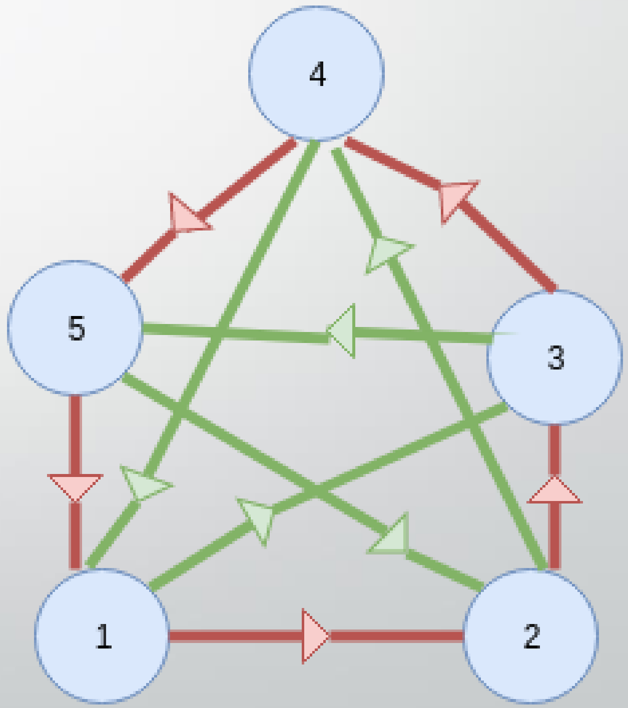
Determination of Edge-Disjoint Hamiltonian Circuit

Mohd. Abdullah(ISM2014004)
Jatin Goel(ICM2014503)
Rohit Raj(ISM2014003)

Edge Disjoint Hamiltonian Circuit

- Graph shown in figure is a complete graph of 5 vertices.
- It has 24 unique Hamiltonian Circuits
- One of the pair of edge-disjoint Hamiltonian Circuit are:
  - Cycle 1 – 5 1 2 3 4 5
  - Cycle 2 – 5 2 4 1 3 5



Approach 1

```
Edge_disjoint_hamiltonian( )
{
        //Initialize path vector
        for i=1 to V
                path[i] = i

        //Generate all Permutation
        while(next_unique_permutation(path))
        {
                // To check given path vector is hamiltonian circuit or not
                check_hamiltonian(path)
                if(true)
                        hamiltonian_cycles.append(path)
        }
        // To Check Disjoint
        for each pair in hamiltonian_cycles
        {
                check_if_disjoint();
                if(true)
                        Edge_disjoint_hamiltonian.append(pair);
        }
}
```

```
check_hamiltonian()
{
        for i=1 to V

        if(!graph[path[i]][path[(i+1)%V]])
                        return false;
        return true;
}
check_if_disjoint()
{
        if(cycle pair have common edges)
                return false;
        return true;
}
Next_unique_permutation
{
        check if next_permutation is
        rotation of one of the previous
        permutation
}
```

Approach 2

```
backtrack(current_node,taken) {
        if taken == no_of_nodes
                e = edge(current_node,start_node)
                if e == -1
                        return
                edge_bitset.set(e)
                all_circuit.insert(circuit,edge_bitset)
                edge_bitset.unset(e)
                return
        for v : adjacency_list[current_node]
                if visited[v.node] is false
                        taken + = 1
                        circuit.insert(v.node)
                        edge_bitset.set(v.edge)
                        visited[v.node] = true
                        backtrack(v.node)
                        taken - = 1
                        visited[v.node] = false
                        circuit.pop()
                        edge_bitset.unset(v.edge)
}
```

```
main() {
        circuit.insert(1)
        taken = 1
        start_node = 1
        visited[1] = true
        backtrack(1)
        // Find all pairs of edge disjoint Hamiltonian circuit
        for(i ← 0 to all_circuit.size())
           for(j ← i+1 to all_circuit.size())
                if(((all_circuit[i].edge_bitset & all_circuit[j].edge_bitset).count()) == 0)
                        print(i,j)
}
```

```
Bitset B[]
{
        B[i]=1: i^th edge is present in circuit
        B[i]=0: i^th edge is not present in circuit
}
```

Time Complexity Analysis

Approach 1

$O((N+N\hat{}3logN)*N!) + O(\ T^2N\ \textit{log}\ N)$

Approach 2

$O((N+E)*(N-1)!) + O(|T^2 *E/32|)$

This approach works well for sparse graph and is a more efficient way to check edge-disjoint Hamiltonian pair.

Notation

N – No of Nodes
E – No of Edges
T – No of Hamiltonian Cycles

Result Comparison

Comparison Between Approach 1 Vs Approach 2 on Complete Graph

No of Nodes	Approach 1(sec.)	Approach 2(sec.)
4	0.002664	0.000036
5	0.045459	0.000087
6	1.14136	0.000643
7	14.5678	0.016820
8	---	0.761507
9	---	47.201474

Comparison Between Approach 1 Vs Approach 2 on Sparse Graph

No of Nodes	Approach 1(sec.)	Approach 2(sec.)
4	0.000195	0.000099
5	0.000235	0.000071
6	0.000337	0.000188
7	0.009584	0.001871
8	0.018894	0.186956
9	0.139235	0.009599
10	3.00345	0.000361