# Tarea 2: Análisis y Diseño de Algoritmos

Juan Diego Collazos Mejia

## 1 Problema: Expert Computer Hackers

Supongamos que estamos administrando un equipo de consultoría de hackers expertos y cada semana debemos elegir un trabajo para que realicen. Los trabajos se dividen en dos tipos:

- **Trabajos de bajo estrés:** generan una ganancia de $L[i] > 0$ dólares en la semana $i$.
- **Trabajos de alto estrés:** generan una ganancia de $H[i] > 0$ dólares en la semana $i$, pero requieren que la semana anterior no se realice ningún trabajo.

Dado un número $N$ de semanas y los arreglos $L[0..N)$ y $H[0..N)$, el problema consiste en encontrar una planificación óptima que maximice la ganancia total.

### 1.1 Especificación

**Entrada:** Dos arreglos $L[0..N)$ y $H[0..N)$ con $N \geq 0$.

**Salida:** Un número natural que representa la máxima ganancia posible en $N$ semanas.

### 1.2 Función Objetivo

Sea $0 \leq n \leq N$, definimos:

$$\Phi(n) \quad \text{como la máxima ganancia posible en } n \text{ semanas.}$$

### 1.3 Reformulación de la salida

$$\Phi(N)$$

### 1.4 Planteamiento Recurrente

$$\Phi(n) = \begin{cases} 0, & \text{si } n = 0, \\ \max(L[0], H[0]), & \text{si } n = 1, \\ \max(\Phi(n-1) + L[n-1], \Phi(n-2) + H[n-1]), & \text{si } n > 1. \end{cases}$$

## 1.5 Análisis de Complejidad

### 1.5.1 Sin Programación Dinámica

Cada llamada recursiva genera dos nuevas llamadas, formando un árbol de recursión. En el peor caso, el número total de llamadas es $O(2^n)$, lo que resulta en una complejidad exponencial.

### 1.5.2 Con Programación Dinámica

Dado que los subproblemas se solapan, podemos almacenar los valores ya calculados, evitando recomputaciones innecesarias. La cantidad de estados es $N$, por lo que la complejidad temporal y espacial se reduce a $O(N)$.

## 1.6 Correctitud

**Teorema:** $\Phi(n)$ es correcto y devuelve la máxima ganancia posible.

**Demostracion:** Inducción sobre $n$.

**Caso Base:**

- $\Phi(0) = 0$, ya que con 0 semanas la ganancia es trivialmente 0.
- $\Phi(1) = \max(L[0], H[0])$, ya que con una semana solo se puede elegir el mayor de los valores en $L[0]$ o $H[0]$.

**Paso Inductivo:** Sea $n > 1$.
**Hipótesis Inductiva:** Procedemos por inducción fuerte.
Suponemos que $\Phi(k)$ es correcto para todo $0 \leq k \leq n-1$.
Queremos demostrar que:

$$\Phi(n) = \max(\Phi(n-1) + L[n-1], \Phi(n-2) + H[n-1])$$

Por hipótesis de inducción, los valores $\Phi(n-1)$ y $\Phi(n-2)$ son correctos, es decir, representan la máxima ganancia posible al considerar hasta $n-1$ y $n-2$ semanas respectivamente.

Ahora bien, en la semana $n$, tenemos dos opciones válidas:

1. **trabajar en el proyecto de baja ganancia** con recompensa $L[n-1]$. En este caso, la mejor decisión previa fue haber seguido una estrategia óptima hasta la semana $n-1$, lo que está representado por $\Phi(n-1)$. Como podemos realizar el trabajo de baja ganancia en la semana $n$, sumamos $L[n-1]$ a la solución previa.
2. **Trabajar en el proyecto de alta ganancia** con recompensa $H[n-1]$. Sin embargo, este trabajo impide haber trabajado en la semana $n-1$, por lo que la mejor decisión previa tuvo que haber sido una estrategia óptima hasta la semana $n-2$, lo que está representado por $\Phi(n-2)$. En este caso, sumamos $H[n-1]$ a la mejor solución previa válida.

Dado que queremos maximizar la ganancia, tomamos el máximo entre ambas opciones. Esto asegura que $\Phi(n)$ almacene el mejor resultado posible para $n$ semanas, lo que completa la demostración.

**Corolario:** Sea $\Phi(n)$ la función definida en la demostración de correctitud. Entonces, el llamado de $\Phi(N)$ resuelve correctamente el problema de maximización de ganancia en $N$ semanas.

## 1.7 Implementación en Python

```
dp = dict()

def phi(n):
  if n in dp:
    ans = dp[n]
  else:
    if n == 0: ans = 0
    elif n == 1: ans = max(H[n - 1], L[n - 1])
    else: ans = max(phi2(n - 1) + L[n - 1], phi2(n - 2) + H[n - 1])
    dp[n] = ans
  return ans

L = [10, 20, 10, 10, 30]
H = [50, 60, 70, 20, 25]
N = len(L)
ans = phi(N) # ans = 160
```

## 1.8 Ejemplo detallado con $n = 5$ y memorización

Dado:

$$L = [10, 20, 10, 10, 30]$$
$$H = [50, 60, 70, 20, 25]$$

Queremos calcular $\Phi(5)$ usando la formulación recurrente con memorización.

$\Phi(1) = \max(10, 50) = 50.$
$\Phi(2) = \max(\Phi(1) + L[1], \Phi(0) + H[1]) = \max(50 + 20, 60) = 70.$
$\Phi(3) = \max(\Phi(2) + L[2], \Phi(1) + H[2]) = \max(70 + 10, 50 + 70) = 120.$
$\Phi(4) = \max(\Phi(3) + L[3], \Phi(2) + H[3]) = \max(120 + 10, 70 + 20) = 130.$
$\Phi(5) = \max(\Phi(4) + L[4], \Phi(3) + H[4]) = \max(130 + 30, 120 + 25) = 160.$

En conclusión, la máxima ganancia posible en 5 semanas es **160**.

# 2 The City of Zion

Dada una secuencia de llegadas de robots $X = \{x_1, x_2, \ldots, x_n\}$ y una función de recarga del EMP $F = \{f(1), f(2), \ldots, f(n)\}$, nuestro objetivo es determinar los momentos óptimos para activar el EMP y maximizar la cantidad de robots destruidos.

## 2.1 Especificación

- Entrada: Dos arreglos $X[0..N)$ y $F[0..N)$ con $N \geq 0$ y un número natural $T$.
- Salida: Un número natural que representa la máxima cantidad posible de robots que se pueden destruir con $T$ unidades de tiempo.

## 2.2 Función Objetivo

Sea $0 \leq i \leq N$ y $0 \leq t \leq T$.

$\phi(i, t)$ : Determina la máxima cantidad de robots que se pueden destruir con $t$ unidades de tiempo y con el EMP cargado $i$ unidades de tiempo.

## 2.3 Reformulacion de la salida

$\phi(0, T)$

## 2.4 Planteamiento Recurrente

$$\phi(i,t) = \begin{cases} 0, & t = 0; \\ \phi(0, t-1) + \min(X[i], F[i]), & t \neq 0 \wedge i = N-1 \\ \max(\phi(i+1, t-1), \phi(0, t-1) + \min(X[i], F[i])), & t > 0 \wedge i < N-1. \end{cases} \tag{1}$$

## 2.5 Complejidad Computacional

### 2.5.1 Sin Programación Dinámica

Cada llamada recursiva genera como maximo dos nuevas llamadas, formando algo similar a un ´arbol binario de recursi´on. En el peor caso, el n´umero total de llamadas es $O(2^n)$, lo que resulta en una complejidad exponencial.

### 2.5.2 Con Programación Dinámica

Dado que los subproblemas se solapan, podemos almacenar los valores ya calculados, evitando recomputaciones innecesarias. Como $0 \leq i \leq N$ y $0 \leq t \leq T$ La cantidad de estados es $O(N \times T)$, por lo que la complejidad temporal y espacial se reduce a $O(N^2)$.

## 2.6 Correctitud

**Teorema:** La función $\phi(i, t)$ devuelve el número máximo de robots que pueden ser destruidos con las restricciones dadas.

### Demostración

Procedemos por inducción doble.

### Inducción sobre $t$ (Tiempo disponible)

**Caso base ($t = 0$)** Si $t = 0$, la definición establece que:

$$\phi(i, 0) = 0$$

Esto es correcto porque si no hay tiempo restante, no podemos activar el EMP en ningún momento, por lo que no es posible destruir robots. No hay alternativas en este caso, lo que garantiza la validez de la ecuación.

**Paso inductivo ($t > 0$) Hipótesis inductiva (HI):** Suponemos que la ecuación es válida para un tiempo $t - 1$, es decir:

$$\phi(i, t - 1)$$

calcula correctamente la cantidad máxima de robots que se pueden destruir en $t - 1$ unidades de tiempo.

Queremos probar que sigue siendo válida para $t$. Para ello, hacemos una inducción interna sobre $N - i$, es decir, sobre los elementos restantes en las listas $X$ y $F$.

### Inducción sobre $N - i$ (Cantidad de elementos restantes en las listas)

**Caso base ($i = N - 1$)** Cuando estamos en la última posición del array ($i = N - 1$), la ecuación toma la forma:

$$\phi(N - 1, t) = \phi(0, t - 1) + \min(X[N - 1], F[N - 1]).$$

Esto significa que en la última posición no podemos seguir cargando el EMP más allá de $i = N - 1$. Solo hay una opción disponible:

- Activar el EMP en este instante, destruyendo $\min(X[N - 1], F[N - 1])$ robots.
- Reiniciar la carga del EMP desde $i = 0$ con $t - 1$ segundos restantes.

Por hipótesis inductiva sobre $t$, sabemos que la ecuación es válida para $t-1$, por lo que $\phi(0, t-1)$ ya calcula correctamente la mejor cantidad de robots destruidos a partir de ese punto.

Dado que esta es la única opción en $i = N - 1$, la ecuación es válida en este caso.

**Paso inductivo ($i < N - 1$) Hipótesis inductiva (HI):** Suponemos que la ecuación es válida para $i + 1$, es decir:

$$\phi(i + 1, t)$$

ya calcula correctamente la mejor cantidad de robots destruidos en los casos en los que el EMP está cargado hasta $i + 1$.

Ahora probamos que la ecuación también es válida para $i$. La ecuación nos da dos opciones:

- **No activar el EMP en $i$, sino seguir cargándolo:**

  $$\phi(i, t) = \phi(i + 1, t - 1).$$

  Esto significa que simplemente avanzamos al siguiente índice y reducimos el tiempo disponible en 1.

  - Como la hipótesis inductiva sobre $N - i$ garantiza que $\phi(i + 1, t - 1)$ es correcto, entonces esta opción ya calcula la cantidad máxima de robots destruidos si optamos por seguir cargando el EMP.

- **Activar el EMP en $i$ y reiniciar la carga:**

  $$\phi(i, t) = \phi(0, t - 1) + \min(X[i], F[i]).$$

  - En este caso, destruimos $\min(X[i], F[i])$ robots.
  - Reiniciamos la carga a $i = 0$ con $t - 1$ segundos restantes.
  - Como la hipótesis inductiva sobre $t$ garantiza que $\phi(0, t - 1)$ es correcto, sabemos que esta opción también nos da la mejor cantidad de robots destruidos para los $t - 1$ segundos restantes.

Dado que tomamos el **máximo** entre estas dos opciones, garantizamos que la cantidad máxima de robots destruidos se mantiene de manera óptima.

Por hipótesis inductiva, la ecuación es válida en $i + 1$, y al probar que se mantiene en $i$, concluimos la inducción sobre $N - i$.

### Finalización de la inducción sobre $t$

Dado que hemos demostrado la fórmula para todos los valores de $N - i$ cuando $t - 1$ es válido, se sigue que también es válida para $t$. **Por lo tanto, la función $\phi(i, t)$ devuelve la cantidad máxima de robots destruidos en cualquier instante de tiempo.**

### Corolario:

Sea $\Phi(i, n)$ la función definida en la demostración de correctitud. Entonces, el llamado de $\Phi(0, T)$ resuelve correctamente el problema de maximización de robots destruidos en $T$ unidades de tiempo.

## 2.7 Implementación en Python

```python
dp = dict()

def phi(i, t):
  if (i, t) in dp:
    ans = dp[i, t]
  else:
    if t == 0:
      ans = 0
    else:
      ans = phi(0, t - 1) + min(X[i], F[i])
      if i < N - 1:
        ans = max(ans, phi(i + 1, t - 1))
    dp[i, t] = ans
  return ans

X = [1, 10, 10, 1]
F = [1, 2, 4, 8]
N = len(F)

T = 4
ans = phi(0, T)
print(ans)
```

## 2.8 Ejemplo de llamadas recursivas para $\Phi(i, t)$

Consideremos los valores siguientes para los arreglos:

- $X = [1, 10, 10, 1]$
- $F = [1, 2, 4, 8]$

Evaluemos $\varphi(0, 5)$ paso a paso:

- $\varphi(0, 5)$ llama a $\varphi(1, 4)$ y $\varphi(0, 4) + \min(1, 1)$.
- $\varphi(1, 4)$ llama a $\varphi(2, 3)$ y $\varphi(0, 3) + \min(10, 2)$.
- $\varphi(2, 3)$ llama a $\varphi(3, 2)$ y $\varphi(0, 2) + \min(10, 4)$.
- $\varphi(3, 2)$ llama a $\varphi(4, 1)$ y $\varphi(0, 1) + \min(1, 8)$.
- Se alcanzan los casos base y se propagan los valores calculados.

**Evaluación de los casos base:**

- $\varphi(i, 0) = 0$ para todo $i$.
- $\varphi(3, 1) = \varphi(0, 0) + \min(1, 8) = 0 + 1 = 1$.

- $\varphi(2,2) = \max(\varphi(3,1), \varphi(0,1) + \min(10,4)) = \max(1, 0+4) = 4$.
- $\varphi(1,3) = \max(\varphi(2,2), \varphi(0,2) + \min(10,2)) = \max(4, 0+2) = 4$.
- $\varphi(0,4) = \max(\varphi(1,3), \varphi(0,3) + \min(1,1)) = \max(4, 0+1) = 4$.
- Finalmente, $\varphi(0,5) = \max(\varphi(1,4), \varphi(0,4) + \min(1,1)) = \max(4, 4+1) = 5$.