Daniel Diamont
EE360C

Programming Assignment 3

PART ONE

Argument for Optimal Substructure

Claim:
Consider any optimal solution to assign *f* total maximum economic growth to the entire Wakandan economy using the *k* denominations of Vibranium that we have available from our limited supply of Vibranium *V*, where the Vibranium is distributed as the optimal investment for each of the *n* available projects. Then, diving the total solution into *i* sub-solutions, each sub-solution must itself provide the optimal economic growth f_x for a subset of the Wakandan economy over the subset of available Vibranium V_x where $\sum_{(x=1)}^{i} f_x = f$ and $\sum_{(x=1)}^{i} V_x = V$.

Proof:
Suppose that one of the i_j sub solutions is *not* optimal. This means that either i_j did not achieve the optimal economic growth for that subset of the economy or used more Vibranium than was necessary to achieve the same economic growth. Then, this implies that more than *f* economic growth could have been achieved for the entire Wakandan economy either by wiser allocation of Vibranium to a project within subset i_j or by using the remaining Vibranium for more investments in the same or other subsets of the economy. However, this is a contradiction, because the claim is that *f* is already the total maximum economic growth to the Wakandan economy that can be achieved by distributing *V* Vibranium over *n* projects.

Value of Optimal Solution

The maximum economic growth for a subset of the Wakandan economy given a project *x* and a subset of Vibranium *y* can be defined in tabular form as:

$$OPT[i,j] = \begin{cases} max_{(0 \le k \le j)}[f_i(k) + OPT[i-1, j-k]], & V>0, \\ 0, & V=0 \end{cases}$$

Clearly, we can see that the maximum economic growth for the entire Wakandan economy given all *n* projects and given all Vibranium *V* can be found at the end of the algorithm in *OPT[n,V]*.

Computing the Optimal Value

Given *V* Vibranium, *n* projects, and economic gain function $g : V' \rightarrow \mathbb{Z}$ that maps Vibranium investments for each project to its respective economic gain.

computeGain(n,V, g):

for all V' ← 0 to V:
OPT[n-1][V'] ← $g_{(n-1)}(V')$

```
for all I ← n – 2 down to 0:
    for all V' ← to V:
        best ← 0
        for all v ← 0 to V':
            this ← g_(I)(V') + OPT[I+1,v]
            if this > best
                best ← this
        endfor
    OPT[I,V'] ← best
    endfor
endfor

return OPT[0,V]
```

Constructing the Optimal Solution

solution(OPT, n,V):

let v = V
let C[i] hold the optimal amount of Vibranium to spend per project, where $i \in (0, n)$.

```
for p ← 0 to n-1:
    max ← 0
    index ← 0

    for i ← 0 to v:
        this ← OPT[p,v] – OPT[p+1,i]
        if max < this ∧ this ∈ g_p
            max ← this
            index ← i

    for x ← v downto 0:
        if g_p(x)=max
            C[p] = x

    v = index
return C
```

PART TWO

This problem is a variation of the 0-1 Knapsack problem with multiple constraints.

Given projects *p*, decision variable *x*, weight *w*, and volume *v*, we want to:

$$\textit{maximize} \sum_{(j=1)}^{n} p_j x_j$$

$\textit{with constraints} \sum_{(j=1)}^{n} w_{(i,j)} x_j \leq W_i, \forall\, i \in [1, m]$, where *m* is the total volume available for weight w_j and $x_j \in [0,1] \forall\, j \in [0, n-1]$

Value of Optimal Solution

The value of the optimal solution can be defined recursively as:

$$OPT[i,j,k]=\begin{cases} max_{(1\leq j\leq m)}[OPT[i-1,w,],p(i)+OPT[i-1,j-j_i,k-k_i]], & if\ j_i\leq j\wedge k_i\leq k, \\ OPT[i-1,j,k], & if\ j_i>j\vee k_i>k, \\ 0, & otherwise \end{cases}$$

The solution of how the maximum amount of Vibranium that the thief can steal is given by *OPT[n,W,V]* where *n* is the number or Vibranium ores available to steal, *W* is the maximum weight that the thief can carry in the knapsack, and *V* is the maximum amount of space available in the knapsack.

Computing the Optimal Value

Given a knapsack of *V* volume and *W* weight, as well as *n* Vibranium ores to possibly steal.

```
computeLoss(n,W,V,w,p,k):

for w ← 0 to W
        for v ← 0 to V
                OPT[0,w,v] ← 0

for i ← 1 to n
        for w ← 1 to W
                for v ← 0 to V
                        if i   w_i>w ∨ v_i>v
                                OPT[i,w,v] ← OPT[i-1,w,v]
                                keep[i][w][v] ← 0
                else
                        best ← 0
                        for all k ← 0 to v
                                this ←    max[OPT[i-1][w][k], p(i)+OPT[i-1][w-w_i][k-k_i]]
                                if best < this
                                        best ← this

                        OPT[i][w][v] ← best
                        keep[i][w][v] ← 1


return OPT[n][W][V]
```

Constructing the Optimal Solution

```
solution(OPT, n, W, V):

        K = W
```

```
J = V

for i ← n downto 1
    if keep[i,K,J] = 1
        output i
        K ← K - w_i
        J ← J - v_i
```