Tarea 1

Análisis De Algoritmo y Estructura De Datos

Integrantes:
Julio Lucero
Eduardo Berrios
Profesor: Cristian Sepulveda

Índice

1. Introducción

En el marco de la Constitución Política de Chile, las municipalidades se erigen como entidades autónomas de derecho público, dotadas de personalidad jurídica y patrimonio propio. Su misión fundamental reside en la satisfacción de las necesidades de la comunidad local y la promoción activa del progreso económico, social y cultural de sus comunas o agrupaciones de comunas. En este contexto, uno de los roles más destacados de los municipios radica en la eficaz asignación y administración de los fondos concursables, mecanismos mediante los cuales el Estado pone a disposición de la ciudadanía recursos destinados a una amplia gama de proyectos, desde emprendimientos hasta instituciones educativas, fundaciones, organizaciones no gubernamentales, juntas de vecinos y diversas instancias de la sociedad civil.

La optimización de la asignación de estos recursos se convierte en un imperativo para los administradores municipales, quienes se esfuerzan por evaluar con precisión el Retorno Social de la Inversión (SROI, por sus siglas en inglés) de cada proyecto. Este indicador se erige como un criterio crucial en la selección de aquellos proyectos que serán financiados, ya que no solo busca maximizar la eficiencia en el uso de los recursos públicos, sino también garantizar que estos se traduzcan en beneficios tangibles y sostenibles para la comunidad.

El desafío que enfrentan los administradores municipales se presenta en términos concretos: seleccionar, entre un conjunto de proyectos con costos y SROI asociados, un subconjunto cuya suma total de costos no supere el presupuesto municipal destinado a fondos concursables, al mismo tiempo que maximice el SROI total. En otras palabras, se trata de un proceso de toma de decisiones estratégicas que combina la asignación de recursos limitados con el objetivo de generar el mayor impacto social y económico posible.

Este informe se adentrará en el análisis de esta problemática crítica, explorando en detalle los desafíos que conlleva la selección de proyectos en el contexto municipal chileno y proponiendo una solución basada en la implementación de un código informático que permita obtener combinaciones óptimas de presupuesto, beneficio y costo. A través de esta investigación, se busca proporcionar a los administradores municipales una herramienta valiosa para la toma de decisiones más fundamentadas y eficientes en la gestión de los fondos concursables, con el fin último de contribuir al bienestar y desarrollo de las comunidades locales.

2. Método

Los materiales a utilizar es el compilador de lenguaje c en esta ocasión el que se utilizo es gcc para compilar el main del programa con una computadora con alta capacidad de procesamiento para procesar las bases de datos de los proyectos que contenían su costo y sroi asociado. El procedimiento que se uso para construir el pseudocódigo y después el código principal, primeramente se implemento un algoritmo para abrir los archivos de entrada en modo lectura, luego se utilizo un estructura de datos para almacenar la información de cada proyecto este incluye su costos y el sroi asociado. Las dificultades que tuvimos la lógica de iterar sobre todas las posibles combinaciones posibles(enumeración exhaustiva) y maximizar el sroi total sin exceder el presupuesto de la municipalidad.

También la lógica que usamos para seleccionar una combinación de proyectos y calcule la suma de sus costos para así poder verificar si este esta dentro del presupuesto, en la figura 1 se aprecia una parte de la lógica de las combinaciones y la selección de los proyectos en este caso consideramos solo 3 proyectos que serian $2^3 = 8$ combinación y así sucesivamente:

```
000  (ningún proyecto es seleccionado)
001  (solo el tercer proyecto es seleccionado)
010  (solo el segundo proyecto es seleccionado)
011  (el segundo y tercer proyecto son seleccionados)
100  (solo el primer proyecto es seleccionado)
101  (el primer y tercer proyecto son seleccionados)
110  (el primer y segundo proyecto son seleccionados)
111  (todos los proyectos son seleccionados)
```

Figura 1: Lógica combinaciones

Otro problemas que se obtuvieron al procesar la información de los proyectos que tenían el sroi alto al igual que su costo, por lo que la solución a este problema es generar un algoritmo que sobre la combinación de proyectos permitiera maximizar el sroi pero manteniendo que no sobrepase el presupuesto. Se presentara el pseudocódigo de todas las funciones implementadas:

Algorithm 1 Leer información de proyectos desde un archivo

Require: Nombre del archivo a leer: nombrearchivo
Ensure: Retorna 1 si la lectura fue exitosa, 0 en caso contrario

```
archivo ← fopen(nombrearchivo, r")
if not archivo then
    return 0
end if
sscanf(nombrearchivo, "knapPI_x_x.txt", &n, &presupuesto_max)
proyecto ← malloc(sizeof(Proyecto) *n)
for i = 0 to n - 1 do
    fscanf(archivo, " %d %d", &proyecto[i].sroi, &proyecto[i].costos)
end for
fclose(archivo)
return 1
```

Algorithm 2 Proyectos seleccionados

Require: Mejor conjunto de proyectos: mejor_conjunto, Conjunto de proyectos: proyecto, Número de proyectos: n

```
costo_total ← 0
sroi_total ← 0
for j = 0 to n - 1 do
    if mejor_conjunto & 1 ≪ j then
        print "Proyecto", j+1, çon SROI", proyecto[j].sroi, ż costo", proyecto[j].costos {Imprime detalles del proyecto}
        sroi_total += proyecto[j].sroi
        costo_total += proyecto[j].costos
    end if
end for
print "Total SROI:", sroi_total, "Total Costos:", costo_total {Imprime los totales}
```

Algorithm 3 Solución de Proyectos

Require: $proyecto, n, presupuesto\_max$

Ensure: $mejor\_conjunto$

```
subconjuntos ← 2^n
maximo_sroi ← 0
mejor_conjunto ← 0
for i = 0 to subconjuntos − 1 do
    sroi_actual ← 0
    costo_actual ← 0
    for j = 0 to n − 1 do
        if i AND 2^j then
            sroi_actual ← sroi_actual + proyecto[j].sroi
            costo_actual ← costo_actual + proyecto[j].costos
        end if
    end for
    if costo_actual ≤ presupuesto_max AND sroi_actual > maximo_sroi then
        maximo_sroi ← sroi_actual
        mejor_conjunto ← i
    end if
end for
return mejor_conjunto
```

Además la dificultad que tuvimos en la ejecución de archivos con proyectos superiores a 32 dado que por enumeración exhaustiva hay una combinación de aproximado 4 mil millones por ende no podíamos solucionar el problema dado que no nos imprimía solo ceros en la consola y la solución fue declararar la función de nuevo pero con otro tipo de datos que fue **LONG** en vez de **INT**.

3. Resultados y Análisis

Para los resultados obtenidos del problema a solucionar fueron exhaustivamente costosos en términos de rendimiento debido a las combinaciones de los proyectos a seleccionar, por ejemplo un archivo de 32 proyectos el computador tiene que procesar aproximadamente 4 mil millones de combinaciones, es por esto que los tiempos de ejecución y el orden de complejidad del algoritmo son muy elevados, en el caso del orden de complejidad desglosamos el algoritmo como sigue:

- $O(2^n)$
  La función solución alberga este orden de complejidad por el numero de subconjuntos posibles de proyectos, siendo n el numero de proyectos

- $O(n)$
  La función proyectos seleccionados tiene esta complejidad por la iteración sobre los proyectos para determinar su inclusión en el conjunto y acumular el sroi y los costos.

Debido el código en general para solucionar el problema es de un orden de complejidad de $O(2^n)$.

A continuación se muestran la tabla 1, de los distintos tiempos de ejecución acorde al tamaño de proyectos según respecta cada archivo entregado, estos datos son los experimentales y ejecutados con el código principal, los cuales al recibir datos de entrada de 36 se demoro 3 horas y 40 minutos.

Tabla 1: Tiempos De Ejecución

Archivos	Tiempos[s]
knapPI_6_500.txt	0.000
knapPI_8_500.txt	0.000
knapPI_10_500.txt	0.000
knapPI_12_500.txt	0.000
knapPI_14_500.txt	0.000
knapPI_16_500.txt	0.007
knapPI_18_500.txt	0.053
knapPI_20_500.txt	0.200
knapPI_24_1000.txt	3
knapPI_28_1000.txt	64
knapPI_32_1000.txt	950
knapPI_36_1000.txt	13.600
knapPI_40_1000.txt	194.766
knapPI_44_1000.txt	2.786.621
knapPI_48_1000.txt	39.887.844
knapPI_52_1000.txt	570.956.546
knapPI_56_1000.txt	8.172.699.651
knapPI_60_1000.txt	116.984.418.351
knapPI_64_1000.txt	1.674.520.626.081
knapPI_68_1000.txt	23.969.169.284.920
knapPI_72_1000.txt	343.095.849.200.399
knapPI_76_1000.txt	4.911.090.590.552.870
knapPI_80_1000.txt	7,03E+32
knapPI_84_1000.txt	1,01E+34
knapPI_88_1000.txt	1,44E+35
knapPI_92_1000.txt	2,06E+36
knapPI_96_1000.txt	2,95E+36
knapPI_100_1000.txt	4,22E+37

Se presenta el gráfico asociado a los datos mostrados de la tabla 1, con los datos experimentales de entrada del 6 al 36 en la figura 2:

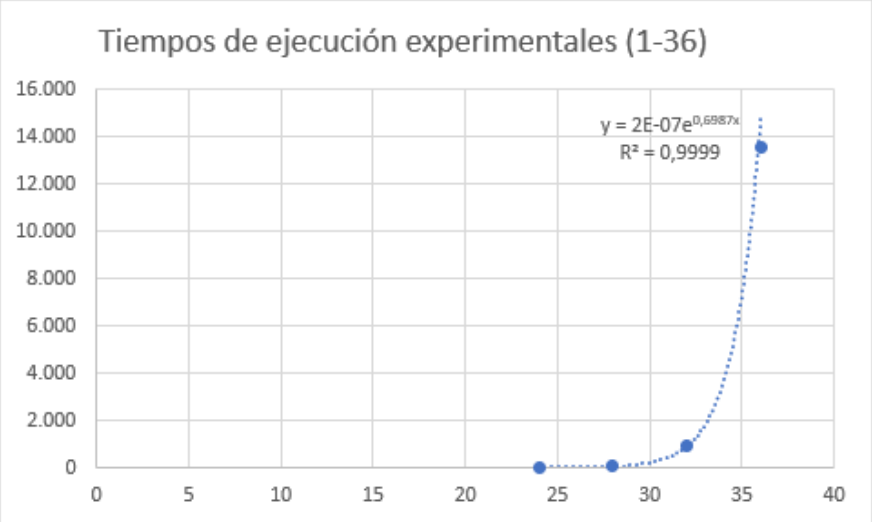

Figura 2: Gráfico Tiempo de Ejecución experimentales

Además se presenta el gráfico en la figura 3 con las predicciones y aproximaciones de los tiempos de ejecución acorde a los valores de entrada del tamaño de archivo 40 al 100.

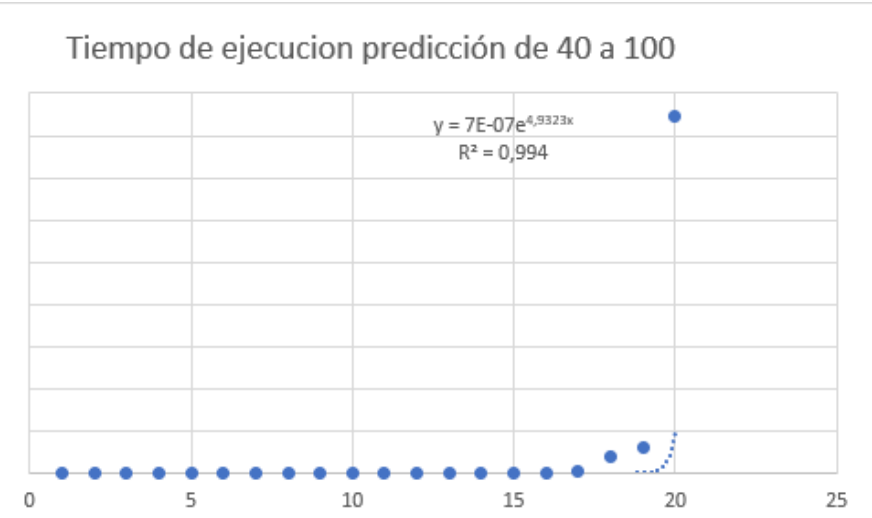

Figura 3: Gráfico Tiempo de Ejecución Predicciones

4. Discurso

- Tiempo de Ejecución Experimentales (1-36):
  El gráfico muestra una tendencia exponencial clara, tal como se esperaría para un algoritmo de complejidad 2^n y la ecuaciones proporcionada $y = 2E{-}07e^{0,587x}$, sugiere que para cada incremento en n entradas, el tiempo de ejecución aumenta exponencialmente. El coeficiente de determinación $R^2 = 0,9999$ es muy cercano a 1, lo que indica que el modelo de regresión exponencial es una excelente representación de los datos experimentales.

- Tiempo de Ejecución predicción (40-100):
  Al igual que el ítem anterior la ecuación $y = 7E{-}07e^{0,5933x}$ predice los tiempos de ejecución de manera exponencial, también se puede observar que el modelo predice un rápido crecimiento del tiempo de ejecución, al punto que para valores ligeramente superiores se vuelven extremadamente altos. El coeficiente de determinación $R^2 = 0,994$ sigue siendo muy alto, pero no como el ítem anterior, lo que sugiere que el modelo es fiable pero no como el anterior.

- Principales Hallazgos:
  Los tiempos de ejecución del algoritmo son consistentes con una función exponencial, lo que confirma la complejidad teórica de 2^n. A medida que el dato de entrada crece, incluso en pequeñas cantidades, el tiempo de ejecución aumenta de manera feroz. Esto hace que sea prácticamente inviable usar el algoritmo para datos de entrada más grandes.

- Limitaciones:
  El crecimiento exponencial del tiempo de ejecución limita la utilidad práctica del algoritmo para valores de entrada más grandes. Incluso un pequeño incremento en el tamaño de entrada puede llevar a tiempos de ejecución prohibitivos. Las predicciones más allá de cierto punto alrededor del 25 en la figura 3, carecen de puntos de datos reales para validar la precisión del modelo.

- Recomendaciones y sugerencias para futuras investigaciones:
  Optimización del Algoritmo: Sería útil investigar técnicas de optimización o enfoques alternativos que puedan reducir el tiempo de ejecución. Esto podría incluir técnicas de programación dinámica o algoritmos heurísticos que ofrezcan soluciones aproximadas en menos tiempo.

5. Conclusión

En conclusión, este informe ha abordado de manera integral la problemática relacionada con la asignación eficiente de recursos en el contexto municipal chileno. El objetivo principal fue encontrar una solución que permitiera optimizar la selección de proyectos en función de su Retorno Social de la Inversión (SROI), maximizando el impacto social y económico en un entorno con recursos limitados.

Después de un extenso proceso de investigación y desarrollo, se diseñó un algoritmo basado en combinatorias que busca la mejor solución posible. Sin embargo, al realizar pruebas y evaluaciones exhaustivas, se llegó a la conclusión de que, a medida que el número de candidatos a la solución óptima aumenta, el rendimiento computacional se deteriora significativamente. A partir de una entrada con 36 candidatos, el algoritmo comienza a experimentar demoras considerables y se requieren hasta 3 horas para obtener una respuesta.

Este hallazgo revela que el orden de complejidad del algoritmo es exponencial, específicamente de 2 a la potencia n. Esto significa que a medida que se incrementa el tamaño del problema, el tiempo requerido para encontrar la solución óptima aumenta de manera exponencial. Aunque el algoritmo es eficaz para problemas pequeños o medianos, su aplicabilidad podría ser limitada en situaciones donde la escala sea significativamente mayor.

Si bien este resultado plantea desafíos en términos de eficiencia computacional, no se debe subestimar la utilidad del algoritmo en contextos más manejables y la valiosa información que puede proporcionar a los administradores municipales para tomar decisiones más informadas.

En última instancia, este informe destaca la importancia de seguir explorando soluciones innovadoras y abordar los desafíos computacionales en la búsqueda de un equilibrio entre la eficiencia y la toma de decisiones bien fundamentadas. La investigación realizada representa un paso hacia adelante en la búsqueda de soluciones más efectivas para la gestión de fondos concursables en las municipalidades chilenas y, sin duda, sienta las bases para futuros desarrollos que puedan superar las limitaciones computacionales identificadas en este estudio.

6. Apéndice

Este apéndice tiene como objetivo proporcionar al lector una guía detallada sobre cómo ejecutar adecuadamente la implementación proporcionada. Aquí se especificarán tanto el formato de los parámetros de entrada como el de las salidas, garantizando un uso eficiente y correcto del programa. Además, se describe cada una de las funcionalidades del software y se mencionan posibles errores que podrían surgir durante su operación. Para una guía visual y paso a paso, por favor consulte el manual de usuario y el vídeo de la explicando la estrategia propuesta , justificando la estructura de datos y la demostración del funcionamiento del programa, https://drive.google.com/drive/folders/1LphR24qgtpDp_nRPIaS_rIw06MFknJ3O.