Hertentamen Algoritmiek
Maandag 8 juli 2019, 14.00 – 17.00 uur

Als er om uitleg, toelichting of motivatie gevraagd wordt bij een opgave, is het belangrijk om die ook te geven.
Als je het antwoord op een onderdeel niet weet, en je hebt dat antwoord nodig bij een later onderdeel, dan kun je het antwoord 'kopen' bij de docent.
Globale puntenverdeling: 1: 29 pt; 2: 28 pt; 3: 25 pt; 4: 18 pt. **Veel succes!**

---

1. (a) Leg uit hoe best-first branch-and-bound werkt voor maximalisatieproblemen in het algemeen. Geef daarbij o.a. aan hoe (deel)oplossingen gegenereerd worden, wat met branch bedoeld wordt en wat met bound, wat best-first betekent, wanneer gesnoeid wordt, enz.

   (b) Bij het knapzak probleem hebben we een verzameling objecten i ($1 \leq i \leq N$), elk met een gewicht w_i en een waarde v_i. Verder hebben we een knapzak met een capaciteit W. In deze knapzak kunnen we objecten uit onze verzameling stoppen, zolang het totaalgewicht niet groter wordt dan W. Doel is om een deelverzameling van (verschillende) objecten in de knapzak te stoppen met een zo groot mogelijke totale waarde.

   Een voorbeeld van een knapzakprobleem met vier objecten is:

object i	w_i	v_i	v_i/w_i
1	6	48	8
2	5	35	7
3	9	54	6
4	7	28	4

   W = 15

   Tijdens het college hebben we een branch-and-bound algoritme voor het knapzakprobleem behandeld. Hierbij wordt gebruikt dat de objecten gesorteerd zijn op de gemiddelde-waarde-per-gewicht v_i/w_i, van groot naar klein. Iedere stap van het algoritme wordt een object wel of juist niet aan de knapzak toegevoegd, te beginnen bij object 1.

   Uitgaande van een deeloplossing, is een bovengrens voor de waarde van elke (complete) uitbreiding van de deeloplossing als volgt te berekenen:

   - Bij aanvang, in de begintoestand: $W * (v_1/w_1)$.
   - Bij een algemene deeloplossing (als we net object i wel of niet hebben gekozen, met $1 \leq i \leq N-1$): $v + (W - w) * (v_{i+1}/w_{i+1})$, met v de totaalwaarde van de reeds gekozen objecten en w het totaalgewicht daarvan.

   Leg uit waarom dit inderdaad bovengrenzen zijn voor de waarde die een uitbreiding van de deeloplossing (begintoestand of algemeen) kan hebben. Geef antwoorden voor een algemeen knapzakprobleem met $N \geq 1$ objecten, dus niet specifiek voor het voorbeeld hierboven.

   (c) Pas het branch-and-bound algoritme met de bovengrens uit het vorige onderdeel toe op het voorbeeld en teken de bijbehorende state-space-tree, met bij elke knoop (deeloplossing) de relevante informatie (waaronder ook de opbouw van de bovengrens). Geef ook aan in welke volgorde de knopen zijn aangemaakt, welke knopen gesnoeid worden en waarom.

   Wat is dus de optimale oplossing?

---

2. Gegeven een array $A = A[0], A[1], \ldots, A[N-1]$ dat $N(\geq 1)$ integers bevat. In deze opgave gaan we op zoek naar de langste, aaneengesloten, stijgende deelrij van A. Bij het volgende array A:

positie i	0	1	2	3	4	5	6	7	8
A[i]	3	1	4	5	2	3	6	2	8

heeft een langste, aaneengesloten, stijgende deelrij lengte 3. Bijvoorbeeld de rij 1,4,5 op de posities 1,2,3 is zo'n deelrij.

We gaan het probleem op drie manieren oplossen: met brute force, met decrease-and-conquer en met divide-and-conquer. **Onder 'stijgend' verstaan we overigens 'strict stijgend': twee dezelfde getallen achter elkaar zijn niet stijgend.**

(a) Geef een eenvoudig (brute force) iteratief algoritme dat de lengte van de langste, aaneengesloten, stijgende deelrij van een array integers $A = A[0], A[1], \ldots, A[N-1]$ bepaalt. Schrijf hiervoor een C++-functie `int aaneengeslotenstijgend1 (int A[ ], int N)` die de gevraagde lengte retourneert.

(b) Geef nu een decrease-by-one algoritme voor bovenstaand probleem. Schrijf daartoe een *recursieve* C++-functie `int aaneengeslotenstijgend2 (int A[ ], int i, int &eindstijgend)`, die het probleem oplost voor het (deel)array $A[0], A[1], \ldots, A[i-1]$ $(i \geq 1)$.

Ook deze functie retourneert de gevraagde lengte. De parameter `eindstijgend` bevat bij aanvang van de functie geen zinvolle waarde. Na afloop bevat hij het (maximale) aantal integers achteraan het (deel)array $A[0], A[1], \ldots, A[i-1]$ dat een aaneengesloten, stijgende deelrij vormt. Bijvoorbeeld, als $i = 4$, dan zal

- bij de rij 0,2,3,7 na afloop `eindstijgend`=4 zijn,
- bij de rij 4,2,3,7 na afloop `eindstijgend`=3 zijn (want 2,3,7 vormt een maximale stijgende deelrij aan het eind),
- bij de rij 4,2,3,0 na afloop `eindstijgend`=1 zijn (want 0 vormt een maximale stijgende deelrij aan het eind).

De aanroep `aaneengeslotenstijgend2 (A, N, eindstijgend);` geeft dan uiteraard het antwoord voor het hele array.

(c) Neem aan dat N een 2-macht is (minstens 1). Geef nu een divide-and-conquer algoritme voor het probleem. Het array dient hiervoor in twee even grote delen te worden verdeeld. Schrijf een *recursieve* C++-functie `int aaneengeslotenstijgend3 (int A[ ], int links, int rechts, int &beginstijgend, int &eindstijgend)` die het probleem oplost voor het deelarray A[links],. . . ,A[rechts] ter lengte een 2-macht (minstens 1 dus). De parameter `eindstijgend` heeft dezelfde betekenis als bij onderdeel (b). Daarnaast is er nu ook een parameter `beginstijgend` die na afloop het (maximale) aantal integers vooraan het (deel)array A[links], . . . , A[rechts] bevat dat een aaneengesloten, stijgende deelrij vormt. Bijvoorbeeld, als $i = 4$, dan zal

- bij de rij 0,2,3,7 na afloop `beginstijgend`=4 zijn,
- bij de rij 2,4,3,7 na afloop `beginstijgend`=2 zijn (want 2,4 vormt een maximale stijgende deelrij aan het begin),
- bij de rij 4,2,3,0 na afloop `beginstijgend`=1 zijn (want 4 vormt een maximale stijgende deelrij aan het begin).

De aanroep `aaneengeslotenstijgend3 (A, 0, N-1, beginstijgend, eindstijgend);` geeft dan uiteraard het antwoord voor het hele array.

3. Gegeven opnieuw een array $A = A[0], A[1], \ldots, A[N-1]$ dat $N(\geq 1)$ integers bevat. Bij opgave 2 waren we op zoek naar de langste, aaneengesloten, stijgende deelrij van A. Bij deze opgave laten we de eis van 'aaneengeslotenheid' vallen. We zoeken dus de langste stijgende deelrij van het array met integers A. Bij het volgende array A:

positie i	0	1	2	3	4	5	6	7	8
A[i]	3	1	4	5	2	3	6	2	8

heeft een langste stijgende deelrij lengte 5. Een deelrij die die lengte haalt, bevindt zich bijvoorbeeld op de posities 0, 2, 3, 6, 8. **Onder 'stijgend' verstaan we overigens 'strict stijgend': twee dezelfde getallen achter elkaar zijn niet stijgend.**

Een gretig algoritme (begin met $A[0]$, en voeg steeds het eerst volgende grotere getal toe aan de deelrij) geeft bij dit voorbeeld toevallig een optimale oplossing, maar bij andere arrays A niet.

(a) We gaan het probleem daarom oplossen met behulp van bottom-up dynamisch programmeren. Daarvoor maken we gebruik van een ééndimensionale tabel `langstestijgend`, waarbij `langstestijgend[i]` is gedefinieerd als: de lengte van de langste stijgende deelrij van array A die eindigt op positie i (waarbij $A[i]$ daadwerkelijk in de deelrij zit). Bij ons array A uit het voorbeeld hierboven ziet dat er als volgt uit:

positie i	0	1	2	3	4	5	6	7	8
A[i]	3	1	4	5	2	3	6	2	8
langstestijgend[i]	1	1	2	3	2	3	4	2	5

Geef nu ook de inhoud van de tabel `langstestijgend` voor het volgende array A:

positie i	0	1	2	3	4	5	6	7	8
A[i]	8	2	6	3	2	5	4	1	3

(b) We gaan nu een recursieve formulering voor `langstestijgend[i]` bedenken voor een willekeurig array A met N integers op posities $0, 1, \ldots, N-1$.

i. Wat is (in het algemeen dus) `langstestijgend[0]` ?

ii. Druk nu `langstestijgend[i]` voor $i \geq 1$ uit in waardes voor `langstestijgend[j]` met $j < i$ en in integers $A[j]$. Je mag dit in woorden doen of in een formule, maar doe het in ieder geval duidelijk en volledig. Motiveer je antwoord.
Hint: als je dit niet onmiddellijk ziet, vraag je dan af hoe in het voorbeeld hierboven `langstestijgend[8]=5` te maken heeft met eerdere waardes `langstestijgend[j]`.

(c) Schrijf een **niet-recursieve** C++-functie `void vullangstestijgend (int A[], int N, int langstestijgend[])` die voor het array A met N integers op posities $0, 1, \ldots, N-1$ de tabel `langstestijgend` vult met de juiste waardes.

(d) De tabel `langstestijgend` bevat alleen nog maar de *lengte* van de langste stijgende deelrij die eindigt op een bepaalde positie, en niet de deelrij zelf. Beschrijf nu (in woorden of (pseudo-)code, maar in ieder geval duidelijk en volledig) hoe je uit de tabel `langstestijgend` ook daadwerkelijk efficiënt een langste stijgende deelrij van array A kunt afleiden.

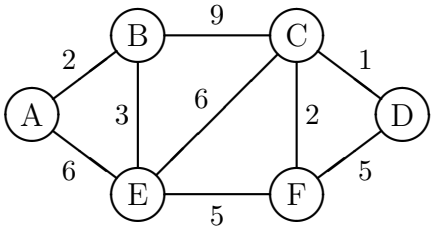
---

4. Het algoritme van Dijkstra bepaalt voor samenhangende, gewogen grafen G met knooppuntenverzameling V de (lengtes van) kortste paden vanuit een gegeven knoop s naar alle andere knopen. Het algoritme wordt beschreven door de volgende pseudo-code:

```
for v ∈ V do
        pad[v] := ∞;
od
pad[s] := 0;
U := ∅;
while ( U ≠ V ) do
        vind knoop v* ∈ V \ U met pad[v*] minimaal;
        U := U ∪ {v*};
        for alle knopen v aangrenzend aan v* do
            if pad[v*]+ gewicht(v*, v) < pad[v] then
                pad[v] := pad[v*]+ gewicht(v*, v);
                nieuwe kandidaattak voor v: (v*, v)
            fi
        od
od
```

(a) Pas het algoritme van Dijkstra toe op onderstaande graaf, beginnend in knoop A. Geef voor elke stap van het algoritme duidelijk aan welke knoop erbij wordt gekozen in U (= verzameling knopen waarvan de kortste afstand vanaf A bekend is) en welke labels door die keuze veranderen en hoe. Licht toe hoe je uitwerking gelezen moet worden.

Geef ook de resulterende boom van kortste paden met daarin de bijbehorende lengtes van de kortste paden vanaf A.



(b) Wat is de worst case tijdcomplexiteit van het algoritme van Dijkstra, zoals dat hierboven beschreven is? Motiveer je antwoord, door een basisoperatie in het algoritme aan te wijzen en te bepalen hoe vaak deze operatie wordt uitgevoerd. Ga ervanuit dat de graaf wordt gerepresenteerd door een adjacency matrix `int graaf[N][N]`. (Dus V bevat N knopen.)

(c) Het algoritme van Prim lijkt sterk op het algoritme van Dijkstra.

i. Wat levert het algoritme van Prim, toegepast op een graaf G, op?

ii. Wat zijn de verschillen tussen het algoritme van Prim en het algoritme van Dijkstra zoals dat hierboven beschreven is?
