F.CS301 Алгоритмын шинжилгээ ба зохиомж

БИЕ ДААЛТ №2

Шалгасан багш: Д.Батмөнх /F.SW811/

Гүйцэтгэсэн оюутан: Г.Төртүшиг /B220000135/

Улаанбаатар хот

2024 он

1. **Divide-and-Conquer буюу хэсэгчлэх арга**

Шийдэх гэж буй асуудлыг шийдвэрлэх боломжтой жижиг хэсгүүдэд хувааж тус бүрээр шийдвэрлэж дараагаар нь хариултуудыг үндсэн асуудлыг шийдлийг гаргаж авахад ашигаладаг.

Жишээ:

Merge sort нь үүний сонгодог жишээ юм.

Жагсаалтыг хоёр хэсэгт хуваана

Хагаслаад харьцуулах үйлдлээ дахин дахин давтана

Эрэмбэлсэн жагсаалтуудаа буцаан нийлүүлнэ.

1. **Dynamic Programming буюу динамик программчлал**

Динамик программчилал нь компьютерын програмчлалын техник бөгөөд алгоритмын бодлогыг эхлээд дэд асуудал болгон хувааж, үр дүнг нь хадгалж, дараа нь ерөнхий шийдлийг олдог. Динамик программчлал нь шийдэлд хүрэх хамгийн дөт замыг олохыг зорьдог бөгөөд ингэхдээ дээрээс доош, доороос дээш гэх хоёр ерөнхий хандлагын дагууд явдаг. Доороос дээш хандлага нь асуудлыг задалж болох хамгийн бага хэмжээнд задалж түүнийхээ шийдлийг дээд төвшний асуудлын шийдлийг бүрдүүлэх байдлаар асуудлыг шийддэг бол дээрээс доош хандлага нь шийдлийн хариуг хадгалан дараах дэд асуудлыг шийдэхэд хариултыг ашиглах байдлаар явагддаг.

Өмнөх Divide-and-Conquer бол динамик программчлалын сонгодог жишээ юм.

1. **Greedy Algorithms**

Шийдвэрлэх гэж буй асуудлын үе шат болгон хамгийн тохиромжтой сонголтыг сонгох зарчмаар явагддаг алгоритм. Энэ алгоритм нь ихэнх тохиолдолд асуудлыг хамгийн үр ашигтайгаар шийдэх шийдэл болж чадахгүй ч түүнтэй дөхөм болж чаддагаараа онцлогтой.

Жишээлбэл аялж буй худалдаачны( travelling salesman problem) бодлого дээр өөрийн байрлаж байгаа хотоос хамгийн ойрхон дараагийн очоогүй хот руу явна гэсэн зарчим баримталдаг.

Грийдийн алгоритм хамгийн үр ашигтайгаар ажилдаг асуудлуудад дараах хоёр шинж чанар байдаг:

* Шийдлийн сонголтуудаас хамгийн тохиромжтойг нь тодорхойлох боломжтой бөгөөд түүнийг сонгох боломжтой байна
* Тухайн асуудлыг шийдэж буй шийдэл нь дэд асуудлуудын шийдлийг агуулж байна.

1. **Recursion vs Divide-and-Conquer**

**Давталт** бол **программчлалын хэв шинж** (Programming Paradigm) бол **Divide-and-Conquer** бол **алгоритмын хэв шинж** (Algorithmic Paradigm) юм. Өөрөөр хэлбэл:

* **Давталт** бол өөрийнх дотор өөрийг нь агуулж болох функц бөгөөд үл ялиг өөр оролтын утгын өөрчлөлттэйгөөр өөрийгөө давтдаг.
* **Divide-and-conquer** бол тухайн шийдвэрлэх гэж буй асуудлуудыг үл давтагдах дэд асуудал болгон салгаж байгаа юм. Харин тэр дэд асуудлуудын шийдлийг олохын тулд бид **давталтыг** ашигладаг

2 болон 3-р даалгавар

|  |  |  |  |
| --- | --- | --- | --- |
|  | Шунахай | Хомхойлох | Динамик программчлал |
| Хандлага(Approach) | Дээрээс-доош | Дээрээс-доош | Доороос-дээш |
| Асуудлын төрөл | Тохиромжтой шийдлийг олох | Шийдвэр гаргах асуудал | Тохиромжтой шийдлийг олох |
| Ерөнхий зарчим | Одоогийнж тохиромжтой шийдэл нь өмнөх асуудлаас үүссэн шийдлийг умартан гаргадаг | Дэд асуудлын шийдэл нэгээс олон удаа дахин тооцоологддог | Дэд асуудлын шийдлыг нэг удаа л тооцоолоод дараа ашиглах үүднээс хүснэгтэд хадгалдаг. |
| Үр дүн | Тохиромжтой шийдлийг гаргаж ч болно, үгүй ч байж болно | Асуудлын шийдлийг олохыг зорьдог боловч хамгийн тохирмжтойг нь зорьдоггүй | Ямагт хамгийн тохиромжтой шийдлийг гаргадаг |
| Давталт | Давталт агуулдаг | Өөрийгөө дууддаг | Өөрийгөө дууддаг |
| Хурд, үр ашиг | Хомхойлох аргачлалаас үр ашиг өндөр, хурдан. | Энэ гурав дундаа хамгийн үр ашиггүй, удаан нь | Үр ашиг өндөртэй боловч шунахайгаас илүү удаан. |
| Нэмэлт санах ой шаардагдах байдал | Нэмэлт санах ой шаардлагагүй | Нэмэлт санах ой шаардлагатай | Шийдлүүдийг дараагийн асуудалд хэрэглэхийн тулд хамгийн ихээр нэмэлт санах ой шаарддаг нь |
| Жишээ | Fractional Knapsack problem, Activity selection problem, Job sequencing problem | Merge sort, Quick sort, Strassen’s matrix multiplication | 0/1 Knapsack, All pair shortest path, Matrix-chain multiplication |

Хавсралт

Дараах хэсэгт хичээлийн агуулгын дагууд хийж гүйцэтгэсэн бөгөөд энд дурдагдсан алгоритм, шийдлүүдийг хүргэв.

1. Merge Sort

if len(arr) > 1:

mid = len(arr) // 2

l = arr[:mid]

r = arr[mid:]

print("Left", l, " Right", r)

merge\_sort(l)

merge\_sort(r)

i = j = k = 0

while i < len(l) and j < len(r):

if l[i] < r[j]:

arr[k] = l[i]

i +=1

else:

arr[k] = r[j]

j += 1

k +=1

while i < len(l):

arr[k] = l[i]

i += 1

k += 1

while j < len(r):

arr[k] = r[j]

j+=1

k +=1

1. Fractal KnapSack

import java.util.Arrays;

class Item {

int value, weight;

Item(int value, int weight) {

this.value = value;

this.weight = weight;

}

}

class FractionalKnapsack {

public static double getMaxValue(Item[] items, int capacity) {

Arrays.sort(items, (a, b) -> Double.compare((double) b.value / b.weight, (double) a.value / a.weight));

double totalValue = 0.0;

for (Item item : items) {

if (capacity >= item.weight) {

// Барааг бүхлээр нь авна

totalValue += item.value;

capacity -= item.weight;

} else {

// Барааны бутархай хэсгийг авна

totalValue += item.value \* ((double) capacity / item.weight);

break; // Дүүрсэн

}

}

return totalValue;

}

public static void main(String[] args) {

Item[] items = {

new Item(60, 10),

new Item(100, 20),

new Item(120, 30)

};

int capacity = 50;

double maxValue = getMaxValue(items, capacity);

System.out.println("Бидний авч чадах хамгийн их нийт хэмжээ = " + maxValue);

}

}

1. 1/0 KnapSack

import java.util.Arrays;

public class Knapsack {

public static int knapsack(int[] weights, int[] values, int capacity) {

int n = values.length;

int[][] dp = new int[n + 1][capacity + 1];

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

for (int w = 1; w <= capacity; w++) {

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

// Барааг нэмж болно

dp[i][w] = Math.max(dp[i - 1][w],

dp[i - 1][w - weights[i - 1]] + values[i - 1]);

} else {

dp[i][w] = dp[i - 1][w];

}

}

System.out.println(Arrays.toString(dp[i]));

}

return dp[n][capacity];

}

public static void main(String[] args) {

int[] weights = {1, 2, 3, 4};

int[] values = {10, 20, 30, 40};

int capacity = 5;

int maxValue = knapsack(weights, values, capacity);

System.out.println("Та хамгийн ихдээ нийт: " + maxValue + " төгрөгний барааг даах нь ээ");

}

}