ALGORITMIEK: opgaven werkcollege 9

Dynamisch programmeren

Opgave 1. (zie tweede editie, opgave 8.1.2)
a. Bereken de binomiaalcoëfficiënt $C(6,3)$ met het DP algoritme.
b. Is het ook mogelijk om $C(n,k)$ te berekenen door de tabel van het DP algoritme kolom voor kolom te vullen in plaats van rij voor rij?

Opgave 2. Levitin, opgave 8.1.12. **(zie tweede editie, opgave 8.1.10.)**

Opgave 3. Levitin, opgave 8.2.1. **(zie tweede editie, opgave 8.4.1.)**

Opgave 4. We bekijken een weegschaalprobleem.
Gegeven zijn $n \geq 1$ verschillende gewichten $g_1, g_2, \ldots, g_n$ en een geheel getal $W \geq 0$. De gewichten zijn alle geheel en positief (> 0). Laten de gewichten opgeslagen zijn in een array `gewicht`.
Vraag: is het mogeljk om een deelverzameling uit de gewichten te kiezen met totaalgewicht precies W?
a. Schrijf een recursieve functie (type bool) voor dit probleem. Hint bij de recursieve formulering die het probleem terugbrengt tot kleinere versies van het probleem: het laatste gewicht (hier g_n) kan wel of niet deel uitmaken van de gezochte deelverzameling.
b. Nu gaan we het probleem via dynamisch programmeren oplossen. Definieer daartoe een geschikt boolean array `wegen`. Leid uit de recursieve formulering uit **a.** een recurrente betrekking af voor `wegen`$[i][j]$.
c. Denk na over de berekeningsvolgorde en schrijf een algoritme dat het array vult. De gevraagde waarde komt uiteindelijk in `wegen`$[n][W]$ te staan.
d. Leg uit hoe we uit het boolean array `wegen` ook *een* oplossing (= een deelverzameling met totaalgewicht W) kunnen vinden.
e. Als we alleen in het eindantwoord geïnteresseerd zijn kunnen we een eendimensionaal array gebruiken. Leg uit hoe het algoritme uit **c.** dan moet worden aangepast.

Opgave 5. We bekijken het Japanse spel Pachinko, gespeeld op een speciale machine (een soort flipperkast), die we als volgt modelleren.
We hebben een rechtopstaand rooster met hoogte m en breedte n. Bovenin kan men een balletje in een kolom gooien, die dan loodrecht naar beneden valt totdat hij op een obstakel (een *) botst of totdat hij in een poort (= een vakje met een `getal` erin) terechtkomt, of totdat hij onderaan uit het rooster valt. In dat laatste geval heb je niets verdiend. Komt het balletje in een poort, dan ben je klaar en heb je `getal` euro gewonnen. Als het balletje onderweg op een * stuit, valt hij verder omlaag in de kolom links of rechts van de *: de kans dat hij naar links valt is gelijk aan de kans dat hij naar rechts valt (beide dus kans 0,5).
Aanname: het rooster bevat geen obstakels direct naast elkaar (noch in dezelfde rij, noch in dezelfde kolom en noch op dezelfde diagonaal). Hetzelfde voor de poorten. Verder bevatten de eerste en de laatste kolom geen obstakels of poorten.
Gevraagd wordt de maximale te verwachten opbrengst en de kolom waarin je het balletje moet gooien om die te behalen.

Z.O.Z.

Voorbeeld: de maximaal verwachte opbrengst voor onderstaand rooster is 7,50 euro.

```
. . . 1 . . . .
. . . . . . . .
. . * . . . * .
. . . . * . . .
. 1 . . . . . .
. . . * . * . .
. . . . . . . .
. . 9 . 7 . 7 .
```

a. Geef een recursief algoritme dat de maximale verwachte opbrengst berekent.
b. Geef een voorbeeld waaruit blijkt dat het gebruik van alleen recursie voor het oplossen van dit probleem i.h.a. inefficiënt is.
c. Gebruik bottom up dynamisch programmeren om de maximale verwachte opbrengst te berekenen. Dus: kies een geschikt (tweedimensionaal) array D, stel een recurrente betrekking op die aangeeft waaruit $D[i][j]$ berekend wordt, geef een berekeningsvolgorde aan en schrijf een algoritme dat het array vult. Geef ook aan hoe je de corresponderende kolom vindt.
d. Denk je dat het hier handiger is om top down dynamisch programmeren (= recursie met array) te gebruiken dan bottom up of niet? Motiveer je antwoord.