125. Maximum Cut and Bin Packing Problem

AIM : To solve the Maximum Cut and Bin Packing Problem by using racibity and approximation algorithm

PROGRAM:

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.adj = [[] for _ in range(vertices)]

    def add_edge(self, u, v):
        self.adj[u].append(v)
        self.adj[v].append(u)

    def maximum_cut_approx(self):
        partition = [-1] * self.V  # Partition array where -1 means unassigned
        cut_size = 0

        for v in range(self.V):
            if partition[v] == -1:
                partition[v] = 1  # Assign vertex v to set 1

                # Count edges crossing the cut between set 1 and set 2
                for neighbor in self.adj[v]:
                    if partition[neighbor] == -1:
                        partition[neighbor] = 0  # Assign neighbor to set 2
                        cut_size += 1

        return partition, cut_size

# Example usage for Maximum Cut Problem:
g = Graph(6)
```

```
g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 3)
g.add_edge(2, 4)
g.add_edge(3, 4)
g.add_edge(3, 5)
g.add_edge(4, 5)

print("Maximum Cut Problem - Approximation Algorithm:")
print("Graph edges:")
for i in range(g.V):
    for j in g.adj[i]:
        if i < j:  # Print each edge only once
            print(f"{i} - {j}"
partition, cut_size = g.maximum_cut_approx()
print("Approximate Partition of Vertices:", partition)
print("Cut Size:", cut_size)
```

OUTPUT:

```
Maximum Cut Problem - Approximation Algorithm:
Graph edges:
0 - 1
0 - 2
1 - 3
2 - 4
3 - 4
3 - 5
4 - 5
Approximate Partition of Vertices: [1, 0, 0, 1, 0, 0]
Cut Size: 4
```

TIME COMPLEXITY: O(V+E)