III B.Tech.

Computer Science & Engineering

CSE304: PYTHON PROGRAMMING WITH WEB FRAMEWORKS

UNIT-II: Recursion and Algorithms

By
Mrs. S. KAMAKSHI, AP-III / CSE
School of Computing

Recursion

- Direct Recursion: Call of a function by itself
- Indirect Recursion: A function f1 calling a function f2 which in turn calls f1
  - Nested to any depth
- Recursion vs. Iteration
  - In many cases iterative version out performs its recursive counterpart in terms of time and space complexities

Recursion vs. Iteration

Iterative Function

```
def add_Iter(*numbers):
   sum = 0
   for x in numbers:
      sum += x
   return sum
print(add_Iter(24, 35, 11, 56, 29, 87, 34))
```

Recursive Function

```
def add_Rec(*numbers):
   if len(numbers)==1:
      return numbers[0]
   else:
      return numbers[0] + add_Rec(*numbers[1:])
print(add_Rec(24, 35, 11, 56, 29, 87, 34))
```

```
def fact_Iter(n):
   f = 1
   for i in range(1,n+1):
      f *= i
   return f
print(fact_Iter(10))
```

```
def fact_Rec(n):
   if n==1:
      return 1
   else:
      return n * fact_Rec(n-1)
print(fact_Rec(10))
```

Recursion vs. Iteration

```
def fib_Iter(n):
    f0 = 0
    f1 = 1
    for i in range(3, n+1):
        f0, f1 = f1, f0 + f1
    return f1
print(fib_Iter(20))
```

```
def fib_Rec(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fib_Rec(n-1) +
                          fib_Rec(n-2)
print(fib_Rec(20))
```

Binary Tree Traversal

```
tree_edges = {'Root': 'A','A':{'left':'B','right':'C'}, 'B':{'left':'D', 'right':'E'},
                         'C':{'left':None,'right':'F'}, 'D':{'left':'G','right':'H'},
                         'E':{'left':'I','right':None}, 'F':{'left':'J','right':'K'} }
def inorder(node):
   if node:
      if tree_edges.get(node, None):
         inorder(tree_edges[node].get('left',None))
      print(node, end=' ')
      if tree_edges.get(node, None):
         inorder(tree_edges[node].get('right', None))
inorder(tree_edges['Root'])
```

Binary Tree Traversal

```
tree_edges = {'Root': 'A','A':{'left':'B','right':'C'}, 'B':{'left':'D', 'right':'E'},
                               'C':{'left':None,'right':'F'}, 'D':{'left':'G','right':'H'},
                               'E':{'left':'I','right':None}, 'F':{'left':'J','right':'K'} }
def preorder(node):
    if node:
        print(node, end=' ')
        if tree_edges.get(node, None):
            preorder(tree_edges[node].get('left',None))
        if tree_edges.get(node, None):
            preorder(tree_edges[node].get('right', None))
preorder(tree_edges['Root'])
```

Binary Tree Traversal

```
tree_edges = {'Root': 'A','A':{'left':'B','right':'C'}, 'B':{'left':'D', 'right':'E'},
              'C':{'left':None,'right':'F'}, 'D':{'left':'G','right':'H'},
              'E':{'left':'I','right':None}, 'F':{'left':'J','right':'K'} }
def postorder(node):
    if node:
        if tree_edges.get(node, None):
            postorder(tree_edges[node].get('left',None))
        if tree_edges.get(node, None):
            postorder(tree_edges[node].get('right', None))
        print(node, end=' ')
postorder(tree_edges['Root'])
```

Merge Sort

```
def mergesort(A, low, high):
    if low < high:
        mid = (low + high)//2
        mergesort(A, low, mid)
        mergesort(A, mid+1, high)
        merge(A, low, mid, high)
```

Merge Sort

```
def merge(A, low, mid, high):
    i, j, k = low, mid+1, 0
    T = [0]*(high-low+1)
    while i<=mid and j<=high:
        if A[i] < A[j]:
            T[k] = A[i]
            i += 1
        else:
            T[k] = A[j]
            j += 1
        k += 1
    if i > mid:
        T[k:high-low+1] = A[j:high+1]
    else:
        T[k:high-low+1] = A[i:mid+1]
    A[low:high+1] = T[0:high-low+1]
```

Merge Sort

```
L = [41, 35, 53, 21, 62, 11, 98, 44, 73]
mergesort(L, 0, len(L)-1)
print(L)
```

Breadth First Search

```
G = {'A':['B', 'C', 'D'], 'B':['A', 'D'],
     'C':['A', 'D'], 'D':['A', 'B', 'C']}

def BFS(G, s):
   keys = G.keys()
   parent = dict.fromkeys(keys, None)
   color = dict.fromkeys(keys, 'WHITE')
   dist = dict.fromkeys(keys, 9999999)
   Q = [s]
   dist[s] = 0
   color[s] = 'GRAY'
   while Q:
      u = Q.pop(0)
      for v in G.get(u, None):
         if color[v] == 'WHITE':
            dist[v] = dist[u] + 1
            parent[v] = u
            color[v] = 'GRAY'
            Q.append(v)
      color[u] =  'BLACK'
   return parent, dist
P, D = BFS(G, 'A')
for v in G:
   print (v, '  ', P[v], '  ', D[v])
```

Depth First Search

```
G = {'A':['B', 'C', 'D'], 'B':['A', 'D'],
    'C':['A', 'D'], 'D':['A', 'B', 'C']}

def DFS(G):
    keys = G.keys()
    global time, d, f, parent, color
    time = 0
    color = dict.fromkeys(keys, 'WHITE')
    d = dict.fromkeys(keys, 0)
    f = dict.fromkeys(keys, 0)
    parent = dict.fromkeys(keys, None)
    for u in G:
        if color[u] == 'WHITE':
            DFS_Visit(u)
    return parent, d, f

def DFS_Visit(u):
    global d, f, time, parent, color
    time += 1
    d[u] = time
    color[u] = 'GRAY'
    for v in G.get(u, None):
        if color[v] == 'WHITE':
            parent[v] = u
            DFS_Visit(v)
    time += 1
    f[u] = time
    color[u] = 'BLACK'
P, D, F = DFS(G)
for v in G:
    print (v, '  ', P[v], '  ', D[v], '  ', F[v])
```

Prim's Algorithm for finding MWST

```
G = {'A':[('B', 3), ('C', 1), ('D', 5)], 'B':[('A', 3), ('D', 6)],
     'C':[('A', 1), ('D', 2)], 'D':[('A', 5), ('B', 6), ('C', 2)]}

def extract_min(Q):
   v = min(Q, key=Q.get)
   Q.pop(v)
   return v

def PRIMS(G,s):
   keys = G.keys()
   parent = dict.fromkeys(keys, None)
   visited = dict.fromkeys(keys, False)
   dist = dict.fromkeys(keys, 9999999)
   Q = dict.fromkeys(keys, 9999999)
   Q[s] = 0
   dist[s] = 0
   visited[s] = True
   parent[s] = None
   cost = 0
   print ("Edges of Minimum Weight Spanning Tree")
   while Q:
      u = extract_min(Q)
      visited[u] = True
      if not u == s:
         print (parent[u], "-->", u)
         cost += dist[u]
      for (v, edg_w) in G.get(u, None):
         if not visited[v] and dist[v] > edg_w:
            dist[v] = edg_w
            Q[v] = edg_w
            parent[v] = u
   print("Total Weight: ", cost)

PRIMS(G, 'A')
```

Kruskal's Algorithm for finding MWST

```
G = [('A', 'B', 3), ('A', 'C', 1), ('A', 'D', 5),  ('B', 'D', 6),
          ('C', 'D', 2)]

def edg_w(t):
   return t[2]

def find_set(S, v):
   for i in range(0, len(S)):
      if v in S[i]:
         return I

def union(S, i, j):
   S[i] = S[i] | S[j]
   S.remove(S[j])
```

```
def KRUSKAL(G):
   G = sorted(G, key = edg_w)
   V = set({})
   for x,y,z in G:
      V = V | {x} | {y}
   S = []
   for u in V:
      S.append({u})
   cost = 0
   print ("Edges of Minimum Weight Spanning Tree")
   for (u, v, weight) in G:
      i = find_set(S, u)
      j = find_set(S, v)
      if not (i == j):
         print (u, "-->", v)
         cost += weight
         union(S, i, j)
   print("Total Weight: ",cost)

KRUSKAL(G)
```