

## ALGORITMIEK: opgaven werkcollege 9

### Dynamisch programmeren

#### Opgave 1. (uit een oud tentamen)

Langs een lang stuk snelweg zijn  $n \geq 1$  posities, die steeds 500 meter van elkaar liggen, aangewezen als plekken waar reclameborden mogen worden neergezet. De (verwachte) opbrengst als gevolg van zo'n reclamebord hangt af van de positie waar het staat en is gegeven middels een 1-dimensionaal array:  $\text{opbrengst}[i]$  ( $> 0$ ;  $i = 1, \dots, n$ ) = opbrengst van een reclamebord op plek  $i$ . Reclameborden moeten altijd meer dan ( $>$ ) 1 km van elkaar staan. De bedoeling is om reclameborden zo te positioneren dat de totaalopbrengst maximaal is.

- Wat is de maximale totaalopbrengst als  $n = 1$ ? En als  $n = 2$ ? En als  $n = 3$ ?
- Geef een recursieve formulering voor  $\text{maxtotaal}(n)$ , de maximale totaalopbrengst. De basisgevallen ( $n = 1, 2, 3$ ) zijn in **a** berekend.
- Het probleem kan recursief worden opgelost, maar dat is in dit geval niet efficiënt. Leg uit waarom niet, en geef aan hoe dynamisch programmeren de efficiëntie aanzienlijk kan vergroten. Bespreek vervolgens (kort) zowel de top down methode als de bottom up methode voor dynamisch programmeren en leg het verschil uit.
- Geef in pseudocode of in C++ een bottom up dynamisch programmeren algoritme dat de maximale totaalopbrengst berekent. Formuleer daarbij duidelijk wat voor array je gebruikt en geef de recurrente betrekking volgens welke het array gevuld wordt.

#### Opgave 2. (Levitin, opgave 8.2.1. )

Pas de bottom-up variant van dynamisch programmeren toe op de volgende instantie van het knapzakprobleem:

object	gewicht	waarde
1	3	25
2	2	20
3	1	15
4	4	40
5	5	50

knapzakcapaciteit  $W = 6$

#### Opgave 3. (Levitin, opgave 8.1.12. )

*World Series odds* Beschouw twee teams  $A$  en  $B$ , die een reeks wedstrijden spelen totdat een van de teams  $n$  wedstrijden gewonnen heeft.

Neem aan dat de kans dat een wedstrijd door  $A$  gewonnen wordt elke wedstrijd hetzelfde is, namelijk  $p$ . Gelijke spelen komen niet voor, dus de kans dat een wedstrijd door  $A$  verloren wordt is  $q = 1 - p$ . Laat  $P(i, j)$  de kans zijn dat team  $A$  de reeks wedstrijden wint, op het moment dat  $A$  nog  $i$  wedstrijden nodig heeft om aan de  $n$  te komen en  $B$  nog  $j$ .

- Stel een recurrente betrekking op voor  $P(i, j)$  die gebruikt kan worden door een DP algoritme.
- Bepaal de kans dat  $A$  de reeks wedstrijden wint, als het om vier gewonnen wedstrijden gaat (dus  $n = 4$ ) en de kans dat  $A$  een wedstrijd wint gelijk is aan 0.4 (dus  $p = 0.4$ ).
- Geef pseudocode voor een DP algoritme dat dit probleem oplost, en bepaal zijn tijd- en ruimtecomplexiteit.

**Opgave 4.** We bekijken een weegschaalprobleem.

Gegeven zijn  $n \geq 1$  verschillende gewichten  $g_1, g_2, \dots, g_n$  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 mogelijk 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) terecht komt, 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.

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.

### Opgave 6.

- 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 7. (tentamenopgave juni 2014)

Jan heeft recentelijk een vakantiehuisje gekocht in een bergachtig gebied. Hij houdt van wandelen en nodigt regelmatig mensen uit voor een wandeling door de streek. Hij heeft een globale kaart van het gebied met de gemiddelde hoogtes per hectare. Omdat Jan het gebied nog niet erg goed kent en niet wil verdwalen, besluit hij alleen maar recht naar het zuiden en recht naar het oosten te lopen van hectare naar hectare op de kaart. Hij wandelt zo van de linkerbovenhoek van de kaart, waar zijn huis staat, naar de rechteronderhoek, waar hij zijn gasten op een maaltijd trakteert. De linkerbovenhoek bevindt zich op positie (rij, kolom) =  $(0, 0)$ , de rechteronderhoek op  $(n - 1, n - 1)$ .

*Voorbeeld:* een hoogtekaart  $H[i][j]$  voor  $n = 6$ :

0	3	5	6	5	4
1	2	2	2	7	3
1	8	7	6	6	2
3	9	7	4	5	3
7	8	7	5	6	2
4	5	8	4	5	0

Het hoogteverschil tussen twee aangrenzende hectaren is gedefinieerd als het absolute verschil tussen de twee hoogtes van deze hectaren en is dus altijd  $\geq 0$ .

Het hoogteverschil van een wandelroute is gelijk aan de som van de hoogteverschillen op deze route. Het hoogteverschil van de wandelroute  $(3, 0), (3, 1), (3, 2), \dots, (3, 5)$  is  $6 + 2 + 3 + 1 + 2 = 14$ .

- a.** Voor zijn oma zoekt Jan naar de route met zo min mogelijk hoogteverschil. Beschrijf een *gretig* algoritme voor het vinden van zo'n route. Welke route vindt hij wanneer hij dit algoritme toepast op bovenstaand voorbeeld? Wat is daarvan het totale hoogteverschil?

Jan gaat het probleem oplossen met bottom-up dynamisch programmeren. Hierbij gebruikt hij een tweedimensionaal array  $M$ , waarbij  $M[i][j]$  het minimale hoogteverschil is voor een wandelroute van  $(i, j)$  naar  $(n - 1, n - 1)$ . We willen uiteindelijk  $M[0][0]$  weten.

**b.** Leg uit waarom  $M$  voor  $0 \leq i < n$  en  $0 \leq j < n$  voldoet aan:

$$M[i][j] = \begin{cases} 0 & \text{als } i = n - 1 \text{ en } j = n - 1 \\ M[n - 1][j + 1] + \mathbf{abs}(H[n - 1][j + 1] - H[n - 1][j]) & \text{als } i = n - 1 \text{ en } 0 \leq j < n - 1 \\ M[i + 1][n - 1] + \mathbf{abs}(H[i + 1][n - 1] - H[i][n - 1]) & \text{als } j = n - 1 \text{ en } 0 \leq i < n - 1 \\ \min(M[i + 1][j] + \mathbf{abs}(H[i + 1][j] - H[i][j]), \\ M[i][j + 1] + \mathbf{abs}(H[i][j + 1] - H[i][j])) & \text{als } 0 \leq i < n - 1 \text{ en } 0 \leq j < n - 1 \end{cases}$$

Hierin is  $\mathbf{abs}(a)$  is de absolute waarde van  $a$  en geeft  $\mathbf{min}(a, b)$  het minimum van  $a$  en  $b$ .

**c.** Schrijf een algoritme in pseudocode of C++ dat, gebruikmakend van het array  $M$  en de recurrente betrekking uit vraag **b.**, het gevraagde minimale hoogteverschil berekent. Je mag ervan uitgaan dat functies  $\mathbf{min}$  en  $\mathbf{abs}$  al bestaan.

Geef duidelijk aan in welke volgorde je het array  $M$  vult, en waarom juist in die volgorde.

Wanneer we het algoritme uit vraag **c.** toepassen op het voorbeeld, krijgen we het volgende array  $M$ :

12	11	9	8	7	6
11	10	10	10	7	5
15	8	7	6	6	4
15	11	9	6	5	3
11	10	9	7	6	2
14	13	10	6	5	0

**d.** Bepaal uit deze  $M$  het minimale hoogteverschil en een bijbehorende optimale wandelroute van  $(0, 0)$  naar  $(n - 1, n - 1)$ . Leg daarbij in woorden uit hoe je deze wandelroute uit de arrays  $M$  en  $H$  bepaalt.

**Opgave 8.** (toegift voor wiskundestudenten; met dank aan een student-assistent uit het verleden)

Zij  $p(n)$  het aantal niet-dalende rijtjes van positieve gehele getallen (dus  $> 0$ ) die sommeren tot  $n$ . Beschrijf een DP-algoritme dat  $p(n)$  berekent.

Hint: Zij  $F(n, m)$  het aantal zulke rijtjes waarvan alle getallen maximaal  $m$  zijn. Bedenk een recursieve formulering voor  $F(n, m)$ . We zoeken dan  $F(n, n)$ .

Opmerking. De functie  $p(n)$  heet de *partitiefunctie* en geeft het aantal cykeltypes/conjugatieklassen van de symmetrische groep  $S_n$ . Opgave 2.58 van Algebra 1 (versie 2012) is om  $p(n)$  te berekenen voor  $n \leq 8$ . Met dit DP-algoritme is dit een eitje.