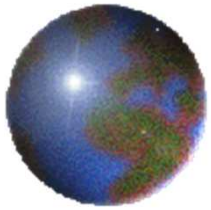
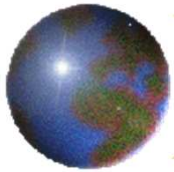


CSH2G3: Design & Analysis of Algorithm

Backtracking strategy

Rimba Whidiana Ciptasari



What is backtracking ?

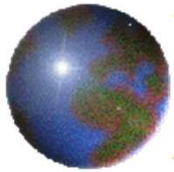
- Difficult combinatorial problem
- Improvement over exhaustive search
- Based on the construction of a *state-space-tree*

Topic assignments

- 0/1 Knapsack problem
- Permutation problem
- Subset-sum problem
- Missionaries-cannibals problem
- Wolf, sheep and cabbage problem

Highlighted points

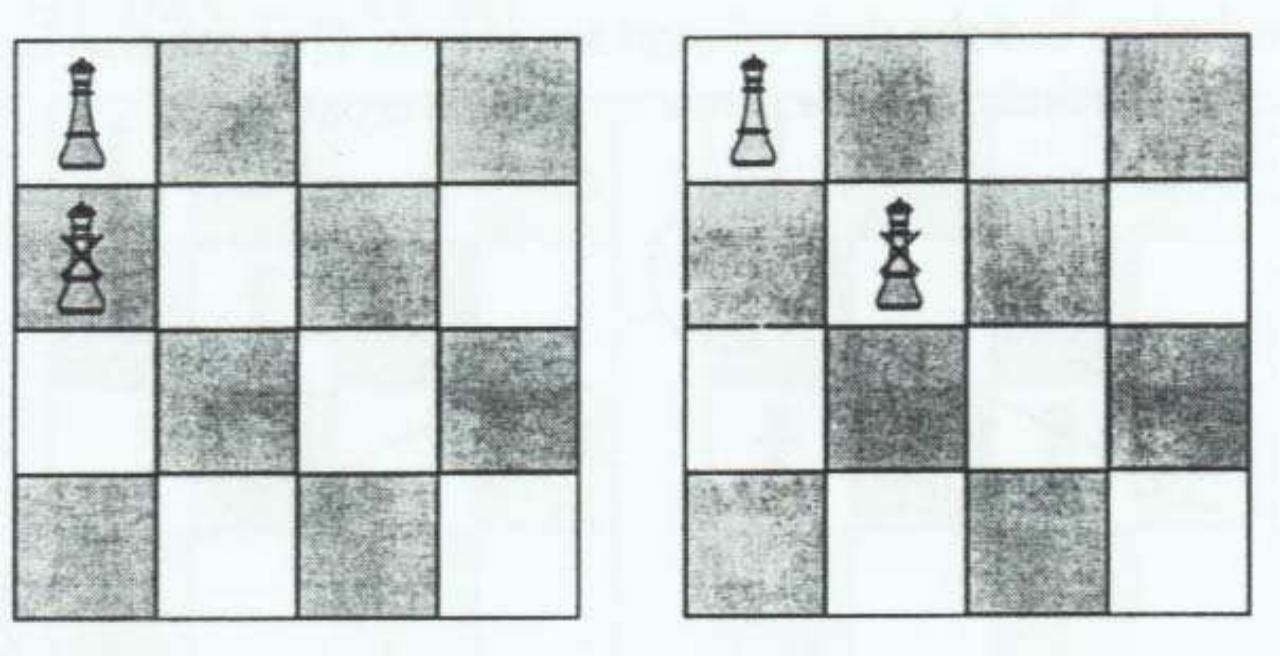
- Problem definition
  - State description
  - Promising condition
- State-space tree
- Time complexity

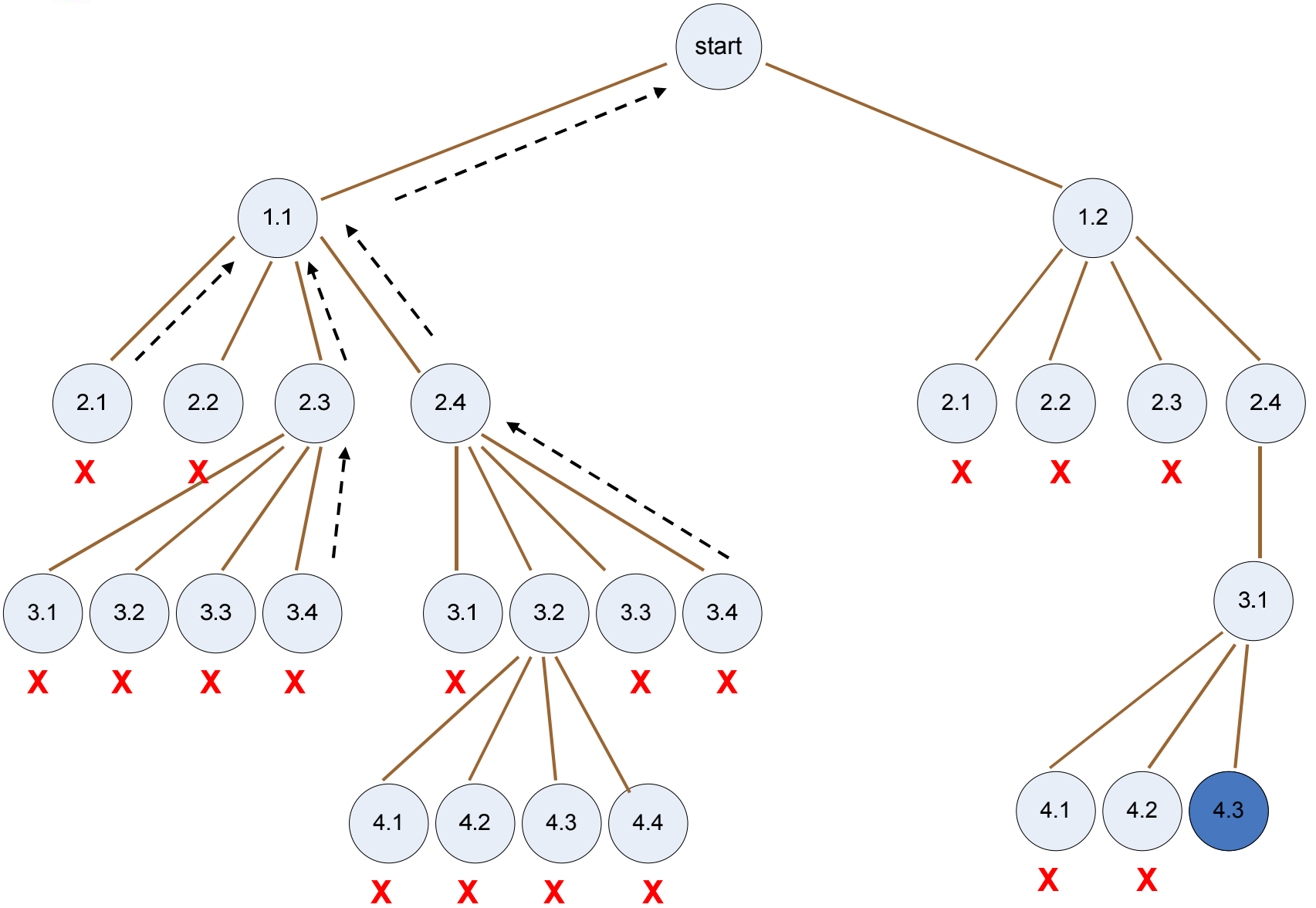


Problem 1: The N-Queens Problem

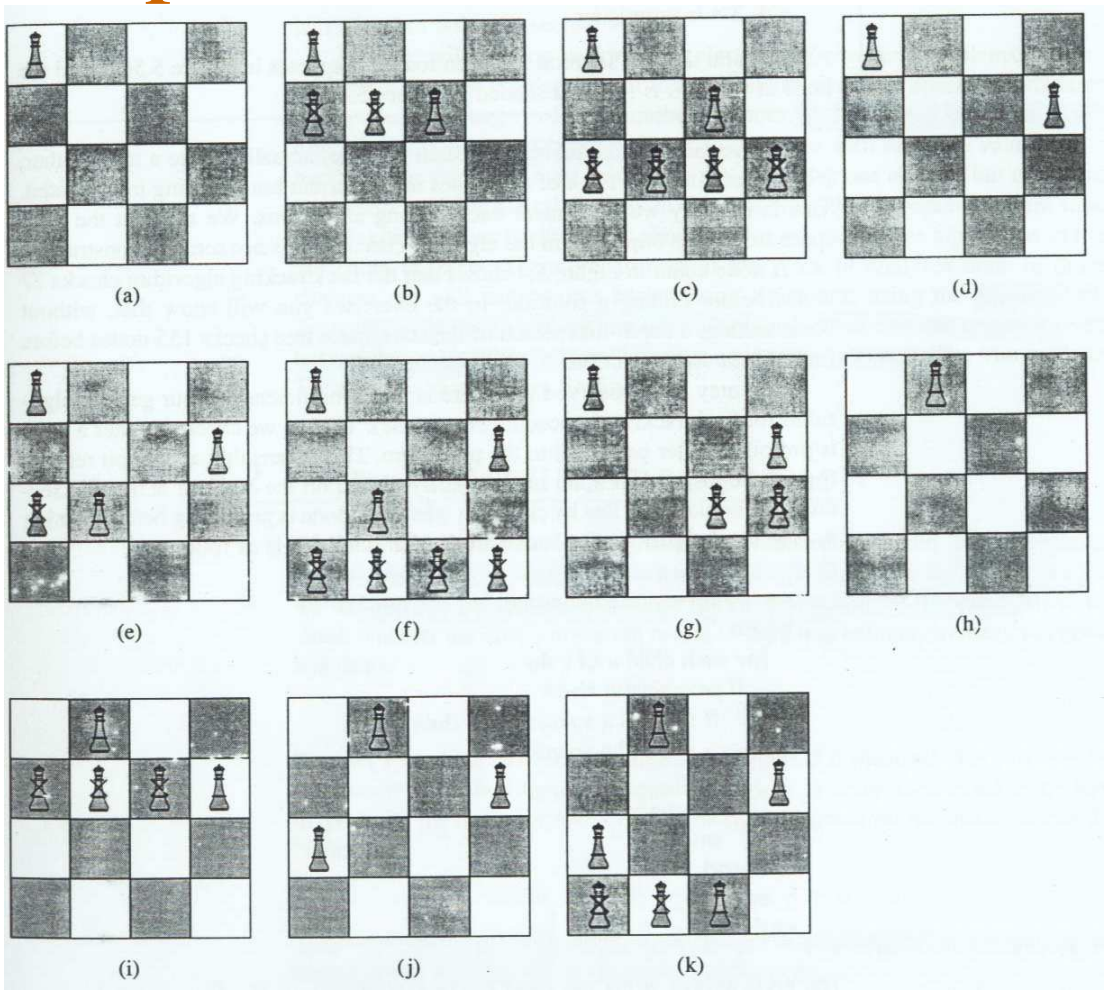
***N-queens problem**: Place N queens on an N x N grid (chess board) so that no more than 1 queen is on any vertical, diagonal, or horizontal line (i.e., no queens can attack each other in chess). If a queen can attack another, it is called a "conflict".*

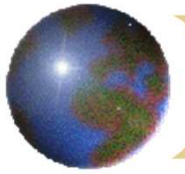
4-queens problem



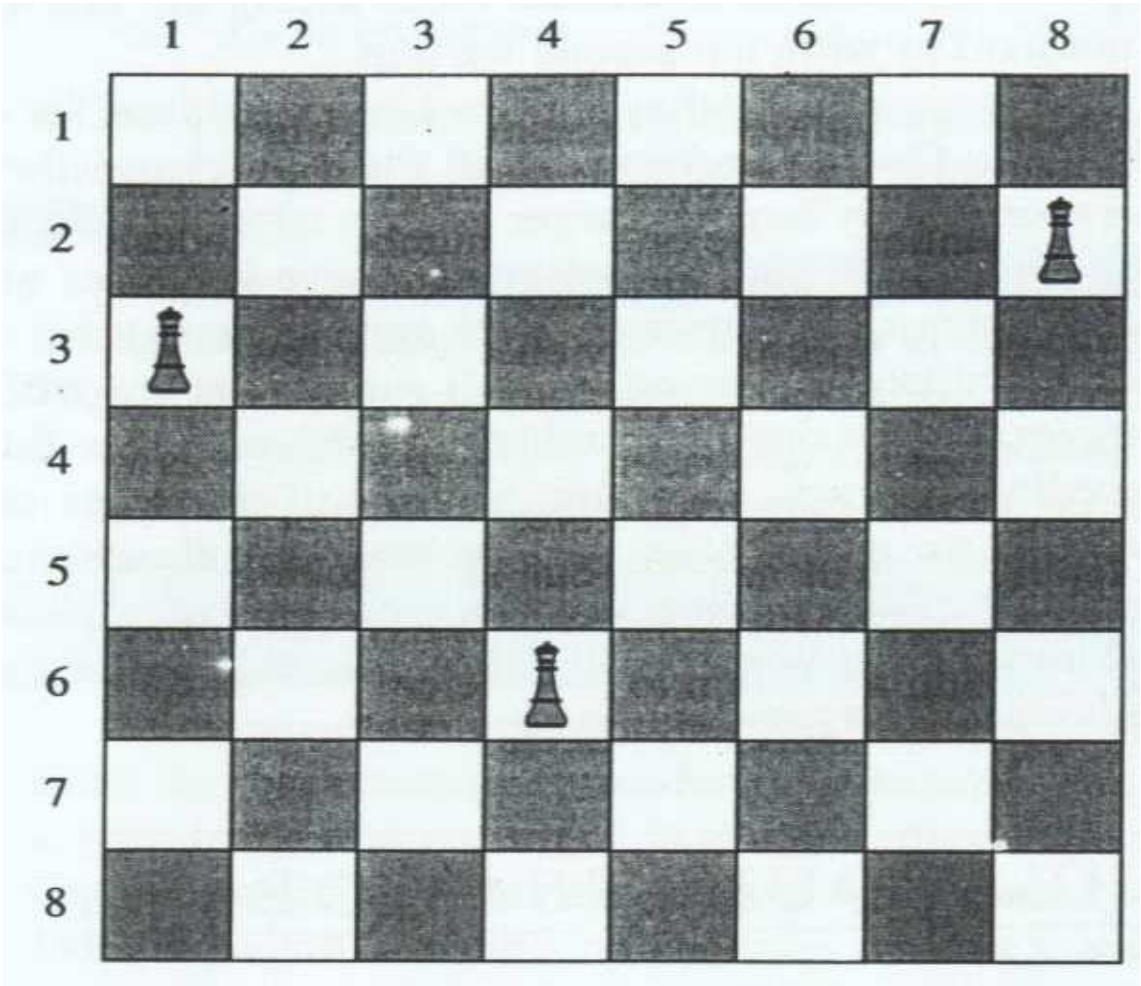

start
1.1
1.2
2.1
2.2
2.3
2.4
2.1
2.2
2.3
2.4
3.1
3.2
3.3
3.4
3.1
3.2
3.3
3.4
3.1
4.1
4.2
4.3
4.4
4.1
4.2
4.3

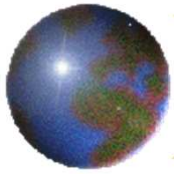
4-queens problem





How to check the diagonal/column ?





How to check the diagonal/column ?

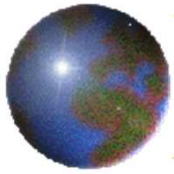
Let *col(i)* be the column where the queen in the *i*th row is located.

- Check column → *col(i) = col(k)*
- Check diagonal → *col(i) – col(k) = |i-k|*

Examples. In the figure, the queen in row 6 is being threatened in its left diagonal by the queen in row 3, and in its right diagonal by the queen in row 2.

col(6) – col(3)= 4 – 1= 3 = 6 – 3

col(6) – col(2)= 4 – 8= -4 =2 – 6



Backtracking algorithm for the n *queens*

Problem: position *n* queen on the chessboard so that there are no two queen in the same row, column, or diagonal

Input: positive integer *n*

Output: all possible ways *n* queens can be placed on a *n* x *n* chessboard. Each output consists of an array *col*[1]..*col*[*n*], where *col*[*i*] is the column where the queen *i* th row is placed.

Backtracking algorithm for the n queens

```
Procedure queens(i:index;n:integer);
Var j:index;
Begin
   if promising(i) then
        if i=n then
              write(col[1] through col[n])
        else
              for j:=1 to n do
                    col[i+1]:=j;
                    queens(i+1,n)
              end
        end
   end
End;
```

Backtracking algorithm for the n queens

```
function promising(i:index):boolean;
Var k:index;
Begin
   k:=1;
   promising:=true;
   while k<i and promising do
        if col[i]=col[k] or abs(col[i]-col[k])=i-k
   then
                promising:=false
        end
        k:=k+1
   end
End;
```

Complexity

- Top level call to *queens* is

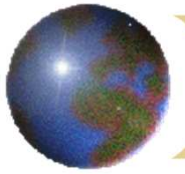
```
queens(0);
```

- Total number of nodes:

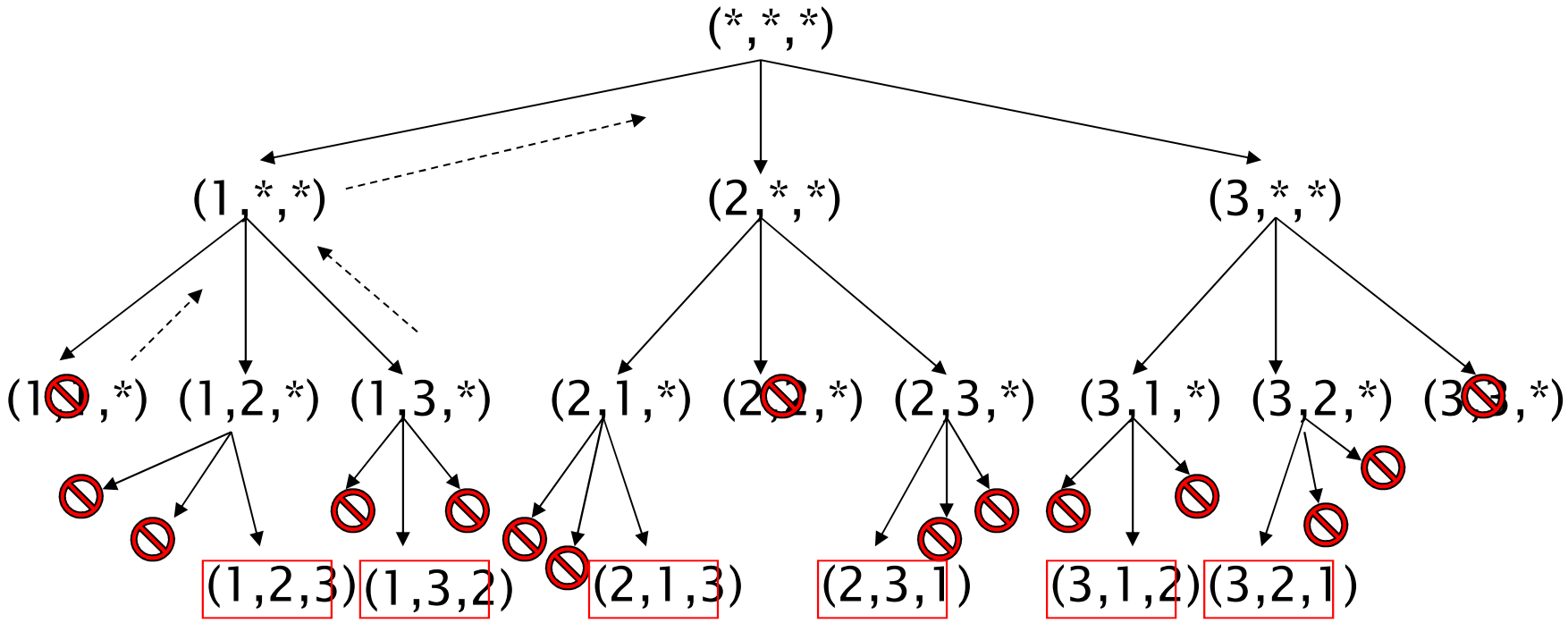
$$1+n+n^2+n^3+\ldots+n^n=\frac{n^{n+1}-1}{n-1}$$

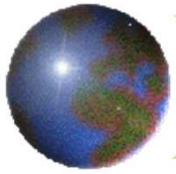
Exercise

1. Generate all permutations of {1,2,3} using backtracking strategy
2. Solve the following instance of the subset sum problem: $S = \{3,5,6,7\}$ & $d = 15$



Solution 1





Problem 2: Subset-sum problem

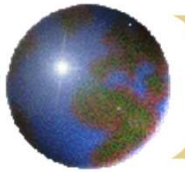
- Description: find a subset of a given set $S=\{s_1,\ldots,s_n\}$ of n positive integers whose sum is equal to a given positive integer d.
- For example, for $S=\{3,5,6,7\}$ and $d=15$
- It's convenient to sort the elements in increasing order:

$$s_1 \le s_2 \le \ldots \le s_n$$

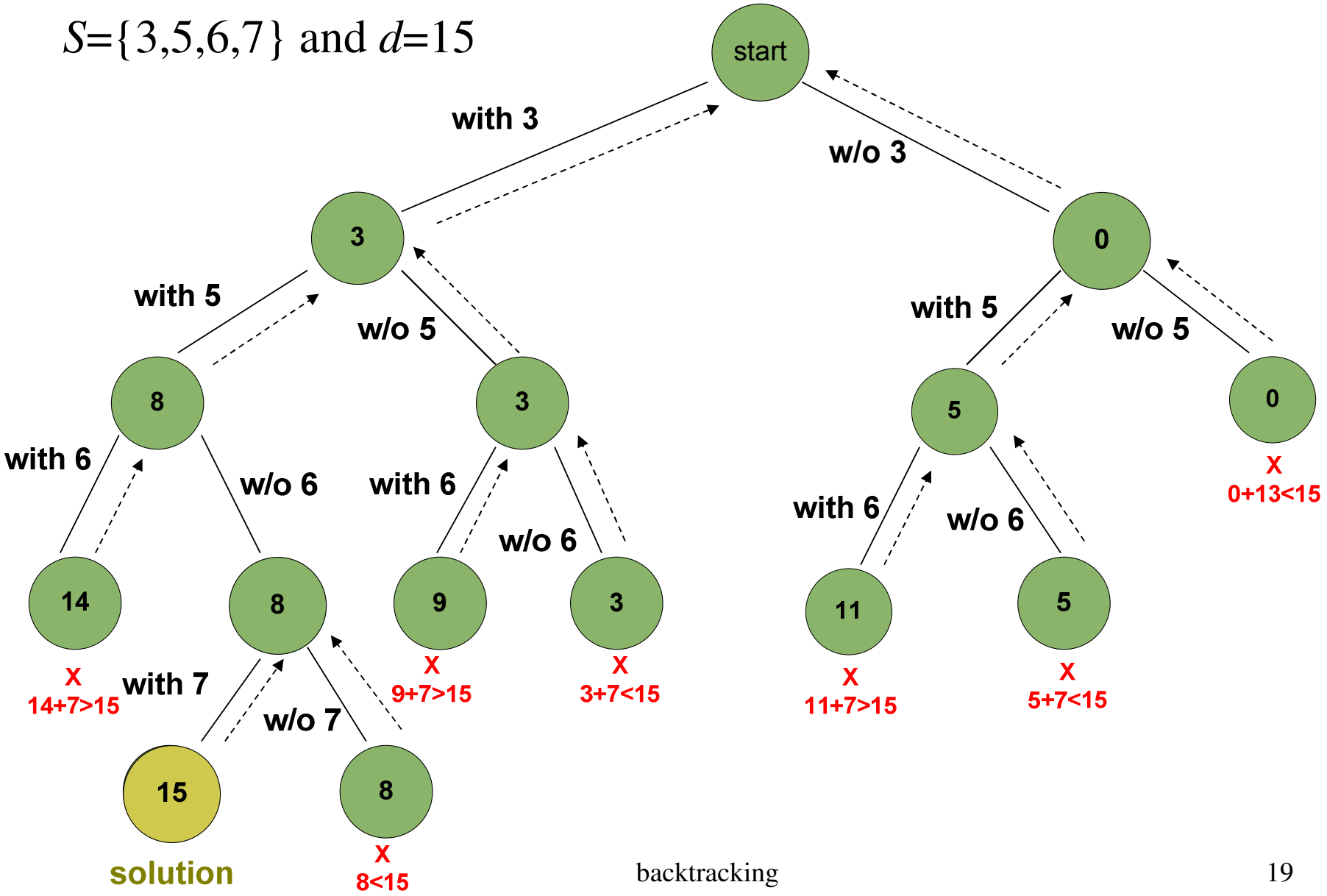
Nonpromising conditions

$$s' + s_{i+1} > d$$

$$s' + \sum_{j=i+1}^{n} s_j < d$$



$S=\{3,5,6,7\}$ and $d=15$



Complexity

$$1 + 2 + 2^2 + 2^3 + \ldots + 2^n = 2^{n+1} - 1$$

Subset-sum algorithm

- **Problem**. Given *n* positive integer and a positive integer *W*, determine all combinations of the integers that sum to *W*
- **Input**. Positive integer *n*, array *w* containing *n* positive integers sorted in nondecreasing order, & positive integer *W*
- **Output**. All combinations of the integers that sum to *W*

Subset-sum algorithm

```
Procedure sum_of_subset(i:index;weight,total:integer);
Begin
   if promising(i) then
        if weight=W then
               write(include[1]through include[i])
        else
               include[i+1]='yes';
               sum_of_subset(i+1,weight+w[i+1],total-w[i+1]);
               include[i+1]='no';
               sum_of_subset(i+1,weight,total-w[i+1]);
        end
   end
End;
```

Subset-sum algorithm

```
function promising(i:index):boolean;
Begin
   promising = (weight+total≥W)and(weight=W or
   weight+w[i+1]≤W)
End;
```

Top level call:

```
sum_of_subset(0,0,total);
```

Assignment

- Solve the following game by backtracking:
  - The knight's tour
  - Missionaries and Cannibals
  - Hi-Q game