Шинжлэх Ухаан Технологийн Их Сургууль
Мэдээлэл, Холбооны Технологийн Сургууль

F.CSM301 Алгоритмын шинжилгээ ба зохиомж

Бие даалт №2

Шалгасан багш: Д. Батмөнх
Гүйцэтгэсэн оюутан: Б.Төгөлдөр /B222270007/

Улаанбаатар хот
2024 он

Гарчиг

1. Алгоритмын тайлбар

1.1 Divide-and-Conquer

Divide-and-Conquer нь үндсэн асуудлыг дэд бодлогод хувааж, тус тусад нь шийдэж, дараа нь нэгтгэж анхны асуудлын шийдлийг олоход ашигладаг бодлого бодох арга юм.[1]

Жишээ бодлого: Merge sort алгоритмыг ашиглан массивыг эрэмбэлэх.

Энэхүү бодлогыг бодохдоо өгөгдсөн массивыг рекурсивээр 2 хэсэгт хуваан жижиг хэсэгт хуваасан массивуудыг эрэмбэлэн эрэмбэлэгдсэн массивуудыг буцаан нэгтгэж үндсэн массивыг эрэмбэлнэ.

Программчлалын C хэл дээрх бодолт

```
void mergeSort(int array[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        // Массивийг 2 хэсэгт хуваан эрэмбэлэх
        mergeSort(array, left, mid);
        mergeSort(array, mid + 1, right);

        // 2 массивийг нэгтгэх
        merge(array, left, mid, right);
    }
}

void merge(int array[], int left, int mid, int right) {
    int leftSize = mid - left + 1;
    int rightSize = right - mid;

    // Массивийг 2 хэсэгт хуваах
    int leftArray[leftSize], rightArray[rightSize];
    for (int i = 0; i < leftSize; i++)
        leftArray[i] = array[left + i];
    for (int j = 0; j < rightSize; j++)
        rightArray[j] = array[mid + 1 + j];

    // 2 хуваасан массивийг нэгтгэх
    int i = 0, j = 0, k = left;
    while (i < leftSize && j < rightSize) {
        if (leftArray[i] <= rightArray[j]) {
            array[k] = leftArray[i];
            i++;
        } else {
            array[k] = rightArray[j];
```

```
            j++;
        }
        k++;
    }

    while (i < leftSize) {
        array[k] = leftArray[i];
        i++;
        k++;
    }

    while (j < rightSize) {
        array[k] = rightArray[j];
        j++;
        k++;
    }
}
```

1.2 Dynamic Programming

Динамик программчлалыг (DP) оновчлолын асуудлуудад ашигладаг бөгөөд асуудлыг зөвхөн нэг удаа шийдэж, дараагийн тооцоололд ашиглахаар хариуг хадгалдаг ба дараа дараагийн тооцоололдоо өмнө тооцоолсон утгаа ашигладаг.[1]

Жишээ бодлого: Fibonacci дарааллын өгөгдсөн n-р тоог олох $F(0) = 0$ ба $F(1) = 1$ $n > 1$,$F(n) = F(n\text{-}1)+F(n\text{-}2)$.

Энэхүү бодлогыг бодохдоо тооцоолсон утгаа fib массивт хадгалж дараа дараагийн алхамдаа өмнө тооцоолсон утгаа ашиглах замаар бодлогын хариуг олно.

Программчлалын Java хэл дээрх бодолт

```
public int fibonacci(int n) {
    if (n == 0) {
        return 1;
    } else if (n == 1) {
        return 1;
    }
    int[] fib = new int[n + 1];
    fib[0] = 0;
    fib[1] = 1;

    for (int i = 2; i <= n; i++) {
        fib[i] = fib[i - 1] + fib[i - 2];
    }

    return fib[n];
}
```

1.3 Greedy Algorithms

Хомхойлох алгоритмууд нь үндсэн асуудлын оновчтой шийдлийг олно гэж найдан алхам тутамд тухайн агшинд оновчтой сонголтыг сонгодог алгоритм юм.[1]Гэхдээ эцсийн үр дүн үргэлж сайн байна гэж хэлж чадахгүй.
Жишээ бодлого: Тухайн өдрийн хувьцааны үнийг харуулсан массив өгөгдөнө.Массивын индекс бүр нэг өдрийг төлөөлнө. Нэг удаад нэг хувьцаа авч болно, авсан хувьцаагаа дараагийн өдөр тухайн өдрийн үнээр зарж болно. Хамгийн их ашгийг буцаа.

Энэхүү бодлогыг бодохдоо маргаашийн хувьцааны үнэ өнөөдрөөс үнэтэй эсэхийг шалгаад , үнэтэй байвал аваад зарах замаар бодсон.

```
public int maxProfit(int[] prices) {
    int max = 0;
    int start = prices[0];
    int len = prices.length;
    for(int i = 1;i<len; i++){
        if(start < prices[i]) max += prices[i] - start;
        start = prices[i];
    }
    return max;
}
```

2. Алгоритмын харьцуулалт

2.1 Recursion ба Divide-and-Conquer

Жишээ бодлого: Өгөгдсөн a тооны n зэргийг олох.

Recursion аргыг ашиглан бодсон бодолт:

```
long power(int a, int n) {
    if (n == 0)
        return 1;
    else
        return a * power(a, n - 1);
}
```

Тайлбар: a тооны n зэргийг олохдоо рекурсивээр функцыг дуудна. Функц дуудагдах бүрд n-ийн утгыг 1-р бууруулна. n = 0 болсон үед рекурсив дуудалт зогсоно.

Divide-and-Conquer аргыг ашиглан бодсон бодолт:

```
long power(int a, int n) {
    if (n == 0)
        return 1;
    else if (n % 2 == 0)
        return power(a, n / 2) * power(a, n / 2);
    else
        return a * power(a, n / 2) * power(a, n / 2);
}
```

Тайлбар:a тооны n зэргийг олохдоо n зэргийг 2 хувааж функцийг дахин дуудна. Хэрэв n нь сондгой тоо бол рекурсив дуудалтын үр дүнг a-р үржүүлнэ.

2.2 Divide-and-Conquer ба Dynamic Programming

Жишээ бодлого: Climbing Stairs. Та шатаар өгсөж байна. Оргилд хүрэхэд n алхам шаардлагатай. Та шат өгсөх бүрдээ нэг эсвэл хоёр алхам хийх боломжтой. Оргилд хүрэхийн тулд хэдэн өөр аргаар өгсөх боломжтой вэ?

Divide-and-Conquer аргыг ашиглан бодсон бодолт:

```
public class Solution {
    public int climbStairs(int n) {
    //Нэг юм уу тэг шат байвал 1 аргаар шатыг дуусгана
        if (n == 0 || n == 1) {
            return 1;
        }
        // Шатны тоог хоёр тэнцүү хэсэгт хуваана
```

```
        int left = n / 2;
        int right = n - left;
        // Хуваасан хэсгүүд дээр рекурсивээр climbStairs() функцыг дуудаж,
        // хоёр талын үржвэрийг нэмнэ
        return (climbStairs(left) * climbStairs(right)) + (climbStairs(left - 1) *
            climbStairs(right - 1));
    }
}
```

Тайлбар: Энэхүү бодлогыг бодохдоо рекурсив байдлаар бодлогыг жижиг хэсгүүдэд хуваан хариуны үржвэрийг хооронд нь нэмэх замаар бодсон.

Dynamic Programming аргыг ашиглан бодсон бодолт:

```
class Solution {
    public int climbStairs(int n) {
        //Нэг юм уу тэг шат байвал 1 аргаар шатыг дуусгана
        if (n == 0 || n == 1) {
            return 1;
        }
        //Массиваа зарлаад эхний 2 индекст утга олгоно
        int[] a = new int[n+1];
        a[0] = a[1] = 1;

        //Өмнөх 2 шатны аргын нийлбэр одоогийн шатны аргатай тэнцүү гэх замаар бодсон
        for (int i = 2; i <= n; i++) {
            a[i] = a[i-1] + a[i-2];
        }
        return a[n];
    }
}
```

Тайлбар: Энэхүү бодлогыг бодохдоо анхны утга болох **a[0] = a[1] = 1;**-г олгож дараа дараагийн утгыг тооцоолохдоо өмнө тооцоолсон 2 шатыг давах аргын нийлбэрийг **a** массивт хадгалах замаар бодлогыг бодсон.

2.3 Dynamic Programming ба Greedy algorithm

Жишээ бодлого: Best Time To Buy and Sell Stock. Хувьцааны үнэтэй массив өгөгдсөн ба тэндээс авч заран хамгийн их ашгийг хийх.

Dynamic Programming аргыг ашиглан бодсон бодолт:

```
class Solution {
    public int maxProfit(int[] prices) {
        int n = prices.length;
        if (n == 0) return 0;

        int[][] dp = new int[n][2];
```

```
        dp[0][0] = 0;            // 0-р өдөр ямар ч хувьцаагүй үед ашиг 0 байна
        dp[0][1] = -prices[0];   // 0-р өдөр хувьцаатай болбол -prices[0] буюу үнээ хасаж х

        for (int i = 1; i < n; i++) {
            // i өдөр ямар ч хувьцаагүй үеийн хамгийн их ашиг
            dp[i][0] = Math.max(dp[i-1][0], dp[i-1][1] + prices[i]);
            // i өдөр хувьцаатай үеийн хамгийн их ашиг
            dp[i][1] = Math.max(dp[i-1][1], dp[i-1][0] - prices[i]);
        }

        // Сүүлийн өдөр ямар ч хувьцаагүй байх үеийн хамгийн их ашгийг буцаана
        return dp[n-1][0];
    }
}
```

Тайлбар: Энэхүү бодлогыг бодохдоо **dp** массивт тооцоолсон утгуудыг хадгалж дараа дараагийн тооцоололдоо ашигласан.

Greedy algorithm аргыг ашиглан бодсон бодолт:

```
class Solution {
    public int maxProfit(int[] prices) {
        int max = 0;
        int start = prices[0];
        int len = prices.length;
        for(int i = 1;i<len; i++){
            if(start < prices[i]) max += prices[i] - start;
            start = prices[i];
        }
        return max;
    }
}
```

Тайлбар: Энэхүү бодлогыг бодохдоо маргаашийн хувьцааны үнэ өнөөдрөөс үнэтэй эсэхийг шалгаад , үнэтэй байвал аваад зарах замаар бодсон.

Номзүй

[1] “Geeks for geeks.” https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorit