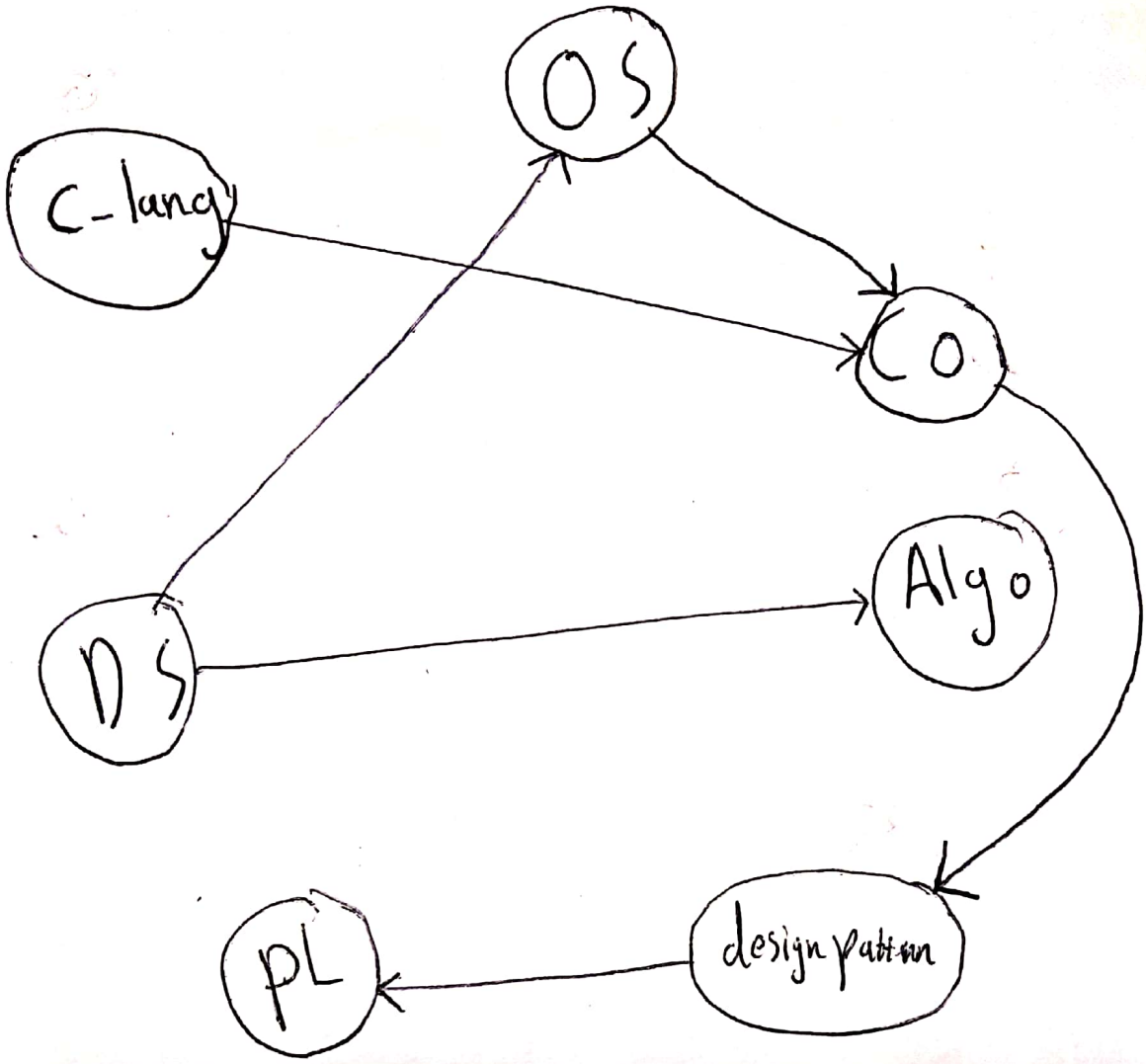
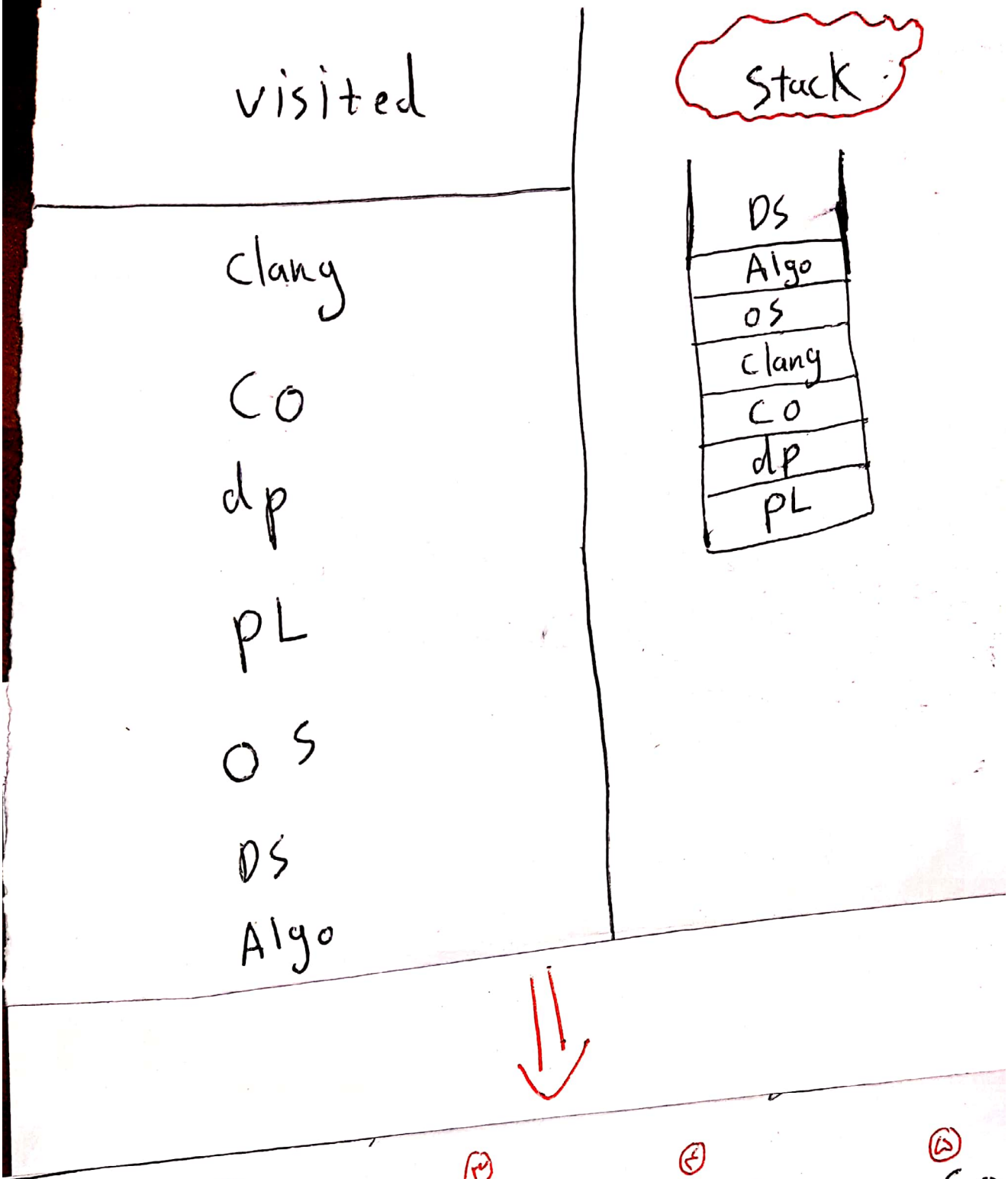
گراف حاصل :



ابتدا با استفاده از متودی که در صفحه (۳) (برای توضیح بیشتر) آمده ، چک می‌کنیم که گراف حاصل ما حتماً DAG باشد تا بتوان از top sort استفاده کرد ، سپس با استفاده از متودی که در صفحهٔ (۴) آمده ، Stack حاصل نشان دهندهٔ ترتیبی است که "علی" باید درس هایش را انتخاب کند !

Scanned by CamScanner



① DS → ② Algo → ③ OS → ④ C_lang → ⑤ CO

→ ⑥ design_pattern → ⑦ PL

« ترتیب انتخاب دروس »

Scanned by CamScanner

```go
//////////// function to check if a directed gragh is cyclic or not , this algorithm is based on DFS
func (g *Graph) IsCyclic() bool {
  // Mark all the vertices as not visited and not part of recursion
  // stack
  visited := make(map[*Vertex]bool)
  recStack := make(map[*Vertex]bool)
  for _, v := range g.Vertices {
    visited[v] = false
    recStack[v] = false
  }
  // Call the recursive helper function to detect cycle in different
  // DFS trees
  for _, v := range g.Vertices {
    if g.isCyclicUtil(v, visited, recStack) {
      return true
    }
  }
  return false
}

func (g *Graph) isCyclicUtil(v *Vertex, visited map[*Vertex]bool, recStack map[*Vertex]bool) bool {
  if !visited[v] {
    // Mark the current node as visited and part of recursion stack
    visited[v] = true
    recStack[v] = true
    // Recur for all the vertices adjacent to this vertex
    temp := g.AdjacenyList[v]
    for temp != nil {
      if temp.StartPoint == v {
        if !visited[temp.EndPoint] && g.isCyclicUtil(temp.EndPoint, visited, recStack) {
          return true
        } else if recStack[temp.EndPoint] {
          return true
        }
      }
      temp = temp.Next
    }
  }
  recStack[v] = false // remove the vertex from recursion stack
  return false
}
```

Scanned by CamScanner

```go
func (g *Graph) TopologicalSort(v *Vertex, visited map[*Vertex]bool, stack *Stack) {
  visited[v] = true
  // Recur for all the vertices
  // adjacent to this vertex
  temp := g.AdjacenyList[v]
  for temp != nil {
    if temp.StartPoint == v {
      if !visited[temp.EndPoint] {
        g.TopologicalSort(temp.EndPoint, visited, stack)
      }
    }
    temp = temp.Next
  }
  // Push current vertex to stack
  // which stores result
  stack.Push(v)
}

func (g *Graph) GetTopList() *Stack {
  stack := CreateStack()
  visited := make(map[*Vertex]bool)
  for _, v := range g.Vertices {
    visited[v] = false
  }
  // Call the recursive helper function
  // to store Topological
  // Sort starting from all
  // vertices one by one
  for _, v := range g.Vertices {
    if !visited[v] {
      g.TopologicalSort(v, visited, stack)
    }
  }
  return stack
}
```

Scanned by CamScanner