# Trabajo Práctico 2: Programación Dinámica para el Reino de la Tierra

Facultad de Ingeniería de la Universidad de Buenos Aires

Teoría de Algoritmos

Cátedra Buchwald-Genender


Gómez Belis, Sofía
Padrón: 109358
email: sgomezb@fi.uba.ar

Llanos Pontaut, Valentina
Padrón: 104413
email: vllanos@fi.uba.ar

Orsi, Tomas Fabrizio
Padrón: 109735
email: torsi@fi.uba.ar

# Indice

# 1 Análisis del problema

## 1.1 Descripción y objetivo

El objetivo de este trabajo práctico es implementar un algoritmo de programación dinámica para ayudar a los Dai Li, la policía secreta de la ciudad Ba Sing Se del Reino de la Tierra, en su combate contra la Nación del Fuego. Como somos sus jefes estratégicos, debemos reportarles las estrategias que deberían emplear en cada minuto del ataque con el fin de eliminar la máxima cantidad de maestros Fuego.

Afortunadamente, las mediciones sísmicas de los Dai Li nos permitieron obtener información de la cantidad de enemigos que llegan en cada minuto. Además, sabemos que la cantidad de enemigos no es acumulativa; después de cada ataque, los enemigos se retiran. Tenemos entonces, una lista de valores $x_i$ que ilustran este hallazgo. Por otro lado, la potencia de los ataques de fisura de la policía secreta dependen de cuánto tiempo fue cargada la energía. La función monótona creciente $f(\cdot)$ indica que, si transcurrieron $j$ minutos desde que se utilizó este ataque, entonces se podrá eliminar hasta $f(j)$ soldados enemigos. Como este valor puede ser mayor a la cantidad de maestros Fuego, en el minuto $k$ se podrá eliminar $min(x_k, f(j))$ soldados, perdiendo además la energía acumulada. En el primer minuto, en caso de decidir atacar, les corresponde $f(1)$ de energía.

Sabiendo la duración en minutos del ataque ($n$), y los valores de $x_i$ y $f(j)$, podemos analizar el problema presentado e informarles a los Dai Li la secuencia de estrategias que les permitirá eliminar la mayor cantidad de enemigos empleando ataques de fisuras.

## 1.2 Análisis y ecuación de recurrencia

La resolución de un problema por medio de la programación dinámica implica reutilizar las soluciones a subproblemas más pequeños en subproblemas más grandes que los incluyan. En el contexto actual, el foco está puesto en maximizar la cantidad de enemigos eliminados dados $n$ minutos. Esta variable $n$ es crucial en el análisis del problema planteado puesto que si se tiene $k$ minutos, con $k < n$, necesariamente la cantidad de enemigos eliminados será menor o igual a la solución en el minuto $n$, siendo igual en el caso de que no lleguen más enemigos entre $k$ y $n$ minutos. De esta forma, la cantidad de minutos que tiene el ataque repercute en el resultado final del combate. Por ejemplo, si $n = 0$, se puede afirmar que la cantidad de enemigos eliminados será también 0. Entonces, nuestros subproblemas estarán dados por la cantidad máxima de enemigos que se pueden eliminar en cada minuto $i \leq n$.

Sabiendo la forma de los subproblemas, debemos analizar cómo se componen para resolver subproblemas más grandes. Si queremos obtener la solución óptima en el minuto $i$, vamos a poder utilizar las soluciones parciales calculadas hasta entonces, pero no nos interesa si existen o no problemas más grandes. En cada minuto $i < n$ hay solamente dos estrategias posibles: atacar o cargar.

**Atacar** implica enfrentarse con el enemigo, derrotando $f(j)$ soldados (con $j$ siendo la cantidad de minutos desde el ataque anterior) y perdiendo la energía acumulada. En cambio, definimos **cargar** como la acción de decidir no atacar y acumular más energía para un ataque futuro.

Sin embargo, en el caso del minuto $i = n$, ¿tiene sentido cargar sabiendo que no van a llegar más enemigos en el futuro? No. Ésto se debe a que la cantidad de maestros Fuego que se podrá eliminar en ese minuto será mayor o igual a 0, pero si cargamos energía, será definitivamente nula. Por lo tanto, en el último minuto conviene siempre atacar. Ahora bien, como establecimos antes, en el minuto $i \leq n$ no importa si $i = n$ o $i < n$, solamente debemos calcular el óptimo actual. En base al análisis previamente presentado, siempre va a ser mejor atacar a cargar, más allá de que en la solución final (problema mayor) se lleve a cabo la estrategia opuesta debido a que eso esté contemplado en el óptimo del minuto $n$.

Si en el minuto $i$ los Dai Li atacan, la cantidad de enemigos eliminados en ese instante será $min(f(j), x_i)$. El valor de $x_i$ es conocido, pero la energía acumulada ($f(j)$) por la policía secreta de la ciudad depende de los minutos que pasaron desde el último ataque. De esta manera, tenemos una segunda variable involucrada en el problema: $j$. Si el último ataque fue realizado hace un minuto, actualmente se podrá eliminar $min(f(1), x_i)$ soldados y la cantidad de enemigos eliminados acumulada será la suma entre este valor y el correspondiente en el ataque anterior.

¿Qué sucede si el óptimo hace dos minutos es mayor que en el minuto anterior o su suma con $min(f(2), x_i)$ lo es? Como queremos maximizar el resultado final, claramente nos conviene haber atacado hace dos minutos, lo cual también indica que en el minuto $i-1$ se cargó energía. Tenemos varias opciones para el ataque anterior, más precisamente $1 \leq j \leq i$, pero utilizaremos aquel que nos lleve a la mejor solución. Entonces, el óptimo para el minuto $i$ será la suma entre $min(f(j), x_i)$ y el óptimo en el minuto $i - j$.

Sabiendo la forma de los subproblemas y la manera en que éstos se combinan, podemos plantear la **ecuación de recurrencia** para el minuto $i$:

$$OPT[i] = max(min(f(j), x_i) + OPT[i - j] \forall j \in [1; i])$$

Como caso base, tenemos que en el minuto 0 se eliminan 0 enemigos.

Encontrada la ecuación de recurrencia, procedemos a aplicarla iterativamente de manera bottom up, construyendo las soluciones a los subproblemas de $i < n$ hasta llegar a la solución del problema original con $i = n$. Esta técnica es justamente programación dinámica. Empleamos `memoization` guardando los resultados calculados previamente en un arreglo. El procedimiento explicado nos permite realizar una exploración implícita del espacio de soluciones. La solución final será óptima porque en el minuto $n$ elegimos haber atacado hace $j$ minutos, donde $j$ maximiza la ecuación de recurrencia.

Como conclusión, el uso de esta ecuación de recurrencia en nuestra implementación nos permite determinar la cantidad máxima de enemigos que se pueden atacar. En cada minuto calculamos el máximo número de adversar-

ios eliminados si se decide atacar en ese instante. Este valor está determinado por el tiempo entre éste y el ataque anterior. Al quedarnos con un $j$ que maximice la suma final de enemigos derrotados en el minuto $i$, nos aseguramos de obtener una solución óptima a ese subproblema. Luego, en el último minuto del combate, utilizaremos las soluciones a estos subproblemas de forma tal que tendremos en cuenta el ataque anterior que ocurrió en el minuto $k = n - j$, donde $OPT[k] + min(f(j), x_n)$ maximiza la cantidad de enemigos eliminados en total. A su vez, el $OPT[k]$ tiene lo mismo en consideración.

Guardar los óptimos en un arreglo de soluciones parciales para cada minuto nos facilita reconstruir la estrategia de ataque óptima que permite obtener el resultado para $OPT[n]$. Sabiendo el valor de la cantidad de enemigos que llegan en el minuto $n$, $x_n$, podemos comparar resultados parciales con $OPT[n]$ para obtener $j$. Es decir, como ya determinamos que en el minuto $n$ vamos a atacar, $j$ va a corresponder al resultado que cumpla con $OPT(n) = OPT(k) + min(f(j), x_n)$, siendo $k$ el minuto en que previamente se realizó un ataque ($k = n - j$). Entre $k$ y $n$, la estrategia empleada es recargar fuerzas.

Repetimos el procedimiento para $k$, hasta llegar al minuto 0.