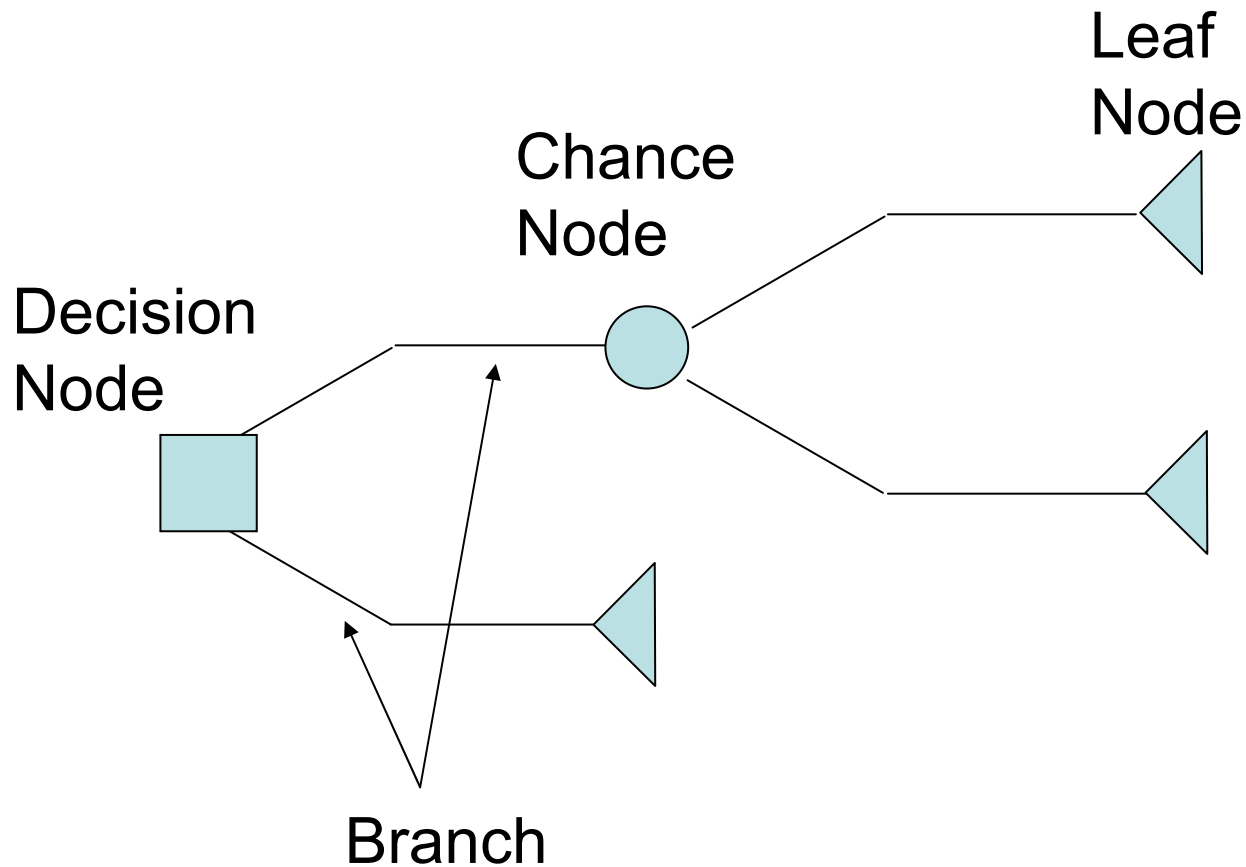


Decision Tree:

Decision trees are typically used to support decision-making in an uncertain environment.



TYPES of NODES:

1. Decision node

The branches originating from a decision node represent options available;

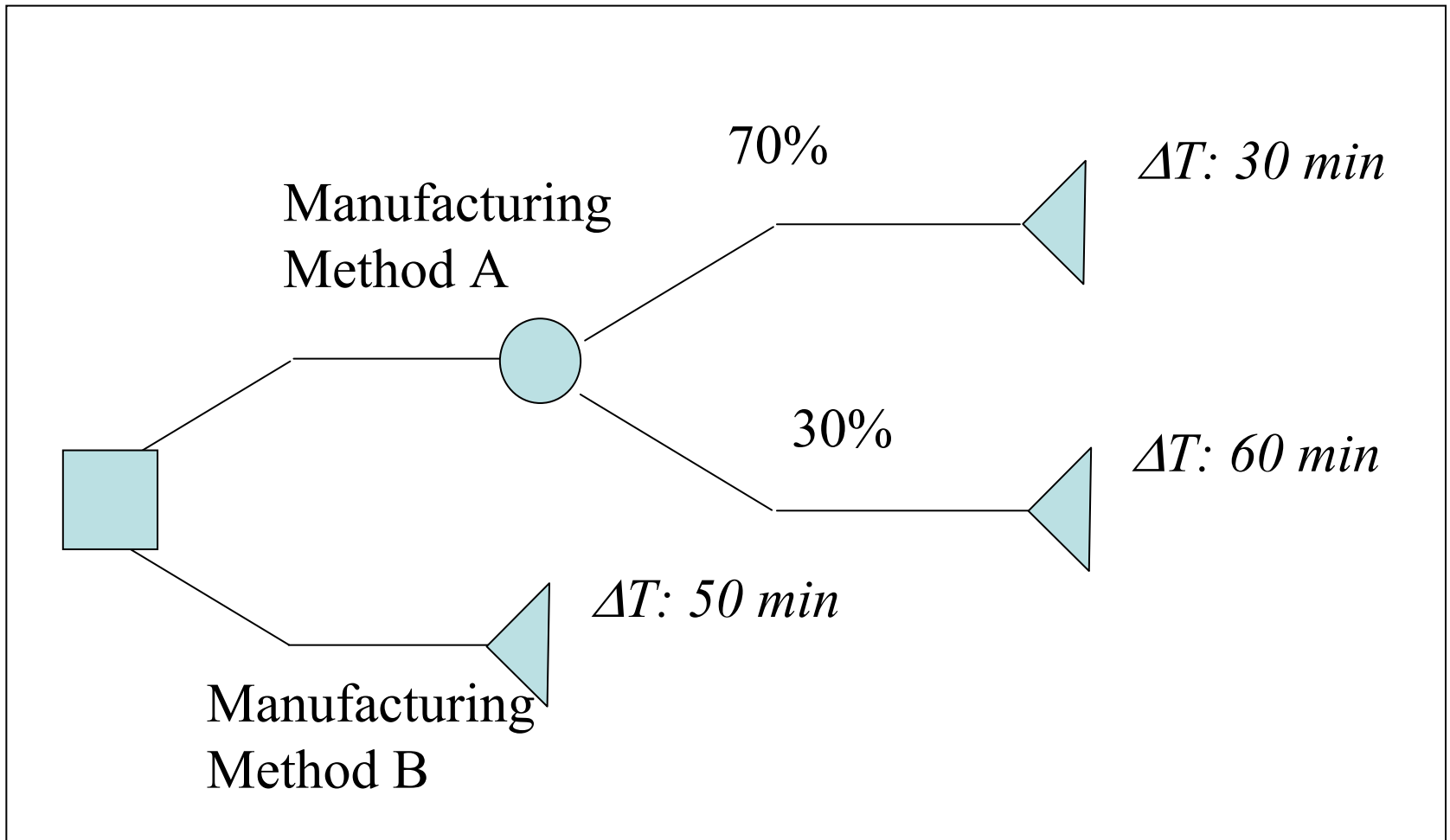
2. Chance node

The branches originating from a chance node represent uncontrollable events. At each chance node, each branch is assigned a conditional probability equal to the probability of the event represented by the branch, conditioned upon the knowledge available at the node.

3. Leaf node

Leaf nodes represent the possible endpoints, (i.e. the results of the decisions and chance outcomes associated with the path from the start of the tree)

Example:



Criterion: Minimum Production Time

Question: Which method to choose?

Decision Tree Analysis:

The objective of finding the optimal solution—that is, the best set of choices at the decision nodes—can be achieved by applying a “*roll-up*” process to the decision tree.

Starting with the leaf nodes and progressing recursively toward the root, we label each node by the value of the situation it represents. Each chance node is labeled with the expected value of its successors, and each decision node is labeled with the value of the choice.

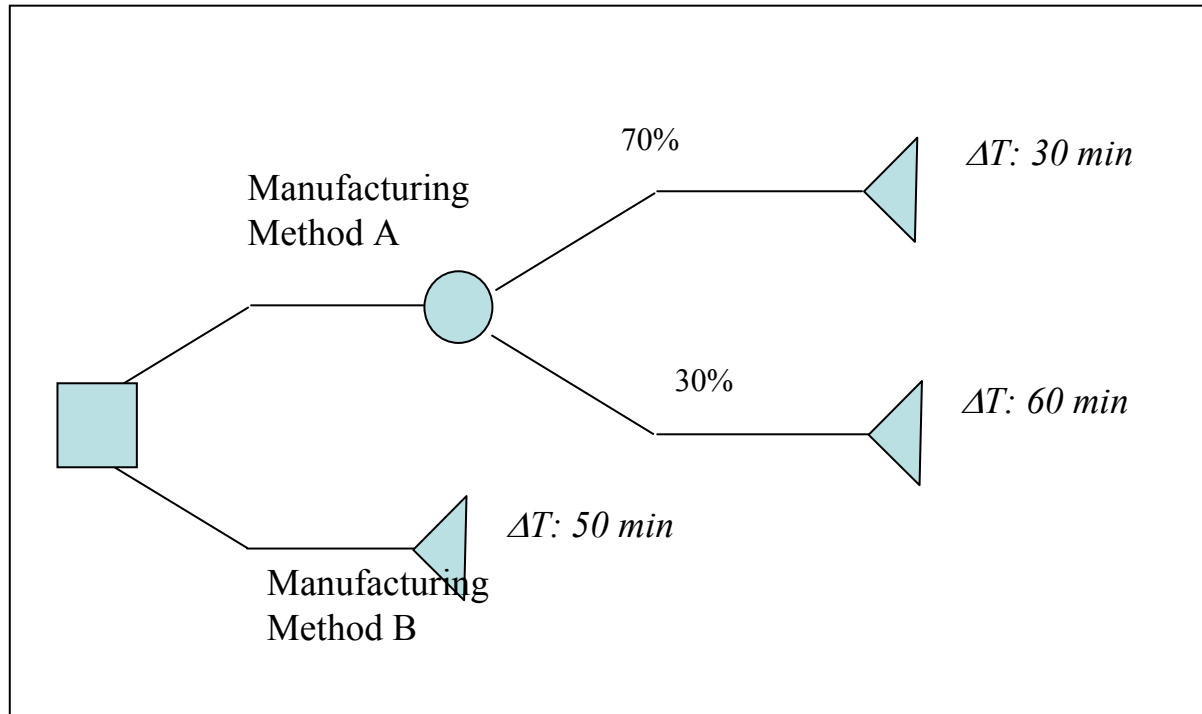
The rules of the “roll-up” analysis:

1. The expected value at a chance node can be calculated by multiplying values along the branches by its probability and adding the results together.

$$\textit{ExpectedValue} = \sum_{i=1}^N \textit{Value}_n \times \textit{Probability}_n$$

2. The expected value at the decision node is that of the best option (e.g. MIN(a,b) or MAX(a,b))

Example Problem



Criterion: Minimum Production Time

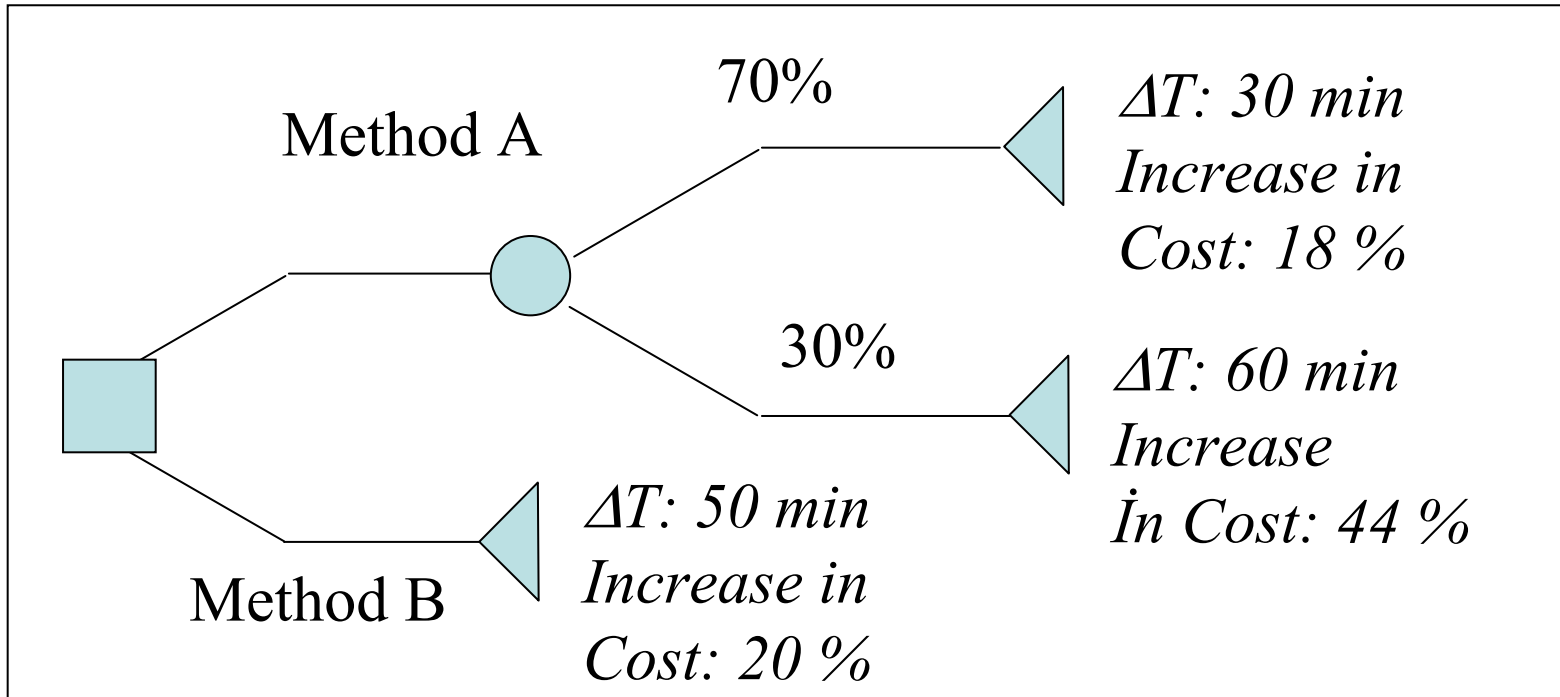
Rule 1: Expected Production Time = $(30 \cdot 0.7 + 60 \cdot 0.3) = 39$

Rule 2: $\text{MIN}(39, 50) = 39$

Conclusion: Method A is a better choice than Method B based on the criterion.

Extended version of the example problem:

Criterion: Minimum Production Cost



Rule 1: Expected Production Cost = $(18 \cdot 0.7 + 44 \cdot 0.3) = 26$

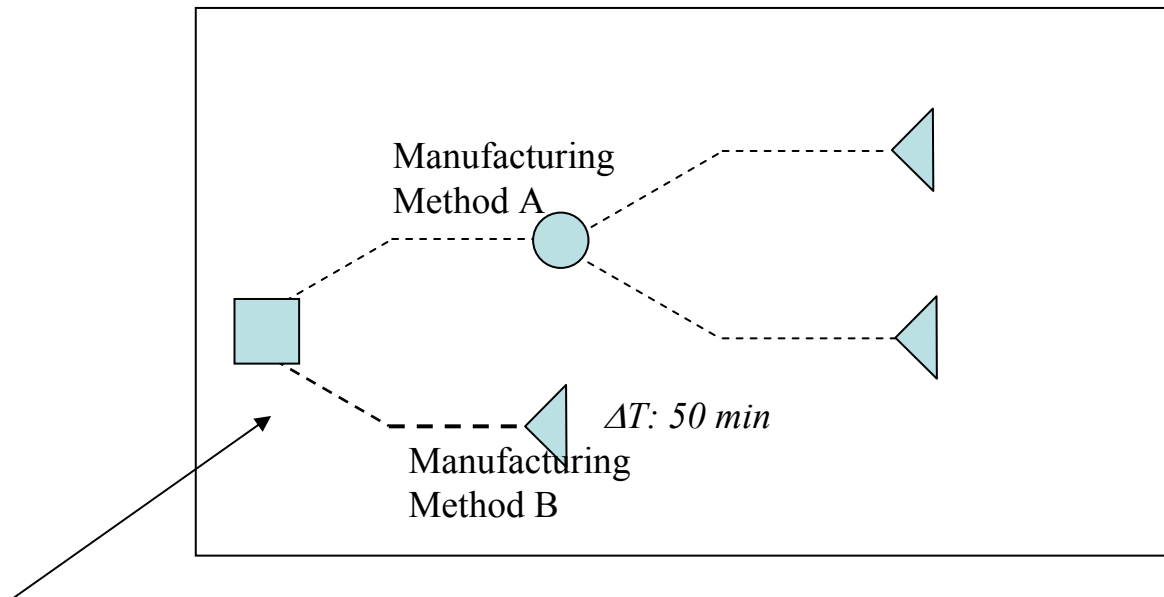
Rule 2: $\text{MIN}(26, 20) = 20$

Conclusion: Method B is a better choice than Method A based on the criterion.

Question: If time and cost are equally important, what to do?
(you will attempt to find an answer to this question in Project 03)

Why do we need pdfs/cdfs ?

In our example, we assumed that production time for Method B would be 50 minutes. However, in reality, we know that this will very unlikely to take *exactly* 50 minutes, so we might want to conduct sensitivity analyses later.



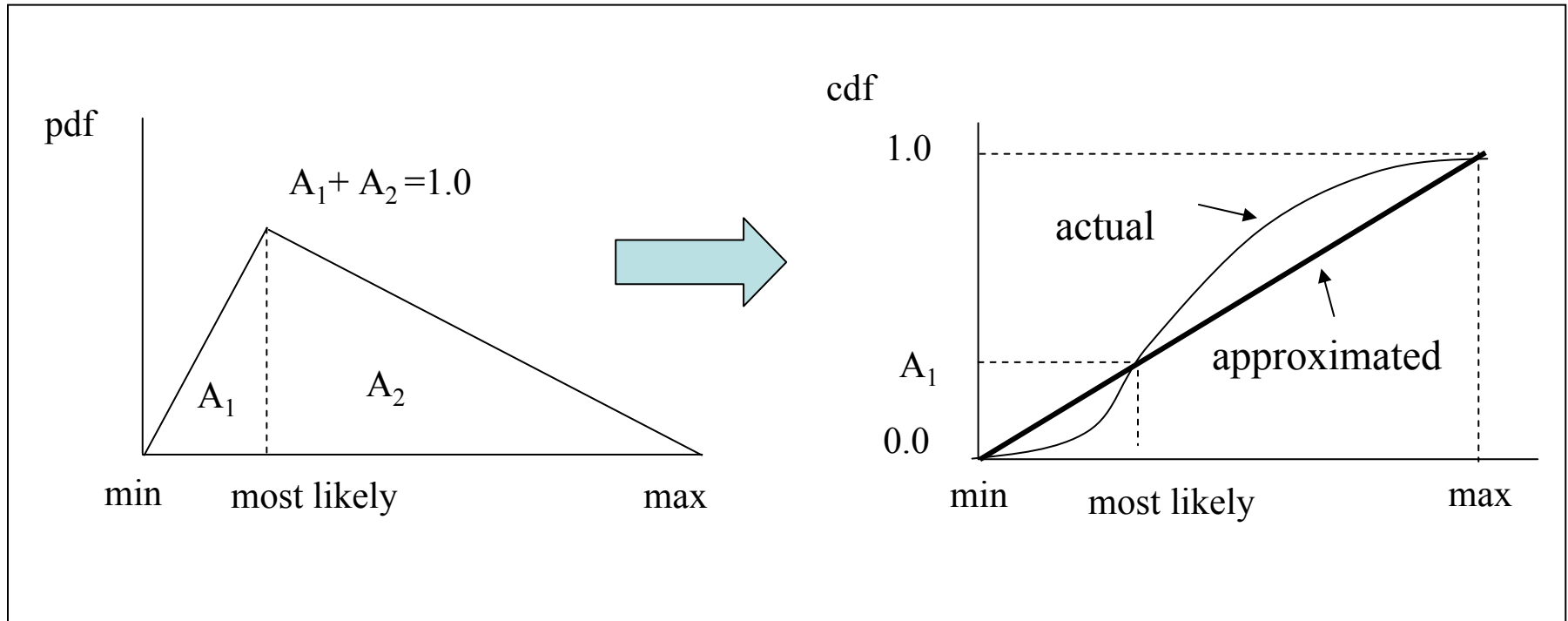
exactly 50 minutes ?? (not realistic)

Min: 34, Most Likely: 50, Max: 55 (more realistic)



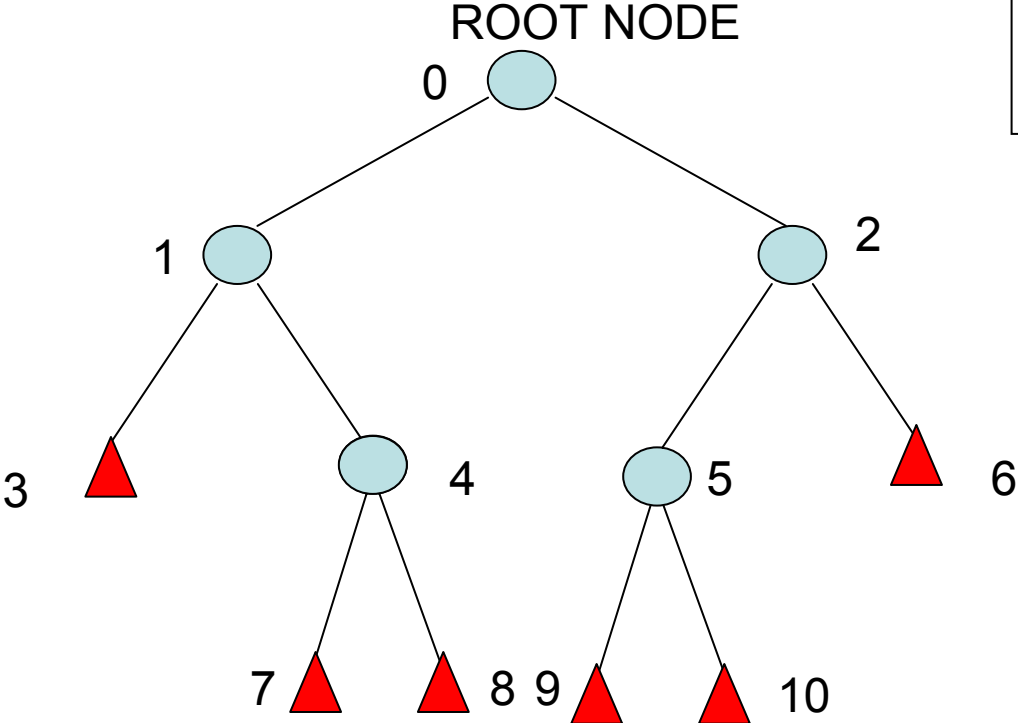
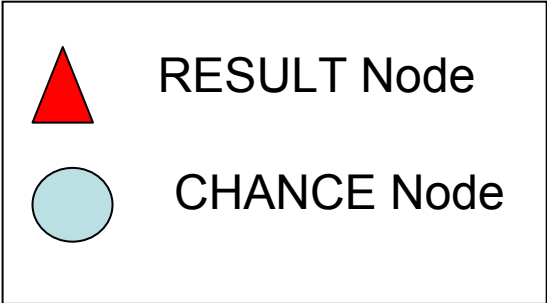
A triangular pdf

A triangular pdf and cdf



Now, instead of single numbers along the branches, probability density functions (one for production time and another for cost) are defined for each branch.

Project 03:



Paths:

- 0-1-3
- 0-1-4-7
- 0-1-4-8
- 0-2-5-9
- 0-2-5-10
- 0-2-6

Pseudo-code for Running Monte Carlo Iterations

```
for i = 0 to N      // N: number of Monte Carlo iterations
{
    total_time = total_cost = 0;
    node_index = 0;      // start traversing the tree from the root node

    do
        r1 = generate_random_number();
        node_index = decide_which_branch_to_take (r1);
        r2 = generate_random_number();
        time = find_the_corresponding_time_value_from_CDF(node_index,r2);
        cost = find_the_corresponding_cost_value_from_CDF(node_index ,r2);
        total_time = total_time + time;
        total_cost = total_cost + cost;
    while (node_index != RESULT node)
        path_no = node[node_index].number_of_paths;
        node[node_index].path[path_no].total_time = total_time;
        node[node_index].path[path_no].total_cost = total_cost;
        node[node_index].number_of_paths++;
}
```

Generating Random Variables using the “C” library

Since computers are deterministic devices, it is somewhat odd to think of them as capable of generating random numbers. Methods that generate series of values which appear to be random are used. Such methods are called *pseudo-random number generators*.

```
/* function to generate pseudo-random number between 0 and 1 */
```

```
#include <stdlib.h>
```

```
double simple_uniform()
```

```
{
```

```
    return ((double) rand() / RAND_MAX);
```

```
}
```

max random
integer



Setting the Seed in the “C” Library


`srand(int)` function allows the programmer to reset the seed value of the random number generator.

```
#include <time.h>
```

```
...
```

```
...
```

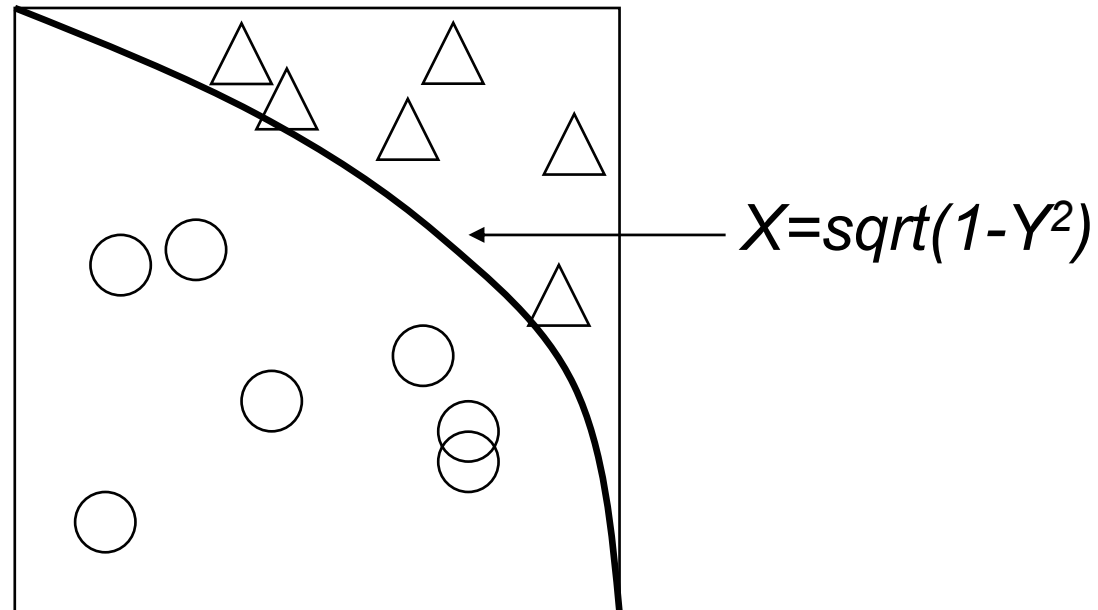
```
srand( (unsigned)time( NULL ) );
```



The seed value of the random number generator is connected to the clock of the computer so that a different number will be generated each time the `rand()` is called.

Example: One application of probabilistic simulation is Monte Carlo integration, where random number generators are used to approximate complicated integrals.

Consider the problem of integrating the first quadrant of the unit circle. We know the area under the curve is $(\pi/4)$.



```
#include <stdlib.h>
main()
{
    int i;
    int count=0;
    double x,y;
    int num_trials = 1000000;
    for(i=1;i < num_trials; i++)
    {
        x = simple_uniform();
        y = simple_uniform();
        if(x*x + y*y < 1.0)
            count++;
    }
    printf("Area = %f\n", count / num_trials);
}
```