.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Programación Dinámica

3 de febrero de 2024

Ariel Aguirre
100199

Lucia Magdalena Berard
101213

Víctor Bravo
98.882

Maximiliano Prystupiuk
94.853

1. Algoritmo

1.1. Problema

Luego de haber analizado a todos los rivales gracias a tu ayuda, Scaloni definió un cronograma de entrenamiento. Tiene definido qué hacer para cada día de acá al mundial que viene, e incluso más. Para hacerlo más simple, para los próximos n días. El entrenamiento del día i demanda una cantidad de esfuerzo e_i, y podemos decir que la **ganancia** que nos da dicho entrenamiento es igual al esfuerzo. El entrenamiento que corresponde al día i (así como su esfuerzo y ganancia) son inamovibles: el Chiqui Tapia alquiló las herramientas específicas para cada día, y la AFA está muy ocupada organizando el torneo de 2^{30} equipos del año que viene para andar moviendo cosas. Si la cantidad de energía que se tiene para un día i es $j < e_i$, entonces la ganancia que se obtiene en ese caso es justamente j. (si se tiene más energía que e_i, no es que se pueda usar más para tener más ganancia).

A su vez, los jugadores no son máquinas. La cantidad de energía que tienen disponible para cada día va disminuyendo a medida que pasan los entrenamientos. Suponiendo que los entrenamientos empiezan con los jugadores descansados, el primer día luego de dicho descanso los jugadores tienen energía s_1. El segundo día luego del descanso tienen energía s_2, etc... Para cada día hay una cantidad de energía, y podemos decir que $s_1 \geq s_2 \geq \ldots \geq s_n$. Scaloni puede decidir dejarlos descansar un día, haciendo que la energía "se renueve"(es decir, el próximo entrenamiento lo harían con energía s_1 nuevamente, siguiendo con s_2, etc...). Obviamente, si descansan, el entrenamiento de ese día no se hace (y no se consigue ninguna ganancia).

Scaloni no sabe bien cómo hacer para definir los días que deba entrenarse y los días que convenga descansar de tal forma de tener la mayor ganancia posible (y tener mayores probabilidades de ganar el mundial que viene), pero Menotti, exponente del juego bonito en Argentina, le recomendó usar Programación Dinámica para resolver este problema. Nos está pidiendo ayuda para poder resolver este problema.

1.2. Diseño

1.2.1. Análisis del problema

- **Dado:**
  - Una secuencia de energías disponibles S[0], S[1], S[2], ..., S[n] que representa la energía disponible en un día dado si no se ha descansado desde el último entrenamiento.
  - Una secuencia de esfuerzo requerido o ganancia potencial E[0]. E[1], E[2], ..., E[n] para cada día.
- **Objetivo:**
  - Determinar la máxima ganancia posible al final del periodo, considerando la posibilidad de entrenar o descansar cada día.
- **Restricciones:**
  - Si se entrena, la ganancia de un día es el mínimo entre el esfuerzo requerido y la energía disponible.
  - Si se descansa, la energía se reinicia al valor inicial S[0].
  - Descansar implica no entrenar, por lo que la ganancia de los días de descanso es cero.
  - Todos los entrenamientos son beneficiosos, es decir, $e_i > 0$

1.2.2. Ecuación de recurrencia

Primero, tomaremos las siguientes reglas de notación:

- **i**: Es un índice para los días. Ejemplo: El día $i = 3$, o el esfuerzo para el día $i = 2$ es e_i.
- **j**: Es un índice para las energías. Por ejemplo, al día $i = 2$ se entrenó con energía s_j con $j = 3$, entonces la ganancia es mín(e_2, s_3).

Luego, para encontrar la ganancia óptima, observamos que tenemos los siguientes posibles casos:

- Si hoy entreno con un s_j y $j \neq 0$, entonces ayer también entrené y fue con energía s_{j-1}
- Si hoy entreno y tengo toda la energía (s_0) entonces ayer descansé s_{None}. Tomaremos $j = None$ para indicar que un día se descansó.
- Si hoy i descanso, entonces ayer pude o haber descansado también o haber entrenado, pero con varias posibilidades de energía que pueden ir desde s_0 hasta s_j con $j \leq i$

Como ejemplo, para un entrenamiento de tres días, podemos tomar las decisiones anteriores y armar el siguiente árbol, donde cada columna es un día y cada rectángulo es una energía

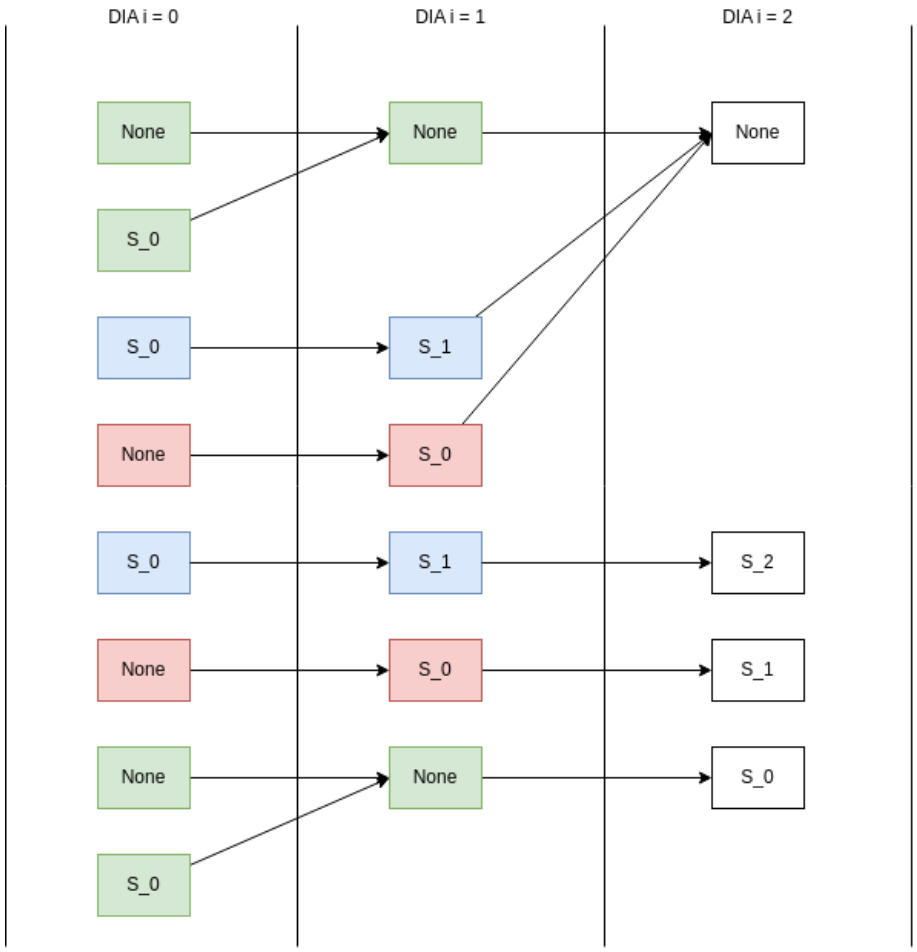

Figura 1: Árbol de posibilidades

Podemos notar que la ganancia óptima consiste en analizar cada posibilidad del árbol y obtener la máxima ganancia.

Si nos fijamos, hay cálculos que se repiten, como la ganancia para un día $i = 1$ y descanso con energía $None$ coloreados con verde. Ambos deben calcular previamente las ganancias para el día

anterior $i = 0$ con posible energía s_0 o descanso. Otros dos casos similares están coloreados de rojo y azul.

Esto, nos hace pensar que podemos aprovechar dicha situación mediante programación dinámica.

El siguiente pseudocódigo nos ayuda a encontrar la máxima ganancia para un cierto día i. Ejemplo de uso: Para calcular el máximo dado $DIA = 3$, $E = [1, 5, 4]y$S=[10, 2, 2], llamaríamos a $gananciaAlDia(S, E, DIA)$.

```
Funcion gananciaAlDia(S, E, i):
    Sea G una lista y D un numero
    Para energia al dia j, desde j=0 hasta dia i+1:
        Calcular G[j] = gananciaAlDiaConEnergia(S, E, i, j)
    Calcular ganancia si ayer hubiera descansado como D = gananciaAlDiaConEnergia(S
    , E, i, None)
    Retornar el maximo entre todas las ganancias G[j] y D

  Function gananciaAlDiaConEnergia(S, E, i, j):
    Si dia i < 0:
        Retornar 0

    Si j es None: # Hoy descanso
        Retornar gananciaAlDia(S, E, i-1)

    Si j == 0: # Hoy es el primer dia de entrenamiento, Ayer descanse
        Retornar ganancia(E[i], S[i]) + gananciaAlDiaConEnergia(S, E, i-1, None)

    # Hoy entreno, ayer tambien entrene
    Retornar ganancia(E[i], S[i]) + gananciaAlDiaConEnergia(S, E, i-1, j-1)

Funcion ganancia(e, s)
    Retornar el minimo entre e y s
```

1.2.3. Pseudocódigo Iterativo

```
Funcion algoritmo(DIAS, E, S)
    Inicializar matriz M y lista D
    Para cada dia i desde 0 hasta DIAS-1
        Para cada j desde 0 hasta i
            Calcular la ganancia para el dia i con j dias de energia
            Si j es 0, sumar la ganancia con D[i-1]
            De lo contrario, sumarla con M[i-1][j-1]
            Almacenar el resultado en M[i][j]
        Actualizar D[i] con la maxima ganancia hasta el dia i
    Retornar M, D

Funcion reconstruccion(DIAS, M, D)
    Inicializar lista entrena
    Determinar la ganancia maxima y la posicion de energia correspondiente
    Para cada dia i desde DIAS-1 hasta 0
        Si j es None, marcar el dia i como descanso y actualizar j
        De lo contrario, marcarlo como entrenamiento y actualizar j
    Retornar entrena

Funcion max_ganancia(M, D, DIAS)
    Retornar la ganancia maxima para el ultimo dia

Funcion pos_energia_ganancia(M, dia, ganancia)
    Retornar la posicion de la energia correspondiente a la ganancia en el dia dado

Funcion gan(e, s)
    Retornar el minimo entre e y s
```

1.3. Análisis

```python
import sys
import csv

def entrena_to_str(entrena):
    r = []
    for e in entrena:
        if e:
            r.append('Entreno')
        else:
            r.append('Descanso')
    return ', '.join(r)

def max_ganancia(M, D, DIAS):
    return max(M[DIAS-1] + [D[DIAS-1]])

def pos_energia_ganancia(M, dia, ganancia):
    for i, g in enumerate(M[dia]):
        if ganancia == g:
            return i
    return None

def reconstruccion(DIAS, M, D):
    entrena = [None]*DIAS
    max_gan = max_ganancia(M, D, DIAS)
    j = pos_energia_ganancia(M, DIAS-1, max_gan)

    for i in range(DIAS-1, -1, -1):
        if j is None:
            entrena[i] = False
            j = pos_energia_ganancia(M, i-1, D[i])
        else:
            entrena[i] = True
            if j == 0:
                j = None
            else:
                j -= 1
    return entrena

def gan(e, s):
    return min(e, s)

def algoritmo(DIAS, E, S):
    M = [[0]*DIAS for _ in range(DIAS)]
    D = [0]*DIAS

    for i in range(DIAS):
        for j in range(i+1):
            g = gan(E[i], S[j])
            if j == 0:
                M[i][j] = g + D[i-1]
            else:
                M[i][j] = g + M[i-1][j-1]

        if i > 0:
            D[i] = max(M[i-1] + [D[i-1]])

    return M, D

def write(filename, data):
    with open(filename, 'w+') as f:
        writer = csv.writer(f, delimiter=';')
        for e in data:
            writer.writerow(e)

def load(filename):
    with open(filename) as f:
        lines = f.readlines()

    data = []
    for e in lines:
```

```
71          data.append(int(e.rstrip()))
72
73      DIAS = data[0]
74      E = data[1:DIAS+1]
75      S = data[DIAS+1:]
76      return DIAS, E, S
77
78
79  def main():
80      filename = sys.argv[1]
81      DIAS, E, S = load(filename)
82      M, D = algoritmo(DIAS, E, S)
83      entrena = reconstruccion(DIAS, M, D)
84      print('M: ', M, ' --- D: ', D)
85      print('max ganancia', max_ganancia(M, D, DIAS))
86      print(entrena_to_str(entrena))
87
88
89  if __name__ == '__main__':
90      main()
```

1.4. Análisis de Complejidad

Complejidad Temporal:

- La función `algoritmo` tiene dos bucles anidados que iteran sobre `DIAS`. Por cada día i, itera hasta $i + 1$, lo que da una complejidad temporal de $O(\texttt{DIAS}^2)$.
- Las funciones `max_ganancia`, `pos_energia_ganancia` y `reconstruccion` tienen todas complejidades lineales en términos de `DIAS`. Sin embargo, estas no afectan significativamente la complejidad general del algoritmo, que sigue siendo $O(\texttt{DIAS}^2)$.

Complejidad Espacial:

- El algoritmo utiliza una matriz M de tamaño `DIAS` $\times$ `DIAS`, por lo que la complejidad espacial es $O(\texttt{DIAS}^2)$.

Impacto de la Variabilidad de los Esfuerzos

- **Valores de Esfuerzo (E) y Energía (S):** La variabilidad en los valores de esfuerzo y energía afecta las decisiones de optimización en cada paso del algoritmo. Sin embargo, no cambia la complejidad computacional del algoritmo, que está determinada por el número de días (`DIAS`) y no por los valores específicos dentro de `E` y `S`.
- **Optimalidad:** La optimalidad del algoritmo depende de cómo maneje los valores variables de `E` y `S`. Si los esfuerzos requeridos son muy altos en comparación con la energía disponible, el algoritmo puede favorecer más días de descanso para maximizar la ganancia total. La variabilidad en estos valores puede conducir a diferentes patrones de entrenamiento/descanso, pero el algoritmo está diseñado para encontrar la secuencia óptima independientemente de estos valores.
- **Tiempo de Ejecución:** La variabilidad de `E` y `S` no afecta la complejidad temporal del algoritmo, que es cuadrática en términos del número de días. La eficiencia temporal está más relacionada con `DIAS` que con la naturaleza específica de los valores de esfuerzo y energía.

2. Mediciones

Se realizaron mediciones de tiempo en la resolución de distintas variaciones del problema, incrementando N, el numero de días.

Esto lo hicimos utilizando un generador de datos, de 500 hasta 20.000, con saltos de 500.

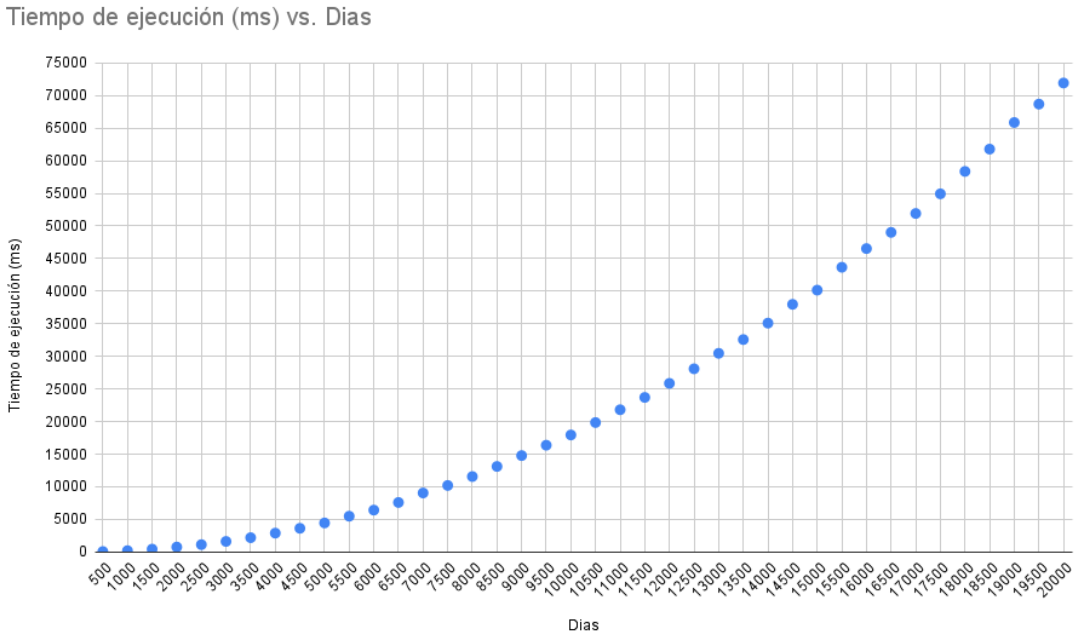

Gráfico de tiempo de ejecución en función de los elementos (días de entrenamiento).

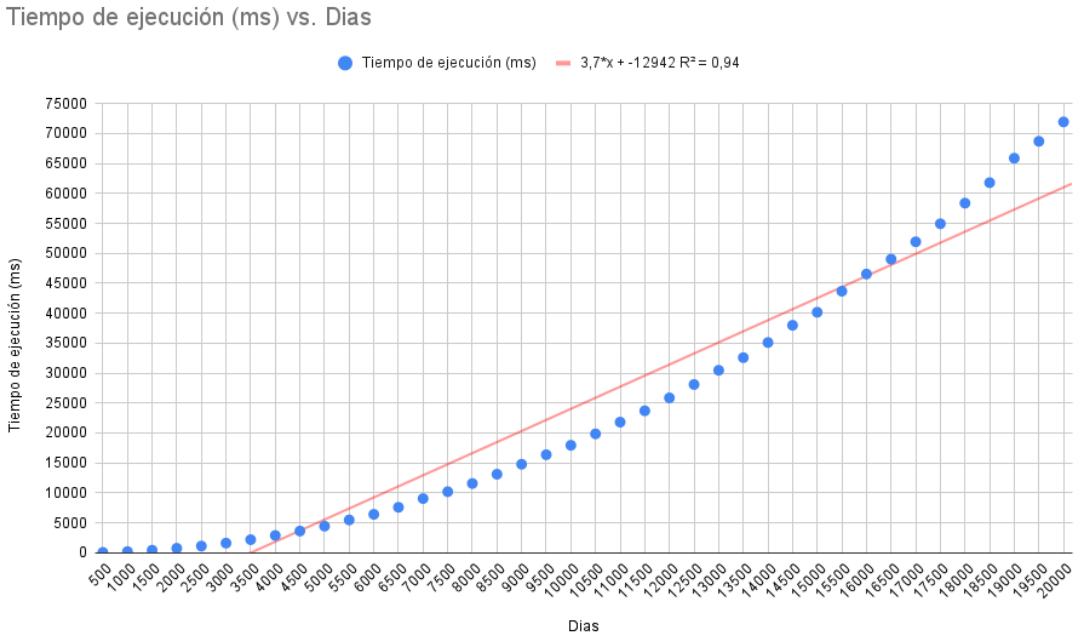

Gráfico con un ajuste lineal.

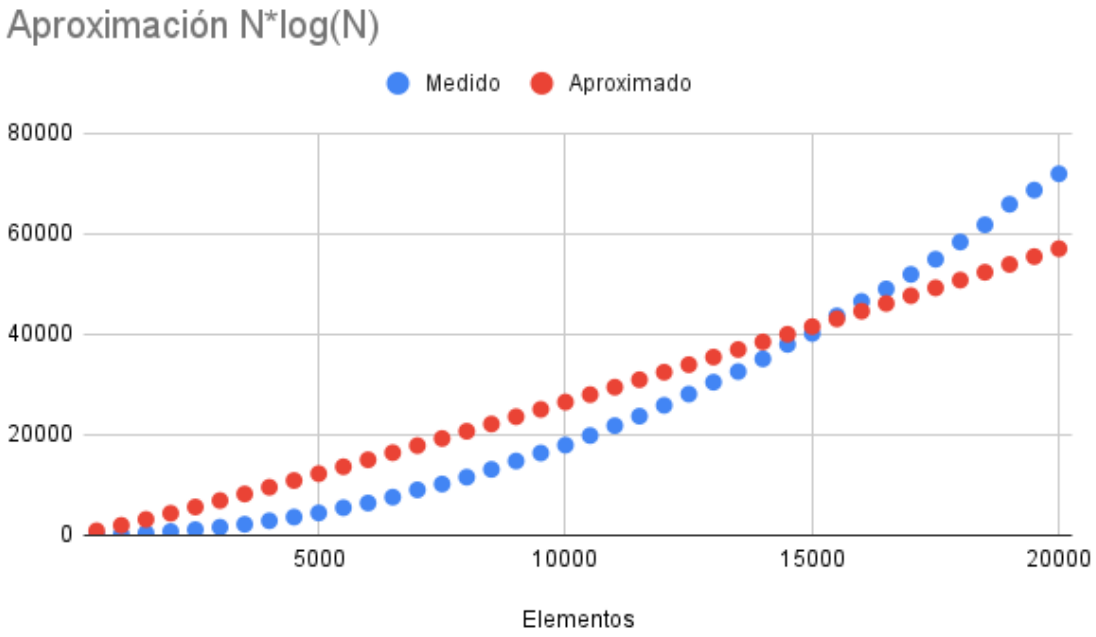

Gráfico con un ajuste de tipo $NLog(N)$. Para éste caso el valor de R^2 es de 0.8914.

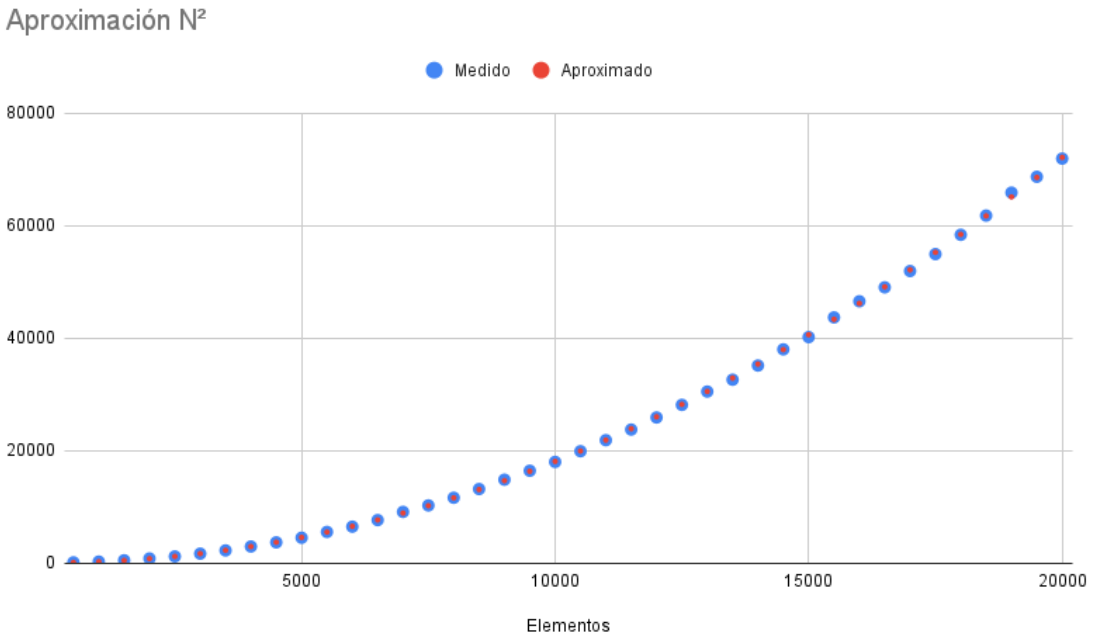

Gráfico con un ajuste de tipo N. Para éste caso el valor de R^2 es de 0,9999, que es el valor que más se ajusta a los datos medidos.

Para estos últimos dos casos, de N^2 y $NLog(N)$ los coeficientes se encontraron manualmente, porque además de las restricciones de minimizar las diferencias de cuadrados, dichos coeficientes debían ser positivos, porque no tiene sentido computacional una complejidad de $N-N$, por ejemplo, ya que $-N$ no representa nada.

En el repositorio puede referirse a /src/data/run-results.ods para ver los cálculos y fórmulas.

Cabe destacar que el limitante al generar los datos y realizar las mediciones no fue temporal, sino espacial. Con nuestra implementación, en una computadora con 16GB de RAM, y minimizando la cantidad de otros programas abiertos, el límite de cantidad de elementos rondaba los 20.000.

3. Mediciones extra

Realizamos también otras mediciones, pero conceptuales o de casos de interés

3.1. Caso base

Un caso en el que la energía coincida con la ganancia día a día.

Como era de esperar, el algoritmo dió que conviene entrenar todos los días.

3.2. Caso cíclico de ganancia

Es un caso en el que los valores de la ganancia se repiten en el ciclo 100-100-90-1, es decir 100-100-90-1-100-100-90-1-100-100-90-1-100-100-90-1... Las energías, por el contrario, cumplen un ciclo, y caen al valor mínimo: 100-100-90-1-1-1....-1

El algoritmo se adecuó a dicho ciclo y entrenó en los días de mayor ganancia, descansando en el de ganancia 1, para adecuarse al siguiente ciclo.

3.3. Caso inicio y fin

Un caso en el que convenga entrenar el primer y último día

El algoritmo cumple con esto, a la vez que pospone el descanso para el ante último día

4. Conclusiones

Nuestra conclusión es que el algoritmo propuesto tiene complejidades tanto temporales como espaciales de $O(N)$ En la práctica el limitante fue la complejidad espacial, que consumía la memoria disponible.