**Rubric**

| Checking if neighbours exists for a vertex | **2** |
| --- | --- |
| Deadend condition handled | **2** |
| Finding max/min edge | **5** |
| Checking if enough points available for next jump | **2** |
| Updating variables for next iteration | **2** |
| Returning result | **2** |
| **Total** | **15** |

**Set A**

**For Adjacency Lis**t

| def traverse\_max\_edge (adj\_list, s, p):  current\_vertex = s  points = p  while True:  neighbors = adj\_list[current\_vertex]  if not neighbors:  print(f"Reached a dead end")  break  max\_edge = find\_max\_edge(neighbors)  destination, weight = max\_edge  if weight > points:  break  points -= weight  current\_vertex = destination  return current\_vertex | def find\_max\_edge(neighbors):  max\_edge = None  max\_weight = float('-inf')    for edge in neighbors:  if edge[1] > max\_weight:  max\_edge = edge  max\_weight = edge[1]    return max\_edge |
| --- | --- |

**For Adjacency Matrix**

| def traverse\_max\_edge (adj\_matrix, s, p):  current\_vertex = s  points = p  vertices = len(adj\_matrix)  while True:  neighbors = []  for v in range(vertices):  if adj\_matrix[current\_vertex][v] > 0:  neighbors += [(v, adj\_matrix[current\_vertex][v])]  if not neighbors:  print(f"Reached a dead end")  break  max\_edge = find\_max\_edge(neighbors)  destination, weight = max\_edge  if weight > points:  break  points -= weight  current\_vertex = destination  return current\_vertex | def find\_max\_edge(neighbors):  max\_edge = None  max\_weight = float('-inf')    for destination, weight in neighbors:  if weight > max\_weight:  max\_edge = (destination, weight)  max\_weight = weight  return max\_edge |
| --- | --- |

**Set B**

**For Adjacency Lis**t

| def traverse\_min\_edge(adj\_list, s, p):  current\_vertex = s  points = p  while True:  neighbors = adj\_list[current\_vertex]  if not neighbors:  print(f"Reached a dead end")  break  min\_edge = find\_min\_edge(neighbors)  destination, weight = min\_edge  if weight > points:  break  points -= weight  current\_vertex = destination  return current\_vertex | def find\_min\_edge(neighbors):  min\_edge = None  min\_weight = float('inf')    for edge in neighbors:  if edge[1] < min\_weight:  min\_edge = edge  min\_weight = edge[1]    return min\_edge |
| --- | --- |

**For Adjacency Matrix**

| def traverse\_min\_edge(adj\_matrix, s, p):  current\_vertex = s  points = p  vertices = len(adj\_matrix)  while True:  neighbors = []  for v in range(vertices):  if adj\_matrix[current\_vertex][v] > 0:  neighbors += [(v, adj\_matrix[current\_vertex][v])]  if not neighbors:  print(f"Reached a dead end")  break  min\_edge = find\_min\_edge(neighbors)  destination, weight = min\_edge  if weight > points:  break  points -= weight  current\_vertex = destination  return current\_vertex | def find\_min\_edge(neighbors):  min\_edge = None  min\_weight = float('inf')    for destination, weight in neighbors:  if weight < min\_weight:  min\_edge = (destination, weight)  min\_weight = weight  return min\_edge |
| --- | --- |