Homework

Problems on DH

4.1

(a)

```
(1) do the following for I from 1 to N:
    (1.1) if A[I,2] is empty, I = I + 1;
    (1.2) otherwise do the following:
        (1.2.1) manager = A[I,2];
        (1.2.2) compare A[I,1] and A[manager,1];
        (1.2.3) if A[I,1] > A[manager,1] then do the following:
            (1.2.3.1) total = total + A[I,1];
        (1.2.4) I = I + 1;
(2) output total;
(3) stop;
```

(b)

```
(1) point to the root R of the tree;
(2) now_depth = 0;
(3) while the pointed node has second offspring, do the following:
    (3.1) if the pointed node has first offspring, do the following:
        (3.1.1) salary = the content of the pointed node;
        (3.1.2) manager = the content of the first offspring of the pointed node;
        (3.1.3) if manager > now_depth + 1, then do the following:
            (3.1.3.1) while the depth of the pointed node doesn't equal to manager - 1,
            point to the second offspring of the pointed node;
            (3.1.3.2) compare the content of the pointed node and salary;
            (3.1.3.3) if salary > the content of the pointed node, then
             total = total + salary;
            (3.1.3.4) while the depth of the pointed node dosen't equal to now_depth,
            point to the previous node of the pointed node;
        (3.1.4) otherwise do the following:
            (3.1.4.1) while the depth of the pointed node doesn't equal to manager - 1,
            point to the previous node of the pointed node;
            (3.1.4.2) compare the content of the pointed node and salary;
            (3.1.4.3) if salary > the content of the pointed node, then
            total = total + salary;
            (3.1.4.4) while the depth of the pointed node dosen't equal to now_depth,
            point to the second offspring of the pointed node;
        (3.1.5) point to the second offspring of the pointed node;
        (3.1.6) now_depth = now_depth+1;
    (3.2) otherwise do the following:
        (3.2.1) point to the second offspring of the pointed node;
        (3.2.2) now_depth = now_depth + 1;
(4) if the pointed node has first offspring, do the following:
    (4.1) salary = the content of the pointed node;
    (4.2) manager = the content of the first offspring of the pointed node;
    (4,3) while the depth of the pointed node doesn't equal to manager-1, point to the
    previous node of the pointed node;
    (4.4) compare the content of the pointed node and salary;
    (4.5) if salary > the content of the pointed node, then total = total + salary;
(5) output total;
(6) stop;
```

4.2

(a)

```
(1) point to the root of the tree;
(2) while the tree isn't empty, do the following:
    (2.1) if the pointed node has isn't a leaf, point to its "smallest" offspring
    //(namely the offspring whose label is the smallest);
    (2.2) otherwise do the following:
        (2.2.1) now_depth = the depth of the pointed node;
        (2.2.2) total = total + now_depth;
        (2.2.3) if now_depth = 0, then break the loop;
        (2.2.4) point to the previous node of the pointed node;
        (2.2.5) delete the "smallest" offspring of the pointed node;
(3) output total;
(4) stop;
```

(b)

```
(1) point to the root of the tree;
(2) while the tree isn't empty, do the following:
    (2.1) if the pointed node has isn't a leaf, point to its "smallest" offspring
    //(namely the offspring whose label is the smallest);
    (2.2) otherwise do the following:
        (2.2.1) now_depth = the depth of the pointed node;
        (2.2.2) if now_depth = K, then total = total + 1;
        (2.2.3) if now_depth = 0, then break the loop;
        (2.2.4) point to the previous node of the pointed node;
        (2.2.5) delete the "smallest" offspring of the pointed node;
(3) output total;
(4) stop;
```

(c)

```
subroutine check-even-leaf-of T;
(1) point to the root of the tree T;
(2) if the pointed node has no offsprings, then do the following:
    (2.1) if the depth of the pointed node is even, then flag = true;
(3) N = the outdegree of the pointed node;
(4) do the following for i from 1 to N:
    (4.1) call check-even-leaf-of i(T)//(i(T) means the i_th subtree of T);
(5) if flag = true, then output "Yes";
```

4.8

Proof.

Suppose the maximal distance d occurs between two points p_1, p_2 that neither is vertex. It is obvious that p_1, p_2 occur on two different sides of the polygon respectively (if they are on one side, then the distance between the vertices of the side is bigger than d). Now draw two lines l_1, l_2 respectively through p_1 and p_2 that are both vertical to the line p_1p_2, because neither p_1 nor p_2 is vertex, so both l_1 and l_2 intersect with the polygon at more than one point. Slide l_1 and l_2 till they intersect with the polygon at only one point and name the points $p_1^{'}$ and $p_2^{'}$ respectively. We can see that the distance between $p_1^{'}$ and $p_2^{'}$ is bigger than d. Thus in this way, for any two points on the sides, we can always find two vertices whose distance is bigger than the points. So the maximal distance between any two points on a polygon occurs between two vertices.

4.11

(a)

```
(1) min = V[1];
(2) do the following for j from 1 to N:
    (2.1) if V[i] < min, then min = V[i];
(3) first_max = v[1], second_max = min;
(4) do the following for i from 1 to N:
    (4.1) if V[i] > first_max, then fisrt_max = V[i];
    (4.2) if second_max < V[i] < first_max, second_max = V[i];
(5) output first_max, second_max;
(6) stop;
```

(b)

```
subroutine find-two-maximal of V:
(1) if N is 2, then do the following:
    (1.1) compare V[1] and V[2];
    (1.2) if V[1] > V[2], then first_max = V[1], second_max = V[2];
    (1.3) otherwise first_max = V[2], second_max = V[1];
(2) otherwise do the following:
    (2.1) split V into two vectors, Vleft and Vright;//(if length of vector is odd, then
    the element in the middle would appear in both Vleft and Vright)
    (2.2) call find-two-maximal of Vleft, placing returned values in first_maxLeft and
    second_maxLeft;
    (2.3) call find-two-maximal of Vright, placing returned values in first_maxRight and
    second_maxRight;
    (2.4) set first_max to bigger of first_maxLeft and first_maxRight;
    (2.5) set second_max to bigger of the bigger of second_maxLeft and second_maxRight
    and the smaller of first_maxLeft and first_maxRight;
(3) return with first_max and second_max;
```

4.13

(a)

```
At the very first, we can transform the problem by view the I_th item as Q[I] items,
so now the problem becoming whether we should pick the item instead of how many items
we should pick. Let M be the number of all items.
(1) do the following for i from M to 1:
    (1.1) do the following for j from 1 to C:
        (1.1.1) if j < W[i], then dp[i][j] = dp[i+1][j];
        (1.1.2) otherwise do the following:
            (1.1.2.1) if dp[i+1][j] > dp[i+1][j-W[i]] + V[i],then do the following:
                (1.1.2.1.1) dp[i][j] = dp[i+1][j];
                (1.1.2.1.2) dp2[i][j] = 0;
            (1.1.2.2) otherwise do the following:
                (1.1.2.2.1) dp[i][j] = dp[i+1][j-W[i]] +V[i];
                (1.1.2.2.2) dp2[i][j] = 1;
(2) do the following for k from 1 to M-1:
    (2.1) if dp[k][C-W[k-1]] doesn't equal to dp[k+1][C-W[k-1]], then F'[k] = 1;
    //(W[0]=0)
    (2.2) otherwise F'[k] = 0;
Finally we reorganise the items, merge the same items and thus transform F' into F, the
max profit is dp[1][C].
```

(b)

```
F=[0,1,3,2,1]
```

The total profit of the knapsack is 194

4.14

(a)

```
subroutine find-the-most-valuable:
(1) least = 1;
(2) do the following for K from 1 to N:
    (2.1) if P[K] / W[K] < P[least] / W[least], then least = K;
(3) most = least;
(4) do the following for I from 1 to N:
    (4.1) if P[I] / W[I] > P[most] / W[most] and Q[I] isn't 0, then most = I;
(5) stop;
main routine:
(1) now_c = 0;
(2) while now_C <= C, do the following:
    (2.1) call find-the-most-valuable;
    (2.2) if now_C + W[most] * Q[most] <= C, then do the following:
        (1.2.1) F[most] = Q[most];
        (1.2.2) Q[most] = 0;
    (2.3) otherwise do the following:
        (1.3.1) F[most] = (C - now_C) / W[most];
        (1.3.2) Q[most] = Q[most] - C + now_c;
(3) output F;
(4) stop;
```

(b)

```
F = [0,1,1.8,5,1]
```

The total profit of the knapsack is 200.

Extra Problem

$$
\begin{array}{r}
109.75 \\
96\,\overline{)\,10536} \\
96 \\
\hline
936 \\
864 \\
\hline
720 \\
672 \\
\hline
480 \\
480 \\
\hline
0
\end{array}
$$

即 $a=4$

$b=2$