|  |  |
| --- | --- |
| **Disjoint Set in C++** | |
| #include <bits/stdc++.h>  using namespace std;  vector<int> parent, rankVec; // Renamed rank to rankVec  void makeSet(int n) {  parent.resize(n + 1);  rankVec.resize(n + 1, 0); // Use rankVec here  for (int i = 0; i <= n; i++) {  parent[i] = i;  }  }  int findUPar(int node) {  if (node == parent[node])  return node;  return parent[node] = findUPar(parent[node]);  }  void unionByRank(int u, int v) {  int ulp\_u = findUPar(u); // ultimate parent of u  int ulp\_v = findUPar(v); // ultimate parent of v  if (ulp\_u == ulp\_v) return; // already in the same set  // Union by rank  if (rankVec[ulp\_u] < rankVec[ulp\_v]) { // Use rankVec here  parent[ulp\_u] = ulp\_v;  }  else if (rankVec[ulp\_u] > rankVec[ulp\_v]) { // Use rankVec here  parent[ulp\_v] = ulp\_u;  }  else {  parent[ulp\_v] = ulp\_u;  rankVec[ulp\_u]++; // Use rankVec here  }  }  int main() {  int n = 7; // Number of elements  makeSet(n);  unionByRank(1, 2);  unionByRank(2, 3);  unionByRank(4, 5);  unionByRank(6, 7);  unionByRank(5, 6);  // Check if 3 and 7 are in the same set  if (findUPar(3) == findUPar(7)) {  cout << "Same\n";  } else {  cout << "Not same\n";  }  unionByRank(3, 7);  // Check again if 3 and 7 are in the same set  if (findUPar(3) == findUPar(7)) {  cout << "Same\n";  } else {  cout << "Not same\n";  }  return 0;  } | 1. **makeSet(n)**    * Initializes:      + parent = [0, 1, 2, 3, 4, 5, 6, 7]      + rankVec = [0, 0, 0, 0, 0, 0, 0, 0]    * Each element is its own parent initially, and the rank is 0. 2. **unionByRank(1, 2)**    * findUPar(1) returns 1 (root of 1).    * findUPar(2) returns 2 (root of 2).    * rankVec[1] (0) < rankVec[2] (0), so parent[2] = 1.    * Updated:      + parent = [0, 1, 1, 3, 4, 5, 6, 7]      + rankVec = [0, 1, 0, 0, 0, 0, 0, 0] 3. **unionByRank(2, 3)**    * findUPar(2) returns 1 (after path compression).    * findUPar(3) returns 3.    * rankVec[1] (1) > rankVec[3] (0), so parent[3] = 1.    * Updated:      + parent = [0, 1, 1, 1, 4, 5, 6, 7]      + rankVec = [0, 1, 0, 0, 0, 0, 0, 0] 4. **unionByRank(4, 5)**    * findUPar(4) returns 4.    * findUPar(5) returns 5.    * rankVec[4] (0) < rankVec[5] (0), so parent[5] = 4.    * Updated:      + parent = [0, 1, 1, 1, 4, 4, 6, 7]      + rankVec = [0, 1, 0, 0, 1, 0, 0, 0] 5. **unionByRank(6, 7)**    * findUPar(6) returns 6.    * findUPar(7) returns 7.    * rankVec[6] (0) < rankVec[7] (0), so parent[7] = 6.    * Updated:      + parent = [0, 1, 1, 1, 4, 4, 6, 6]      + rankVec = [0, 1, 0, 0, 1, 0, 1, 0] 6. **unionByRank(5, 6)**    * findUPar(5) returns 4 (path compression for 5).    * findUPar(6) returns 6.    * rankVec[4] (1) > rankVec[6] (1), so parent[6] = 4.    * Updated:      + parent = [0, 1, 1, 1, 4, 4, 4, 6]      + rankVec = [0, 1, 0, 0, 2, 0, 0, 0] 7. **Checking if 3 and 7 are in the same set**    * findUPar(3) returns 1.    * findUPar(7) returns 6 (path compression for 7 → 6 → 4).    * They are not in the same set, so it prints "Not same". 8. **unionByRank(3, 7)**    * findUPar(3) returns 1.    * findUPar(7) returns 4 (path compression for 7 → 6 → 4).    * rankVec[1] (1) < rankVec[4] (2), so parent[1] = 4.    * Updated:      + parent = [0, 4, 1, 1, 4, 4, 4, 4]      + rankVec = [0, 1, 0, 0, 2, 0, 0, 0] 9. **Checking if 3 and 7 are in the same set again**    * findUPar(3) returns 4 (path compression for 3 → 1 → 4).    * findUPar(7) returns 4.    * They are now in the same set, so it prints "Same".   **Final Parent and Rank Arrays:**   * parent = [0, 4, 1, 1, 4, 4, 4, 4] * rankVec = [0, 1, 0, 0, 2, 0, 0, 0] |
| **Output:-** Not same  Same | |