Informe del Proyecto Final: Programación Funcional para el Riego Óptimo

Informe de Proyecto

Proyecto: Fundamentos de Programación Funcional y Concurrente

Universidad del Valle

Integrantes del grupo:

Juan Esteban Franco López 2259475

Andrés Eduardo Narváez Cañas 2259545

Alejandro Garzón Mayorga – 2266088

Andres Felipe Chaparro - 2266252

Fecha: 17 Diciembre de 2024

GitHub: https://github.com/AlejandroGarz/Proyecto

Introducción

En este informe trabajamos la implementación de un conjunto de funciones en Scala para resolver el problema del **riego óptimo** de tablones de cultivo en una finca. Estas funciones se diseñaron para calcular la programación de riego que minimice el costo total, considerando los costos de movilidad y las prioridades de cada tablón.

También se implementaron las funciones secuenciales y paralelizadas con el objetivo de optimizar el rendimiento

Se describen las funciones implementadas, su funcionamiento y casos de prueba.

Descripción de Funciones Implementadas

Generación de Entradas Aleatorias

- **fincaAlAzar(long: Int): Finca**

Descripción: Esta función genera una finca con un número determinado de tablones (“long”). Cada tablón se caracteriza por:

- Tiempo de supervivencia: aleatorio entre 1 y long * 2.
- Tiempo de regado: aleatorio entre 1 y long.
- Prioridad: aleatorio entre 1 y 4.

Ejemplo del proceso: Si long = 3, la función podría generar la finca:

Vector((4, 2, 3), (6, 1, 2), (10, 3, 4))

- **distanciaAlAzar(long: Int): Distancia**

Descripción: Crea una matriz simétrica de distancias entre tablones, con valores aleatorios entre 1 y long * 3. La diagonal principal contiene ceros.

Ejemplo del proceso: Si long = 3, podría generar:

```
Vector(
  Vector(0, 4, 6),
  Vector(4, 0, 2),
  Vector(6, 2, 0)
)
```

Exploración de Entradas

- **tsup(f: Finca, i: Int): Int**

Descripción: Devuelve el tiempo de supervivencia del tablón i.

- **treg(f: Finca, i: Int): Int**

Descripción: Devuelve el tiempo de regado del tablón i.

- **prio(f: Finca, i: Int): Int**

Descripción: Devuelve la prioridad del tablón i.

Ejemplo del proceso: Dada la finca:

Vector((4, 2, 3), (6, 1, 2), (10, 3, 4))

- tsup(f, 0) devuelve 4.
- treg(f, 1) devuelve 1.
- prio(f, 2) devuelve 4.

Cálculo del Tiempo de Inicio de Riego

- **tIR(f: Finca, pi: ProgRiego): TiempoInicioRiego**

Descripción: Calcula el tiempo de inicio de riego para cada tablón, según una programación pi.

Funcionamiento:

- El primer tablón en la programación empieza a regarse en tiempo 0.
- Cada tablón posterior comienza su riego cuando termina el riego del anterior.

Ejemplo del proceso:

Finca:

Vector((4, 2, 3), (6, 1, 2), (10, 3, 4))

Programación de riego:

Vector(2, 0, 1)

Resultado:

Vector(2 -> 0, 0 -> 3, 1 -> 9)

Cálculo de Costos

- **costoRiegoTablon(i: Int, f: Finca, pi: ProgRiego): Int**

Descripción: Calcula el costo de regar un tablón considerando:

Penalización si el riego comienza después del tiempo de supervivencia.

Ejemplo del Proceso:

Finca = Vector((4, 2, 3), (6, 1, 2), (10, 3, 4))

Programación de riego = Vector(1, 2, 0)

costoRiegoTablon(0, finca, pi) => 2

- **costoRiegoFinca(f: Finca, pi: ProgRiego): Int**

Descripción: Suma los costos de riego de todos los tablones.

- **costoMovilidad(f: Finca, pi: ProgRiego, d: Distancia): Int**

Descripción: Calcula el costo de mover el sistema de riego según la matriz de distancias.

Ejemplo del proceso: Dada una programación pi = Vector(0, 2, 1) y la matriz de distancias:

```
Vector(
  Vector(0, 4, 6),
  Vector(4, 0, 2),
  Vector(6, 2, 0)
)
```

El costo de movilidad es 6 + 2 = 8.

Generación de Programaciones de Riego

- **generarProgramacionesRiego(f: Finca): Vector[ProgRiego]**

Descripción: Genera todas las permutaciones posibles de programaciones de riego para los tablones.

Ejemplo del proceso: Finca:

Vector((4, 2, 3), (6, 1, 2))

Resultado:

Vector(Vector(0, 1), Vector(1, 0))

Programación de Riego Óptima

- **ProgramacionRiegoOptimo(f: Finca, d: Distancia): (ProgRiego, Int)**

Descripción: Hace la programación de riego que minimice la suma de costos de riego y de movilidad.

Funcionamiento:

- Genera todas las programaciones posibles.
- Calcula los costos de riego y movilidad para cada programación.
- Devuelve la programación con el menor costo total.

Ejemplo del proceso: Finca:

Vector((4, 2, 3), (6, 1, 2))

Matriz de distancias:

```
Vector(
  Vector(0, 4),
  Vector(4, 0)
)
```

Resultado:

(Vector(1, 0), 10)

- **Generación de Programación de Riego Paralelizada**

Descripción: Esta función se encarga de optimizar el riego en diferentes áreas de cultivo utilizando paralelismo buscando así hacer el proceso más rápido y eficiente. El objetivo al utilizar paralelización es determinar cuánto y cuándo regar cada área, asegurando un uso adecuado del agua, ahorrando recursos y maximizando la productividad de los cultivos

Funcionamiento: Esta función genera una secuencia desde cero hasta la longitud de la finca, posteriormente cada valor de la secuencia lo incluye en un vector. Una vez en este vector se empiezan a calcular todas las permutaciones posibles las cuales van a significar todas las programaciones de riego posible. Seguidamente el vector con las posibles programaciones de riego se convierte a una estructura paralela. Posteriormente la función retorna un vector secuencial de las programaciones posibles de la finca que se pasa como parámetro

.

Ejemplo del proceso:
finca_1 = vector((1, 2, 4), (3, 4, 1), (4, 2, 3))

Posibles programaciones de riego = Vector(Vector(0, 1, 2), Vector(0, 2, 1), Vector(1, 0, 2), Vector(1, 2, 0), Vector(2, 0, 1))

Adicional: En la función actual no se utilizan las propiedades de la colección paralela (en este caso Vector), ya que no se realizan operaciones adicionales sobre esta e inmediatamente la convierte a un nuevo vector secuencial.

---

costoRiegoFincaPar

Descripción: Calcula el costo total de regar toda la finca sumando los costos individuales de cada tablón. Esta implementación aprovecha la paralelización para mejorar el rendimiento.

Parámetros:

- f: Representación de la finca (vector de tuplas).
- pi: Programación de riego (vector de índices).

Funcionamiento: La función calcula en paralelo los costos individuales de cada tablón y devuelve su suma.

Ejemplo del Proceso:
Finca = Vector((4, 2, 3), (6, 1, 2), (10, 3, 4))
Programación de riego = Vector(1, 2, 0)
Resultado => 12

---

costoMovilidadPar

Descripción: Calcula el costo total de mover el sistema de riego entre tablones de acuerdo con una matriz de distancias. Se implementa en paralelo para manejar grandes entradas de datos.

Parámetros:

- f: Representación de la finca (vector de tuplas).
- pi: Programación de riego (vector de índices).
- d: Matriz de distancias (vector de vectores).

Funcionamiento: La función suma las distancias entre los tablones según el orden definido por la programación de riego.

Ejemplo del Proceso:
Matriz de distancias:
Vector(
Vector(0, 4, 6),
Vector(4, 0, 2),
Vector(6, 2, 0)
)
Programación de riego = Vector(0, 2, 1)
Resultado => 6 + 2 = 8

Casos de Prueba

costoRiegoTablon

- **Caso 1:**
  Finca = Vector((4, 2, 3))
  Programación = Vector(0)
  Resultado esperado = 2

- **Caso 2:**
  Finca = Vector((4, 2, 3), (6, 1, 2))

Programación = Vector(1, 0)
Resultado esperado = 3

- **Caso 3:**
  Finca = Vector((4, 2, 3), (6, 1, 2), (10, 3, 4))
  Programación = Vector(2, 1, 0)
  Resultado esperado = 3

---

costoRiegoFincaPar

- **Caso 1:**
  Finca = Vector((4, 2, 3))
  Programación = Vector(0)
  Resultado esperado = 2

- **Caso 2:**
  Finca = Vector((4, 2, 3), (6, 1, 2))
  Programación = Vector(1, 0)
  Resultado esperado = 5

- **Caso 3:**
  Finca = Vector((4, 2, 3), (6, 1, 2), (10, 3, 4))
  Programación = Vector(2, 1, 0)
  Resultado esperado = 8

---

costoMovilidadPar

- **Caso 1:**
  Matriz = Vector(
  Vector(0, 4),
  Vector(4, 0)
)
  Programación = Vector(0, 1)
  Resultado esperado = 4

- **Caso 2:**
  Matriz = Vector(
  Vector(0, 4, 6),

Vector(4, 0, 2),
Vector(6, 2, 0)
)
Programación = Vector(0, 2, 1)
Resultado esperado = 8

- **Caso 3:**
  Matriz = Vector(
  Vector(0, 3, 5),
  Vector(3, 0, 7),
  Vector(5, 7, 0)
)
  Programación = Vector(1, 2, 0)
  Resultado esperado = 10

Comparación de rendimiento

Tamaño de la finca (#)	Funcion	Tiempo secuencial (ms)	Tiempo paralelo (ms)
10	generarProgramacionesRiego	1944.652	2,544.603
5	generarProgramacionesRiego	1.948	94,498
10	costoRiego	18.60	82.312
5	costoRiego	17.717	83.89
10	programacionRiegoOptimo	14804.278	38364.517
5	programacionRiegoOptimo	32.799	84.783

En el problema del riego óptimo no se evidencia que el paralelismo represente una mejora en el rendimiento por diferentes razones:

Se probó con máximo fincas de tamaño 10 por la limitación de memoria al momento de las permutaciones; es probable que para fincas más grandes represente una optimización.

Al manejar colecciones paralelas con tamaños pequeños pequeños, paralelizar representa una desventaja en cuanto a dividirlos en los hilos de la CPU, por lo que la versión secuencial resulta más eficiente.

Conclusiones

Como conclusión pudimos hacer la implementación funcional, permitiendo modelar soluciones complejas como el riego óptimo de una finca de manera eficiente.
La generación de programaciones y el uso de matrices de distancias permitieron evaluar las distintas funciones de riego.
Las funciones implementadas son reutilizadas para implementar la paralelización, mejorando así el desempeño en entradas grandes.