```
from pulp import *
```

Best Case Scenario

```
# Create a dictionary of the activities and their durations
activitiesBest = {
    'A': 2,
    'B': 4,
    'C': 4,
    'D1': 8,
    'D2': 24,
    'D3': 24,
    'D4': 40,
    'D5': 8,
    'D6': 10,
    'D7': 10,
    'D8': 4,
    'E': 24,
    'F': 4,
    'G': 12,
    'H': 8

}
# Create a list of the activities
activities_list = list(activitiesBest.keys())
# Create a dictionary of the activity precedences
precedences = {
    'A': [],
    'B': [],
    'C': ['A'],
    'D1': ['A'],
    'D2': ['D1'],
    'D3': ['D1'],
    'D4': ['D2', 'D3'],
    'D5': ['D4'],
    'D6': ['D4'],
    'D7': ['D6'],
    'D8': ['D5', 'D7'],
    'E': ['B', 'C'],
    'F': ['D8', 'E'],
    'G': ['A', 'D8'],
    'H': ['F', 'G']
    }
```

```
# Create the LP problem
prob = LpProblem("Critical Path", LpMinimize)
# Create the LP variables
start_times = {activity: LpVariable(f"start_{activity}", 0, None) for activity in
activities_list}
end_times = {activity: LpVariable(f"end_{activity}", 0, None) for activity in
activities_list}

# Add the constraints
for activity in activities_list:
    # Duration constraint: end time = start time + duration
    prob += end_times[activity] == start_times[activity] + activitiesBest[activity], f"{activity}_duration"

    # Precedence constraints: start time should be after the end time of predecessors
    for predecessor in precedences[activity]:
        prob += start_times[activity] >= end_times[predecessor], f"{activity}_predecessor_{predecessor}"
# Set the objective function
prob += lpSum([end_times[activity] for activity in activities_list]), "minimize_end_times"
# Solve the LP problem
status = prob.solve()
# Print the results
print("Critical Path time:")
for activity in activities_list:
    print(f"{activity} starts at {value(start_times[activity])} hours and ends at {value(end_times[activity])} hours")

if value(end_times[activity]) == max([value(end_times[activity]) for activity
in activities_list]):
    print(f"{activity} ends at {value(end_times[activity])} hours in duration")
```

```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/juliaokeeffe/Library/Python/3.9/lib/python/site-packages/pulp/solverdir/cbc/osx/64/cbc /var/folders/lv
At line 2 NAME          MODEL
At line 3 ROWS
At line 39 COLUMNS
At line 123 RHS
At line 158 BOUNDS
At line 159 ENDATA
Problem MODEL has 34 rows, 30 columns and 68 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 0 (-34) rows, 0 (-30) columns and 0 (-68) elements
Empty problem - 0 rows, 0 columns and 0 elements
Optimal - objective value 882
After Postsolve, objective 882, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 882 - 0 iterations time 0.002, Presolve 0.00
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.01

Critical Path time:
A starts at 0.0 hours and ends at 2.0 hours
B starts at 0.0 hours and ends at 4.0 hours
C starts at 2.0 hours and ends at 6.0 hours
D1 starts at 2.0 hours and ends at 10.0 hours
D2 starts at 10.0 hours and ends at 34.0 hours
D3 starts at 10.0 hours and ends at 34.0 hours
D4 starts at 34.0 hours and ends at 74.0 hours
D5 starts at 74.0 hours and ends at 82.0 hours
D6 starts at 74.0 hours and ends at 84.0 hours
D7 starts at 84.0 hours and ends at 94.0 hours
D8 starts at 94.0 hours and ends at 98.0 hours
E starts at 6.0 hours and ends at 30.0 hours
F starts at 98.0 hours and ends at 102.0 hours
G starts at 98.0 hours and ends at 110.0 hours
H starts at 110.0 hours and ends at 118.0 hours
H ends at 118.0 hours in duration
```

Expected Case

```python
# Create a dictionary of the activities and their durations
activitiesExpected = {
    'A': 4,
    'B': 22,
    'C': 10,
    'D1': 20,
    'D2': 36,
    'D3': 36,
    'D4': 60,
    'D5': 20,
    'D6': 24,
    'D7': 28,
    'D8': 10,
    'E': 52,
    'F': 10,
    'G': 22,
    'H': 20

}

# Create a list of the activities
activities_list = list(activitiesExpected.keys())
# Create a dictionary of the activity precedences
precedences = {
    'A': [],
    'B': [],
    'C': ['A'],
    'D1': ['A'],
    'D2': ['D1'],
    'D3': ['D1'],
    'D4': ['D2', 'D3'],
    'D5': ['D4'],
    'D6': ['D4'],
    'D7': ['D6'],
    'D8': ['D5', 'D7'],
    'E': ['B', 'C'],
    'F': ['D8', 'E'],
    'G': ['A', 'D8'],
    'H': ['F', 'G']
    }



# Create the LP problem
prob = LpProblem("Critical Path", LpMinimize)
# Create the LP variables
start_times = {activity: LpVariable(f"start_{activity}", 0, None) for activity in
activities_list}
end_times = {activity: LpVariable(f"end_{activity}", 0, None) for activity in
activities_list}

# Add the constraints
for activity in activities_list:
    # Duration constraint: end time = start time + duration
    prob += end_times[activity] == start_times[activity] + activitiesExpected[activity], f"{activity}_duration"

    # Precedence constraints: start time should be after the end time of predecessors
    for predecessor in precedences[activity]:
        prob += start_times[activity] >= end_times[predecessor], f"{activity}_predecessor_{predecessor}"
# Set the objective function
prob += lpSum([end_times[activity] for activity in activities_list]), "minimize_end_times"
# Solve the LP problem
status = prob.solve()
# Print the results

print("Critical Path time:")
for activity in activities_list:
    print(f"{activity} starts at {value(start_times[activity])} hours and ends at {value(end_times[activity])} hours")

if value(end_times[activity]) == max([value(end_times[activity]) for activity
in activities_list]):
    print(f"{activity} ends at {value(end_times[activity])} hours in duration")
```

```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/juliaokeeffe/Library/Python/3.9/lib/python/site-packages/pulp/solverdir/cbc/osx/64/cbc /var/folders/lv
At line 2 NAME          MODEL
At line 3 ROWS
At line 39 COLUMNS
At line 123 RHS
At line 158 BOUNDS
At line 159 ENDATA
Problem MODEL has 34 rows, 30 columns and 68 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 0 (-34) rows, 0 (-30) columns and 0 (-68) elements
Empty problem - 0 rows, 0 columns and 0 elements
Optimal - objective value 1636
After Postsolve, objective 1636, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 1636 - 0 iterations time 0.002, Presolve 0.00
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.00

Critical Path time:
A starts at 0.0 hours and ends at 4.0 hours
B starts at 0.0 hours and ends at 22.0 hours
C starts at 4.0 hours and ends at 14.0 hours
D1 starts at 4.0 hours and ends at 24.0 hours
D2 starts at 24.0 hours and ends at 60.0 hours
D3 starts at 24.0 hours and ends at 60.0 hours
D4 starts at 60.0 hours and ends at 120.0 hours
D5 starts at 120.0 hours and ends at 140.0 hours
D6 starts at 120.0 hours and ends at 144.0 hours
D7 starts at 144.0 hours and ends at 172.0 hours
D8 starts at 172.0 hours and ends at 182.0 hours
E starts at 22.0 hours and ends at 74.0 hours
F starts at 182.0 hours and ends at 192.0 hours
G starts at 182.0 hours and ends at 204.0 hours
H starts at 204.0 hours and ends at 224.0 hours
H ends at 224.0 hours in duration
```

Worst-case

```
# Create a dictionary of the activities and their durations
activitiesWorst = {
    'A': 8,
    'B': 40,
    'C': 16,
    'D1': 32,
    'D2': 48,
    'D3': 28,
    'D4': 100,
    'D5': 32,
    'D6': 40,
    'D7': 48,
    'D8': 16,
    'E': 80,
    'F': 16,
    'G': 32,
    'H': 32

}


# Create a list of the activities
activities_list = list(activitiesWorst.keys())
# Create a dictionary of the activity precedences
precedences = {
    'A': [],
    'B': [],
    'C': ['A'],
    'D1': ['A'],
    'D2': ['D1'],
    'D3': ['D1'],
    'D4': ['D2', 'D3'],
    'D5': ['D4'],
    'D6': ['D4'],
    'D7': ['D6'],
    'D8': ['D5', 'D7'],
    'E': ['B', 'C'],
    'F': ['D8', 'E'],
    'G': ['A', 'D8'],
    'H': ['F', 'G']
    }



# Create the LP problem
prob = LpProblem("Critical Path", LpMinimize)
# Create the LP variables
start_times = {activity: LpVariable(f"start_{activity}", 0, None) for activity in
activities_list}
end_times = {activity: LpVariable(f"end_{activity}", 0, None) for activity in
activities_list}

# Add the constraints
for activity in activities_list:
    # Duration constraint: end time = start time + duration
    prob += end_times[activity] == start_times[activity] + activitiesWorst[activity], f"{activity}_duration"
```