7.6 APPROXIMATION ALGORITHM FOR MAXIMUM CUT

Question:
Implement an approximation algorithm for the Maximum Cut problem using a greedy or randomized approach. Compare the results with the optimal solution obtained through an exhaustive search for small graph instances.

Input:
• Graph G = (V, E) with V = {1, 2, 3, 4}, E = {(1,2), (1,3), (2,3), (2,4), (3,4)}
• Edge Weights: w(1,2) = 2, w(1,3) = 1, w(2,3) = 3, w(2,4) = 4, w(3,4) = 2

Output:
• Greedy Maximum Cut: Cut = {(1,2), (2,4)}, Weight = 6
• Optimal Maximum Cut (Exhaustive Search): Cut = {(1,2), (2,4), (3,4)}, Weight = 8
• Performance Comparison: Greedy solution achieves 75% of the optimal weight.

AIM
To implement a greedy approximation (and optional randomized) algorithm for Maximum Cut, and compare with the optimal solution produced via exhaustive search on a small graph.

ALGORITHM
1. Greedy (Deterministic) Construction:
    - Maintain two partitions A and B (initially empty).
2. Process vertices in some order; for each vertex v, compute gain_A = sum of weights to B, gain_B = sum of weights to A; place v in the partition that yields the larger gain.
3. Optionally perform local improvement: flip any vertex whose move increases the cut weight until no improvement remains.
4. Randomized (Optional):
    - Place each vertex into A or B uniformly at random; repeat several trials and keep the best cut.
5. Exhaustive Search (Optimal on Small Graphs):
    - Enumerate all 2^n assignments of vertices to A/B; compute the cut weight; keep the maximum.

PROGRAM

```
from itertools import combinations

def greedy_max_cut(n, edges):
    A = set()
    B = set()
    for v in range(1, n+1):
        if sum(w for u, vv, w in edges if vv == v and u in A) <= sum(w for u, vv, w in edges if vv == v and u in B):
            A.add(v)
        else:
            B.add(v)
    cut = [(u, v, w) for u, v, w in edges if (u in A and v in B) or (u in B and v in A)]
    return cut

def optimal_max_cut(n, edges):
    best = []
    for combo in combinations(range(1, n+1), n//2):
        A = set(combo)
        B = set(range(1, n+1)) - A
        cut = [(u, v, w) for u, v, w in edges if (u in A and v in B) or (u in B and v in A)]
        if sum(w for _, _, w in cut) > sum(w for _, _, w in best):
            best = cut
    return best

n = int(input("Enter number of vertices: "))
m = int(input("Enter number of edges: "))
edges = []
for _ in range(m):
    u, v, w = map(int, input("Edge (u v weight): ").split())
    edges.append((u, v, w))

greedy = greedy_max_cut(n, edges)
optimal = optimal_max_cut(n, edges)
print("Greedy Maximum Cut:", greedy)
print("Weight =", sum(w for _, _, w in greedy))
print("Optimal Maximum Cut:", optimal)
print("Weight =", sum(w for _, _, w in optimal))
```

Input:

- Graph G = (V, E) with V = {1, 2, 3, 4}, E = {(1,2), (1,3), (2,3), (2,4), (3,4)}
- Edge Weights: w(1,2) = 2, w(1,3) = 1, w(2,3) = 3, w(2,4) = 4, w(3,4) = 2

Output:

```
Enter number of vertices: 4
Enter number of edges: 5
Edge (u v weight): 1 2 2
Edge (u v weight): 1 3 1
Edge (u v weight): 2 3 3
Edge (u v weight): 2 4 4
Edge (u v weight): 3 4 2
Greedy Maximum Cut: [(1, 2, 2), (2, 3, 3), (2, 4, 4)]
Weight = 9
Optimal Maximum Cut: [(1, 2, 2), (1, 3, 1), (2, 4, 4), (3, 4, 2)]
Weight = 9
>>>
```

RESULT:

Thus the program is executed successfully and the output is verified.

PERFORMANCE ANALYSIS:

• Time Complexity: Typically O(|V|·deg + improvements); near-linear for sparse graphs.

• Space Complexity: O(2^{|V|}) time by enumerating all partitions; exact but exponential.