最良優先探索のアルゴリズム

① **OPEN**＝(出発節点 S)
// 初期状態を表す出発節点を OPEN リストに入れる
CLOSED＝{ }；
// **CLOSED** リストは調べ終わった節点を入れる集合

② もし，**OPEN** が空ならば，終了［失敗］
// 目標節点がない

③ p:＝先頭要素（**OPEN**）
// OPEN の先頭要素を p とする［選択］

④ もし，p が目標節点 G ならば，終了［成功］
// p が目標節点である

⑤ p を展開；p を OPEN から削除し，**CLOSED** に格納
// p は調査済み

⑥ 子節点 q が **OPEN** にも **CLOSED** にも含まれない場合
q を OPEN に入れる　q から p へのポインタを付加

⑦ **OPEN** の節点の評価値 $h^*(p)$ によるソート（昇順）

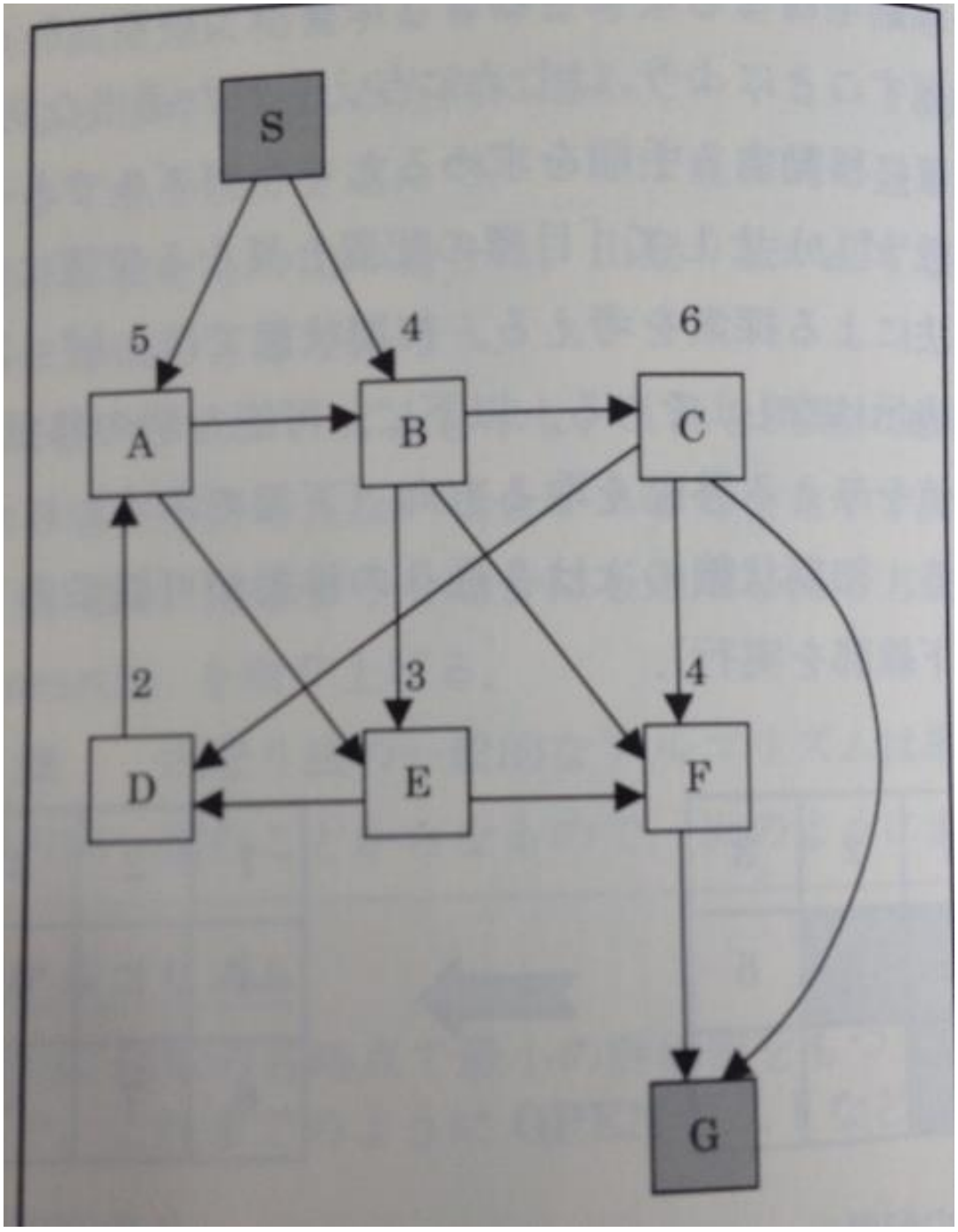
⑧ ②へ戻る
// ②～⑦はループをなす

```
# 資料にある欲張り探索アルゴリズムの実装プログラム
def GS(graph, h, start, goal):
    C = [] # ClosedList
    O = [] # OpenList
    initPath = [start] # Path init
    O.append(start)
    pathPriorityQueue = [initPath] # List of paths
    while len(O) != 0:
        p = O.pop(0) # p is removed from the list O
        tmpPath = pathPriorityQueue.pop(0)
        if p == goal:
            print("Goal! GS Search is over!")
            path=[goal]
            tmpPath.reverse()
            for node in tmpPath:
                child_node = path[-1]
                if graph.has_key(node) and child_node in graph[node]:
                    path = path + [node]
            path.reverse()
            return path
        else:
            P_A = graph[p]
            C.append(p)
            if P_A == []:
                print("No Solution. Path empty.")
                return None
            for q in P_A:
                if q not in O and q not in C:
                    O.append(q)
            sortedO = []
            print ("Open :", O)
            print ("Closed:",C)
            for n in O:
                sortedO.append((n,h[n]))
            sortedO = sorted(sortedO, key=lambda x:x[1])
            O=[]
            for node, n_h in sortedO:
                O.append(node)
            print ("Open after sort: ", O)
            newPath = tmpPath + [O[0]]
            pathPriorityQueue.append(newPath)
            print "path PQ: ",pathPriorityQueue
    return None
graph1={'S':{'A','B'},'A':{'E','B'},'B':{'C','E','F'},
        'C':{'F','G'},'D':{'A'},'E':{'D','F'},'F':{'G'}}

graph1_h = {'S':10,'A':5,'B':4,'C':6,'D':2,'E':3,'F':4,'G':0}

print "GS for graph1:"
print GS(graph1,graph1_h,"S","G")
```

```
GS for graph1:
('Open :', ['A', 'B'])
('Closed:', ['S'])
('Open after sort: ', ['B', 'A'])
path PQ:  [['S', 'B']]
('Open :', ['A', 'C', 'E', 'F'])
('Closed:', ['S', 'B'])
('Open after sort: ', ['E', 'F', 'A', 'C'])
path PQ:  [['S', 'B', 'E']]
('Open :', ['F', 'A', 'C', 'D'])
('Closed:', ['S', 'B', 'E'])
('Open after sort: ', ['D', 'F', 'A', 'C'])
path PQ:  [['S', 'B', 'E', 'D']]
('Open :', ['F', 'A', 'C'])
('Closed:', ['S', 'B', 'E', 'D'])
('Open after sort: ', ['F', 'A', 'C'])
path PQ:  [['S', 'B', 'E', 'D', 'F']]
('Open :', ['A', 'C', 'G'])
('Closed:', ['S', 'B', 'E', 'D', 'F'])
('Open after sort: ', ['G', 'A', 'C'])
path PQ:  [['S', 'B', 'E', 'D', 'F', 'G']]
Goal! GS Search is over!
['S', 'B', 'E', 'F', 'G']
>>>
```


S
5
4
6
A
B
C
2
3
4
D
E
F
G