Math 6367 Optimization 2
HW 02
Prof. Dr. Ronald H.W. Hoppe

29 April 2019
Jonathan Schuba

Assignment:

A student is taking three courses -- Algebra, Geometry, and Optimization. Each with a respective probability of failure. The student has only four hours to study, and wants to minimize the probability of failing all three courses.

Note: this is a shortest path problem, in which the final path length is the product (rather than the sum) of the individual sub-paths.

In [1]:
```
import numpy as np
import pandas as pd

courses = ["Algebra", "Geometry", "Optimization"]
print("Note: The courses are indexed:")
for i, course in enumerate(courses):
    print(f"{i}: {course}")
print()

failure_prob = [[0.8, 0.75, 0.90],
                [0.70, 0.70, 0.70],
                [0.65, 0.67, 0.60],
                [0.62, 0.65, 0.55],
                [0.60, 0.62, 0.50]]
# Transpose the failure matrix, so that we can index it by [course][hours]
failure_prob = np.transpose(failure_prob)


def print_failure_matrix(failure_matrix, course_list):
    # Function to pretty-print the failure matrix
    df = pd.DataFrame(failure_matrix)
    df.columns = [i for i in range(len(failure_matrix[-1]))]
    df.index = course_list
    print("The course-failure probability matrix is:")
    print(df)
    print()

print_failure_matrix(failure_prob, courses)
```

```
Note: The courses are indexed:
0: Algebra
1: Geometry
2: Optimization

The course-failure probability matrix is:
                 0    1     2     3     4
Algebra       0.80  0.7  0.65  0.62  0.60
Geometry      0.75  0.7  0.67  0.65  0.62
Optimization  0.90  0.7  0.60  0.55  0.50
```

In tbe next cell block, we will define a function to perform the Backward Dynamic Programming Algorithm. The algorithm is defined as:

$$f_3(x) = p_3(x)$$
$$f_k(x) = min_{t \leq x}\{p_k(t) f_{k+1}(x-t)\}$$

This function stores and returns all of the values of $f_k(x)$ as well as the argmin, t, which produced that value. The argmin list will be used later in the forward algorithm to determine the path which produces the final minimizing value. In this problem, it is convenient that the indicies of the probability matrix are the same as the values (ie: p[0][1] = 0.7 corresponds to studying algebra for one hour).

In [2]:
```
def backward_dp(failure_prob):
    f = []        # store all values of f_k(x)
    f_arg = []    # store the argmin of t which produced the final value in f_k(x
)
    for k in range(len(failure_prob)-1, -1, -1):
        fk = []
        fk_arg = []
        for i in range(len(failure_prob[0])):
            if k == len(failure_prob)-1:
                fk.append(failure_prob[k][i])
                fk_arg.append(i)
                print(f"f_{k}[{i}] = {failure_prob[k][i]}")
            else:
                if i == 0:
                    t = 0
                    temp = failure_prob[k][t]*f[-1][i-t]
                    fk.append(temp)
                    fk_arg.append(0)
                else:
                    temp = []
                    for t in range(i):
                        temp.append(failure_prob[k][t]*f[-1][i-t])
                    fk.append(min(temp))
                    fk_arg.append(np.argmin(temp))
                print(f"f_{k}[{i}] = min({np.round(temp, 4)}) = {np.round(fk[-1]
,4)}  ")
                print(f"\t argmin = {fk_arg[-1]}")

        f.append(fk)
        f_arg.append(fk_arg)
    return f, f_arg

f, f_arg = backward_dp(failure_prob)
```

```
f_2[0] = 0.9
f_2[1] = 0.7
f_2[2] = 0.6
f_2[3] = 0.55
f_2[4] = 0.5
f_1[0] = min(0.675) = 0.675
         argmin = 0
f_1[1] = min([0.525]) = 0.525
         argmin = 0
f_1[2] = min([0.45 0.49]) = 0.45
         argmin = 0
f_1[3] = min([0.4125 0.42   0.469 ]) = 0.4125
         argmin = 0
f_1[4] = min([0.375 0.385 0.402 0.455]) = 0.375
         argmin = 0
f_0[0] = min(0.54) = 0.54
         argmin = 0
f_0[1] = min([0.42]) = 0.42
         argmin = 0
f_0[2] = min([0.36   0.3675]) = 0.36
         argmin = 0
f_0[3] = min([0.33   0.315  0.3412]) = 0.315
         argmin = 1
f_0[4] = min([0.3    0.2888 0.2925 0.3255]) = 0.2888
         argmin = 1
```

The forward strategy walks through the argmin list to see how many hours to spend on each course. It then returns the optimal strategy.

In [4]:
```
def forward_strategy(f, f_arg):
    prob_of_failing_all_courses = f[-1][-1]

    strategy = [0 for _ in f]

    hours_remaining = len(f[0])-1

    for k in range(len(f)-1, -1, -1):
        hours_to_spend = f_arg[k][hours_remaining]
        strategy[k] = hours_to_spend
        hours_remaining -= hours_to_spend

    strategy.reverse()
    return prob_of_failing_all_courses, strategy

prob_of_failing_all_courses, strategy = forward_strategy(f, f_arg)

print(f"The probability of failing all courses is: {prob_of_failing_all_courses}
")
print("The optimal strategy is to spend:")
for course, hours in zip(courses, strategy):
    print(f"{hours} hours(s) on {course}")
```

```
The probability of failing all courses is: 0.28875
The optimal strategy is to spend:
1 hours(s) on Algebra
0 hours(s) on Geometry
3 hours(s) on Optimization
```