Brute Force

- **Brute force** is a straightforward approach to solve a problem based on the problem's statement and definitions of the concepts involved.
- It is considered as one of the easiest approach to apply and is useful for solving small–size instances of a problem

- Example 1: Computing a^n ($a > 0$, n a nonnegative integer) based on the definition of exponentiation

  $a^n = a * a * a * .... * a$
- The brute force algorithm requires n-1 multiplications.
  The recursive algorithm for the same problem, based on the observation that $a^n = a^{n/2} * a^{n/2}$ requires Θ(log(n)) operations.
- **Example 2:** Computing n! based on the definition n! = 1*2*3*…*n The algorithm requires Θ (n) operations

Brute-Force Search and Sort

- Sequential search in an unordered array and simple sorts – selection sort, bubble sort are brute force algorithms.
- Sequential search: the algorithm simply compares successive elements of a given list with a given search key until either a match is found or the list is exhausted without finding a match.
- The complexity of a sequential search algorithm is $\Theta(n)$ in the worst possible case and $\Theta(1)$ in the best possible case, depending on where the desired element is situated.

Brute-Force -Sort

Selection sort:

- the entire given list of *n* elements is scanned to find its smallest element and exchange it with the first element.
- Thus, the smallest element is moved to its final position in the sorted list.
- Then, the list is scanned again, starting with the second element in order to find the smallest element among the *n – 1* and exchange it with the second element.
- The second smallest element is put in its final position in the sorted list.
- After *n-1* passes, the list is sorted.

Brute-Force – Selection Sort

Algorithm SelectionSort (A[0..n-1])

for i ← 0 to n-2 do

    min ← i

    for j ← i + 1 to n-1 do

        if A[j] < A[min]

            min ← j

    swap A[i] and A[min]

- The basic operation of the selection sort is the comparison -> A[j] < A[min]. The complexity of the algorithm is $\Theta(n^2)$ and the number of key swaps is $\Theta(n)$.

Brute-Force – Bubble Sort

- Bubble sort is another application of a brute force.
- In the algorithm, adjacent elements of the list are compared and are exchanged if they are out of order.
- Algorithm BubbleSort (A[0..n-1])

```
for i ← 0 to  n-2 do
        for j ← 0 to n – 2 – i do
                if A[j+1] < A[j]
                   swap A[j] and A[j+1]
```

Brute-Force -Sort

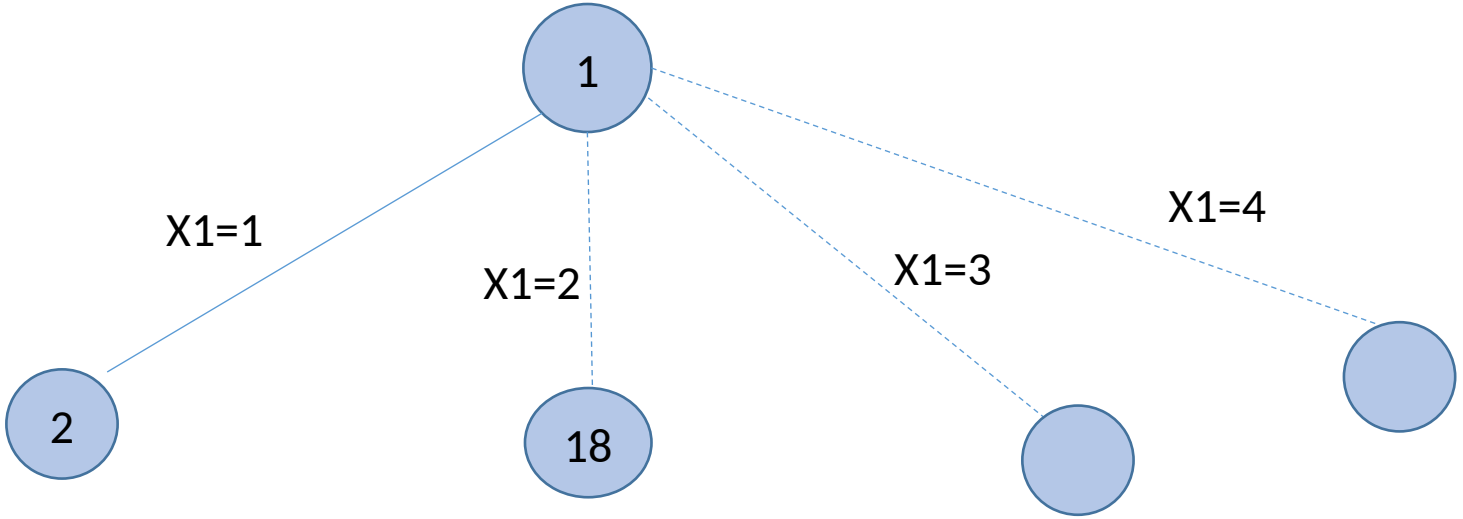
- The basic operation of the bubble sort is comparison - A[j+1] < A[j] and swapping - swap A[j] and A[j+1].
- The number of key comparisons is the same for all arrays of size n and it is $\Theta(n^2)$.
- However, the number of key swaps depends on the input and in the worst case is $\Theta(n^2)$.
- The above implementation of Bubble sort can be slightly improved if we stop the execution of the algorithm when a pass through the list makes no exchanges (i.e. indicating that the list has been sorted).
- Thus, in the best case the complexity will be $\Theta(n)$ and the worst case $\Theta(n^2)$.

Exhaustive Search

- Exhaustive search is used in problems where the solution is an object with specific properties in a set of candidate solutions.
- We have to examine all candidate solutions to find the solution if it exists.
- There are two types of problems that involve exhaustive search –
  - state-space search problems and
  - combinatorial problems.

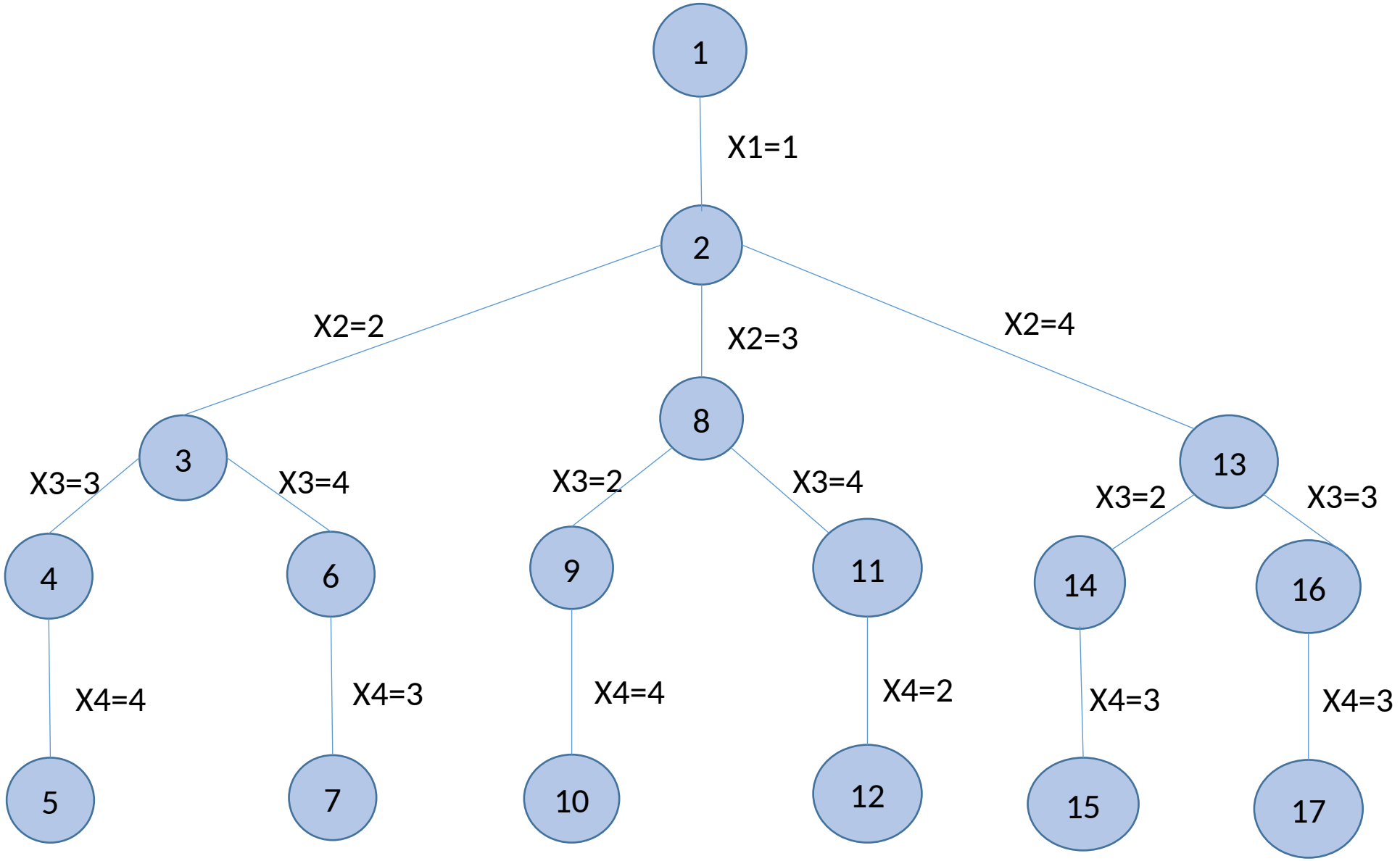
Exhaustive Search: state-space search

- In state-space search the problem is formulated as follows: Given an initial state, a goal state, and a set of operations, find a sequence of operations that transforms the initial state to the goal state. The solution process can be represented as a tree with the initial state being its root, and the goal state being a leaf in the tree, and each edge corresponds to an operation.
- At each node we attempt to apply all available operations. When an operation is applied, a new node is generated. If no operation can be applied, the node is called “dead end”. The process terminates when a node is generated that satisfies the requirements of the goal state or when no more nodes can be generated.
- The search can be performed in a breadth-first manner or in a depth-first manner. The problem-solving method is called “generate-and-test” approach because we generate a node and test to see if it is the solution.
- Exhaustive search is not feasible if the search tree grows exponentially. If this is the case, the generated nodes have to be evaluated (if possible) and so that at each step the best node is expanded. Many Artificial Intelligence problems fall into this category, e.g. the 8-puzzle problem.

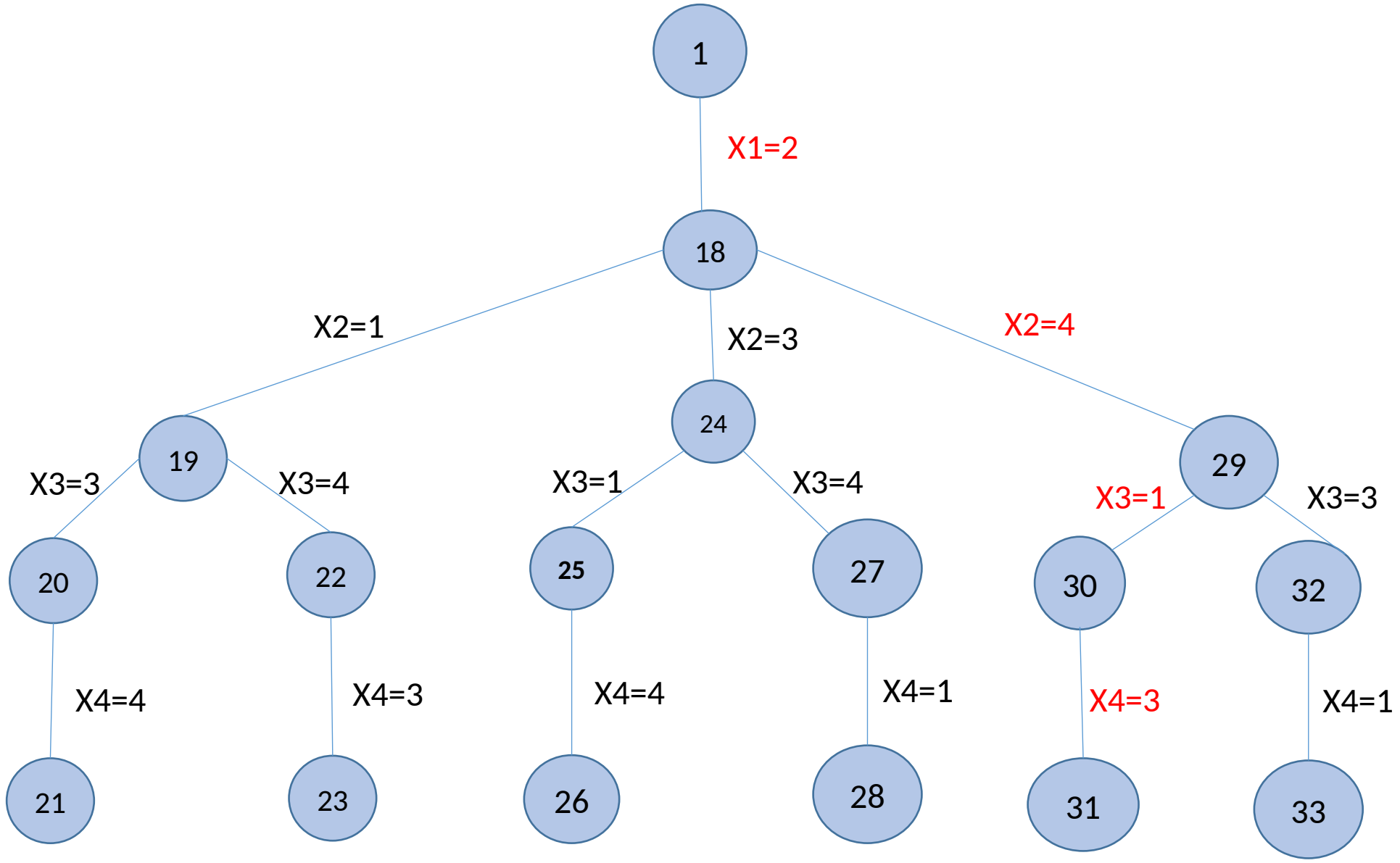


To be continued in next slide

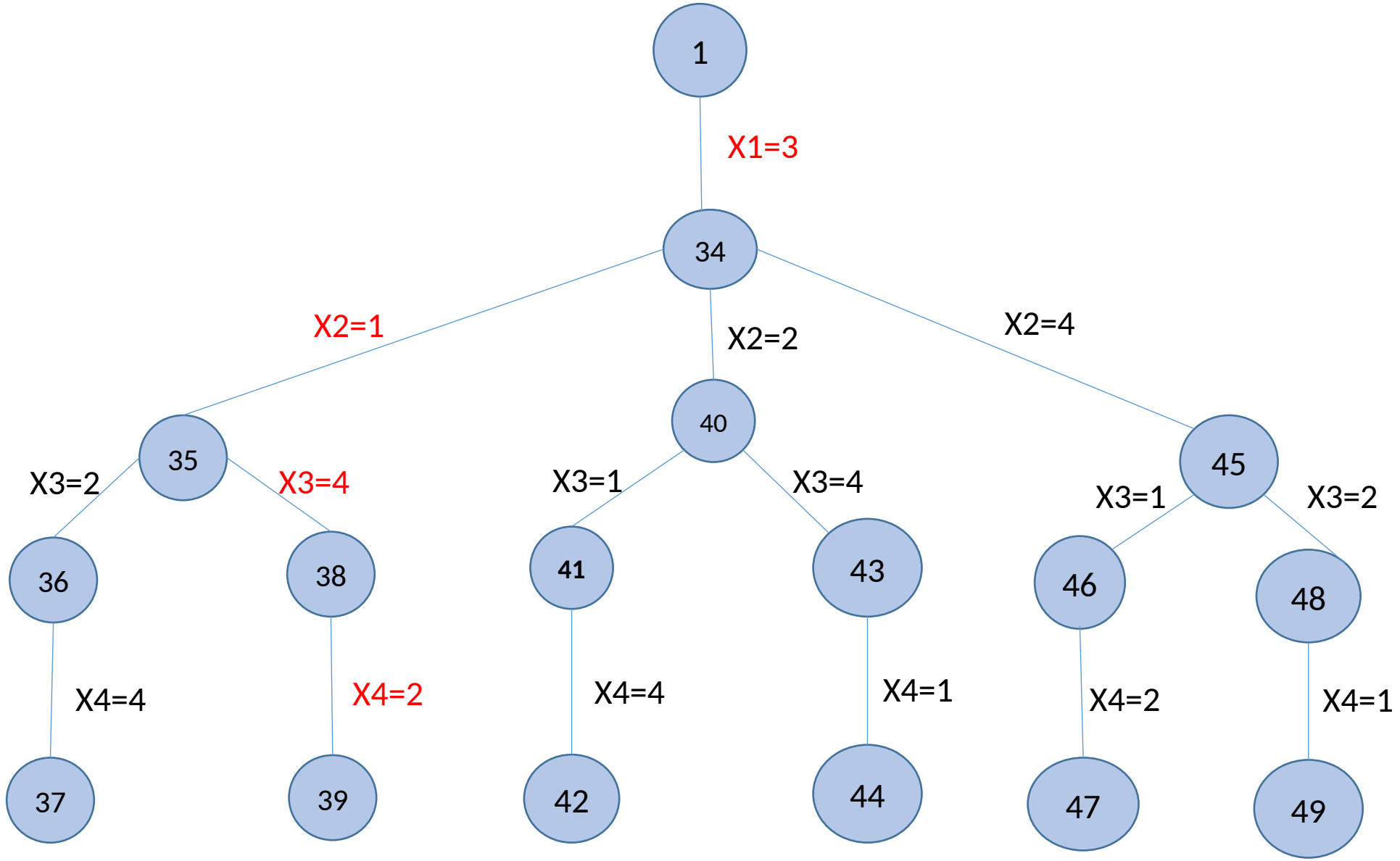
Tree organization of 4 queens solution space in DFS



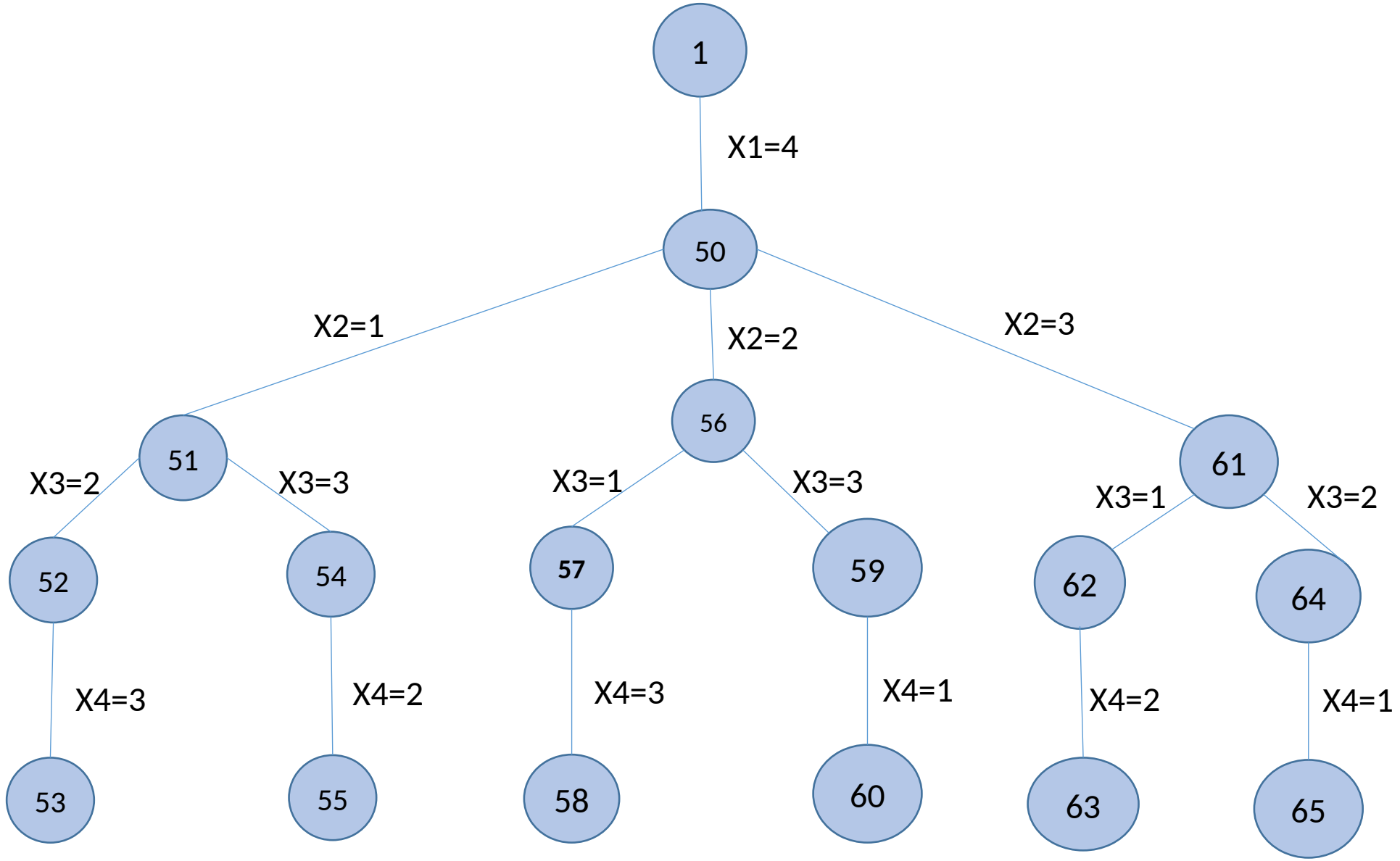
Tree organization of 4 queens solution space in DFS



Tree organization of 4 queens solution space in DFS



Tree organization of 4 queens solution space in DFS



Tree organization of 4 queens solution space in DFS

Exhaustive Search :Combinatorial problems

- In combinatorial problems the solution (if is exists) is an element of a set of combinatorial objects – permutations, combinations, or subsets. The brute-force approach consists in generating the combinatorial objects and testing each object to see if it satisfies some specified constraints. Problems like the traveling salesman, knapsack, and bin-packing can be solved by using an exhaustive search.
- The **traveling salesman problem** tries to find the shortest tour through a given set of n cities that visits each city exactly once before returning to the city whether it started. Since each tour is a permutation of the n cities, and since we can fix the start city, the time efficiency of this algorithm is $\Theta((n-1)!)$.
- The **knapsack problem**, for given **n** items with their benefits and volume, tries to find the most valuable subset (i.e. the one with the greatest benefit) without exceeding the capacity of the knapsack. The solution is one of all subsets of the set of objects and therefore the time efficiency of this algorithm is $\Theta(2^n)$.

Exhaustive Search :Combinatorial problems

- n the assignment problem we have n persons that have to be assigned n jobs, one person per job. The problem is represented as a cost matrix, where c(i,j) is the cost of assigning job j to person i. The solution is a vector (a1, a2, ..an) such that person i is assigned job ai and the sum ∑ c(i,ai) is minimum.
- The solution is a permutation of the n jobs and therefore the time efficiency is Θ(n!).
- In combinatorial problems, The time efficiency is determined by the domain size. Once we have a candidate for a solution we can examine it in polynomial time. Thus, if we have all possible tours, we can find the shortest one in linear time. However, it take Θ((n-1)!) to generate all permutations.
- Exhaustive search algorithms run in a realistic amount of time only on very small instances of the problem at hand. In many cases, there are much better alternative algorithms. For example the 8-puzzle problem is solved using the heuristic A* algorithm. However, in some cases such as the traveling salesman problem, exhaustive search is the only known algorithm to obtain exact solution. Approximate algorithms are used to solve realistic instances of such problems.

The strengths of using a brute force approach

- It has wide applicability and is known for its simplicity.
- It yields reasonable algorithms for some important problems such as searching, string matching, and matrix multiplication.
- It yields standard algorithms for simple computational tasks such as sum and product of n numbers, and finding maximum or minimum in a list.

The weaknesses of the brute force approach

- It rarely yields efficient algorithms.
- Some brute force algorithms are unacceptably slow.
- It is neither as constructive nor creative as some other design techniques.