习题2-1

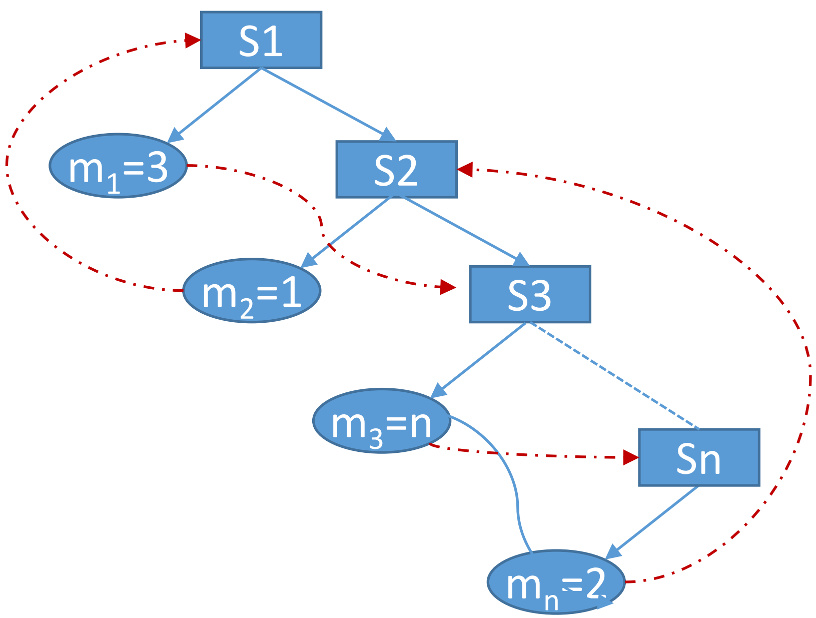
DH第4章练习1、2、8、9、11、12、13、14

4.1. Consider the problem of summing the salaries of employees earning more than their direct manager, assuming each employee has a single manager. The employees are labeled 1, 2, etc. Write algorithms that solve the problem for each of the following representations of the input data:

(a) The input is given by an integer N and a two-dimensional array A, where N is the number of employees, $A[I, 1]$ is the salary of the Ith employee and $A[I, 2]$ is the label of his or her manager.

(b) The input is given by a binary tree constructed as follows: The root of the tree represents the first employee. For every node V of the tree representing the Ith employee,

- V contains the salary of the Ith employee;
- the first offspring of V is a leaf containing the label of the manager of the Ith employee; and
- if there are more than I employees, the second offspring of V is the node that represents the $I + 1$th employee.



思路1:

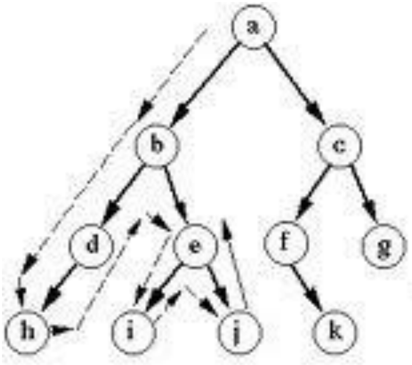
- 遍历一遍树将其转化为（a）；
- 利用（a）求解

思路2:

- 定义函数get_salary_of(Node root, int i)：
  - 沿着second offspring 关系查找以root为根的子树中第i个node所存的salary
- 定义主函数Main(Node root)计算结果：
  - Sum=0
  - Node cnode=root;
  - While(cnode!=null){
    - ms=get_salary_of(root,cnode.first.label);
    - If(cnode.salary>ms)sum+=cnode.salary;
  - }
  - Return sum;

- 4.2.
  - (a) Write an algorithm which, given **a tree *T***, calculates the sum of the depths of all the nodes of *T*
  - (b) Write an algorithm which, given **a tree *T*** and a positive integer *K*, calculates the number of nodes in *T* at depth *K*.
  - (c) Write an algorithm which, given **a tree *T***, checks whether it has any leaf at an even depth.

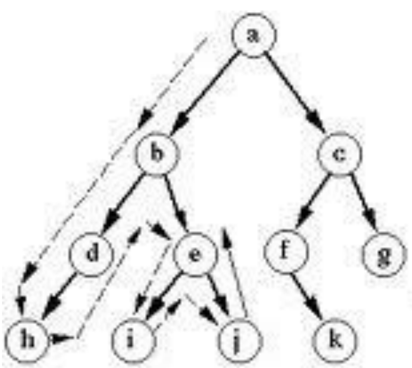
```
DFS-VISIT(G, u):
    Preorder processing of u;
    for each v child of u
        Processing of edge uv(1);
        DFS-Visit(G,v);
        Processing of edge uv(2);
    Postorder processing of u;
```



Depth-first search

- 4.2.
  - (a) Write an algorithm which, given **a tree *T***, calculates the sum of the depths of all the nodes of *T*
  - (b) Write an algorithm which, given **a tree *T*** and a positive integer *K*, calculates the number of nodes in *T* at depth *K*.
  - (c) Write an algorithm which, given **a tree *T***, checks whether it has any leaf at an even depth.

```
DFS-VISIT(T, u):
    Preorder processing of u;
    for each v child of u
        Processing of edge uv(1);
        DFS-Visit(T,v);
        Processing of edge uv(2);
    Postorder processing of u;
```



Depth-first search

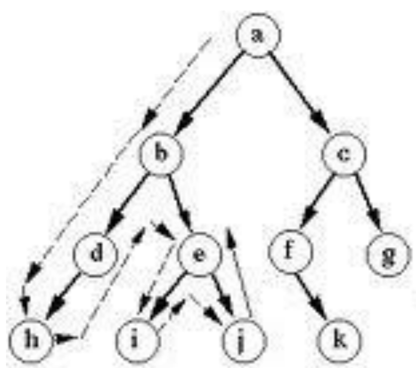
(a)

- Input: T-a tree
- Output: the sum of the depths of all nodes of T

```
sum=0;
DFS-VISIT(T, u, d):
    sum+=d;
    for each v child of u
        d'=d+1;
        DFS-Visit(T,v,d');

主函数：
DFS-Visit(T,T.root,0)
```

- 4.2.
  - (a) Write an algorithm which, given **a tree *T***, calculates the sum of the depths of all the nodes of *T*
  - (b) Write an algorithm which, given **a tree *T*** and a positive integer *K*, calculates the number of nodes in *T* at depth *K*.
  - (c) Write an algorithm which, given **a tree *T***, checks whether it has any leaf at an even depth.

```
DFS-VISIT(T, u):
    Preorder processing of u;
    for each v child of u
        Processing of edge uv(1);
        DFS-Visit(T,v);
        Processing of edge uv(2);
    Postorder processing of u;
```



Depth-first search

(b)

- Input:
  - T-a tree,
  - k- a positive integer
- Output: the number of nodes in T at depth k.

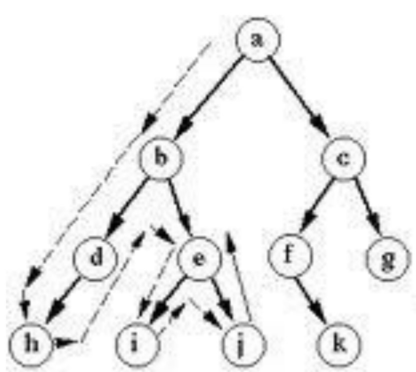
```
DFS-VISIT(T, u, d, k):
    if(d==k) sum++;
    for each v child of u
        d'=d+1;
        DFS-Visit(T,v,d',k);
```

主函数：

```
Sum=0;
DFS-Visit(T,T.root,0,k)
```

- 4.2.
  - (a) Write an algorithm which, given **a tree *T***, calculates the sum of the depths of all the nodes of *T*
  - (b) Write an algorithm which, given **a tree *T*** and a positive integer *K*, calculates the number of nodes in *T* at depth *K*.
  - (c) Write an algorithm which, given **a tree *T***, checks whether it has any leaf at an even depth.

```
DFS-VISIT(T, u):
    Preorder processing of u;
    for each v child of u
        Processing of edge uv(1);
        DFS-Visit(T,v);
        Processing of edge uv(2);
    Postorder processing of u;
```



Depth-first search

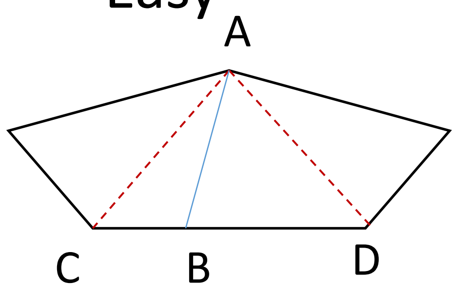
(c)

- Input:
  - T-a tree,
- Output:
  - True-if T has any leaf at an even depth

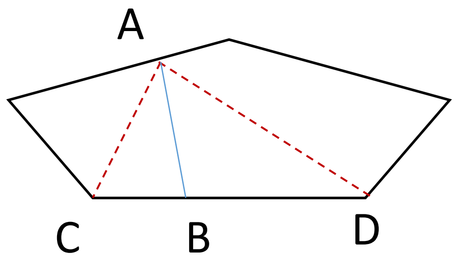
```
DFS-VISIT(T, u, d):
    if(d%2==0&&u is a leaf)
        return true;
    for each v child of u
        d'=d+1;
        if(DFS-Visit(T,v,d'))
            return true;

主函数：
DFS-Visit(T,root,0,)
```

4.8. Prove that the maximal distance between any two points on a polygon occurs between two of the vertices.

- 分情况：
  - Case1: Two point on the same line——Easy
  - Case2:Two point on different line



    - 2.1:one vertex, one non-vertex
      - It is easy to show that
        - AB<AC or AB<AD
        - In another word, for each two points of case 2.1, we can always find two vertices with longer distance.



    - 2.2:two non-vertex
      - It is easy to show that
        - AB<AC or AB<AD
      - AC or AD are of case 2.1, and with the conclusion of case 2.1, we conclude that- for each two points of case 2.2, we can always find two vertices with longer distance.
    - 2.3: two vertexes
      - trivial

4.9. Write a program implementing the maximal polygonal distance algorithm.

- Specification:
  - Input: a simple convex polygon P of n points, with P[i] denotes the i-th point of P in clockwise order, P[i].x and P[i].y denote the coordinate of the i-th point(i=0,…,n-1)
  - Output: a simple integer indicating the maximal Polygoal Distance of P;

1) Let V[i] be the vector obtained by $\langle P[(i+1)\%n].x - P[i].x, P[(i+1)\%n].y - P[i].y\rangle$
2) Let line[i] be the line determined by P[i] and P[(i+1)%n]
3) Let $cl \leftarrow line[0]$; $cv \leftarrow V[0]$;
4) **Find one point p, which has the longest distance to cl;**
5) $\max \leftarrow 0$;
6) for(t $\leftarrow$ 0;t<n;t++){
6.1) tdis $\leftarrow \max\big(dis(P[p], P[cl.v1]), dis(P[p], P[cl.v2])\big)$
6.2) if(tdis>max) max $\leftarrow$tdis;
6.3) ang1 $\leftarrow$angle between cv and V[$cl.v2$];
6.4) ang2 $\leftarrow$angle between $-cv$ and V[p];
6.5) if(ang1<ang2){
6.5.1) $cl \leftarrow line[cl.v2]$; $cv \leftarrow V[cl.v2]$; $p \leftarrow P[(p+1)\%n]$;
6.5.2) }else{
6.5.3) $cl \leftarrow line[p]$; $cv \leftarrow V[p]$; $p \leftarrow P[(cl.v2)\%n]$
6.5.4) }
6.6) }
7) return max;

4.11. Write algorithms that find the two maximal elements in a given vector of N distinct integers (assume $N > 1$).

(a) Using an iterative method.
(b) Using the divide-and-conquer method.

(a)

思路1：

```
m=find_max(V);
sm=find_max(V-max);
```

思路2：

```
m=max(V[1],V[2]);
sm=min(V[1],V[2]);
for(i=3;i<=n;i++){
    if(V[i]>m){
        sm=m;
        m=V[i];
    }else if(V[i]>sm) sm=V[i];
}
```

4.12. Write in detail the greedy algorithm described in the text for finding a minimal spanning tree.

- **Prim**
  - 基本思想：
    - 两类节点集合：
      - V_{MST}:已被纳入MST的节点，初始为∅
      - $V_{\text{Non}-MST}$:尚未被纳入MST的节点，初始为所有节点
    - 随机从$V_{\text{Non}-MST}$中选取并删除一个点v放入V_{MST}
    - While($V_{\text{Non}-MST} \neq \emptyset$) do
      - 选取权值最小的边 $uz \in \{ab|ab \in E, a \in V_{MST}, b \in V_{\text{Non}-MST}\}$
      - $V_{\text{Non}-MST} \leftarrow V_{\text{Non}-MST} - b$;
      - $V_{MST} \leftarrow V_{MST} + b$;
      - $MST \leftarrow MST + ab$
- **Kruskal ?**

capacity with some elements of a given set of available items of various types in the most profitable way. The input to the problem consists of:

- *C*, the total weight capacity of the knapsack;
- a positive integer *N*, the number of item types;
- a vector *Q*, where *Q*[*I*] is the available number of items of type *I*;
- a vector *W*, where *W*[*I*] is the weight of each item of type *I*, satisfying 0 < *W*[*I*] ≤ *C*;and
- a vector *P*, where *P*[*I*] is the profit gained by storing an item of type *I* in the knapsack.

All input values are non-negative integers. The problem is to fill the knapsack with elements whose total weight does not exceed *C*, such that the total profit of the knapsack is maximal. The output is a vector *F*, where *F*[*I*] contains the number of items of type *I* that are put into the knapsack.

4.13. (a) Design a dynamic planning algorithm for the integer-knapsack problem.

(b) What is your algorithm's output for the input

- $N = 5$
- $C = 103$
- $Q = [3,1,4,5,1]$
- $W = [10,20,20,8,7]$
- $P = [17,42,35,16,15]$

and what is the total profit of the knapsack?

S[i][k]=当容量为k,仅使用0~i类物品，所能得到的最高profit

$$S[i][k] = \max_{j=0\sim\lfloor k/w[i]\rfloor} \{S[i-1][k - j*w[i]] + j*p[i]\}$$

4.14. (a) Design a greedy algorithm for the knapsack problem.
(b) What is your algorithm's output for the input given in Exercise 4.13(b), and what is the total profit of the knapsack now?

The **knapsack** problem is a variation of the integer-knapsack problem, in which instead of discrete items, there are materials. The difference is that instead of working with integer numbers, we may put into the knapsack **any *quantity* of material *I*** which does not exceed the available quantity *Q*[*I*]. The vectors *W* and *P* now contain the weight and profit, respectively, of one quantity unit of material *I*. **All input and output values are now non-negative real numbers, not necessarily integers.**

```
Profits=0;
for( i=0;i<n;i++) {
   up[i].unitprofit=p[i]/w[i];//计算单位总量的利润
   up[i].index=I;
}
reorder up[] decreasingly with respects to up[i].unitprofit
for(t=0;t<n;t++) do
   K=min(C, w[up[t].index]*q[up[t].index]);
   profits+=K*up[t].unitprofit;
   C-=K;
return profits;
```