Algorithms
National Taiwan University, Fall 2022

Assignment 4

Yu-Chieh Kuo B07611039[†]

[†]Department of Information Management, National Taiwan University

Problem 1

Let K be the size of the knapsack, $S[i]$ be the size of the i-th item, and $P(n, K)$ be the solution with the number of items n and the size of the knapsack K. The algorithm can be rewrite as Algorithm 1.

Algorithm 1 A modified kanpsack problem using the *belong* flag

```
function ModifiedKnapsack(K,S,P)
    k := K, Solution := ∅
    for i := n...1 do
        if P[i,k].exist = false then
            return 0
        end if
        if P[i,k].belong = true then
            Solution ← S[i]
            k := K − S[i]
        end if
    end for
    if k ≠ 0 then
        return 0
    end if
    return Solution
end function
```

Problem 2

Denoting the solution checking $P[i-1, j].exist$ first by *solA*, and the solution checking $P[i-1, j-k_j].exist$ first by *solB*, we state *solA* $\succ$ *solB* where the symbol $\succ$ represents the performance relationship. $A \succ B$ indicates the algorithm A has a better performance than B. The performance relationship $\succ$ satisfies the completeness and the transitivity.

Adopting *solB* might require additional $if - else$ conditions to complete the algorithm. In the iteration process, *solB* checks whether $k - S[i] \geq 0$ then $P[i, k - S[i]].exist$. If both examinations fails, the algorithm finally checks $P[i-1, k].exist$. However, there might be the more or even repeated procedures, which leads to be more time-consuming to this modification. A rough procedure is described in Algorithm 2.

Algorithm 2 Partial steps in the modified kanpsack

```
function PartialKnapsackStep(S,K)
    In the double for loop
    if k − S[i] ≥ 0 then
        if P[i, k − S[i]].exist then
            if P[i, k − S[i]].belong then
                Do something. Might be repeated and more steps.
            else if P[i − 1, k].exist then
                Do something. Might be repeated and more steps.
            end if
        end if
    else if P[i − 1, k].exist then
        Do something
    end if
end function
```

Problem 3

Given a set S of n item where the i-th item has an interger size $S[i]$ and an integer K, to solve a variation of the knapsack problem where each item has an unlimited supply, the knapsack algorithm can be represented as Algorithm 3.

Problem 4

Here I slightly modify the description in the assignment since it might be vague in the usage of *set*. Given the integers $x_1, \cdots, x_n$ and $S = \sum_{i=1}^{n} x_i$, we denote X with the size n by the set of $x_i,\ i = 1 \ldots n$, where $X[j]$ is the j-th value of X.

Following the algorithm we proposed in Algorithm 3, the algorithm to partition the set into two subsets of equal sum is described as Algorithm 4.

Problem 5

Without recursive steps, we can also design an algorithm to solve the Hanoi Puzzle problem. We define an auxiliary function $move(X, Y)$ to move disks between two legs X, Y and print this move. The algorithm is described as Algorithm 5.

Algorithm 3 A modified unlimited knapsack problem

```
function UnlimitedKnapsack(S,K)
    P[0,0].extst := True
    P[0,0].belong := 0
    for k := 1...K do
        P[0,k].exist := False
    end for
    for i := 1...n do
        for k := 0...K do
            P[i,k].exist := False
            if P[i−1,k].exist = True then
                P[i,k].exist := True
                P[i,k].belong := 0
            else if k − S[i] ≥ 0 then
                if P[i,k − S[i]].exist = True then
                    P[i,k].exist := True
                    P[i,k].belong := P[i,k − S[i]].belong + 1
                end if
            end if
        end for
    end for
    return P
end function
```

Algorithm 4 Equal-sum subsets partition

```
function EqualSumSubsetPartition(S,X)
    P, partitionA, partitionB := ∅
    if S mod 2 = 1 then
        return False
    else
        halfS := S/2
        P ← UnlimitedKnapsack(X, halfS)
        if P[n, halfS].exist = True then
            partitionA ← elements in P.belong = True
            partitionB ← elements in P.belong = False
        else
            return False
        end if
    end if
    return partitionA, partitionB
end function
```

Algorithm 5 Hanoi puzzle

```
function NonRecursizeHanoiPuzzle(n,A,B,C)
    step := 0
    while step < 2^n − 1 do
        if n mod 2 ≠ 0 then
            move(A, C)
            move(A, B)
            move(C, B)
        else
            move(A, B)
            move(A, C)
            move(C, B)
        end if
        step := step + 1
    end while
end function
```