BÀI 2.1. ĐỆ QUY – QUAY LUI

GIẢI THUẬT ĐỆ QUY

Định nghĩa đệ quy: Đưa ra 1 định nghĩa có sử dụng chính khái niệm đang cần định nghĩa

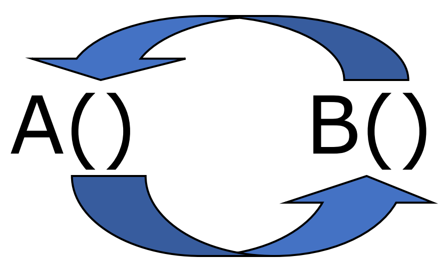
Giải thuật đệ quy

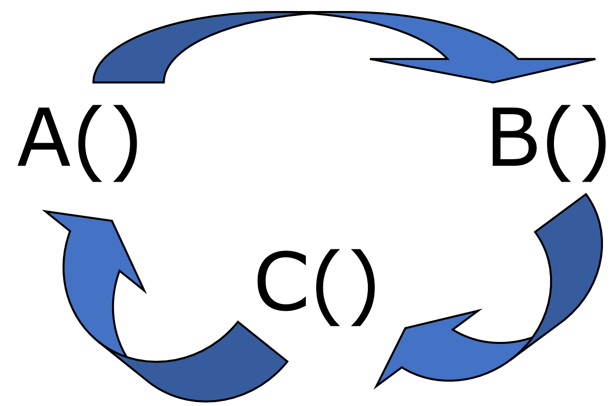
- Nếu bài toán T được thực hiện bằng lời giải của bài toán T' có dạng giống T là lời giải đệ quy
- Giải thuật tương ứng với lời giải như vậy gọi là giải thuật đệ quy.

PHÂN LOẠI GIẢI THUẬT ĐỆ QUY

Đệ quy phân thành 2 loại :

- Đệ quy trực tiếp *(phổ biến nhất)*
- Đệ quy gián tiếp (Tương hỗ):





CÀI ĐẶT ĐỆ QUY

Cấu trúc tổng quát

```
If (suy biến)
  <Giải quyết trường hợp suy biến>;

Else {
    <tiền xử lý đệ qui>;
    <Gọi đệ qui> ;
    <Xử lý hậu đệ qui>;
  }
```

PHÂN LOẠI ĐỆ QUY THEO SỐ LỜI GỌI

1. Đệ quy tuyến tính – gọi đệ quy một lần trong hàm.

```
P (<tham số>) {
if (điều kiện dừng) {
  <Xử lý trường hợp suy biến>
}
else {
<Thực hiện một số công việc (nếu có)>
  P(<tham số>);
 <Thực hiện một số công việc (nếu có)>
}
 }
```

PHÂN LOẠI ĐỆ QUY THEO SỐ LỜI GỌI

Ví dụ: Hàm Fact(n) tính n!

$fact_0 = 1$;
$f_n = n * fact_{n-1;}$ $(n>=1)$

```
long long Fact(int n){
        if (n==0) return 1;
        return n*Fact(n-1);
}
```

PHÂN LOẠI ĐỆ QUY THEO SỐ LỜI GỌI

2. Đệ quy nhị phân.

```
P (<tham số>){
    if (điều kiện dừng){
        <Xử lý trường hợp suy biến>
    }
    else {
       <Thực hiện một số công việc (nếu có)>
        P(<tham số>);
       <Thực hiện một số công việc (nếu có)>
        P(<tham số>);
       <Thực hiện một số công việc (nếu có)>
    }
}
```

PHÂN LOẠI ĐỆ QUY THEO SỐ LỜI GỌI

3. Đệ quy phi tuyến.

```
P (<danh sách tham số>) {
    for (int i = 1; i<=n; i++){
      <Thực hiện một số công việc (nếu có)>
      if (điều kiện dừng){
        <Xử lý trường hợp suy biến>
        }
      else {
            <Thực hiện một số công việc (nếu có)>
            P (<danh sách tham số>);
        }
    }
}
```

ĐÁNH GIÁ VỀ ĐỆ QUY

Ưu điểm:

- Sáng sủa, dễ hiểu, nêu rõ bản chất vấn đề
- Tiết kiệm thời gian hiện thực mã nguồn

Nhược điểm:

- Tốn nhiều bộ nhớ, thời gian thực thi lâu
- Một số bài toán không có lời giải đệ quy

PHƯƠNG PHÁP QUAY LUI (back tracking)

Đặc trưng : là các bước hướng tới lời giải cuối cùng của bài toán hoàn toàn được làm thử.

Tại mỗi bước

- Nếu có một lựa chọn được chấp nhận thì ghi nhận lại lựa chọn này và tiến hành các bước thử tiếp theo.
- Ngược lại, không có lựa chọn nào thích hợp thì làm lại bước trước, xóa bỏ sự ghi nhận và quay về chu trình thử các lựa chọn còn lại

THUẬT TOÁN QUAY LUI

Cần xác định bộ X =(x_1, x_2,..,x_n) thỏa mãn ràng buộc.

Mỗi thành phần x_i ta có n_i khả năng cần lựa chọn.

Ứng với mỗi khả năng $j \in n_i$ dành cho thành phần x_i ta cần thực hiện:

1. Kiểm tra xem j có được chấp thuận cho thành phần x_i hay không? Nếu được:
   - Nếu i là thành phần cuối cùng (i=n) => ghi nhận nghiệm của bài toán.
   - Nếu i chưa phải cuối cùng, gọi đệ quy xác định thành phần thứ i +1.
2. Nếu không có khả năng j nào được chấp thuận thì QUAY LUI lại bước trước đó (i-1)

```
Thuật toán Back-Track ( int i ) {
    For ( j =<Khả năng 1>; j <=ni;  j++ ){
            if (<chấp thuận khả năng j>) {
                    X[i] = <khả năng j>;
                    if ( i ==n) Result();
                    else Back-Track(i+1);
            }
    }
```

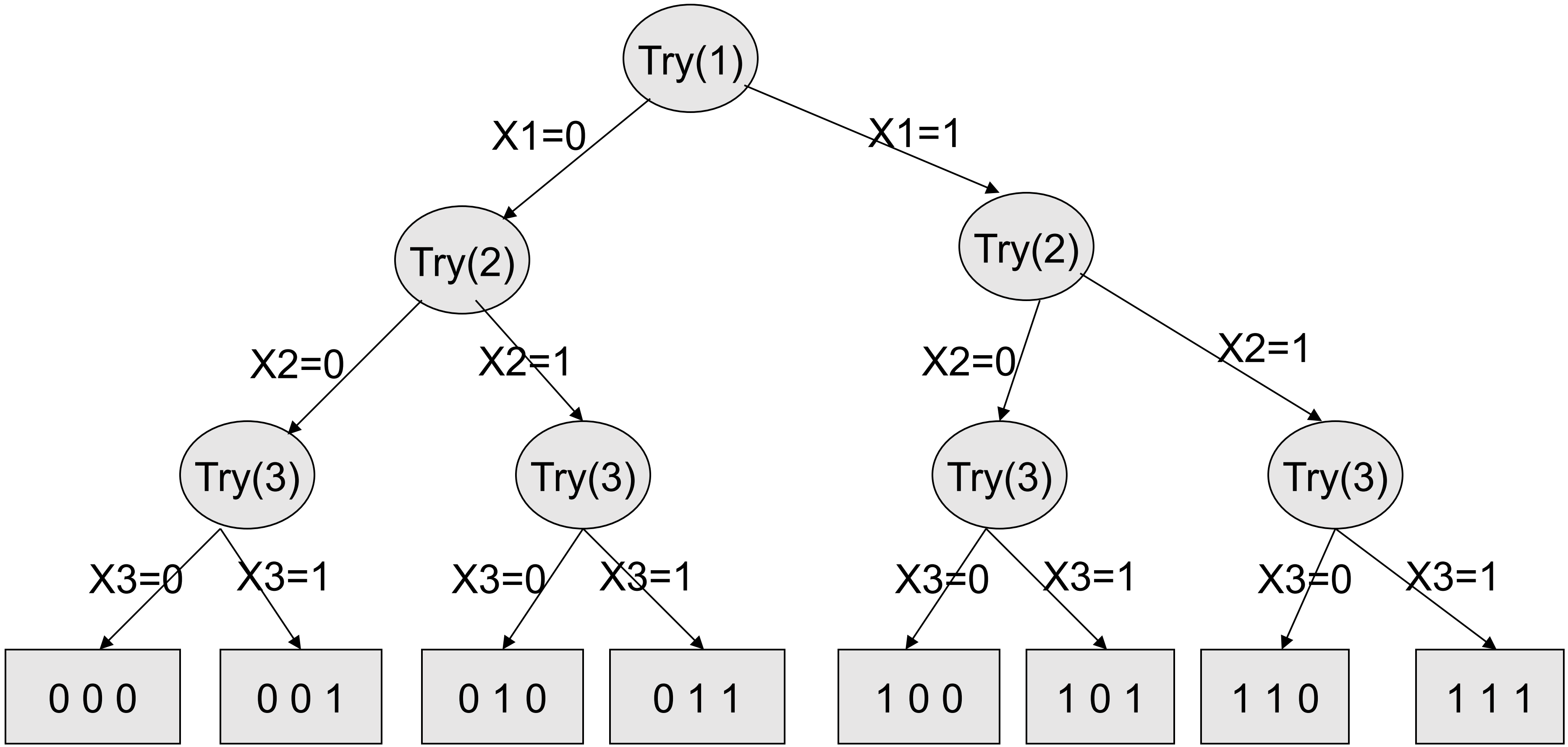
BÀI TOÁN 1: DUYỆT CÁC XÂU NHỊ PHÂN ĐỘ DÀI N

Xâu nhị phân $X = (x_1, x_2,..,x_n)| x_i = 0, 1$. Mỗi $x_i \in X$ có hai lựa chọn $x_i = 0, 1$.

Cả hai giá trị này đều được chấp thuận mà không cần có thêm bất kỳ điều kiện gì.

```
void Try ( int i ) {
        for (int j =0; j<=1; j++){
                X[i] = j;
                if ( i ==n) Result();
                else Try (i+1);
        }
}
```

BÀI TOÁN 1: DUYỆT CÁC XÂU NHỊ PHÂN ĐỘ DÀI N

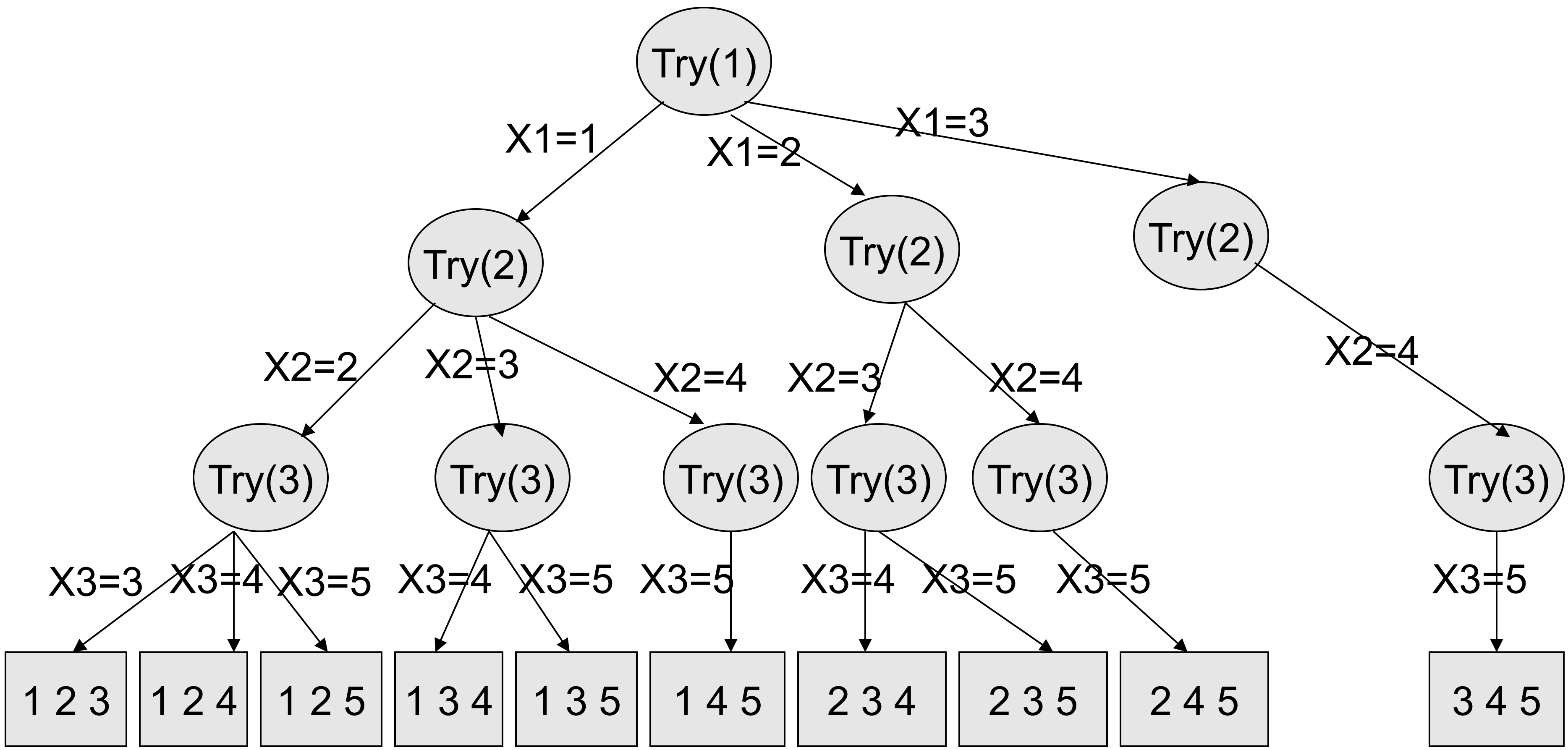


BÀI TOÁN 2: DUYỆT CÁC TẬP CON K PHẦN TỬ CỦA 1,2…N

- Mỗi tập con K phần tử $X = (x_1, x_2,..,x_K)$ là bộ không tính đến thứ tự K phần tử của 1, 2, .., N.
- Mỗi $x_i \in X$ có N-K+i lựa chọn.
- Các giá trị này đều được chấp thuận mà không cần có thêm bất kỳ điều kiện gì.

```
void Try ( int i ) {
        for (int j =X[i-1]+1; j<=N-K+ i; j++){
                X[i] = j;
                if ( i ==K) Result();
                else Try (i+1);
        }
}
```

BÀI TOÁN 2: DUYỆT CÁC TẬP CON K PHẦN TỬ CỦA 1,2…N

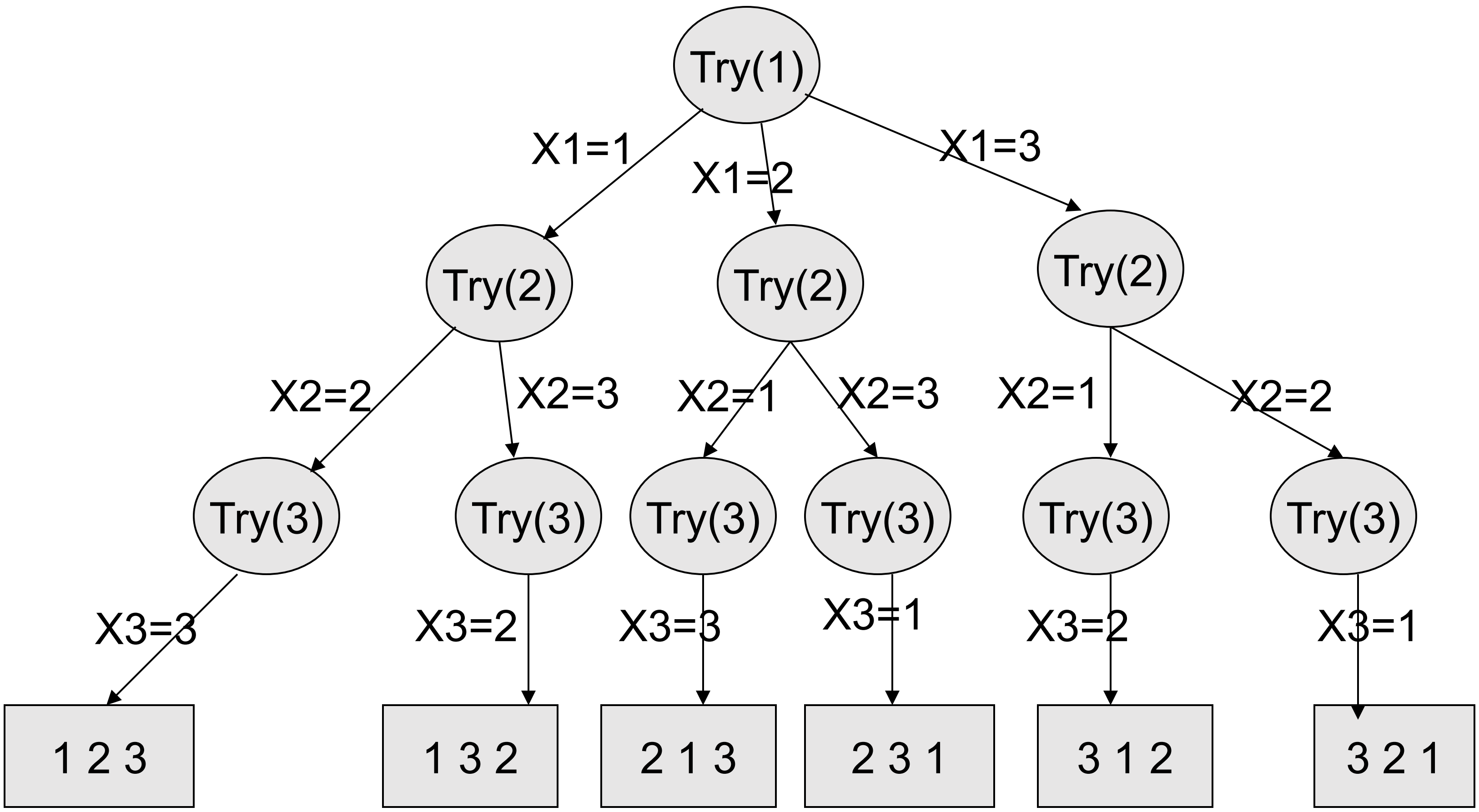


BÀI TOÁN 3: DUYỆT CÁC HOÁN VỊ CỦA 1,2…N

- Mỗi hoán vị $X = (x_1, x_2,..,x_K)$ là bộ có tính đến thứ tự của 1, 2, .., N.
- Mỗi $x_i \in X$ có N lựa chọn. Khi $x_i = j$ được lựa chọn thì giá trị này sẽ không được chấp thuận cho các thành phần còn lại.
- Sử dụng mảng chuaxet[] gồm N phần tử để đánh dấu.

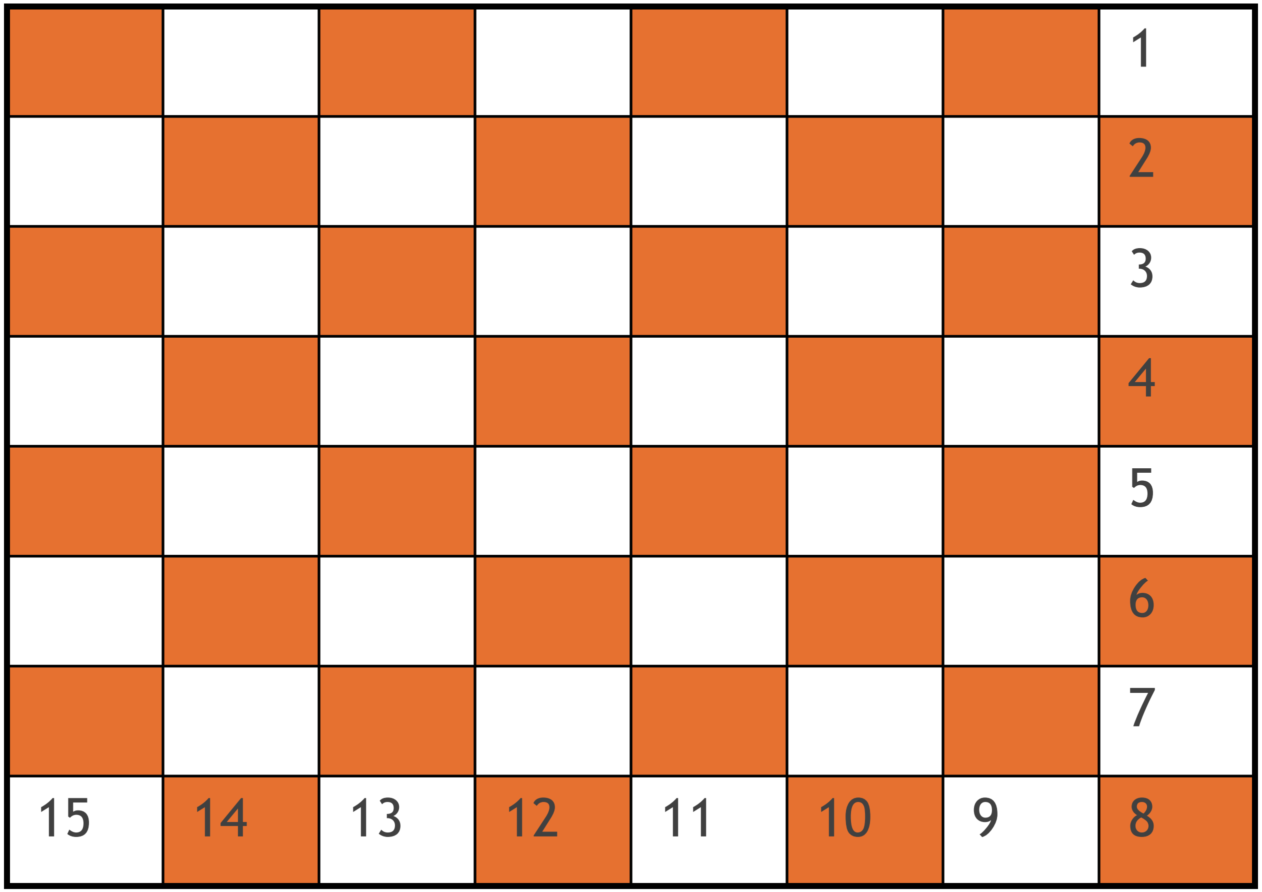
```
void Try ( int i ) {
        for (int j =1; j<=N; j++){
                if (chuaxet[j] ) {
                        X[i] = j;chuaxet[j] = False;
                        if ( i ==N) Result();
                        else Try (i+1);
                        Chuaxet[j] = True;
                }
        }
}
```

BÀI TOÁN 3: DUYỆT CÁC HOÁN VỊ CỦA 1,2…N



BÀI TOÁN 4: BÀI TOÁN XẾP HẬU

Ví dụ với N = 8, ta có bàn cờ 8*8



BÀI TOÁN 4: BÀI TOÁN XẾP HẬU

Bài toán: Trên bàn cờ kích cỡ N×N, hãy đặt N quân hậu mỗi quân trên 1 hàng sao cho tất cả các quân hậu đều không ăn được lẫn nhau.

Phân tích. Gọi X =(x_1, x_2,..,x_n) là một nghiệm của bài toán. Khi đó, x_i = j được hiểu là quân hậu hàng thứ i đặt ở cột j.

Để các quân hậu khác không thể ăn được, quân hậu thứ i cần:

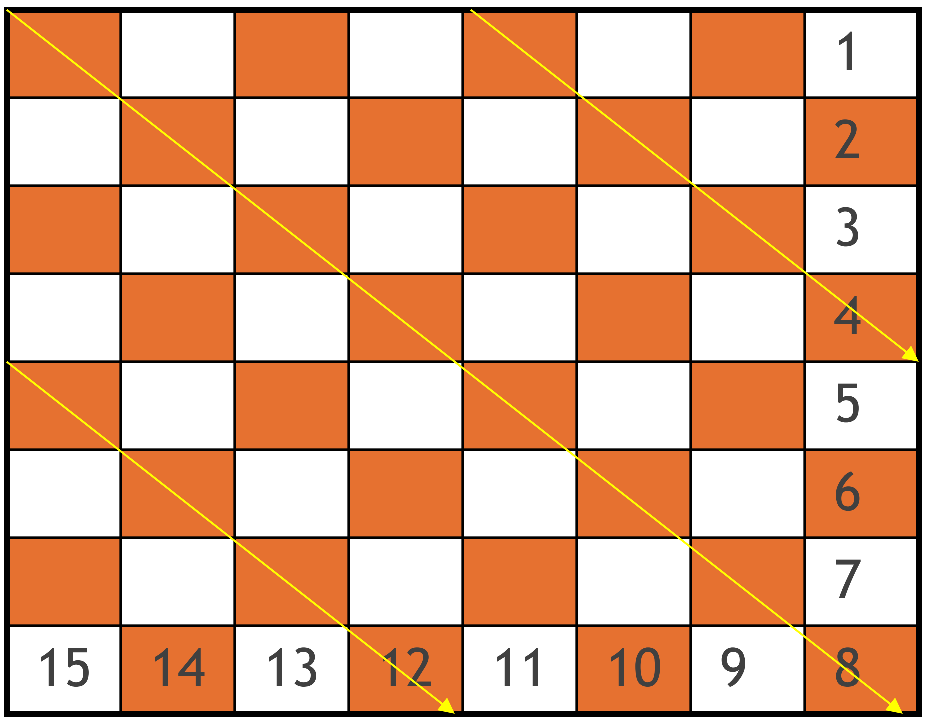
- Không được lấy trùng với bất kỳ cột nào,
- Không được cùng đường chéo xuôi,
- Không được cùng đường chéo ngược.

Ta có n cột A = (a_1,..a_n), có **Xuoi[2*n-1]** đường chéo xuôi, **Nguoc[2*n-1]** đường chéo ngược.

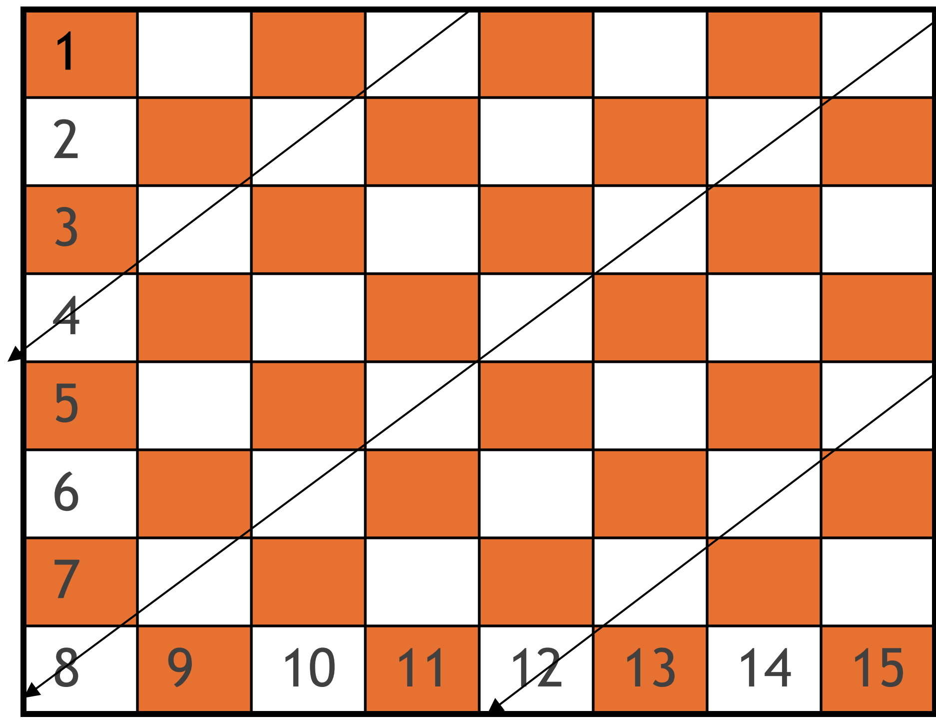
- Nếu x_i =j thì
    - A[j] = True
    - Xuoi[i-j+n] = True
    - Nguoc[i + j -1] = True.

BÀI TOÁN 4: BÀI TOÁN XẾP HẬU

Đường chéo xuôi: Xuoi [i – j + n]



Đường chéo ngược: Nguoc [i + j -1]



BÀI TOÁN 4: BÀI TOÁN XẾP HẬU

Thuật toán:

```
void Try (int    i){
        for(int j=1; j<=n; j++){
                if( A[j] && Xuoi[ i – j + n ] && Nguoc[i + j -1]){
                        X[i] =j;  A[j]=FALSE;
                        Xuoi[ i - j + n]=FALSE;
                        Nguoc[ i + j - 1]=FALSE;
                        if(i==n) Result();
                        else Try(i+1);
                        A[j] = TRUE;
                        Xuoi[ i - j + n] = TRUE;
                        Nguoc[ i + j - 1] = TRUE;

                }
        }
}
```

BÀI TOÁN TỐI ƯU

Phát biểu bài toán

Tìm **min { $f(X) : X \in D$ }.**
hoặc tìm **max { $f(X) : X \in D$ }.**

- *D* là tập hữu hạn các phần tử thỏa mãn tính chất P nào đó.
  $D = \{ X = (x_1, x_2, .., x_n) \in A_1 \times A_2 \times A_n : X \text{ thỏa mãn tính chất } P \}$
- X $\in$D được gọi là một phương án cần duyệt.
- Hàm f(X) được gọi là hàm mục tiêu của bài toán.
- Miền D được gọi là tập phương án của bài toán.
- X $\in$D thỏa mãn tính chất P được gọi là tập các ràng buộc của bài toán.

THUẬT TOÁN NHÁNH CẬN (Branch-And-Bound)

Thuật toán nhánh cận tổng quát

Branch_And_Bound (k) {

    for $a_k \in A_k$ do {

        if (<chấp nhận a_k>){

            $x_k = a_k$;

            if (k==n)

                <Cập nhật kỷ lục>;

            else if ($g(a_1, a_2,..,a_k) \le f^*$)

                Branch_And_Bound (k+1) ;

        }

    }

}

THUẬT TOÁN NHÁNH CẬN (Branch-And-Bound)

Giải bài toán cái túi bằng thuật toán nhánh cận:

Tìm giá trị lớn nhất của hàm mục tiêu f(X) với X∈D. Trong đó, f(X) được xác định như dưới đây:

$$f^* = \max\left\{ f(X) = \sum_{i=1}^{n} c_i x_i : \sum_{i=1}^{n} a_i x_i \le b, x_i \in Z_+, i = 1,2,\ldots,n \right\}$$

$$D = \left\{ X = (x_1, x_2, \ldots, x_n) : \sum_{i=1}^{n} a_i x_i \le b, x_i \in Z_+, i = 1,2,\ldots,n \right\}$$

THUẬT TOÁN NHÁNH CẬN (Branch-And-Bound)

Bước 1. *Sắp xếp các đồ vật thỏa mãn công thức* (2):

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \ldots \geq \frac{c_n}{a_n}$$

Bước 2 (*Lặp*): Lặp trên các bài toán bộ phận cấp *k =1, 2,..,n:*

- *Giá trị sử dụng của k đồ vật trong túi*:

$$\sigma_k = \sum_{i=1}^{k} c_k x_k$$

- *Trọng lượng còn lại của túi*:

$$b_k = b - \sum_{i=1}^{k} a_k x_k$$

- *Cận trên của phương án bộ phận cấp k*:

$$g(x_1, x_2, .., x_k) = \sigma_k - b_k . \frac{c_{k+1}}{a_{k+1}}$$

Bước 3 (*Trả lại kết quả*): Phương án tối ưu và giá trị tối ưu tìm được.

THUẬT TOÁN NHÁNH CẬN (Branch-And-Bound)

Thuật toán Branch_And_Bound (k) {

for j = 1; j<=0; j--){

x[k] = j;

$$\sigma_k = \sigma_k + c_i x_i; \;\; b_k = b_k + a_k x_k;$$

If (k==n) <Ghi nhận kỷ lục>;

else if (δ_k + (c_{k+1}*b_k)/a_{k+1} >FOPT)

Branch_And_Bound(k+1);

$$\sigma_k = \sigma_k - c_k x_k; \quad b_k = b_k - a_k x_k;$$

}

}

THUẬT TOÁN NHÁNH CẬN (Branch-And-Bound)

```
void Back_Track(int i){
    int j, t = ((b-weight)/A[i]);
    for(int j= t; j>=0; j--){
        X[i] = j;  weight = weight+A[i]*X[i];
        cost = cost + C[i]*X[i];
        if (i==n) Update();
        else if ( cost + ( C[i+1]*((b- weight)/A[i+1]))>FOPT)
            Back_Track(i+1);
        weight = weight-A[i]*X[i];
        cost = cost - C[i]*X[i];
    }
}
```