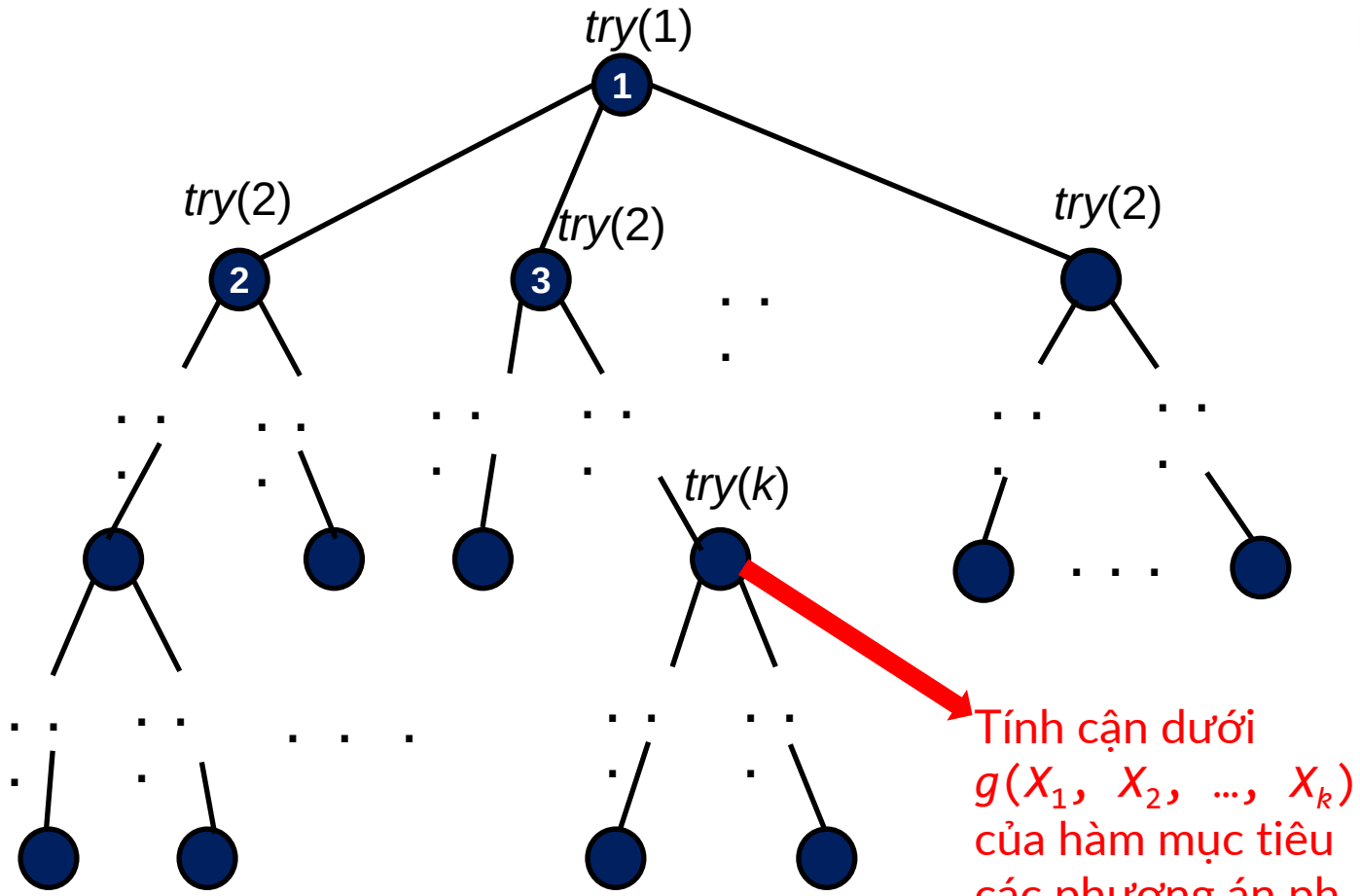
Nhánh và cận

- Một trong số các phương pháp để giải bài toán tối ưu tổ hợp
- Dựa trên kỹ thuật quay lui để xét tất cả các phương án có thể có
- Kết hợp phân tích cận (cận dưới đối với bài toán tìm min và cận trên đối với bài toán tìm max) để cắt bớt nhánh cần phải duyệt

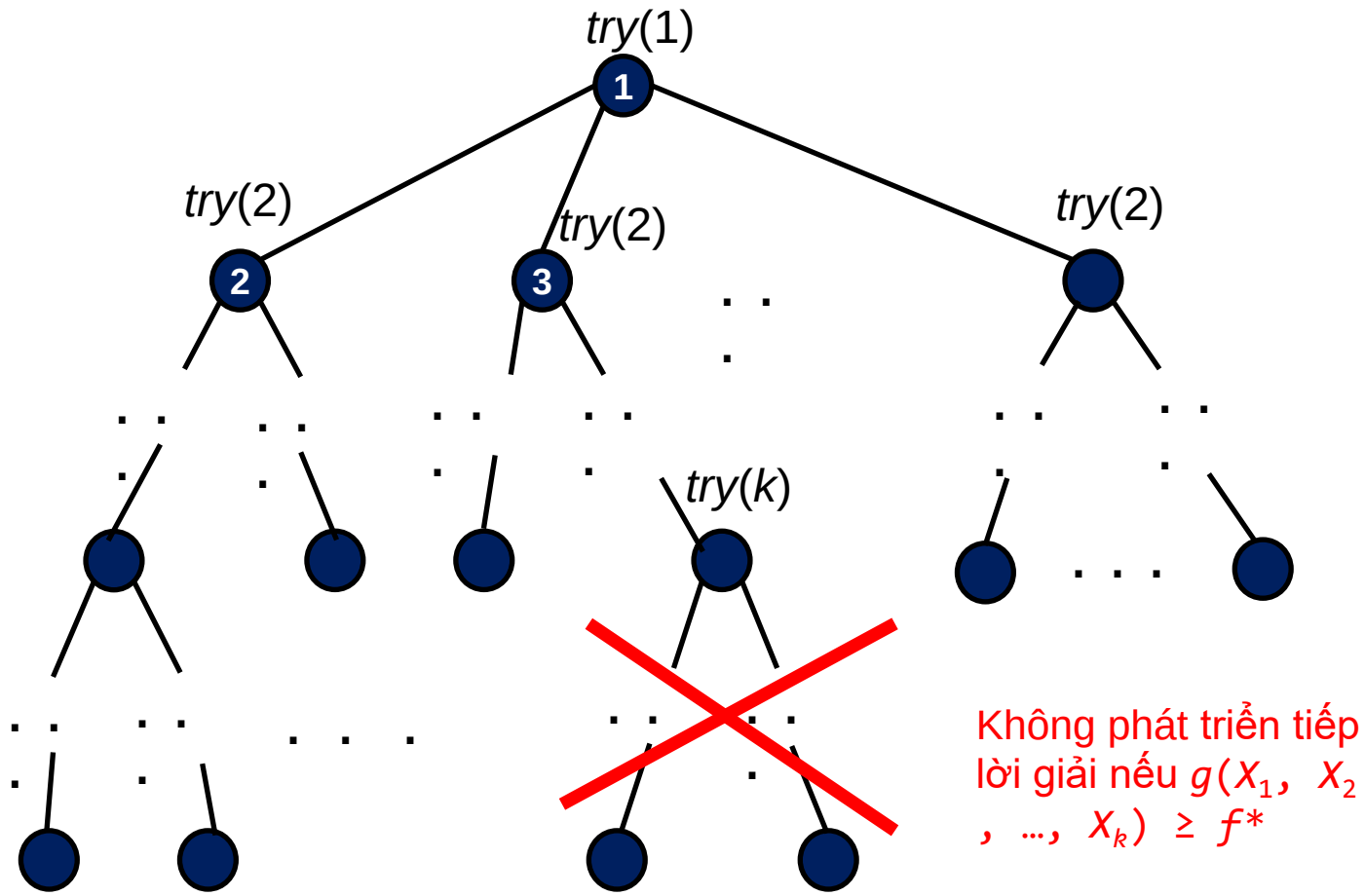


Bài toán tìm min (ký hiệu f^* : kỷ lục)

```
try(k){ // thử các giá trị có thể gán cho X_k
  for v in A_k do {
     if check(v,k){
        X_k = v;
        [Update a data structure D]
        if k = n then updateBest();
        else {
            if g(X_1, X_2, …, X_k) < f* then
                try(k+1);
        }
        [Recover the data structure D]
     }
  }
}
```

Nhánh và cận

- Một trong số các phương pháp để giải bài toán tối ưu tổ hợp
- Dựa trên kỹ thuật quay lui để xét tất cả các phương án có thể có
- Kết hợp phân tích cận (cận dưới đối với bài toán tìm min và cận trên đối với bài toán tìm max) để cắt bớt nhánh cần phải duyệt



Bài toán tìm min (ký hiệu f^* : kỷ lục)

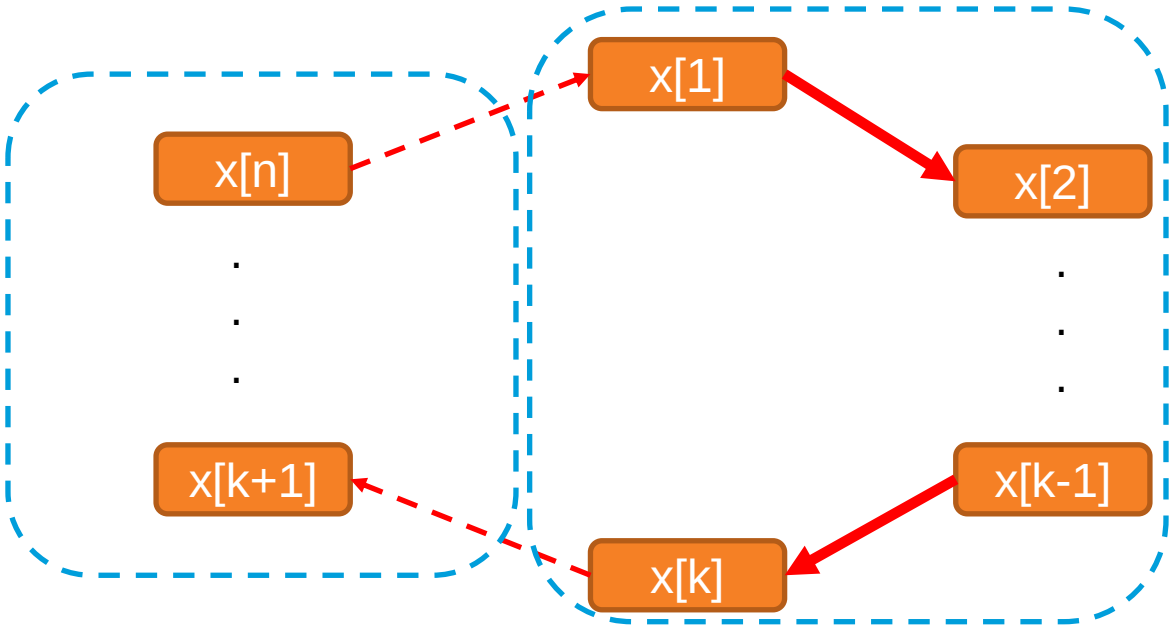
```
try(k){ // thử các giá trị có thể gán cho X_k
  for v in A_k do {
     if check(v,k){
        X_k = v;
        [Update a data structure D]
        if k = n then updateBest();
        else {
            if g(X_1, X_2, …, X_k) < f* then
                try(k+1);
        }
        [Recover the data structure D]
     }
  }
}
```

Nhánh và cận: Bài toán TSP

- Cho 1 điểm 1, 2, . . ., *n*. Khoảng cách đi từ điểm *i* đến điểm *j* là c(*i,j*), với mọi *i, j* = 1, 2, . . ., *n*. Hãy tìm hành trình có tổng khoảng cách di chuyển nhỏ nhất xuất phát từ điểm 1, đi qua các điểm 2, 3, . . ., *n*, mỗi điểm đúng 1 lần rồi quay về 1.

Nhánh và cận: Bài toán TSP

- Biểu diễn phương án: *x*[1, . . ., *n*] trong đó x[*i*] là điểm thứ *i* trên hành trình, *i* = 1, 2, …, *n*. Chu trình sẽ là x[1] → x[2] → . . . → x[n] → x[1]
- Mảng đánh dấu:
  - mark[v] = 1: nghĩa là điểm v đã xuất hiện trên hành trình
- Nhánh và cận
  - Cm: khoảng cách nhỏ nhất trong số khoảng cách giữa 2 điểm




```
try(k){
  for v = 1 to n do {
     if mark[v] = 0 then {
        x[k] = v;
        f = f + d[x[k-1], v]; mark[v] = 1;
        if k = n then {
           if fmin > f + d[x[n],x[1]] then
              fmin = f + d[x[n],x[1]];
        }else{
           if f + Cm*(n-k+1) < fmin then
              try(k+1);
        }
        f = f – d[x[k-1], v]; mark[v] = 0;
     }
  }
}
```