Part 1: Problem Setup

The following assignment is a Critical Path problem where the objective of the problem is to find the minimal amount of time to complete all tasks deemed necessary for the completion and successful delivery of a restaurant reeccomendation applcation. After each essential task was identified, time estimates in terms of hours and preceding steps. The entire tablle used to specifiy the project details can be in WONG_project-plan-v003.xlsx found or below:

taskID	task	predecessorTaskIDs	bestCaseHours	expectedHours	worstCaseHours
A	Describe product		10	18	25
B	Develop marketing strategy		6	20	22
C	Design brochure	A	3	6	10
D	Develop product prototype				
D1	Requirements analysis	A	15	30	60
D2	Software design	D1	10	15	20
D3	System design	D1	9	10	18
D4	Coding	D2, D3	65	100	170
D5	Write documentation	D4	25	30	50
D6	Unit testing	D4	25	50	72
D7	System testing	D4	20	23	30
D8	Package deliverables	D5, D7	6	10	20
E	Survey potential market	B, C	20	50	70
F	Develop pricing plan	D8, E	10	12	15
G	Develop implementation plan	A, D8	15	26	35
H	Write client proposal	F, G	9	15	20

A visual representation of workflow of each task can be seen below in this Visio diagram: image

The cost of labor per hour was estimated to be $800/ hr (representing the time from project start to finish. This does not represent the sum of all hours worked by each project member).

Part 2: Model Specification

Given the information above, a model was developed in Python utilizing the PuLP package to model the project timeline as a linear program. Furthermore, matplotlib was used to develop Gantt charts based off the 3 scenarios. The entire code can be found in this repository (.pf file/.ipynb file).

There are several assumptions made about the following model:

- Each member of the development team charges the same price.
- Each member of the team has unlimited time meaning there will be no bottlenecks if a 2 tasks can be completed simultaneously but require the same team member.

- Each task can be immediately started assuming the preceding tasks are completed.

Part 3: Programming

In order to set up this linear program within Python, 2 dictionaries were initialized: *tasks* and *predecessors*.

Tasks represents the description of each task to be compelted as well as the three time estimates for how long the task will take.

predecessors has the same tasks and holds the TaskID of each task that must be completed before the current task can be started.

```
from pulp import *
import matplotlib.pyplot as plt
from pulp import LpMinimize, LpProblem, LpVariable, lpSum, value

tasks = {
    "A": {"name": "Describe product", "best_case": 10, "expected": 18, "worst_case": 25},
    "B": {"name": "Develop marketing strategy", "best_case": 6, "expected": 20, "worst_case": 22},
    "C": {"name": "Design brochure", "best_case": 3, "expected": 6, "worst_case": 10},
    "D1": {"name": "Requirements analysis", "best_case": 15, "expected": 30, "worst_case": 60},
    "D2": {"name": "Software design", "best_case": 10, "expected": 15, "worst_case": 20},
    "D3": {"name": "System design", "best_case": 9, "expected": 10, "worst_case": 18},
    "D4": {"name": "Coding", "best_case": 65, "expected": 100, "worst_case": 170},
    "D5": {"name": "Write documentation", "best_case": 25, "expected": 30, "worst_case": 50},
    "D6": {"name": "Unit testing", "best_case": 25, "expected": 50, "worst_case": 72},
    "D7": {"name": "System testing", "best_case": 20, "expected": 23, "worst_case": 30},
    "D8": {"name": "Package deliverables", "best_case": 6, "expected": 10, "worst_case": 20},
    "E": {"name": "Survey potential market", "best_case": 20, "expected": 50, "worst_case": 70},
    "F": {"name": "Develop pricing plan", "best_case": 10, "expected": 15, "worst_case": 20},
    "G": {"name": "Develop implementation plan", "best_case": 15, "expected": 26, "worst_case": 35
},
    "H": {"name": "Write client proposal", "best_case": 9, "expected": 15, "worst_case": 20}
}


predecessors = {
    "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"]
}
```

The next piece of the Python script is the “meat” of the code. A critical path minimization problem is initialized here and the activities as well as predecessors are fed into the funciton as inputs. For each activity, an equation is set up with the given time constraints and predecessors and the optimization problem is set up to minimize the total number of time that is needed to complete the entire project. All data is output onto the display and a

gantt chart is generated that displays the total time needed to complete the problem under the three scenarios.

```
def critical_path_analysis(activities, predecessors, scenario="expected"):
    # Create a list of the activities
    activities_list = list(activities.keys())
    # 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_l
ist}
    end_times = {activity: LpVariable(f"end_{activity}", 0, None) for activity in activities_list}

    # Add the constraints
    for activity in activities_list:
        # Access the duration for the current scenario
        duration = activities[activity][scenario]
        prob += end_times[activity] == start_times[activity] + duration, f"{activity}_duration"
        for predecessor in predecessors[activity]:
            prob += start_times[activity] >= end_times[predecessor], f"{activity}_predecessor_{pre
decessor}"

    # 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:
        if value(start_times[activity]) == 0:
            print(f"{activity} starts at time 0")
        if value(end_times[activity]) == max([value(end_times[activity]) for activity in activitie
s_list]):
            print(f"{activity} ends at {value(end_times[activity])} hours in duration")

    # Print solution
    print("\nSolution variable values:")
    for var in prob.variables():
        if var.name != "_dummy":
            print(var.name, "=", var.varValue)

    # Prepare data for Gantt chart
    gantt_data = []
    for activity in activities_list:
        start = value(start_times[activity])
        end = value(end_times[activity])
        gantt_data.append((activity, start, end - start))

    # Plot Gantt chart
    fig, ax = plt.subplots(figsize=(15, 8))
    for i, (task, start, duration) in enumerate(gantt_data):
      task_label = f"{task}: {activities[task]['name']}"
      ax.barh(task_label, duration, left=start, align='center')

    ax.set_xlabel('Time (Hour)')
    ax.set_title(f'{scenario.upper()} Gantt Chart')
    plt.show()
```

The following code was used to call the above function and display the optimization results for all three scenarios.

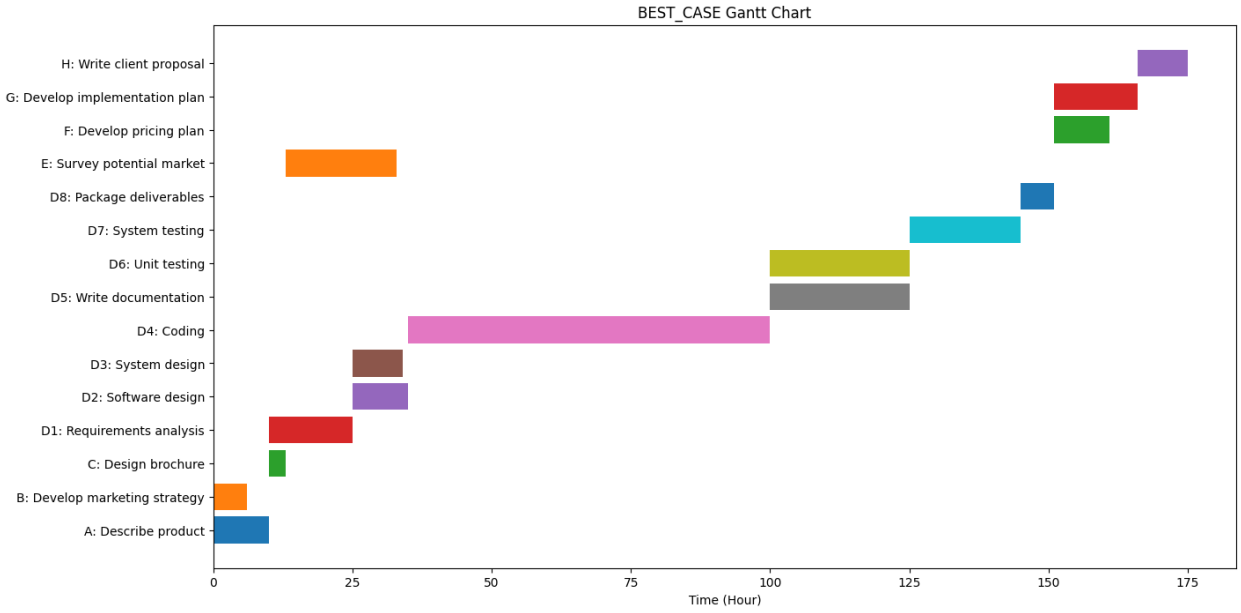
```
# Calculate and print the critical path for each scenario
scenarios = ["best_case", "expected", "worst_case"]
```

```
for scenario in scenarios:
    print(f"--- {scenario.upper()} SCENARIO ---")
    critical_path_analysis(tasks, predecessors, scenario)
    print()
```

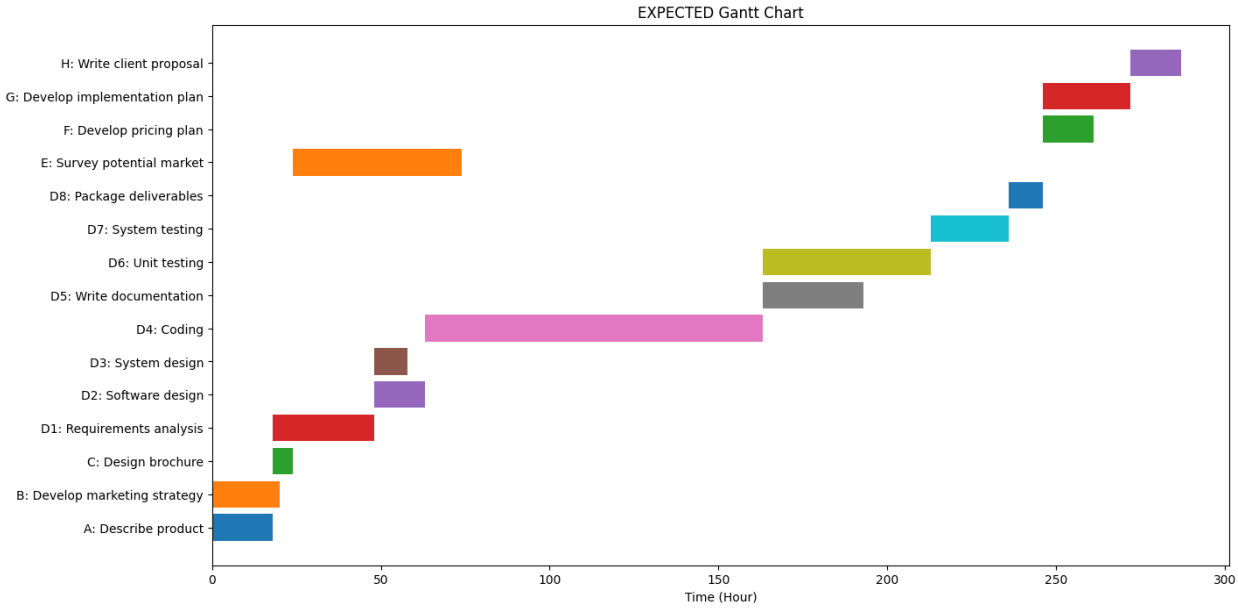
Part 4: Solution

Based off the results of the program here are the following results:

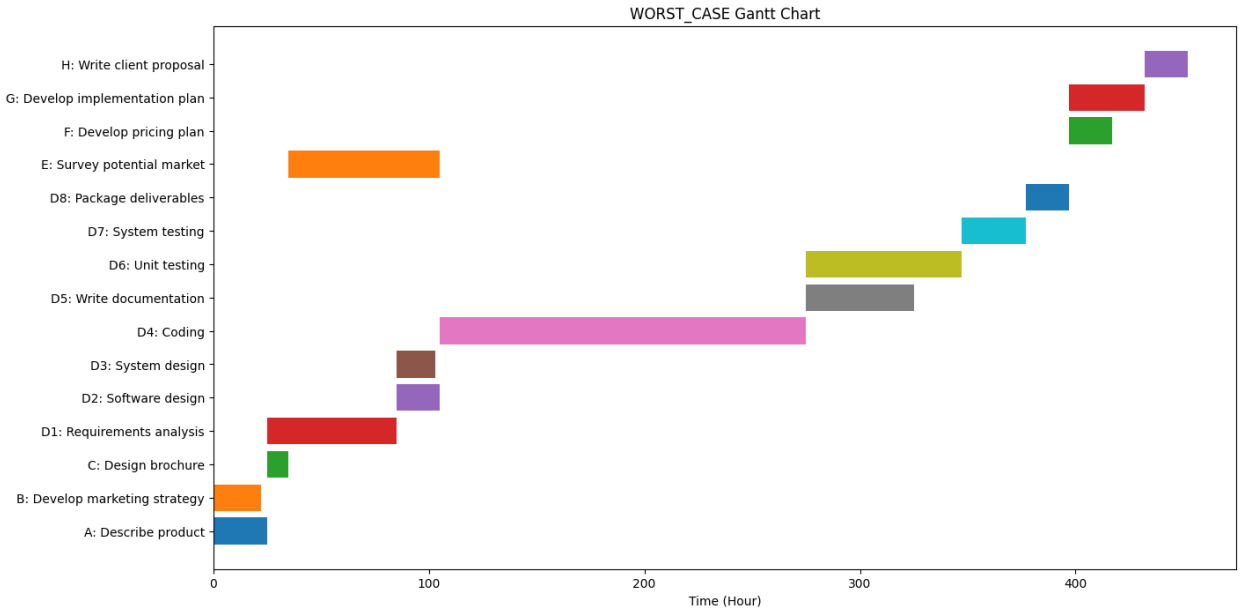
Best Case Scenario Total Time/cost: 175 hours / $105,000 total



Expected Scenario Total Time/Cost: 287 hours / $172,000



Worst Case Scenarip Total Time/Cost: 452 hours / $271,000



Based off these solutions, the project could take anywhere from 19 to 51 days to complete (assuming 9-hour work days)

Part 5: Overview

Here is an example writeup of what the overview could look like:

Hello,

We have finished developing a project timeline for your restaurant recommender system along with key timelines. Based off current work estimates, we believe that the project will take close to 300 billable hours putting the total estimated cost of this project endeavor to just around $200,000. It should be noted that while not expected, significant delays could increase the cost of the project by 50%. Estimated The estimated delivery time will be approximately 1 month. Depending on the needs of your organization, hiring extra help could be an option to increase the speed of delivery.

Given the current project team of 5, adding independent contractors could be used to speed up the delivery of the application. Given a single extra worker (which would mean an reduction in ~17% of work per team member), you could expect to shave off approximately 1 business week.

While linear programming was used in this scenario to simualte a project timeline, there were a number of key assumptions that may decrease the validity of the estimated timeline. For example, if the probability of a task falling behind or being ahead of schedule is already known, then this is a scenario in which a montecarlo simulaiton may be more appropriate to give a better idea of how long a project will take. Similarly, stochastic

programming be more suitable if the modeling prediction will need to adapt to new conditions/ data as time goes on. This may be a more suitable approach if the project timeline is regularly updated as the project progresses (I am thinking this could be especially useful in scenarios in which the project may take longer than a few months, potentially spanning years).