Appendix
AI Programming

Appendix
AI Programming

Search

In the following we give the C# code of NODE and BFS classes.

```
public class Node
{
     public ArrayList state = new ArrayList ();
     public int parent;
     public int depth;
     public int cost;
     public int order;
     public int distanceFromInitial;
     public int distanceToGoal;

     public Node( ) {}

     public Node(ArrayList state)
     {
          this.state =state;
     }

    public Node(ArrayList s,int p,int d,int c,int o)
     {
          this.state =s;
          this.parent =p;
          this.depth =d;
          this.cost =c;
          this.order=o;
     }
}
```

```
public class BFS
{
     public delegate Queue Expand(Node node);
     Stack Solution = new Stack ();
     Node initial = new Node ();
     Node goal = new Node ();

     Queue fringe = new Queue ();
     ArrayList visited = new ArrayList ();

     public BFS(object initial, object goal)
     {
          this.initial=(Node)initial;
          this.goal = (Node)goal;
     }

     public Stack solve(Expand getSuccessor)
     {
          if (treeSearch(getSuccessor))
               getPath(this.goal);
          return Solution ;
     }

     public bool treeSearch(Expand getSuccessor)
     {
          Queue comingSucc = new Queue ();
          Node tempNode = new Node ();
          fringe.Enqueue(initial);

          while(true)
          {
          if (fringe.Count == 0 )
               return false;
          tempNode = (Node) fringe.Dequeue();
          visited.Add (tempNode);

          if (testGoal(tempNode))
               return true;

          comingSucc = getSuccessor(tempNode);
          while (comingSucc.Count !=0)
               fringe.Enqueue(comingSucc.Dequeue());
          }
     }
```

```
      public void getPath(Node c)
      {
            Solution.Push(c);
            while(c.parent > 0)
            {
                  IEnumerator ve =
                     visited.GetEnumerator (c.parent,1);
                  visitedEnum.MoveNext ();
                  c = (Node)ve.Current;
                  Solution.Push (c);
            }
      }

      public bool testGoal(Node node)
      {
            if ( goal.order != node.order)
                  return false;

            goal = node ;
            return true;
      }
} // end class BFS
```

Class **DFS** is the same as **BFS**, but with changing the fringe to be stack.

```
Stack fringe = new Stack ();
```

And using `push()` and `pop()` instead of `enqueue()` and `dequeue()`. Changes will affect the highlighted lines.