**ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ**

**Мэдээлэл холбооны технологийн сургууль**

****

**БИЕ ДААЛТ №2**

**2024-2025 оны хичээлийн жилийн намар F.CSM301**

**Хичээл заасан багш:**   Д. Батмөнх

**Бие даалтын ажил гүйцэтгэсэн:**  С.Түвшинтөгс - B221960004

**Улаанбаатар хот**

**2024 он**

Гарчиг

* 1  Алгоритмын тайлбар                                                                                              1  
  1.1 Divide-and-Conquer …………………………………………………….……….1
* 1.2 DynamicProgramming……………………………………………………………2
* 1.3 GreedyAlgorithms……………………………………………………………….. 3
* 2  Алгоритмын харьцуулалт                                                                                       4  
  2.1 RecursionбаDivide-and-Conquer………………………………………………... 4
* 2.2 Divide-and-ConquerбаDynamicProgramming …………………………………...4
* 2.3 DynamicProgrammingбаGreedyalgorithm……………………………………….5

             I

1. Алгоритмын тайлбар

1.1 Divide-and-Conquer

Divide-and-Conquer нь асуудлыг жижиг дэд бодлогууд болгон задалж, дэд бодлого бүрийг тус тусад нь рекурсивээр бодож, тэдгээрийн шийдлүүдийг нэгтгэн үндсэн бодлогыг боддог алго- ритм юм.[1]

Жишээ бодлого: Merge sort алгоритмыг ашиглан массивыг эрэмбэлэх.

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

Программчлалын С хэл дээрх бодолт

void mergeSort(int array[], int left, int right) {

    if (left < right) {

       int mid = left + (right - left) / 2;

       // Массивыг 2 хэсэгт хуваан эрэмбэлэх

        mergeSort(array, left, mid);

        mergeSort(array, mid + 1, right);

       // 2 массивыг 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;

   // Массивыг 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 массивыг нэгтгэх

    int i = 0, j = 0, k = left;

    while (i < leftSize && j < rightSize) {

       if (leftArray[i] <= rightArray[j]) {

            array[k] = leftArray[i];

            i++;} else {

1

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

 Программчлалын С хэл дээрх бодолт

int fibonacci(int n) {

    if(n == 0) return 0;

    else if(n == 1) return 1;

   int[] dp = new int[n+1];

   \\base утгууд

    dp[0] = 0;

    dp[1] = 1;

   for(int i = 2; i < n+1; i++){

      dp[i] = dp[i-1] + dp[i-2];

}  return dp[n];

 }

2

1.3 Greedy Algorithms

Хомхойлох алгоритмууд нь үндсэн асуудлын оновчтой шийдлийг олно гэж найдан алхам ту- тамд тухайн агшинд оновчтой сонголтыг сонгодог алгоритм юм.[1]

Жишээ бодлого: Өгөгдсөн зоосны багцаас өгөгдсөн үнийн дүнг хамгийн бага тооны зоо- соор гаргах авах.

Энэхүү бодлогыг бодохдоо хамгийн том зоосноос аль болох ихийг авах байдлаар бодно.

void coinChange(int coins[], int n, int amount) {

    int coinCount = 0;

   for (int i = 0; i < n; i++) {

        while (amount >= coins[i]) {

           amount -= coins[i];

           coinCount++;

        }

}

   printf("\nЗоосны тоо: %d\n", coinCount);

}

3

2. Алгоритмын харьцуулалт

2.1 Recursion ба Divide-and-Conquer

Жишээ бодлого: Өгөгдсөн x тооны n зэргийг олох. Recursion аргыг ашиглан бодсон бодолт:

long power\_recursive(int x, int n) {

    if (n == 0)

       return 1;

    else

       return x \* power\_recursive(x, n - 1);

}

Тайлбар: x тооны n зэргийг олохдоо рекурсивээр функцийг 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\_recursive(int x, int n) {

    if (n == 0)

       return 1;

    else if (n % 2 == 0)

       return power\_recursive(x, n / 2) \* power\_recursive(x, n / 2);

    else

       return x \* power\_recursive(x, n / 2) \* power\_recursive(x, n / 2);

}

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

2.2 Divide-and-Conquer ба Dynamic Programming

Жишээ бодлого: Knapsack Problem. Тус бүр нь w[i] жин болон v[i] үнэтэй n ширхэг бараа болон хулгайчийн цүнхний багтаамж W өгөгдсөн. Өгөгдсөн бараа дундаас цүнхэнд багтахаар хамгийн их үнэтэй бараануудыг сонгох.

Divide-and-Conquer аргыг ашиглан бодсон бодолт:

int knapsack(int W, int weights[], int values[], int n) {

    // Base case

   if (n == 0 || W == 0)

        return 0;

   // Case 1: Одоогийн барааг авахгүйгээр

    int exclude = knapsack(W, weights, values, n - 1);

4

   // Case 2: Хэрэв цүнхэнд багтахаар бол одоогийн барааг оруулах

    int include = 0;

    if (weights[n - 1] <= W)

       include = values[n - 1] + knapsack(W - weights[n - 1], weights,

        values, n - 1);

   // Одоогийн барааг оруулсан болон оруулаагүй утгын ихийг авах

   return (include > exclude) ? include : exclude;

}

Тайлбар: Энэхүү бодлогыг бодохдоо одоогийн барааг авах болон аваагүй байх үнийг тоо- цон рекурсив байдлаар бодлогыг жижиг хэсгүүдэд хуваан 2 хариуны ихийг авах замаар бодсон.

Dynamic Programming аргыг ашиглан бодсон бодолт:

// Туслах функц

int max(int a, int b) {

   return (a > b) ? a : b;

}

int knapsack\_dp(int W, int weights[], int values[], int n) {

    int dp[n + 1][W + 1];

   // bottom-up аргачлалаар массивын утгуудыг олох

    for (int i = 0; i <= n; i++) {

       for (int w = 0; w <= W; w++) {

            if (i == 0 || w == 0)

} }

   dp[i][w] = 0; // Base case

else if (weights[i - 1] <= w)

   dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]],

   dp[i - 1][w]);

else

   dp[i][w] = dp[i - 1][w];

   return dp[n][W];

}

Тайлбар: Энэхүү бодлогыг бодохдоо base утга болох dp[i][w] = 0;-г олгож дараа дараа- гийн утгыг тооцоолохдоо өмнө тооцоолсон буюу одоогийн барааг аваагүй үнэ болон авсан үнийн харьцуулж ихийг dp массивт хадгалах замаар бодлогыг бодсон.

2.3 Dynamic Programming ба Greedy algorithm

Жишээ бодлого: Coin Change Problem. Зоосны багц өгөгдсөн ба amount үнийг өгөгдсөн зоос- нуудаас хамгийн цөөхөн тооны зоосоор гаргаж авах.

Dynamic Programming аргыг ашиглан бодсон бодолт:

5

// Туслах функц

int min(int a, int b) {

   return (a < b) ? a : b;

}

void coin\_change\_dp(int coins[], int n, int amount) {

    int dp[amount + 1];

   dp[0] = 0;

    for (int i = 1; i <= amount; i++) {

       dp[i] = INT\_MAX;

    }

   // DP массивыг дүүргэх

    for (int i = 1; i <= amount; i++) {

       for (int j = 0; j < n; j++) {

            if (coins[j] <= i && dp[i - coins[j]] != INT\_MAX) {

               dp[i] = min(dp[i], dp[i - coins[j]] + 1);

            }

} }

   if (dp[amount] == INT\_MAX)

        printf("Өгөгдсөн дүнг гарган авах боломжгүй.\n");

   else

        printf("Хамгийн бага зоосны тоо: %d\n", dp[amount]);

}

Тайлбар: Энэхүү бодлогыг бодохдоо dp массивт тооцоолсон утгуудыг хадгалж дараа да- раагийн тооцоололдоо ашигласан. Хэрэв одоогийн зоосыг авч болохоор байвал одоогийн зоосыг оруулаагүй өмнөх зоосны тоотой харьцуулан аль багийг нь авах замаар бодсон.

Greedy algorithm аргыг ашиглан бодсон бодолт:

int coin\_change\_greedy(int coins[], int n, int amount) {

    int count = 0;

    for (int i = 0; i < n; i++) {

        while (amount >= coins[i]) {

              amount -= coins[i];

              count++; }

}

   return count;

}

Тайлбар: Энэхүү бодлогыг бодохдоо хамгийн том зоосноос эхлэн авах боломжгүй болтол авч дараагийн том зоос уруу шилжих замаар бодсон.

6

Номзүй

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

7