Informe Tarea
INFO145 - Diseño y Análisis de Algoritmos

Acádemico: Dr. Héctor Ferrada {hferrada@inf.uach.cl}
Instituto de Informática, Universidad Austral de Chile
Alumnos: - Felipe Córdova {felipe.cordova@alumnos.uach.cl}
- Sebastián Montecinos {sebastian.montecinos02@alumnos.uach.cl}

Junio 26, 2023

Resumen

Nuestro trabajo se basó en la resolución de dos problemas algorítmicos utilizando diferentes estrategias. El primer problema involucra calcular formas de subir una escalera, evitando escalones rotos y con saltos restringidos. Se aplican soluciones de Fuerza Bruta y Programación Dinámica. El segundo problema busca encontrar la ruta más económica entre capitales continentales y de un archipiélago, considerando ciudades, islas y rutas marítimas. Se proporciona una descripción en Fuerza Bruta y se implementa un algoritmo greedy. Para ambos problemas se presenta pseudocódigo relevante, tiempo de ejecución, gráficos y respuestas a preguntas predefinidas. Además, se explica por qué el enfoque de Programación Dinámica y, respectivamente, Greedy es superior al enfoque de Fuerza Bruta en términos de eficiencia y calidad de la solución. Se concluye con la validación de las hipótesis iniciales.

En este informe se utiliza una abreviatura para referirse a las estrategias utilizadas. “Fuerza Bruta” se representa como “FB” y “Programación Dinámica” como “PD”. Esta notación permite distinguir claramente ambas estrategias en el documento, facilitando una comunicación precisa.

Índice

1. Introducción

En los cursos previos y actuales, se reconoce los algoritmos y las estructuras de datos como inseparables. Utilizaremos esta relación para resolver los problemas propuestos, proponiendo soluciones eficientes que serán implementadas en C++ para lograr un mejor performance empírico.

El problema 1, titulado: "Subiendo la Escalera con Programación Dinámica", nos enfrentamos al desafío trivial en nuestra vida cotidiana de subir una escalera de n escalones, donde aleatoriamente r de ellos están rotos y no pueden ser pisados. Además, debemos tener en cuenta que solo podemos avanzar en saltos de potencias de p, es decir, siguiendo las habilidades de *Super Mario*. Nuestro objetivo es determinar la cantidad de formas posibles de alcanzar el n-ésimo escalón sin pisar los dañados. Para resolver este problema, usaremos dos enfoques: una solución basada en FB y otra basada en PD.

El problema 2, titulado: "Ruta en Grafos Mediante Algoritmos Greedy", tenemos la misión de encontrar la ruta más económica entre dos capitales: una ubicada en el continente y otra en un archipiélago. El país está conformado por ciudades y rutas, mientras que el archipiélago tiene islas y rutas marítimas. Para abordar este desafío, consideraremos un grafo dirigido $G = (V, E)$, que representa el continente y otro grafo no dirigido $G' = (V', E')$, que representa el archipiélago, cada nodo es una ciudad o isla, dependiendo del grafo, y las rutas terrestres o rutas marítimas son las aristas. Nuestro objetivo es encontrar la ruta de costo mínimo entre la capital continental s y la capital regional z del archipiélago, teniendo en cuenta la existencia de puertos marítimos en algunas ciudades y la limitación de islas habilitadas para recibir barcos desde el continente. Para resolver este problema, usaremos dos enfoques: una descripción de algoritmo en FB y otra basada en algoritmo greedy.

El objetivo de este informe es demostrar nuestra capacidad para aplicar la teoría en la práctica, presentando soluciones respaldadas por un análisis sólido. Concluiremos con observaciones obtenidas durante la experimentación y evaluaremos nuestras hipótesis iniciales en base a los resultados. Finalmente, compartiremos nuestra experiencia en el desarrollo de este trabajo.

2. Hipótesis

2.1. Problema 1

Nuestra hipótesis es que el algoritmo de PD será más eficiente que el algoritmo de FB debido a su capacidad para evitar cálculos repetitivos. Aunque PD requiere para un costo en memoria, creemos que su beneficio en tiempo compensará ese costo. Sin embargo, pese a esta suposición, esperamos que el algoritmo de PD no pueda resolver este problema en menos de $O(n)$ debido a la necesidad de recorrer la escalera completa para alcanzar la cima. Por otro lado, FB explorará todas las combinaciones posibles y repetirá los mismos cálculos una y otra vez. Este enfoque es similar al problema clásico estudiado en clases, como el cálculo de la serie Fibonacci. Por lo tanto, esperamos que el tiempo de ejecución de FB siga una tendencia exponencial $O(c^n)$, al igual que el problema conocido ya mencionado.

2.2. Problema 2

Creemos que por el enunciado del problema, necesitamos implementar una variante del algoritmo greedy de Dijkstra (visto en clases) unas cuantas veces para encontrar la ruta mas económica que une s y z. Sabemos que Dijkstra se puede utilizar tanto como para grafos dirigidos, que es el caso del grafo del continente, como para grafos no dirigidos, como es el caso del archipiélago. La única condición es que el peso de las aristas sea positivo. Por lo mismo, en la parte donde se unen los puertos del continente, con las islas del archipielago, tendremos que buscar otro método, ya que estas aristas pueden tener costo negativo.

3. Subiendo la Escalera

3.1. Algoritmo de FB

Desarrollamos un algoritmo recursivo que implementa FB al problema en C++. A continuación, se presenta el pseudocódigo que representa dicho algoritmo.

Algoritmo 1: Cálculo de formas posibles de subir la escalera con Fuerza Bruta.

```
formasFB(n, k, B[1..k], R[1..r])
    if n = 0 then
        return 1;
    end
    total ← 0;
    for i = 1 to k do
        if n - B[i] ≥ 0 and n ∉ R then
            total ← total + formasFB(n − B[i], k, B, R);
        end
    end
    return total;
```

El algoritmo recibe 4 parámetros:

- n: Tamaño de la escalera que se desea subir.
- k: Largo del arreglo B.
- $B[1..k]$: Arreglo de enteros que contiene las k potencias de p, los cuales son los únicos valores posibles para saltar y llegar a la cima de la escalera, verificando $p^k \leq n$.
- $R[1..r]$: Arreglo de enteros que almacena los r escalones rotos los cuales no pueden ser pisados, cada r_i es distinto entre sí, $\forall i \in \{1..r\}$. En la implementación realizada, se verifica que el valor de estos índices estén en el rango válido de la escalera, es decir, $1 \leq r_i \leq n-1$. Esto es ya que si el escalón n estuviera roto, la misión sería directamente imposible.

A continuación una breve explicación del algoritmo:

- Si $n = 0$, hemos alcanzado la cima de la escalera. Esta es la condición de borde, la cual es necesaria para un algoritmo recursivo.
- Si $n \neq 0$, se salta la sentencia if y ejecuta el bucle for que itera k veces, es decir, sobre todos los elementos de B. Una vez en el loop, se verifica si es posible restar el valor actual $B[i]$ a n. Esto se traduce con $n - B[i] > 0$. Además se comprueba que el valor actual de $n \notin R$. Si ambas condiciones se cumplen, se realiza una nueva llamada recursiva. Esta vez actualizando $n - B[i]$ como nuevo n, mientras que los demás parámetros se mantienen sin cambios.
- Finalmente, retorna la variable $total$, la cual es un contador que comienza en cero y luego va incrementando con la cantidad formas que se puede subir la escalera.

Para obtener el tiempo asintótico del algoritmo analizamos lo siguiente:

- El número de llamadas recursivas en el algoritmo depende de tres factores: la profundidad de la recursión determinada por el parámetro n, el bucle que itera k veces en cada nivel de recursión y la cantidad de r escalones rotos. En el nivel más profundo de recursión, se realizan k^0 llamadas recursivas, en el siguiente nivel k^1, y así sucesivamente hasta k^{n-r}. Notar que debemos restar los r escalones rotos, ya que son las veces que la recursión no se invoca de nuevo.

$\therefore$ Se concluye que $\boldsymbol{T(n,k,r) = O(k^{n-r})}$ $\rightarrow$ Orden Exponencial.

3.2. Algoritmo de PD

Desarrollamos el algoritmo que implementa PD al problema en C++. A continuación, se presenta el pseudocódigo que representa dicho algoritmo.

Algoritmo 2: Cálculo de formas posibles de subir la escalera con Programación Dinámica.

```
formasPD(n, k, B[1..k], R[1..r])
    aux ← Arreglo[1..n+1] de enteros;
    aux[u] ← 0, ∀u ∈ {1..n + 1};
    aux[1] ← 1;
    for i = 2 to n do
        for j = 1 to k do
            if i - B[j] ≥ 0 and i ∉ R then
                aux[i] ← aux[i] + aux[i − B[j]];
            end
        end
    end
    return aux[n];
```

Este algoritmo recibe los mismos 4 parámetros que el algoritmo de FB explicado en la sección 3.1. A continuación una breve explicación:

- Se crea un arreglo $aux[1..n+1]$ de enteros, inicializando cada celda con ceros. El cual se utilizará para alamacenar los resultados parciales. Luego se asigna $aux[1] = 1$, ya que es la única forma de subir una escalera de tamaño 1.
- Luego se ejecuta un bucle anidado de claro costo $O(n \cdot k)$. El segundo for itera sobre los elementos de B y se verifica si se puede restar $B[j]$ a i, evitando números negativos. Así, $B[j]$ es apto para ser utilizado en la creación de una nueva forma. Además, se comprueba que $i \notin R$, lo cual tiene costo $O(r)$, de lo contrario, si $i \in R$, se excluye inmediatamente. Si se cumplen ambas condiciones se incrementa el valor de $aux[i]$ sumándole $aux[i - B[j]]$. Esto se hace para acumular todas las formas posibles de construir el numero i utilizando los valores en B, evitando los valores de R.
- Finalmente, se retorna $aux[n]$. Este número representa la cantidad de formas posibles de construir el número n, lo que viene siendo nuestra escalera.

$\therefore$ Es claro que $\boldsymbol{T(n,k,r) = O(n \cdot k \cdot r)}$ $\rightarrow$ Linealmente dependiente de n, k y r a la vez.

3.3. Experimentación

Después de implementar los algoritmos FB y PD en C++, procedimos a ejecutar el código para distintas entradas de n, p y r. Es importante destacar que ambos algoritmos están incluidos en el archivo *SubiendoEscalera.cpp*, lo cual nos permite utilizar la misma entrada para comparar su eficiencia y velocidad. Para obtener medidas más precisas del tiempo de ejecución, hemos incorporado la biblioteca *ctime*.

De manera inmediata, notamos la evidente lentitud del algoritmo FB en comparación con PD. Para valores grandes de n y valores pequeños de p y r, siempre existirán numerosas formas de llegar al final de la escalera para dichas entrada, independientemente de la ubicación de los r escalones rotos. Esto, en ciertas ocasiones, generaba un desbordamientos, o más conocido como $Overflow$.

El método $formasFB(...)$ lo definimos de tipo int, ya que para valores grandes de n, el tiempo de ejecucion es extremadamente grande, por lo que no es factible ejecutarlo en la práctica. Incluso, la consola de ejecución nos advertia la ineficiencia del algoritmo, ya que nos mostraba el conocido mensaje de $Terminado(Killed)$. Esto no significa que el algoritmo este incorrecto, sino que nunca terminará de ejecutarse y el sistema operativo decide terminar la ejecución, debido a que ha consumido demasiados recursos como memoria o tiempo.

El método $formasPD(...)$, decidimos que sea del tipo de dato unsigned long long int (recordemos que C++ es de tipado fuerte), el cual proporciona un rango mucho mas amplio para almacenar el valor correspondiente en la variable $aux[n]$. Al utilizar este tipo de dato, evitamos valores negativos y generar $Overflow$ para muchos mas valores de n, así podremos trabajar con números considerablemente más grandes.

3.4. Presentación y Análisis de Resultados

Se realizaron pruebas ejecutando el algoritmo de **Fuerza Bruta**, el cual nos entregó los resultados para generar los gráficos que se muestran a continuación.

Para la Figura 1, exploramos el impacto de los r escalones rotos en el tiempo de ejecución del algoritmo. Mantuvimos constantes los valores de n y p, mientras incrementamos el valor de r de uno en uno.

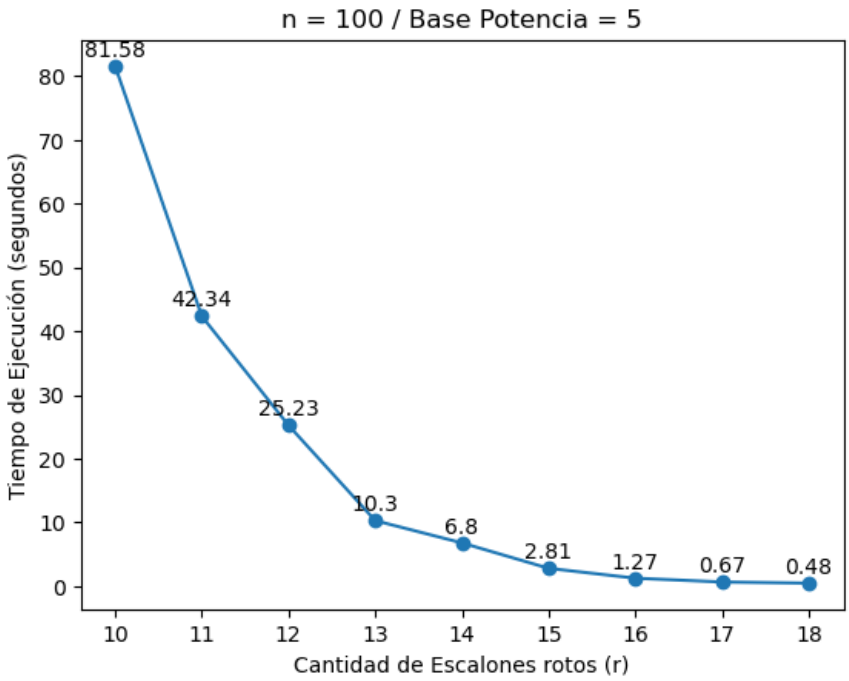

Figura 1: Gráfico de algoritmo FB, r v/s t

El gráfico se presenta en escala lineal, lo cual permite interpretar inmediatamente la curva resultante como exponencial, es decir, mientras el valor de r aumenta, el tiempo de ejecución disminuye de manera exponencial.

Para la Figura 2, se realizó un incremento lineal del valor de n, partiendo desde $n = 115$ hasta $n = 140$, con un incremento de 5 unidades en cada ejecución. Es importante destacar, que tanto la base de la potencia p como la cantidad de escalones rotos r, se mantuvieron constantes en todas las pruebas.

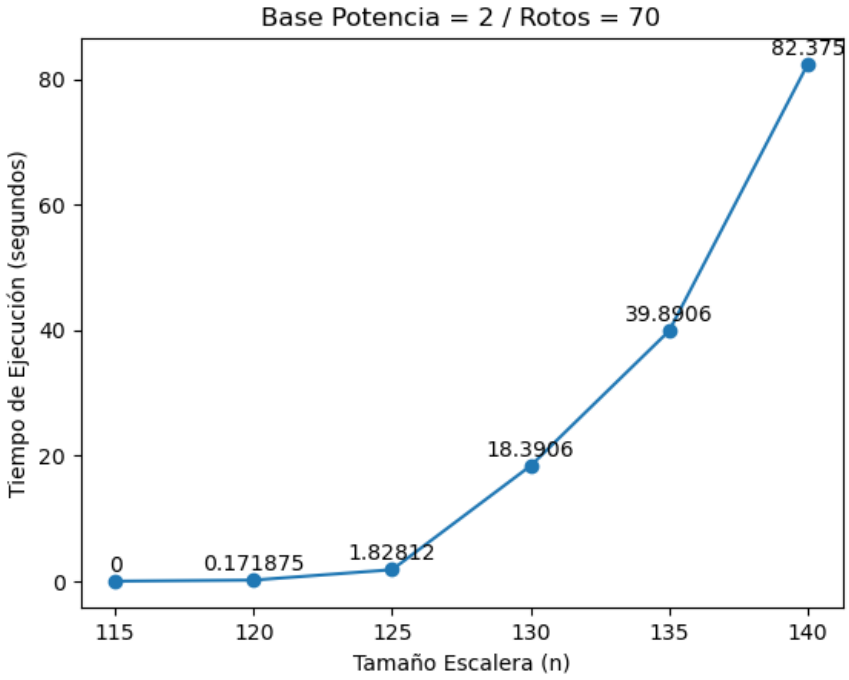

Figura 2: Gráfico de algoritmo FB, n v/s t

Al igual la Figura 1, este gráfico se encuentra en escala lineal, lo que significa que a medida que aumentamos el tamaño de la escalera (n), el tiempo de ejecución aumenta de manera exponencial.

$\therefore$ Queda demostrado gráficamente que el tiempo de ejecución asintótico de $formasFB(...)$ es $O(k^{n-r})$.

Realizamos pruebas para evaluar el rendimiento del algoritmo de **Programación Dinámica**. En el gráfico izquierdo, utilizamos valores de n que sean potencias de 10, con una base de potencia $p = 2$ y un $r = 100$ escalones rotos. En el gráfico derecho, utilizamos valores de n que sean potencias de 2, con una base de potencia $p = 2$ y un $r = 6000$ escalones rotos. Como resultado, los gráficos generados se presentaron en una escala logarítmica.

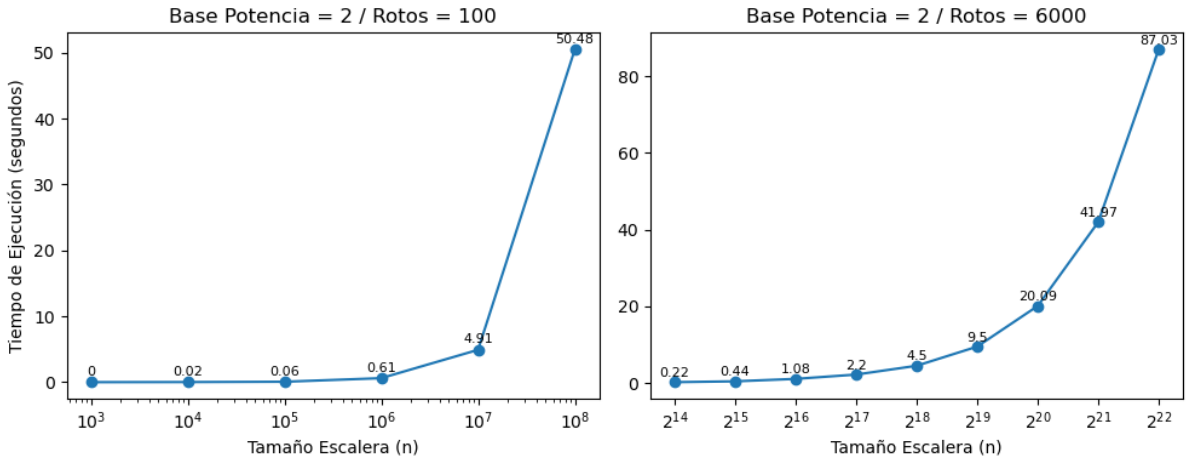

Figura 3: Gráficos de algoritmo PD, n v/s t

En los gráficos presentados en la Figura 3, se puede observar una curva exponencial. Sin embargo, debido

al uso de una escala logarítmica en el eje X, esta curva exponencial se transforma en una líneal recta. Este fenómeno es consecuencia de la naturaleza de la escala logarítmica, la cual comprime los valores más grandes en dicho eje, suavizando así el crecimiento exponencial y convirtiéndolo en un crecimiento más lineal en el gráfico.

En la Figura 4, analizamos el impacto del valor de r en el tiempo de ejecución. Mantuvimos constantes los valores de n y p, mientras aumentábamos de manera exponencial el valor de r.

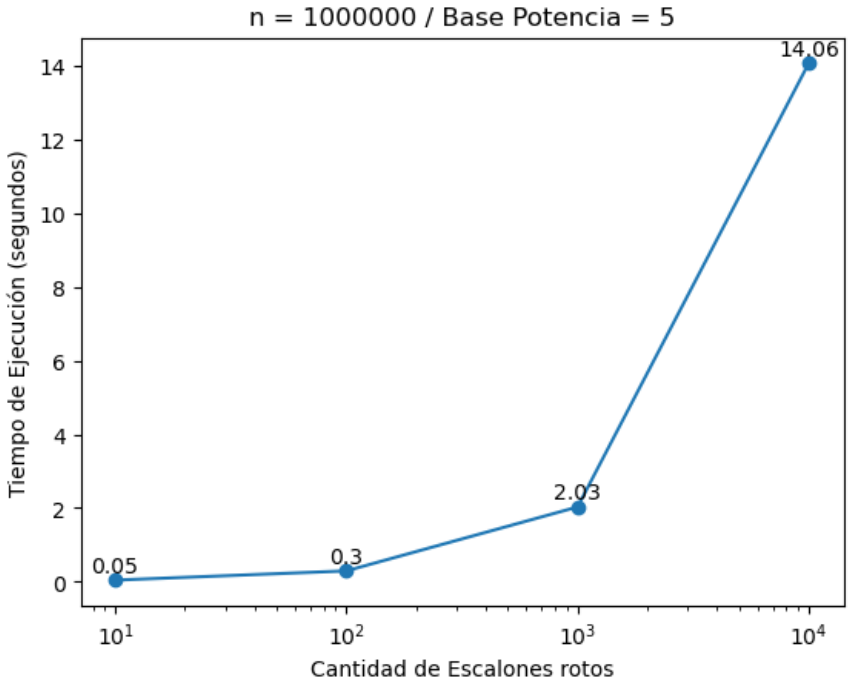

Figura 4: Gráficos de algoritmo PD, r v/s t

Al igual que en la Figura 3, este gráfico está representado en una escala logarítmica, lo cual transforma la curva exponencial en una curva líneal.

$\therefore$ Esto confirma que el tiempo el tiempo asintótico de $formasPD(...)$ es $O(n \cdot k \cdot r)$. La linealidad en los gráficos indica que un aumento en las variables n, k o r tiene un impacto lineal en el tiempo de ejecución del algoritmo.

3.5. Análisis Teórico

I) **¿Es posible esperar un compartamiento muy diferente en la eficiencia de la PD si el valor de r es muy pequeño o muy grande?**
R: Sí. Como demostramos, $formasPD(...) = O(n \cdot r)$ para un valor constante de k. Esto implica que para valores pequeños de r, el algoritmo se comporta de manera casi lineal, lo que lo convierte en un algoritmo altamente eficiente. Sin embargo, a medida que r es más grande y se acerca relativamente a n, el algoritmo PD tiende hacía un tiempo de ejecución cuadrático. Pese a esto, sigue siendo más eficiente que el enfoque de FB, ya que $formasFB(...) = O(k^{n-r})$. Este caso exponencial es uno de los peores en la clasificación de algoritmos, siendo superado en terminos de ineficiencia solo por los algoritmos de orden factorial $O(n!)$.

II) **¿Qué se espera en los casos que el valor de la potencia p sea muy grande o pequeño?**
R: Cuando la base de la potencia p es más grande, el tiempo de ejecución disminuye. Esto se debe ya que al aumentar p, se generan menos potencias que son menores o iguales que n. Por ejemplo, si $n = 90$

y $p = 2$, las potencias generadas serían $B = \{1, 2, 4, 8, 16, 32, 64\}$. Sin embargo, si $n = 90$ y $p = 10$, las potencias serían $B = \{1, 10\}$.

En este caso, el tamaño del arreglo B, que se denota como k, es más pequeño cuando p es más grande. Esto debido a que al tener una base mayor, se requieren menos potencias para alcanzar o superar el valor de n. Por lo tanto, el número de elementos k se reduce.

III) **Explique de manera clara y objetiva por qué la utilización de la programación dinámica resulta beneficiosa para la solución de este problema.**
R: Explicaremos 2 razones por las cuales PD es el mejor enfoque al problema:

- Evita el recálculo de soluciones: Al utilizar PD, se almacenan los resultados parciales de subproblemas previamente resueltos en un arreglo auxiliar. Esto evita tener que recalcular las mismas soluciones una y otra vez, lo que reduce drásticamente el tiempo de ejecución. En este problema al almacenar la cantidad de formas de subir los escalones anteriores, se evita repetir los cálculos para esos subproblemas.
- Permite resolver problemas más grandes: La PD proporciona una solución más eficiente en términos de tiempo de ejecución, lo que permite abordar una entrada más grandes en un tiempo razonable. En el caso de subir una escalera, si el número n de escalones es grande, el enfoque de fuerza bruta se vuelve inviable, debido a su alta complejidad asintótica.

3.6. Conclusión Problema 1

Al comparar el algoritmo de Programación Dinámica con el de Fuerza Bruta, creemos que todos los fundamentos presentados en esta sección respaldan nuestra hipótesis planteada en **2.1**, la cual afirmaba que el algoritmo de PD sería más eficiente que el algoritmo de FB, debido a su capacidad para aprovechar la naturaleza repetitiva del problema y evitar recalcular soluciones conocidas.

Al analizar los resultados obtenidos, hemos observado que el algoritmo de PD se comporta mejor que FB, aunque requiere un costo adicional en memoria para almacenar las soluciones parciales, este costo se ve claramente compensado por el beneficio en términos de tiempo. Por otro lado, el algoritmo de FB, como su nombre lo indica, debía explorar todas las combinaciones posibles y repetir los mismos cálculos. Esto confirma la similitud entre este problema y el problema de Fibonacci, donde el tiempo asintótico aumenta de manera exponencial de acuerdo al tamaño de la entrada.

Este estudio contribuye al campo de la optimización de algoritmos y nos brinda una comprensión más profunda de cómo seleccionar y diseñar algoritmos eficientes en situaciones donde la naturaleza repetitiva del problema puede ser aprovechada.

4. Ruta en Grafos

4.1. Descripción Algoritmo FB

Una solución utilizando un algoritmo de Fuerza Bruta para este problema, sería probar todas las posibles combinaciones de rutas desde la capital continental s hasta la capital regional del archipiélago z y calcular el costo total de cada una.

El algoritmo de Fuerza Bruta seguiría estos pasos:

- Generar todas las combinaciones posibles de rutas desde s hasta z, teniendo en cuenta las ciudades que contengan puertos y las islas del archipiélago que esten habilitados para recibir barcos.
- Para cada combinación de rutas, calcular el costo total sumando el costo de las rutas terrestres y el costo de la ruta marítima correspondiente.
- Comparar los costos totales de todas las combinaciones y quedarnos con la ruta de menor costo.
- Finalmente retornar la ruta mas económica.

Debemos tener en cuenta que el enfoque de FB para resolver este problema puede llegar a ser ineficiente en terminos de tiempo de ejecución, sobretodo cuando la cantidad de ciudades, islas y combinaciones posibles son muy grandes. En esos casos sería mejor opcion utilizar algoritmos mas eficientes, como búsqueda en anchura (BFS) o el algoritmo de Dijkstra.

4.2. Descripción Algoritmo Greedy

Hemos propuesto un algoritmo greedy para resolver este problema que se basa en una variante del algoritmo de Dijkstra visto en clases. Este algoritmo permite calcular la distancia mínima entre un nodo inicial y todos los demás nodos en un grafo. Este algoritmo funciona en grafos con pesos en sus aristas, ya sean **dirigidos o no dirigidos**. Sin embargo, es importante destacar que el algoritmo **no puede manejar aristas con pesos negativos**. Por lo tanto, el requisito fundamental es que todas las aristas del grafo deben tener pesos no negativos para su correcto funcionamiento.

Utilizaremos el algoritmo de Dijkstra, el cual invocaremos en multiples ocasiones. Este algoritmo se nombra de la siguente forma: $Dijkstra(G, x, \omega(.))$, en donde $G = (V, Ad)$ y Ad es la matriz de adyacencia, x es el nodo inicial y la función de costo $\omega(.) : E \to \mathbb{R}^+$. Es importante destacar que este algoritmo también es aplicable a grafos no dirigidos. En un grafo no dirigido, cada arista puede considerarse como una arista dirigida en ambas direcciones. Por lo tanto, el algoritmo puede manejar eficientemente tanto grafos dirigidos como no dirigidos sin requerir modificaciones adicionales.

A continuación se describe el algoritmo, el cual consta de 3 grandes pasos:

I)
- Ejecutar el algoritmo de Dijkstra en el grafo $G = (V, E)$ con la función de costo ω, comenzando desde el nodo s. Esto nos entregará las rutas de costo mínimo para llegar desde s a cada puerto p_i donde $1 \leq i \leq k$. Denotaremos el costo de s a p_i como $C(S, p_i)$
- Luegos crearemos un conjunto llamado $Puertos$ que contendrá los k costos obtenidos en el paso anterior: $Puertos = \{C(s, p_1), C(s, p_2), \ldots, C(s, p_k)\}$. Cada elemento de este conjunto representará el costo mínimo de llegar desde s hasta un puerto específico p_i.
- El tiempo de ejecución de esta parte del algoritmo es $O((n + |E|) \cdot \log n)$, donde $n = |V|$ es el número de ciudades del continente y $|E|$ es el número de aristas en el grafo. Esto se debe al uso de Dijkstra, el cual emplea una estructura de datos como una cola de prioridad basada en heap para mantener y actualizar las distancias mínimas durante el proceso.

II)
- Ejecutar el algoritmo de Dijkstra en el grafo $G' = (V', E')$, donde $|V'| = m$, con la funcion de costo ω', comenzando desde cada una de las islas q_j donde $1 \leq j \leq \lfloor \log m \rfloor$. Esto implica realizar el algoritmo de Dijkstra $\log m$ veces, una vez para cada isla q_j. Denotaremos el costo de q_j a z como $C(q_j, z)$
- Luego crearemos un conjunto llamado $Islas$ que contendrá los $\log m$ costos obtenidos en el paso anterior: $Islas = \{C(q_1, z), C(q_2, z), \ldots, C(q_t, z)\}$, donde $t = \lfloor \log m \rfloor$. Cada elemento de este conjunto representará el costo mínimo de llegar desde una isla específica hasta la isla z.
- El tiempo de ejecución de esta parte del algoritmo es $O(t^2 \cdot (m + |E'|))$, donde $t = \lfloor \log m \rfloor$ es el número de iteraciones (número de islas habilitadas para recibir barcos) y $|E'|$ es el número de aristas del grafo G'. Esto se debe a que se realizan t iteraciones del algoritmo de Dijkstra, y cada iteración tiene una complejidad de $O((m + |E'|) \cdot \log m)$.

III)
- Una vez tengamos los conjuntos $Puertos$ e $Islas$ podremos ejecutar el ultimo paso de este algoritmo. El cual consta de minimizar el costo total de la ruta desde la capital s hasta la isla z. Esto se logra calculando el costo $C(s \to p_i \to q_j \to z)$ utilizando los conjuntos previamente calculados.
- Esta última parte consiste en un doble bucle for que itera sobre los elementos de los conjuntos $Puertos$ e $Islas$. Para cada combinación de puertos e islas, se calcula el costo de la ruta de acuerdo a la fórmula $C(s \to p_i \to q_j \to z) = C(s, p_i) + costoBarco(p_i, q_j) + C(q_j, z)$. Se guarda el mínimo costo encontrado y se registra el par (i, j) correspondiente.
- El tiempo de ejecución de esta parte del algoritmo es $O(k \cdot \log m)$, donde k es la cantidad de puertos y m es la cantidad de islas habilitadas para recibir barcos.

4.3. Pseudocódigo

Algoritmo 3: Cálculo del coste mínimo para unir s y z, algoritmo greedy

```
costoMinSZ(G, ω(.), G', ω'(.), costoBarco(.), s, z)
    Puertos = {C(s,p_1), C(s,p_2), ··· , C(s,p_k)}; # Se crea a partir de Dijkstra(G, s, ω(.))
    islas ← ∅;
    for i = 1 to log m do
        C(q_j, z) ← extraer costo de q_j a z luego de ejecutar Dijkstra(G', q_j, ω'(.));
        islas ← islas ∪ C(q_j, z);
    end
    costoMin ← +∞;
    best_i ← 0;
    best_j ← 0;
    for i=1 to k do
        for j=1 to ⌊log m⌋ do
            costo ← C(s, p_i) + costoBarco(p_i, q_j) + C(q_j, z);
            if costo ≤ costoMin then
                costoMin ← costo;
                best_i ← i;
                best_j ← j;
            end
        end
    end
    return {costo, best_i, best_j};
    # El camino de s a best_i se extrae de Dijkstra(G, s, ω)
    # El camino de best_j a z se extrae de Dijkstra(G', q_j, ω')
```

$\therefore$ El costo es $\boldsymbol{T(n,m,k) = O((n+|E|)\cdot \log n) + O((m+|E'|)\cdot \log^2 m) + O(k\cdot \log m)}$

4.4. Análisis Teórico

I) **¿Cómo afecta la variación de valores de k en el tiempo de ejecución del pseudocódigo?**
R: Este valor k influye directamente en el cálculo del costo mínimo, ya que hay un bucle for que depende de este valor. El tiempo asíntotico del algoritmo contiene la siguente expresión: $O(k \cdot \log m)$. Esto significa que es directamente proporcional al valor de k, mientras mas grande k, más veces se ejecutará el bucle, por lo tanto mayor tiempo de ejecución.

II) **¿Es posible aplicar alguna mejora a su propuesta greedy a fin de acelerar el tiempo de ejecución de su algoritmo para ciertos casos especiales?**
R: Una posible mejora sería utilizar una estructura de datos más eficiente para la cola de prioridad, como un $Heap$, el cual puede mejorar el tiempo de ejecución asintótico. Otra mejora sería preprocesar el grafo para eliminar las rutas que son claramente ineficientes. Por ejemplo, si hay dos rutas desde una ciudad a una isla y una es claramente más barata que la otra, podríamos eliminar la ruta más cara del grafo antes de ejecutar el algoritmo.

4.5. Experimentación y Análisis de Resultados

Realizamos un experimento para analizar el impacto del valor de k en este problema. mantuvimos n y m constantes. A continuación, se muestra el gráfico obtenido como resultado.

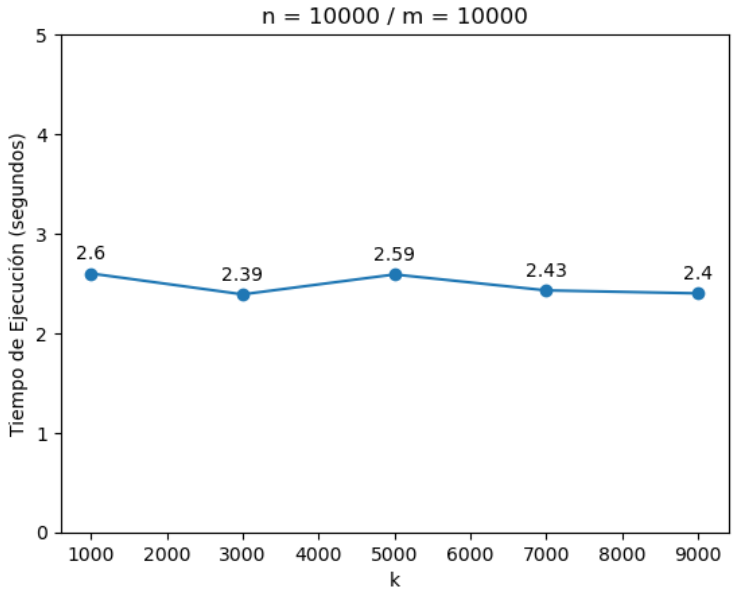

Figura 5: Gráfico de algoritmo Greedy, k v/s t

La Figura 5, nos muestra que el valor de k no es significativo en la ejecución del código. Según la teoría, se esperaba que este valor fuera relevante, pero la práctica demostró lo contrario.

A continuación realizamos otro experimento, el cual consiste en ir cambiando los valores de n para ver como afecta en el tiempo de ejecución. En este caso, mantuvimos k y m constantes.

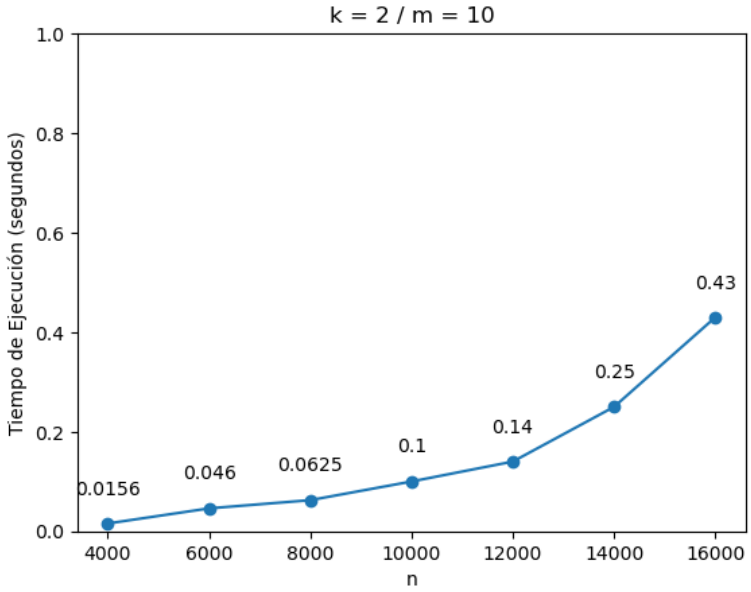

Figura 6: Gráfico de algoritmo Greedy, n v/s t

El gráfico en la Figura 6 muestra una clara dependencia del tiempo de ejecución con respecto a n. A medida que aumentamos el valor de n, esperamos ver una curva que se asemeje a una función logarítmica. Sin embargo, debido a limitaciones en nuestro hardware, no pudimos aumentar aún más el tamaño de n.

4.6. Conclusión Problema 2

Como mencionamos en la hipótesis, efectivamente tuvimos que implementar el algoritmo de Dijkstra varias veces, una vez para encontrar la ruta de mínimo costo desde s hasta cada puerto p_i, y luego $\lfloor \log m \rfloor$ veces para encontrar la ruta de mínimo costo desde cada isla q_j habilitada para recibir barcos hasta z.

Para encontrar la ruta de costo mínimo entre los puntos s y z, considerando la necesidad de utilizar un barco, nos vimos en la necesidad de implementar un bucle anidado for para evaluar todas las posibles combinaciones. En este caso, no fue posible utilizar el algoritmo de Dijkstra debido a la presencia de rutas marítimas con posibles costos negativos. Por lo tanto, requerimos de un enfoque diferente para abordar este aspecto particular del problema.

5. Conclusión Final

Este trabajo es de suma importancia para nuestra formación como profesionales de la informática, ya que nos proporciona experiencia práctica, habilidades de trabajo en equipo y una visión más sólida de los desafíos que encontraremos en nuestra carrera. A través de la resolución de estos problemas algorítmicos, pudimos aplicar los conocimientos teóricos adquiridos durante el curso y poner a prueba las habilidades del diseño y análisis de algoritmos.

La capacidad de resolver problemas de manera eficiente es fundamental en numerosas áreas, como la inteligencia artificial, la cual hoy en día está en constante evolución, así como la optimización de procesos, análisis de datos y seguridad informática, entre otras. Esto nos permitió desarrollar y mejorar nuestras habilidades en la resolución de problemas complejos, así como también en la comprensión de la eficiencia y complejidad de los algoritmos.

Además, nos brindó la oportunidad de trabajar en equipo. Aunque fuimos dos integrantes, la dedicación, el esfuerzo y tiempo invertido, que creemos que esta más que evidenciado, fueron suficiente por encima de cualquier otra consideración, incluso más allá de la nota que obtendremos. Lo que realmente disfrutamos fue el proceso de desarrollar el trabajo, aplicar lo aprendido, lo cual es fundamental, y enfrentar los desafíos que se presentaron en el camino. Estas habilidades de trabajo en equipo y colaboración son esenciales en el futuro entorno laboral que enfrentaremos, donde los proyectos suelen requerir la colaboración de varios profesionales, los cuales pueden tener diferente formación a la nuestra, y por lo tanto poseer diferentes perspectivas y conocimientos.