INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ANÁLISIS Y DISEÑO DE ALGORITMOS

PRÁCTICA 09

Rodrigo García Mayorga

2012630554

Introducción

Este programa utiliza varios conceptos de programación, pero principalmente:

1. Programación Dinámica (Dynamic Programming):

- Se usa para optimizar el cálculo del agua atrapada
- Almacena resultados intermedios (los máximos a la izquierda y derecha) para evitar recálculos
- Utiliza subestructuras óptimas para resolver el problema general

2. Características adicionales:

- Gestión dinámica de memoria (mediante `malloc` y `free`)
- Programación estructurada
- Uso de arrays unidimensionales
- Operadores ternarios para simplificar comparaciones

El algoritmo tiene una complejidad:

- Temporal: O(n), donde n es el número de elementos en el array
- Espacial: O(n) por los dos arrays auxiliares (`maxIzquierda` y `maxDerecha`)

La programación dinámica es especialmente útil en este caso porque:

- Evita recalcular los máximos repetidamente
- Permite resolver el problema de manera eficiente
- Divide el problema en subproblemas más pequeños y manejables

Desarrollo

Desarrollo del Programa de Cálculo de Agua Atrapada

Descripción del Algoritmo

El programa implementa una solución para calcular la cantidad de agua que puede quedar atrapada entre elevaciones usando programación dinámica. El algoritmo se desarrolla en los siguientes pasos:

1. Estructura Principal

- Se implementa la función `calcularAguaAtrapada` que recibe:
- Un arreglo de enteros (`altura[]`) que representa las elevaciones
- El tamaño del arreglo (`n`)

2. Manejo de Memoria

```
int* maxIzquierda = (int*)malloc(n * sizeof(int));
int* maxDerecha = (int*)malloc(n * sizeof(int));
```

- Se crean dos arreglos dinámicos para almacenar:
    - Máximas alturas a la izquierda de cada posición
    - Máximas alturas a la derecha de cada posición

3. Proceso de Cálculo

El algoritmo se divide en tres fases principales:

1. Cálculo de máximos izquierdos:

```
maxIzquierda[0] = altura[0];
for (int i = 1; i < n; i++) {
    maxIzquierda[i] = (altura[i] > maxIzquierda[i-1]) ?
                      altura[i] : maxIzquierda[i-1];
}
```

2. Cálculo de máximos derechos:

```
maxDerecha[n-1] = altura[n-1];
for (int i = n-2; i >= 0; i--) {
    maxDerecha[i] = (altura[i] > maxDerecha[i+1]) ?
                    altura[i] : maxDerecha[i+1];
}
```

3. Cálculo del agua atrapada:

```
for (int i = 1; i < n-1; i++) {
    int minAltura = (maxIzquierda[i] < maxDerecha[i]) ?
                    maxIzquierda[i] : maxDerecha[i];
    if (minAltura > altura[i]) {
        aguaTotal += minAltura - altura[i];
    }
}
```

Complejidad del Algoritmo

- Temporal: O(n)

- Un recorrido para máximos izquierdos
- Un recorrido para máximos derechos
- Un recorrido final para calcular el agua

- Espacial: O(n)

- Dos arreglos auxiliares de tamaño n

Ejemplos de Uso

El programa incluye dos casos de prueba:

1. Ejemplo 1: `[0,1,0,2,1,0,1,3,2,1,2,1]`

  - Resultado: 6 unidades de agua

2. Ejemplo 2: `[4,2,0,3,2,5]`

  - Resultado: 9 unidades de agua

Consideraciones de Implementación

- Se utiliza asignación dinámica de memoria con liberación adecuada
- Se implementan validaciones básicas (n <= 2)
- Se usa el operador ternario para código más conciso
- La solución es eficiente tanto en tiempo como en espacio

Enlace del código (github):
https://github.com/LowisN/ADA_Practica9