Analysis of different type of Algorithms

o Divide and Conquer Algorithm

o Greedy algorithm

o Dynamic Programming algorithm

o Brute force algorithm

o Backtracking algorithms

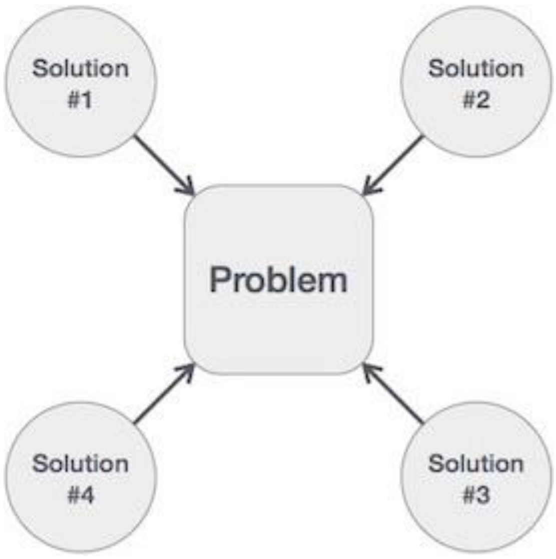
o Branch-and-bound algorithms

o Stochastic algorithms

Different Types of Algorithms in Data Structure.

What is Algorithm?

An Algorithm is a sequence of steps that describe how a problem can be solved. Every computer program that ends with a result is basically based on an Algorithm. These can also be used to solve mathematical problems and on many matters of day-to-day life. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.



The Algorithm are different Categories which are described as below:

1. **Search −** Algorithm to search an item in a data structure.
2. **Sort −** Algorithm to sort items in a certain order.
3. **Insert −** Algorithm to insert item in a data structure.

4. **Update −** Algorithm to update an existing item in a data structure.
5. **Delete −** Algorithm to delete an existing item from a data structure.

These are the categories by which every problem become easy and can easily solved.

How to Write an Algorithm?

here are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.

As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm.

We write algorithms in a step-by-step manner, but it is not always the case. Algorithm writing is a process and is executed after the problem domain is well-defined. That is, we should know the problem domain, for which we are designing a solution.

Example

Let's try to learn algorithm-writing by using an example.

Problem − Design an algorithm to add two numbers and display the result.

```
Step 1 − START
Step 2 − declare three integers a, b & c
Step 3 − define values of a & b
Step 4 − add values of a & b
Step 5 − store output of step 4 to c
Step 6 − print c
Step 7 − STOP
```

Algorithms tell the programmers how to code the program. Alternatively, the algorithm can be written as

```
Step 1 − START ADD
Step 2 − get values of a & b
Step 3 − c ← a + b
Step 4 − display c
Step 5 − STOP
```

In design and analysis of algorithms, usually the second method is used to describe an algorithm. It makes it easy for the analyst to analyze the algorithm ignoring all unwanted definitions. He can observe what operations are being used and how the process is flowing.

Advantages and Disadvantages of Algorithms.

Advantages of Algorithm

1. It is a step wise representation of a solution for a give problem
2. Algorithm use a definite procedure
3. It does not depend upon any programming language which make algorithm easily understandable without any programming knowledge.
4. Every step in algorithm has logical reason so it can be easily debugged.
5. Algorithm divide the problem into smaller problems due to which every problem can easily solved.

Disadvantages of Algorithm

1. Algorithm is Time Consuming
2. Every problem cannot be converted into algorithm which makes it difficult
3. Difficult to show looping and branching in Algorithm

Types of Algorithm

Based on how they function, we can divide Algorithms into multiple types. Let's take a look at some of the important ones.

1. Recursive Algorithm

This is one of the most interesting Algorithms as it calls itself with a smaller value as inputs which it gets after solving for the current inputs. In more simpler words, It's an Algorithm that calls itself repeatedly until the problem is solved.

Example:

To find factorial using recursion, here is the pseudo code:

```
 Fact(x)
     If x is 0      /*0 is the base value and x is 0 is base case*/
         return 1
     return (x*Fact(x-1))  /* breaks the problem into small problems*/
```

2. Divide and Conquer Algorithm

This is another effective way of solving many problems. In Divide and Conquer algorithms, divide the algorithm into two parts, the first parts divide the problem into smaller subproblems of the same type. Then on the second part, these smaller problems are solved and then added together (combined) to produce the final solution of the problem.

Example:

Merge sort , Quick sort

3. Dynamic Programming Algorithm

These algorithms work by remembering the results of the past run and using them to find new results. In other words, dynamic programming algorithm solves complex problems by breaking it into multiple simple subproblems and then it solves each of them once and then stores them for future use.

In this type of algorithm we use sequence of decisions. Example- Floyed warshall algorithm

Example:

Fibonacci sequence, here is the pseudo code :

```
Fib(n)
    if n=0
            return 0
        else
            prev_Fib=0,curr_Fib=1
        repeat n-1 times  /*if n=0 it will skip*/
            next_Fib=prev_Fib+curr_Fib
            prev_Fib=curr_Fib
            curr_Fib=new_Fib
        return curr_Fib
```

4. Greedy Algorithm

These algorithms are used for solving optimization problems. In this algorithm, we find a locally optimum solution (without any regard for any consequence in future) and hope to find the optimal solution at the global level.

The method does not guarantee that we will be able to find an optimal solution.

The algorithm has 5 components:

- The first one is a candidate set from which we try to find a solution.
- A selection function which helps choose the best possible candidate.
- A feasibility function which helps in deciding if the candidate can be used to find a solution.
- An objective function which assigns value to a possible solution or to a partial solution
- Solution function that tells when we have found a solution to the problem.

Huffman Coding and Dijkstra’s algorithm are two prime examples where Greedy algorithm is used.

In Huffman coding, The algorithm goes through a message and depending on the frequency of the characters in that message, for each character, it assigns a variable length encoding. To do Huffman coding, we first need to build a Huffman tree from the input characters and then traverse through the tree to assign codes to the characters.

Example:
Dijkstra's algorithm, prims algorithm, Kruskal's algorithm

5. Brute Force Algorithm

This is one of the simplest algorithms in the concept. A brute force algorithm blindly iterates all possible solutions to search one or more than one solution that may solve a function. Think of brute force as using all possible combinations of numbers to open a safe.

Example:
1. Exact string
2. Matching algorithm

6. Backtracking Algorithm

Backtracking is a technique to find a solution to a problem in an incremental approach. It solves problems recursively and tries to get to a solution to a problem by solving one piece of the problem at a time. If one of the solutions fail, we remove it and backtrack to find another solution.

In other words, a backtracking algorithm solves a subproblem and if it fails to solve the problem, it undoes the last step and starts again to find the solution to the problem.

N Queens problem is one good example to see Backtracking algorithm in action. The N Queen Problem states that there are N pieces of Queens in a chess board and we have to arrange them so that no queen can attack any other queen in the board once organized.

Example:
Queens Problem

Branch and bound

What is Branch and bound?

Branch and bound is one of the techniques used for problem solving. It is similar to the backtracking since it also uses the state space tree. It is used for solving the optimization problems and minimization problems. If we have given a maximization problem then we can convert it using the Branch and bound technique by simply converting the problem into a maximization problem.

Let's understand through an example.

Jobs = {j1, j2, j3, j4}

P = {10, 5, 8, 3}

d = {1, 2, 1, 2}

The above are jobs, problems and problems given. We can write the solutions in two ways which are given below:

Suppose we want to perform the jobs j1 and j2 then the solution can be represented in two ways:

The first way of representing the solutions is the subsets of jobs.

S1 = {j1, j4}

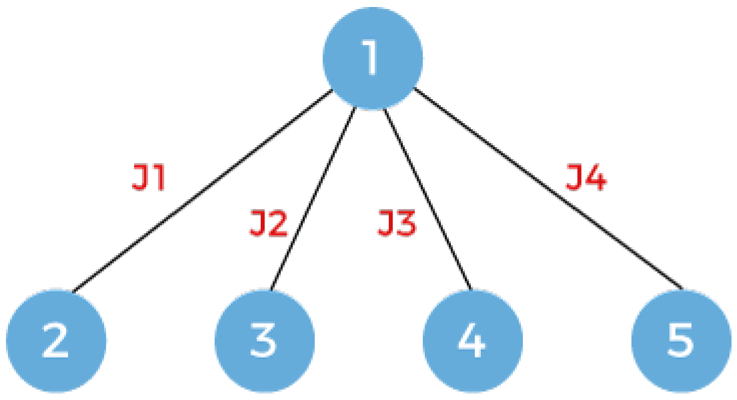
The second way of representing the solution is that first job is done, second and third jobs are not done, and fourth job is done.

S2 = {1, 0, 0, 1}

The solution s1 is the variable-size solution while the solution s2 is the fixed-size solution.

First, we will see the subset method where we will see the variable size.

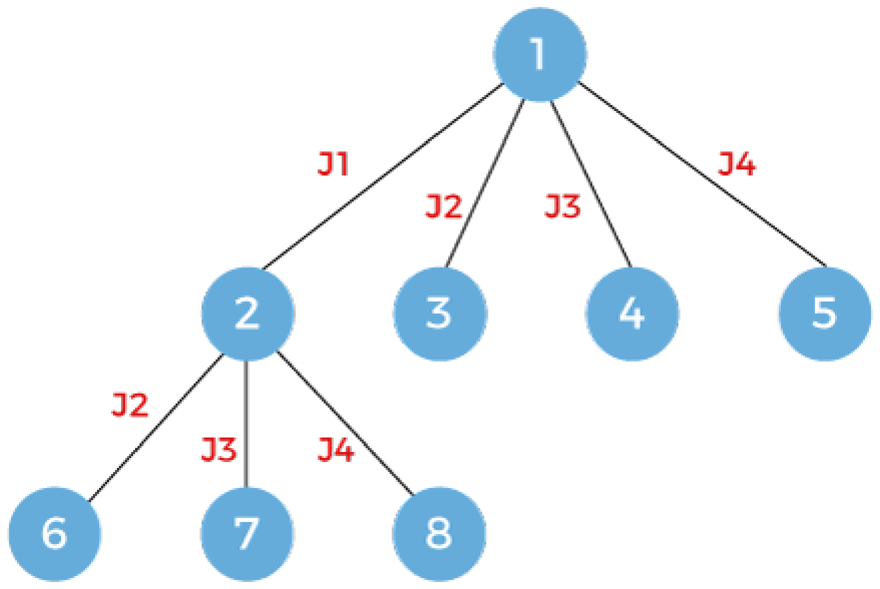
First method:



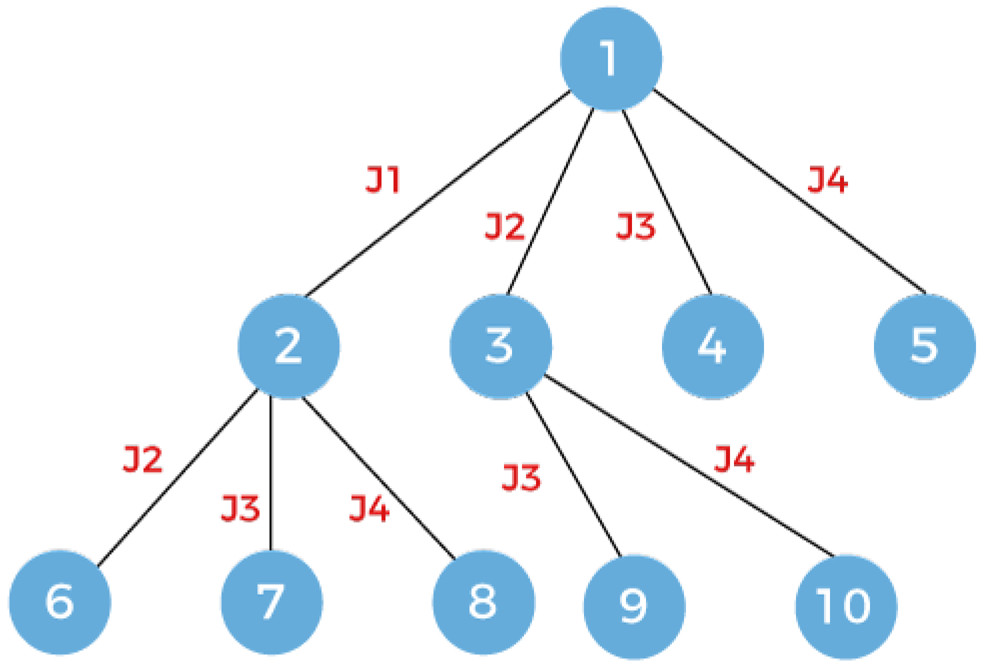
In this case, we first consider the first job, then second job, then third job and finally we consider the last job.

As we can observe in the above figure that the breadth first search is performed but not the depth first search. Here we move breadth wise for exploring the solutions. In backtracking, we go depth-wise whereas in branch and bound, we go breadth wise.

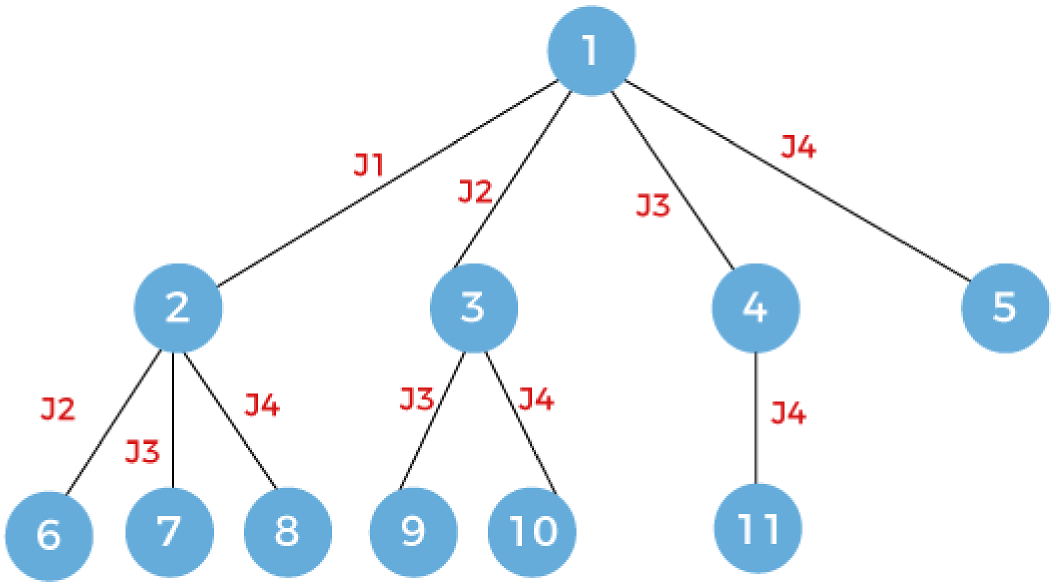
Now one level is completed. Once I take first job, then we can consider either j2, j3 or j4. If we follow the route then it says that we are doing jobs j1 and j4 so we will not consider jobs j2 and j3.



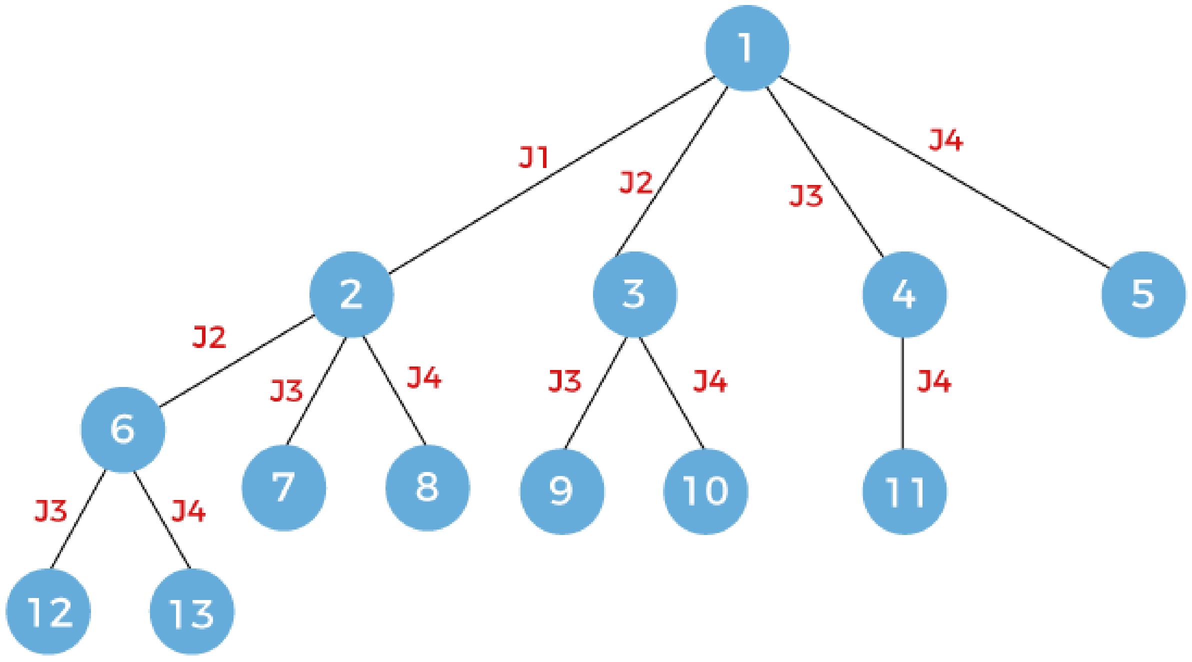
Now we will consider the node 3. In this case, we are doing job j2 so we can consider either job j3 or j4. Here, we have discarded the job j1.



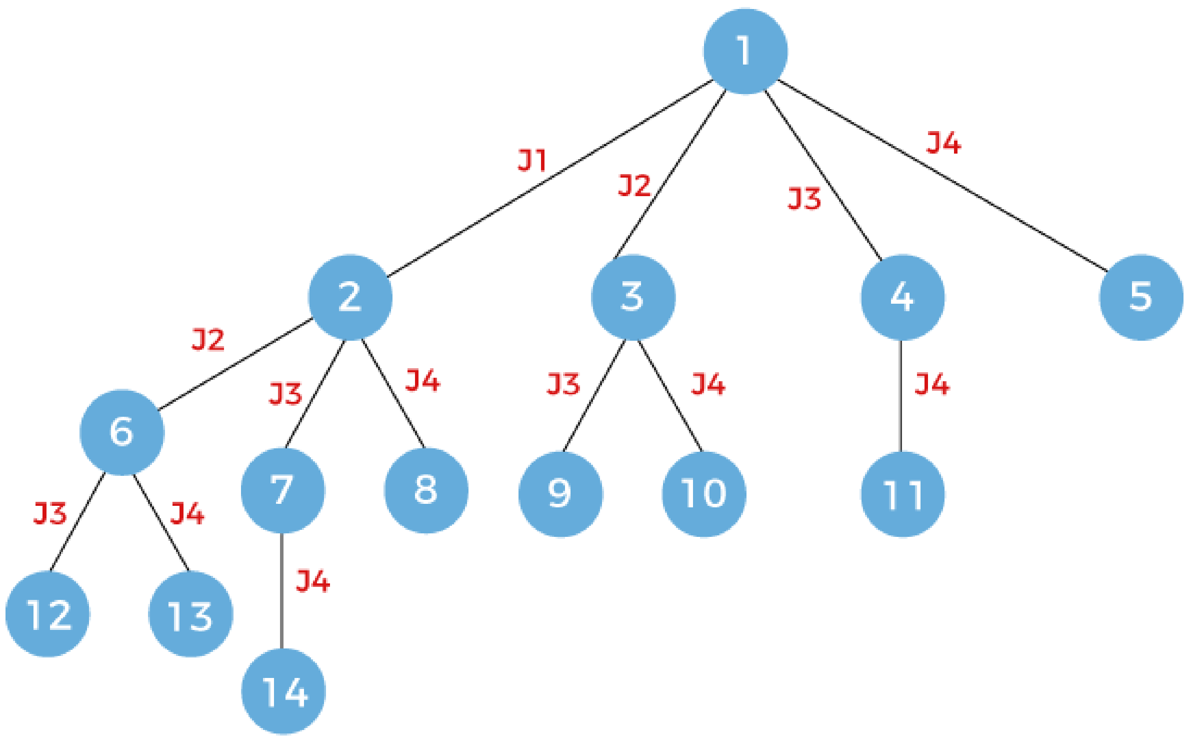
Now we will expand the node 4. Since here we are doing job j3 so we will consider only job j4.



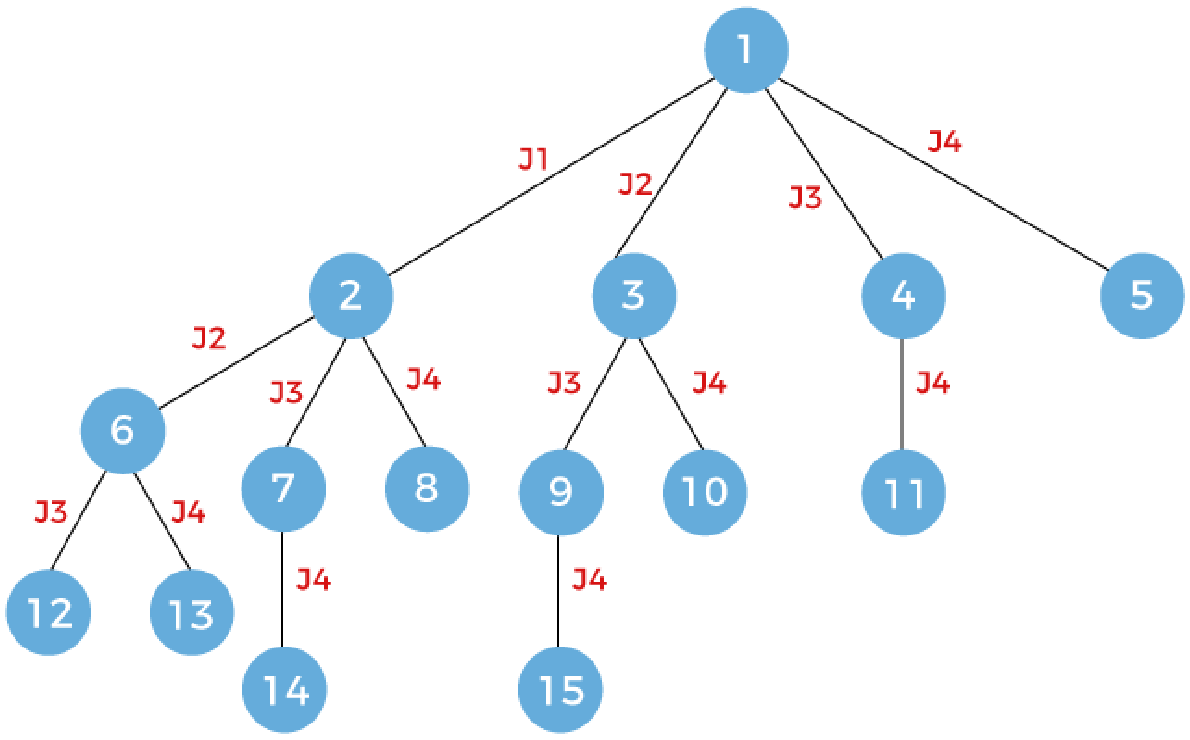
Now we will expand node 6, and here we will consider the jobs j3 and j4.



Now we will expand node 7 and here we will consider job j4.



Now we will expand node 9, and here we will consider job j4.



The last node, i.e., node 12 which is left to be expanded. Here, we consider job j4.

The above is the state space tree for the solution s1 = {j1, j4}

Stochastic Linear Optimization

INTRODUCTION

The fundamental idea behind stochastic linear programming is the concept of *recourse.* Recourse is the ability to take corrective action after a random event has taken place. A simple example of *two-stage recourse* is the following:

- Choose some variables, ***x,*** to control what happens today.
- Overnight, a random event happens.
- Tomorrow, take some recourse action, ***y,*** to correct what may have gotten messed up by the random event.

EXAMPLE

You are in charge of a local gas company. When you buy gas, you typically deliver some to your customers right away and put the rest in storage. When you sell gas, you take it either from storage or from newly-arrived supplies. Hence, your decision variables are 1) how much gas to purchase and deliver, 2) how much gas to purchase and store, and 3) how much gas to take from storage and deliver to customers. Your decision will depend on the price of gas both now and in future time periods, the storage cost, the size of your storage facility, and the demand in each period. You will decide these variables for each time period considered in the problem. This problem can be modeled as a simple linear program with the objective to minimize overall cost. The solution is valid if the problem data are known with certainty, that is, if the future events unfold as planned.

More than likely, the future will not be precisely as you have planned; you don’t know for sure what the price or demand will be in future periods though you can make good guesses. For example, if you deliver gas to your customers for heating purposes, the demand for gas and its purchase price will be strongly dependent on the weather . Predicting the weather is rarely an exact science; therefore, not taking this uncertainty into account may invalidate the results from your model. Your “optimal” decision for one set of data may not be optimal for the actual situation.

SCENARIOS

Suppose in our example that we are experiencing a normal winter and that the next winter can be one of three scenarios: normal, cold, or very cold. To formulate this problem as a stochastic linear program, we must first characterize the uncertainty in the model. The most common method is to formulate scenarios and assign a probability to each scenario. Each of these scenarios has different data as shown in the following table:

Scenario	**Probability**	**Gas Cost ($)**	**Demand (units)**
Normal	1/3	5.0	100
Cold	1/3	6.0	150

Very Cold	1/3	7.5	180

Both the demand for gas and its cost increase as the the weather becomes colder. The storage cost is constant, say, 1 unit of gas is $1 per year. If we solve the linear program for each scenario separately, we arrive at three purchase/storage strategies:

- Normal – Normal

Year	Purchase to Use	Purchase to Store	Storage	Cost
1	100	0	0	500
2	100	0	0	500

Total Cost = $1000

- Normal – Cold

Year	Purchase to Use	Purchase to Store	Storage	Cost
1	100	0	0	500
2	150	0	0	900

Total Cost = $1400

- Normal – Very Cold

Year	Purchase to Use	Purchase to Store	Storage	Cost
1	100	180	180	1580
2	0	0	0	0

Total Cost = $1580

We do not know which of the three scenarios will actually occur next year, but we would like our current purchasing decision to put is in the best position to minimize our expected cost. Bear in mind that by the time we make our second purchasing decision, we will know which of the three scenarios has actually happened.

FORMULATING A STOCHASTIC LINEAR PROGRAM

Stochastic programs seek to minimize the cost of the first-period decision plus the expected cost of the second-period recourse decision.

$$\min \quad c^T x + E_\omega Q(x,\omega) \quad \text{s.t.} \quad Ax = b \quad x \geq 0$$

min���+���(�,�)s.t.��=��≥0

where

$$Q(x,\omega)=\min \text{s.t.} d(\omega)^{T}yT(\omega)x+W(\omega)y=h(\omega)y\geq 0$$�(�,�)=min�(�)��s.t.�(�)
�+�(�)�=ℎ(�)�≥0

The first linear program minimizes the first-period direct costs, $c^{T}x$��� plus the expected recourse cost, $Q(x,\omega)$�(�,�), over all of the possible scenarios while meeting the first-period constraints, $Ax=b$��=�.

The recourse cost Q� depends both on x� the first-period decision and on the random event, ω�. The second LP describes how to choose $y(\omega)$�(�) (a different decision for each random scenario ω�). It minimizes the cost $d^{T}y$��� subject to some recourse function, $Tx+Wy=h$��+��=ℎ. This constraint can be thought of as requiring some action to correct the system after the random event occurs. In our example, this constraint would require the purchase of enough gas to supplement the original amount on hand in order to meet the demand.

One important thing to notice in stochastic programs is that the first-period decision, x�, is independent of which second-period scenario actually occurs . This is called the *nonanticipativity property*. The future is uncertain and so today’s decision cannot take advantage of knowledge of the future.