

CTF PROJECTS & RESEARCH - 2025 RECRUITMENT TASK

PROJECT NAME: Intelligent Traffic Signal System for Ambulance Priority Using GPS

TASK:

Dijkstra's shortest path algorithm

- 1. Performing removal operation wrongly:** u is a vertex from Q to be removed but the code uses k , which is wrong for removal.
- 2. Priority Queue operation:** Priority queue shall need adjustment based on the neighbor node n instead of v .
- 3. Incorrect updation of distance:** Instead of updating the distance $\text{dist}[n]$ (representing the neighbor), the distance $\text{dist}[v]$ is updated.
- 4. Incorrect variable names:** The code inconsistently uses k , n , v , and neighbor—creating confusion. For example, k is used but not initialized, and it mixes k with neighbor n in the loop.

Pseudocode :

```
function Dijkstra(Graph, source):
    dist[source] <- 0
    for each vertex v in Graph:
        if v ≠ source:
            dist[v] <- ∞
    Q ← priority queue of all vertices with dist[] as the key
    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q
        for each neighbor v of u:
            alt <- dist[u] + length(u, v)
            if alt < dist[v]:
                dist[v] <- alt
                decrease priority of v in Q
    return dist
```

Code in C++ :

```
#include <iostream>
#include <vector>
#include <queue>
#include <limits.h>

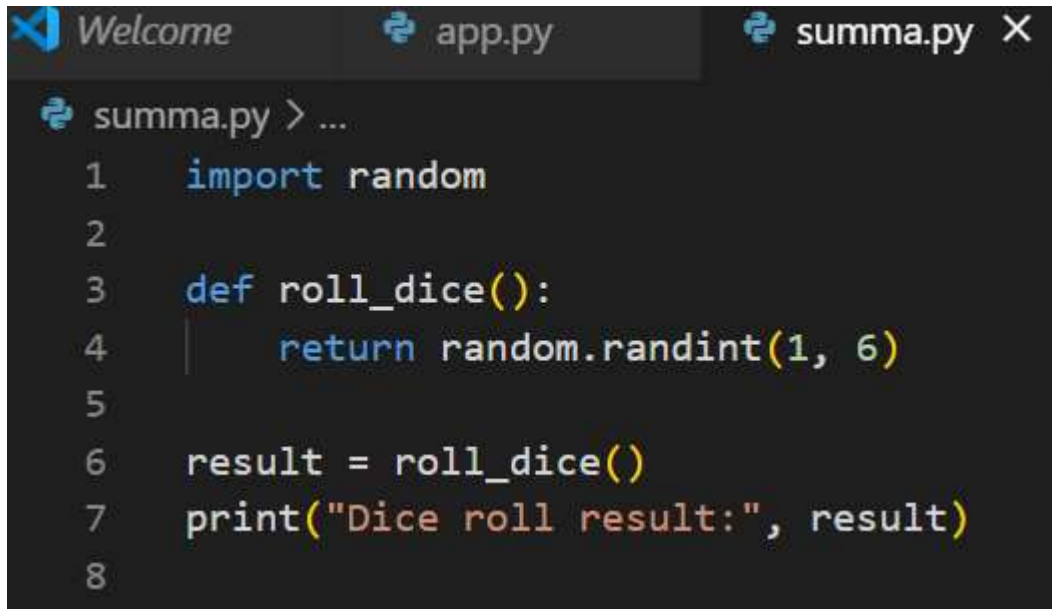
using namespace std;
struct Node {
    int vertex, distance;
    bool operator>(const Node& other) const {
        return distance > other.distance;
    }
};

vector<int> Dijkstra(const vector<vector<pair<int, int>>>& Graph, int source) {
    int n = Graph.size();
    vector<int> dist(n, INT_MAX);
    dist[source] = 0;
    priority_queue<Node, vector<Node>, greater<Node>> pq;
    pq.push({source, 0});
    while (!pq.empty()) {
        int u = pq.top().vertex;
        pq.pop();
        for (const auto& neighbor : Graph[u]) {
            int v = neighbor.first;
            int weight = neighbor.second;
            int alt = dist[u] + weight;
            if (alt < dist[v]) {
                dist[v] = alt;
                pq.push({v, dist[v]});
            }
        }
    }
    return dist;
}
```

2. Write a python code

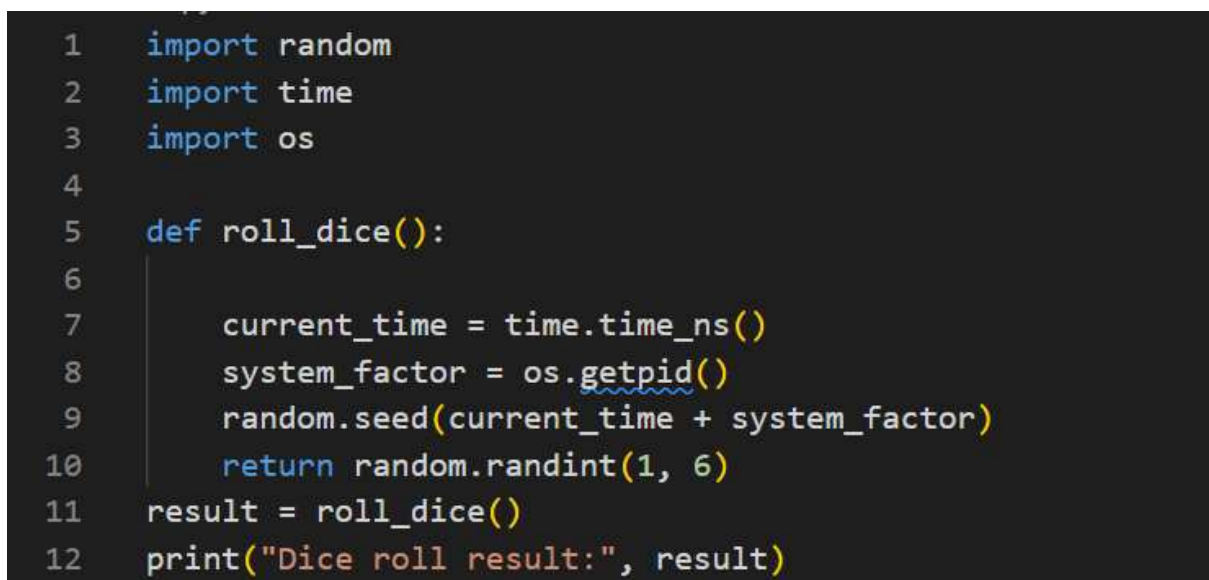
1. Generates a random number between 1 and 6 simulating a dice roll.

2. when the same code is simulated in 2 system at a time, output should not be the same.



The screenshot shows a code editor with three tabs: 'Welcome', 'app.py', and 'summa.py'. The 'summa.py' tab is active, displaying a Python script. The script imports the 'random' module, defines a function 'roll_dice()' that returns a random integer between 1 and 6, and then calls this function to print the result. The code is as follows:

```
summa.py > ...
1  import random
2
3  def roll_dice():
4      return random.randint(1, 6)
5
6  result = roll_dice()
7  print("Dice roll result:", result)
8
```



The screenshot shows a code editor with a single tab containing a Python script. This script imports 'random', 'time', and 'os' modules. The 'roll_dice()' function generates a unique seed by combining the current time in nanoseconds with the current process ID (obtained via 'os.getpid()'). This seed is used to initialize the random number generator before returning a random integer between 1 and 6. The code is as follows:

```
1  import random
2  import time
3  import os
4
5  def roll_dice():
6
7      current_time = time.time_ns()
8      system_factor = os.getpid()
9      random.seed(current_time + system_factor)
10     return random.randint(1, 6)
11
12 result = roll_dice()
13 print("Dice roll result:", result)
```