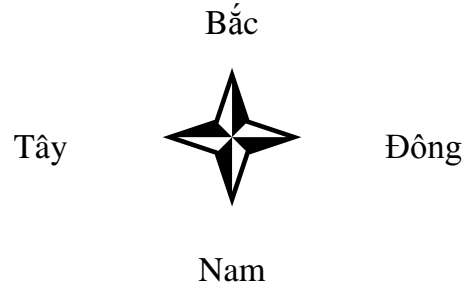
```

Bài toán Tìm đường trong mê cung

Phát biểu: Một người máy tìm đường đi trong mê cung. Từ vị trí *xuất phát*, người máy tìm vị trí *đích* (cổng thoát). Tại mỗi thời điểm, người máy chỉ có thể đi một bước theo một trong bốn hướng (duyệt lần lượt): Bắc, Đông, Nam, Tây.

#	S	#	#	.	#
#	#
.	#	.	#	.	#
.	.	.	.	#	#
.	.	#	#	.	G
#	#
#	#	#	.	#	#

(a)



#	S	#	#	.	#
#	x	x	x	x	#
.	#	.	#	.	#
.	.	.	.	#	#
.	.	#	#	.	G
#	#
#	#	#	.	#	#

(b)

#	S	#	#	.	#
#	x	x	.	.	#
.	#	x	#	.	#
.	x	x	.	#	#
.	x	#	#	x	G
#	x	x	x	x	#
#	#	#	.	#	#

(c)

	Bắc	Đông	Nam	Tây	
xx####	xx####	xx####	xx####	xx####	xx####
#x#..#	#x#..#	#x#..#	#x#..#	#x#..#	#x#..#
#x#..#	#x#..#	#x#..#	#x#..#	#x#..#	#x#..#
#x#.#	#x#.#	#x#.#	#x#.#	#x#.#	#x#.#
###...	###...	###...	###...	###...	###...
G...##	G...##	G...##	G...##	G...##	G...##

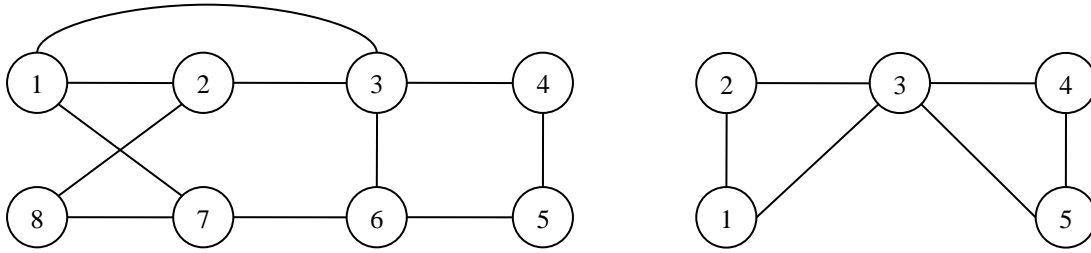
Giải thuật

```

bool Find_Path(r, c) {
    if ((r, c) ∉ Maze) return false;
    if (Maze[r][c] == 'G') return true;
    if (Maze[r][c] == 'x') return false;
    if (Maze[r][c] == '#') return false;
    Maze[r][c] = 'x';
    if (Find_Path(r - 1, c) == true) return true;
    if (Find_Path(r, c + 1) == true) return true;
    if (Find_Path(r + 1, c) == true) return true;
    if (Find_Path(r, c - 1) == true) return true;
    Maze[r][c] = '.';
    return false;
} Find_Path(r0, c0);

```

Chu trình Hamilton



Giải thuật

```

bool promising(int pos, int v) {
    if (pos == n && G[v][path[1]] == 0) return false;
    else
        if (G[path[pos - 1]][v] == 0) return false;
        else
            for (int i = 1; i < pos; i++)
                if (path[i] == v)
                    return false;
    return true;
}

Hamiltonian(bool G[1..n][1..n], int path[1..n], int pos) {
    if (pos == n + 1)
        print(path);
    else
        for (v = 1; v ≤ n; v++)
            if (promising(pos, v)) {
                path[pos] = v;
                Hamiltonian(G, path, pos + 1);
            }
}

path[1 .. n] = -1;
path[1] = 1;
Hamiltonian(G, path, 2);

```

Bài toán Tổng các tập con

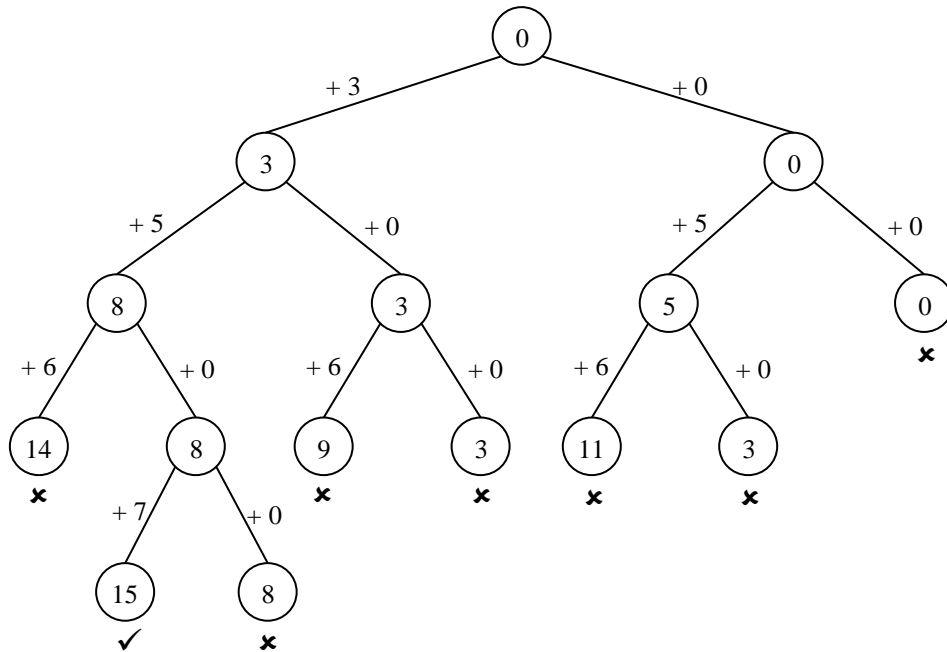
Phát biểu: Tìm tập con của tập đã cho $\mathcal{W} = \{w_1, w_2, \dots, w_n\}$ gồm n số nguyên dương sao cho tổng các phần tử của tập con này bằng với w .

Dạng 1

Một lời giải S sẽ là một vector kích thước n :

$$S = \{s_1, s_2, \dots, s_n\}$$

với $s_i = 1$ hoặc 0 , tương ứng với w_i có thuộc về tập con cần tìm hay không.

**Giải thuật**

```

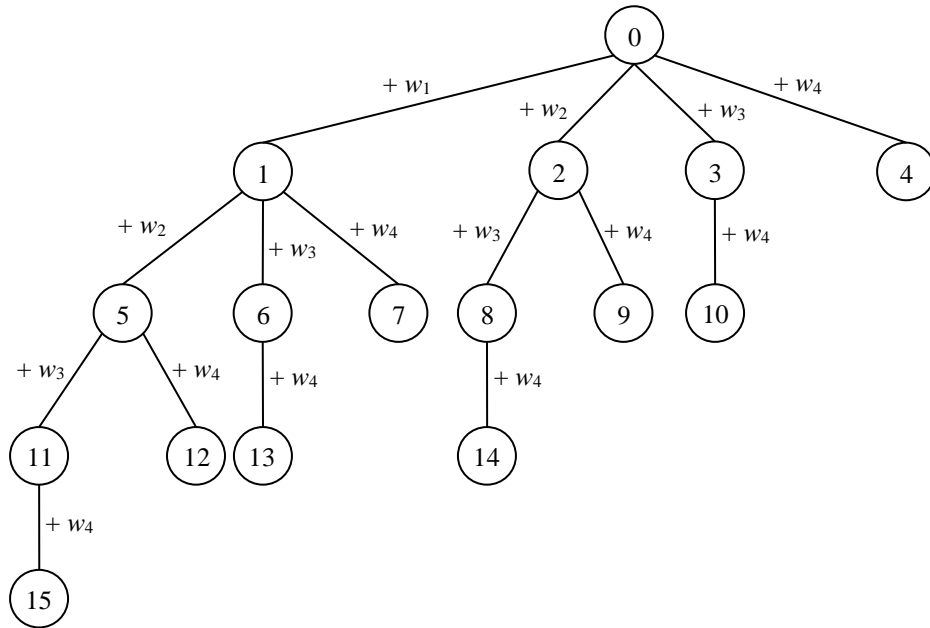
sum_of_subsets(k, sum, total, r[1 .. n]) {
    if (sum == W)
        print(r);
    else
        if ((sum + total ≥ W) && (sum + val[k] ≤ W)) {
            r[k] = true;
            sum_of_subsets(k + 1, sum + val[k], total - val[k], r);

            r[k] = false;
            sum_of_subsets(k + 1, sum, total - val[k], r);
        }
}

bool r[1 .. n] = {false};
int total = Σi=1n val[i];
sort(val, n);
if (val[1] ≤ W ≤ total)
    sum_of_subsets(1, 0, total, r);

```

Dạng 2: Giả sử tập ban đầu có 4 giá trị: $\mathcal{W} = \{w_1, w_2, w_3, w_4\}$.



Giải thuật (Quay lui vết cặn)

```

subset_sum(r[1..n], size, sum, start) {
    if (sum == W) Print(r);
    else
        for (i = start; i ≤ n; i++) {
            r[size] = val[i];
            subset_sum(r, size + 1, sum + val[i], i + 1);
        }
}
subset_sum(r, 1, 0, 1);

```

Giải thuật

```

subset_sum(r[1 .. n], size, sum, start, total) {
    if (W == sum) print(r);
    else {
        lost = 0;
        for (i = start; i ≤ n; i++) {
            if ((sum + total - lost ≥ W) && (sum + val[i] ≤ W)) {
                r[size] = val[i];
                subset_sum(r, size + 1, sum + val[i], i+1, total - val[i] - lost);
            }
            lost += val[i];
        }
    }
}

total =  $\sum_{i=0}^{n-1} val[i]$ ;
sort(val, n);
if (val[1] ≤ W ≤ total) subset_sum(r, 1, 0, 1, total);

```


Nhánh cận: Giới thiệu chung

Đây là một cải tiến của kỹ thuật quay lui, tuy vẫn sử dụng khái niệm *cây không gian các trạng thái* để xử lý vấn đề. Sự khác biệt nằm ở chỗ:

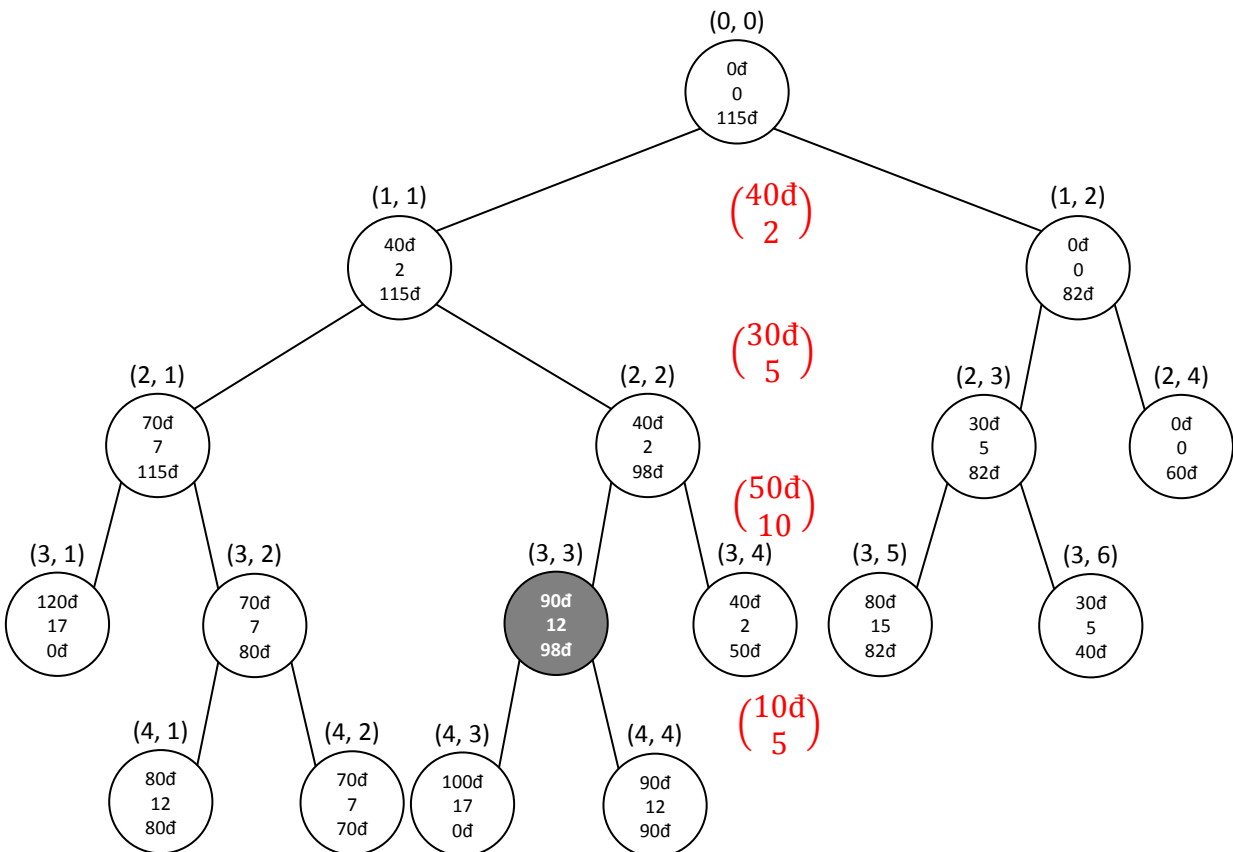
1. Không cố định một phép duyệt cây cụ thể
2. Chỉ được sử dụng cho những vấn đề tối ưu

Ví dụ: (Bài toán túi xách 0–1)

Xét thể hiện dữ liệu sau với $n = 4$, $W = 16$, đồ vật thứ i sẽ có cân nặng w_i , giá trị v_i (tất cả đều là những số nguyên dương):

i	v_i	w_i	$\frac{v_i}{w_i}$
1	40đ	2	20đ
2	30đ	5	6đ
3	50đ	10	5đ
4	10đ	5	2đ

Nhánh cận với tìm kiếm rộng (*breadth-first search with branch-and-bound pruning*)



Giải thuật

```

struct node {
    int    level;
    int    profit;
    int    weight;
};

Knapsack_B&B_BrFS(n, val[1..n], wei[1..n], W) {
    Queue Q;
    node u, v;
    initialize(Q);
    u.level = 0;
    u.profit = u.weight = 0;
    maxprofit = 0;

    enqueue(Q, u);
    while (!empty(Q)) {
        u = dequeue(Q);
        v.level = u.level + 1;

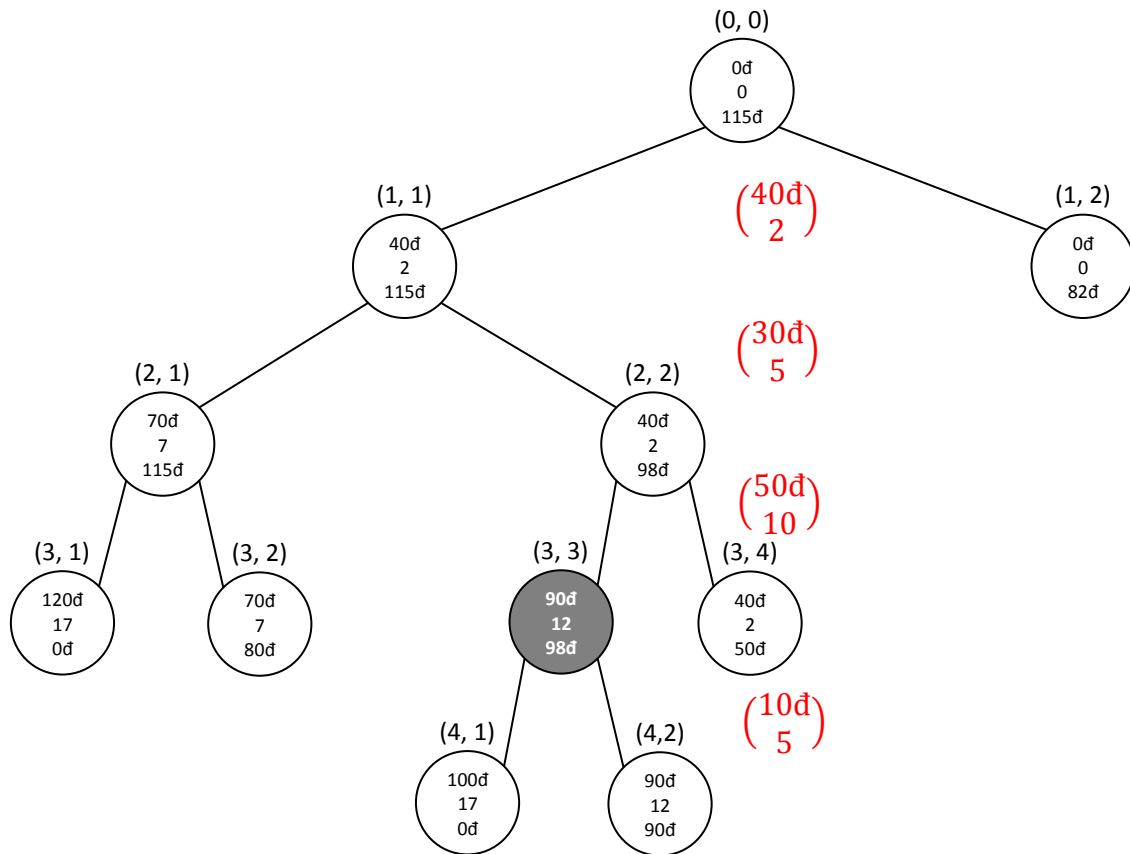
        v.weight = u.weight + wei[v.level];
        v.profit = u.profit + val[v.level];
        if (v.weight ≤ W && v.profit > maxprofit)
            maxprofit = v.profit;
        if (Bound(v) > maxprofit)
            enqueue(Q, v);

        v.weight = u.weight;
        v.profit = u.profit;
        if (Bound(v) > maxprofit)
            enqueue(Q, v);
    }
    return    maxprofit;
}

Bound(node u) {
    if (u.weight ≥ W)
        return    0;
    p = u.profit;
    j = u.level + 1;
    accWeight = u.weight;
    while (j ≤ n && accWeight + wei[j] ≤ W) {
        accWeight += wei[j];
        p += val[j];
        j++;
    }
    if (j ≤ n)
        p += (W - accWeight) * val[j] / wei[j];
    return    ⌈p⌉;
}

```

Nhánh cận với tìm kiếm dựa trên giá trị tốt nhất (best-first search with branch-and-bound pruning)



Giải thuật

```
struct node {
    int    level;
    int    profit;
    int    weight;
    float  bound;
};

Knapsack_B&B_BeFS(n, val[1..n], wei[1..n], W) {
    PriorityQueue pQ;
    node u, v;

    initialize(pQ);
    u.level = 0;
    u.profit = u.weight = 0;
    maxprofit = 0;
    u.bound = Bound(u);
```

```

enqueue(pQ, u);
while (!empty(pQ)) {
    u = dequeue(pQ);
    if (u.bound > maxprofit) {
        v.level = u.level + 1;

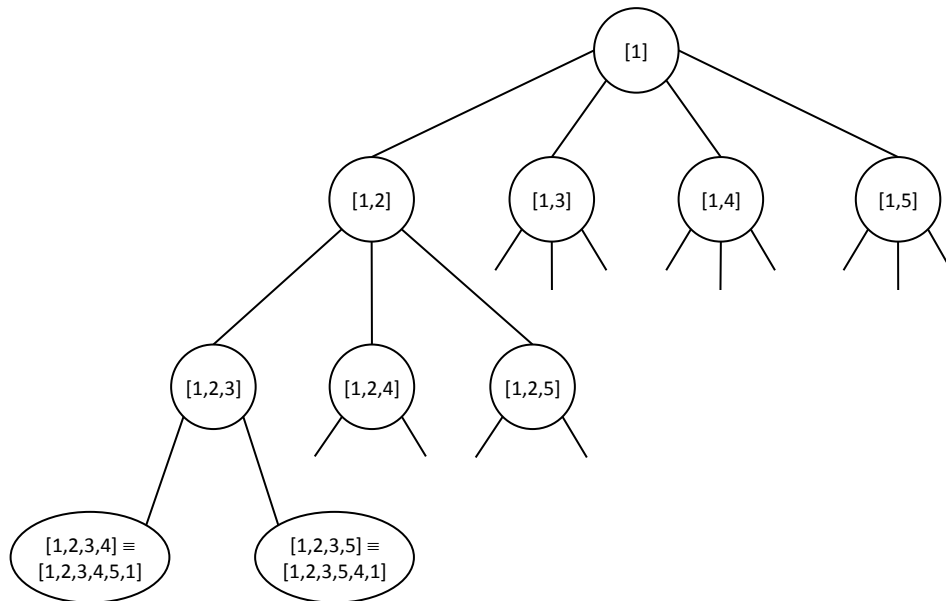
        v.weight = u.weight + wei[v.level];
        v.profit = u.profit + val[v.level];
        if (v.weight ≤ W && v.profit > maxprofit)
            maxprofit = v.profit;
        v.bound = Bound(v);
        if (v.bound > maxprofit)
            enqueue(pQ, v);

        v.weight = u.weight;
        v.profit = u.profit;
        v.bound = Bound(v);
        if (v.bound > maxprofit)
            enqueue(pQ, v);
    }
}
return maxprofit;
}

```

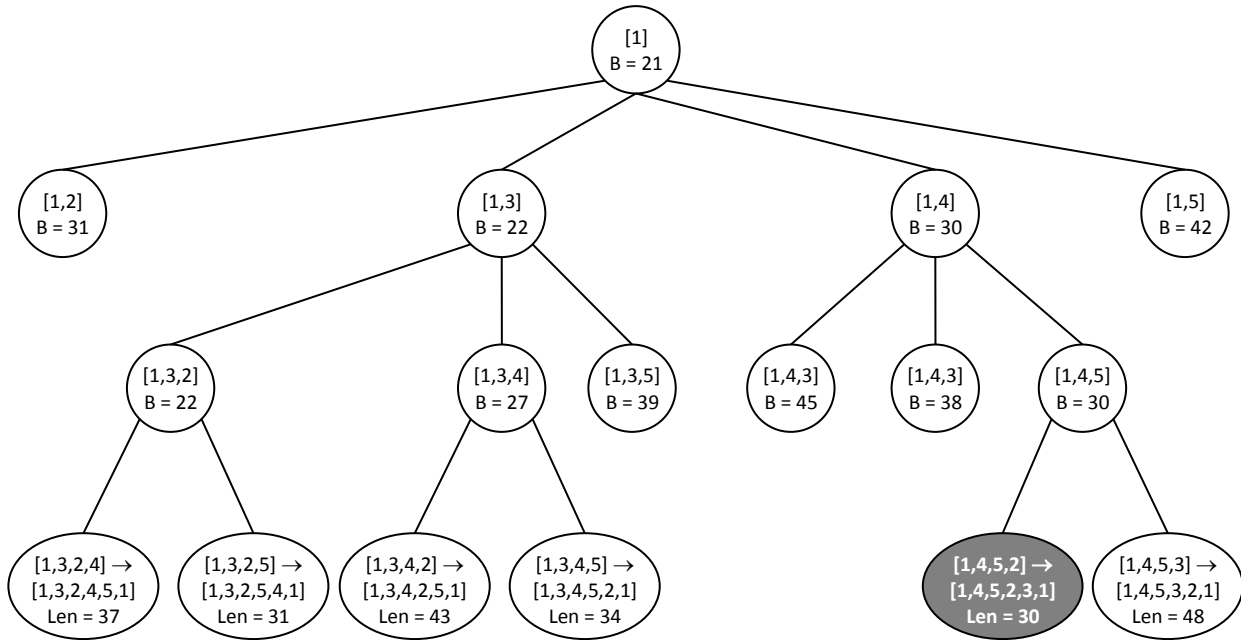
Bài toán Đường đi người bán hàng

Ví dụ: Giả sử đồ thị chứa 5 đỉnh



Ví dụ: Xét đồ thị được biểu diễn bởi ma trận kề sau:

	v_1	v_2	v_3	v_4	v_5
v_1	0	14	4	10	20
v_2	14	0	7	8	7
v_3	4	5	0	7	16
v_4	11	7	9	0	2
v_5	18	7	17	4	0



Giải thuật

```

struct node {
    int level;
    ordered_set path;
    float bound;
};

TSP_BB_BeFS(n, G[1..n][1..n]) {
    PriorityQueue pQ;
    node u, v;
    initialize(pQ);

    u.level = 0;
    u.path = [1];
    u.bound = TSP_Bound(u);

    minlen = +∞;
  
```

```

enqueue(pQ, u);
while (!empty(pQ)) {
    u = dequeue(pQ);
    if (u.bound < minlen) {
        v.level = u.level + 1;
        for (each  $i \in [2, n]$  &&  $i \notin u.path$ ) {
            v.path = u.path;
            add i to the right-end of v.path;
            if (v.level == n - 2) {
                // let  $j \in [2, n]$  and  $j \notin v.path$ 
                add j to the right-end of v.path;
                add 1 to the right-end of v.path;
                if (Length(v) < minlen) {
                    minlen = Length(v);
                    opttour = v.path;
                }
            }
            else {
                v.bound = TSP_Bound(v);
                if (v.bound < minlen)
                    enqueue(pQ, v);
            }
        }
    }
}
}
}

```