Programmeeropdracht 3 — Goedkoop Tanken
Algoritmiek, voorjaar 2025

Inleiding

De vriendinnen Sophie en Caroline zitten gezellig in hun Hummer, op weg naar een klimaatconferentie. Het is een lange reis, en onderweg moet er dan ook af en toe getankt worden. De prijzen bij de pompstations op de route variëren nogal, en het is dan ook een hele puzzel om te bepalen bij welke stations er getankt moet worden, en hoeveel. Doel van de vriendinnen is om zo goedkoop mogelijk op hun bestemming te komen. Immers, de beschikbare gelden voor klimaatbeleid zijn niet zo groot meer. Helaas komen de vriendinnen er samen niet uit. Daarom vragen ze jou om een computerprogramma te schrijven dat de minimale reiskosten en de daarbij behorende tankstrategie bepaalt.

Specificatie

De dieseltank van de Hummer van Sophie en Caroline heeft een gegeven inhoud (capaciteit) c. Met deze inhoud moeten ze het doen; ze hebben geen jerrycans bij zich om extra diesel mee te nemen. Op de route naar hun bestemming komen ze n pompstations tegen. Elk van deze stations heeft een positie (een x-coördinaat langs de route, in kilometers), een hoeveelheid diesel dat het nog beschikbaar heeft (meer dan dat kunnen de vriendinnen daar dus niet tanken), en een prijs per liter.

Sophie en Caroline beginnen met een lege tank bij het eerste pompstation, het station met de laagste x-coördinaat. Daar kunnen ze dus meteen tanken. Hun Hummer verbruikt vervolgens één liter diesel per kilometer. Doel is dus om zo goedkoop mogelijk de klimaatconferentie te bereiken, die gehouden wordt op een x-coördinaat voorbij het laatste pompstation. Daar aangekomen mag de tank weer leeg zijn. Sterker nog: voor een zo goedkoop mogelijke reis, *moet* de tank daar weer precies leeg zijn.

Alle getallen die het probleem specificeren (c, n, de posities, de hoeveelheden diesel en de prijzen) zijn integers. Alle posities (van de pompstations en de conferentie) zijn verschillend. Er is gegeven dat $1 \leq c \leq 100$, $1 \leq n \leq 500$, en dat de hoeveelheden diesel en de prijzen positief zijn.
Een voorbeeld van een complete specificatie is de volgende serie getallen:

```
60
4
10 10 15
15 100 20
40 30 18
80 35 25
125
```

De inhoud van de tank is hier $c = 60$, en er zijn $n = 4$ pompstations. Elk station heeft achtereenvolgens een positie, een hoeveelheid diesel en een prijs. De reis begint op $x = 10$ en de bestemming ligt op $x = 125$. In dit geval kost de goedkoopste reis 2315 euro. Dit is te bereiken door achtereenvolgens 10, 50, 30 en 25 liter te tanken. In dit voorbeeld tanken de vriendinnen dus op alle n stations. In het algemeen hoeft dat niet het geval te zijn.

Eisen aan het programma

Je moet een C++-programma schrijven dat **met behulp van dynamisch programmeren**[1] het probleem voor Sophie en Caroline oplost. Dat kun je doen met een twee-dimensionaal array `kosten`, waarbij `kosten[i][j]` de minimale kosten voorstelt om te arriveren op station i met nog j liter diesel in je tank. Wanneer we de n stations nummeren als $0, 1, 2, \ldots, n-1$ en de bestemming 'station n' noemen, gaat het ons dus uiteindelijk om `kosten[n][0]`.

Het is aan te raden om, voordat je begint met programmeren, goed na te denken over de recurrente betrekking waaraan `kosten[i][j]` voldoet. In het verslag wordt hier ook naar gevraagd.

Het kan overigens gebeuren dat de vriendinnen met geen mogelijkheid op de klimaatconferentie kunnen komen, simpelweg doordat ze niet voldoende diesel kunnen tanken om de afstanden tussen de pompstations (en/of tussen het laatste pompstation en de conferentie) te overbruggen. In dat geval moet je programma dat ook rapporteren.

Natuurlijk willen Sophie en Caroline niet alleen weten hoeveel geld hun reis kost, maar ook bij welk pompstation ze hoeveel diesel moeten tanken om voor dat geld op de conferentie te komen. Je programma moet die informatie dus ook opleveren. In principe kunnen er overigens meerdere oplossingen zijn die dezelfde minimale kosten geven. Het maakt dan niet uit welke oplossing je programma oplevert.

Een complete specificatie van het probleem met de hand invoeren wordt veel werk. Daarom moet je programma een tekstbestand in kunnen lezen, dat de vorm heeft als in het eerder gegeven voorbeeld. Hoewel er direct boven dat voorbeeld voorwaarden worden beschreven waaraan de getallen voldoen, moet je programma die voorwaarden nog wel controleren. Ook moet het controleren dat alle posities in oplopende volgorde staan.

Jokers

Je programma moet allereerst voldoen aan de eisen in de vorige paragraaf. Een deel van de 10 punten kun je echter alleen verdienen als je ook rekening houdt met jokers. Sophie en Caroline hebben namelijk met de spaarpunten van eerdere tankbeurten een aantal jokers verdiend. Met een joker krijgen ze 50% korting op de totale prijs van een tankbeurt. Als die totale prijs daarmee een niet-geheel getal wordt, wordt er naar boven afgerond. Elke joker kan maximaal één keer worden benut, en per tankbeurt kan maximaal één joker worden ingezet.

Het ligt wellicht voor de hand om te denken: "Als ik met jokers moet werken, bereken ik eerst de goedkoopste reis zonder jokers, en dan halveer ik de kosten van de duurste tankbeurten." Dit werkt echter in het algemeen niet. Als je jokers kunt inzetten, zul je soms andere hoeveelheden diesel per station gaan tanken.

Dat is in het bijzonder het geval bij ons voorbeeld. Met twee jokers kunnen de reiskosten in het eerder gegeven voorbeeld teruggebracht worden van 2315 euro naar 1383 euro. Je krijgt deze kosten door achtereenvolgens 5, 60 (met joker), 15 en 35 (met joker) liter te tanken.

Je zult het gebruik van jokers daarom moeten meenemen bij het dynamisch programmeren. Je krijgt dan een driedimensionaal array `kosten`, waarbij `kosten[i][j][k]` de minimale kosten voorstelt om te arriveren op station i met nog j liter diesel in je tank, en met gebruikmaking van k jokers. Bij de eerste i pompstations heb je dan dus k jokers ingezet.

[1] Misschien denk je het probleem ook op een andere manier op te kunnen lossen, maar dat is nu niet de bedoeling.

Sophie en Caroline hebben maximaal 10 jokers. Het exacte aantal wordt opgegeven door de gebruiker van het programma.

Voor u beschikbaar

Op de website van het vak

`https://liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

is een pakketje beschikbaar, met daarin

- een skeletprogramma,
- een tekstbestand `tankinfo1.txt` met het voorbeeld uit deze specificatie,
- een C++-progamma waarmee je zelf random instanties kunt genereren.

In het skeletprogramma wordt een klasse `TankPlanner` gedefinieerd, die door het hoofdprogramma gebruikt wordt. Het programma wordt onder Linux gecompileerd met het commando `make`. Vervolgens kun je het met het commando `./TankPlanner` runnen. Je krijgt dan een menu aangeboden, met de keuze om een instantie in te lezen of op te lossen met top-down of bottom-up dynamisch programmeren. Bedoeling is om de `TODO`'s in tankplanner.cc en tankplanner.h uit te voeren.

De meeste functies die je moet implementeren worden toegelicht in het skeletprogramma, speciaal in `tankplanner.h`. Goed om te weten: als er in het commentaar boven een functie `Pre:` staat, dan volgt de **preconditie** voor die functie: een aantal aannames waar je vanuit mag gaan als je in die functie binnenkomt. Je hoeft dat dus in de functie niet meer te controleren. De gebruiker van de functie moet daar van tevoren voor zorgen. Net zo staat `Post:` voor de **postconditie** van de functie: zaken die na afloop van de functie moeten gelden. Daar moet je in de functie dus voor zorgen.

De belangrijkste functies zijn `losOpTD` en `losOpBU`. Beide functies bepalen de minimale kosten voor de ingelezen instantie, met dynamisch programmeren. `losOpTD` gebruikt top-down dynamisch programmeren, `losOpBU` bottom-up dynamisch programmeren. Ze zijn allebei gebaseerd op dezelfde recurrente betrekking voor deze kosten.

Beide functies krijgen een parameter `nrJokers` mee, voor het aantal jokers dat Sophie en Caroline tot hun beschikking hebben. Als het je niet lukt om in je programma rekening te houden met de jokers, kun je deze parameter gewoon negeren.

Van deze twee functies is `losOpBU` de allerbelangrijkste: als het bottom-up dynamisch programmeren hier niet goed genoeg in te herkennen is, zal je oplossing voor de opdracht **nooit** voldoende worden, hoe goed de rest ook is. Daarnaast dient deze functie niet alleen de minimale kosten, maar ook een tankstrategie te retourneren die overeenkomst met deze minimale kosten.

Enkele tips bij het programmeren

- Kijk in het skeletprogramma waar allemaal `TODO` staat, voor met name de functies die geïmplementeerd moeten worden.
- In het skeletprogramma wordt gebruik gemaakt van `pair`, om een integer en een boolean in op te slaan. Als je niet weet hoe je die gebruikt, zoek het op in de C++ reference.

- Het is verstandig om eerst na te denken over hoe je de data uit een invoerbestand, met name de informatie van de tankstations, opslaat in je klasse. Implementeer vervolgens achtereenvolgens de betrekkelijk eenvoudige memberfuncties `leesIn` en `drukAfInstantie`.
- Wellicht ten overvloede: als je in C++ `ifstream fin` gebruikt voor je invoer-tekstbestand, dan kun je heel eenvoudig een getal inlezen, b.v. met

  ```
  fin >> getal;
  ```

- Denk, voordat je begint met `losOpTD` en `losOpBU`, eerst goed na over de recurrente betrekking die beide functies moeten gebruiken. En over hoe je bij `losOpBU` vervolgens ook de optimale tankstrategie kunt afleiden, uitgaande van de tabel(len) die je vult tijdens het berekenen van de minimale kosten.
- Bij de memberfuncties van klasse `TankPlanner`, behalve bij `leesIn`, moet je steeds eerst controleren of er al een geldige instantie is (ingelezen). Het is handig om daarvoor een boolean membervariabele te gebruiken die in `leesIn` een waarde krijgt.
- Houd er rekening mee dat de posities van de stations (hun x-coördinaat) ook negatief kunnen zijn.
- Hoewel er geen vaste bovengrenzen zijn gegeven voor de hoeveelheid en de prijs van de diesel die op een station te krijgen is, mag je ervanuitgaan dat de totale kosten van de reis royaal in een `int` passen.

Let op: het is niet toegestaan om de headers van de gegeven public memberfuncties in `tankplanner.h` te veranderen. Dan zou onze automatische test (met een ander main programma) bij de beoordeling namelijk niet goed kunnen werken. Je mag wel functies toevoegen. Verder kan het inleveren van lelijke, niet-elegante, of erg inefficiënte code 0.5 punt aftrek opleveren.

Algemene opmerkingen

- Maak / behoud een verstandige klassenstructuur.
- Je klassen mogen alleen `public` membervariabelen en -functies hebben die buiten de klasse bekend moeten zijn. Andere membervariabelen en -functies moeten `private` zijn.
- Als er in je klassen dynamisch geheugen wordt gealloceerd, denk dan ook aan een destructor.
- Functies mogen niet te lang zijn (maximaal 35 regels).
- Gebruik constantes waar dat zinvol is. Kijk bijvoorbeeld in bestand `constantes.h` in het skeletprogramma.
- Het werkende programma mag er op het scherm eenvoudig uitzien, maar moet wel duidelijk zijn. De enige te gebruiken headerfiles zijn in principe `iostream`, `sstream`, `iomanip`, `fstream`, `cstring`, `string`, `vector`, `utility`, `set`, `unordered_set`, `cstdlib`, `ctime` en `climits`. Wil je een andere headerfile gebruiken, vraag de docent. De headerfile `algorithm` mag in ieder geval niet gebruikt worden.

- Boven elke functie in `tankplanner.h` moet een commentaarblokje komen met daarin een korte beschrijving van wat de functie doet. Noem daarin tevens de gebruikte parameters: geef hun betekenis en geef aan hoe ze eventueel veranderd worden door de functie. Geef ook aan wat de memberfuncties met de membervariabelen van het object doen. Let verder op de layout (consequent inspringen) en op het overige commentaar bij de programmacode (zinvol en kort).
- Als je bij de vereiste controles in de memberfuncties een fout ontdekt, geef dan ook een passende foutmelding aan de gebruiker.
- Het programma moet onder Linux bij LIACS getest zijn en werken. Dat kan vanuit huis bijvoorbeeld op de huisuil, zie de instructie op de website.

Verslag

Het verslag moet getypt zijn in LaTeX, en moet bevatten:

- Een korte introductie, met uitleg over het probleem en de opdracht.
- Een paragraaf waarin je uitlegt wat `kosten[i][j][k]` (of `kosten[i][j]`, als je aan de jokers niet bent toegekomen) voorstelt. Vervolgens behandel je de recurrente betrekkeing waar `kosten[i][j][k]` aan voldoet:
  - Leg uit hoe je `kosten[i][j][k]` kunt berekenen uit waardes van `kosten` voor deelproblemen, en waarom dat zo is. Wees precies in je uitleg en geef zo mogelijk een formule.
  - Vermeld ook wat `kosten[0][j][k]` is voor alle mogelijke j en k, en waarom.
- Een paragraaf waarin je uitlegt hoe je een tankstrategie bepaalt die overeenkomt met de minimale kosten. Dat wil zeggen: hoe je uit de tabel(len) die je vult tijdens `losOpBu`, terugrekent hoeveel diesel Sophie en Caroline bij elk pompstation moeten tanken, en waar ze een joker inzetten.
- Een paragraaf waarin je uitlegt wat de worst-case tijdcomplexiteit van je algoritme met bottom-up dynamisch programmeren is (grote O), uitgedrukt in de capaciteit van de dieseltank c, het aantal pompstations n, en het aantal jokers.
- Een paragraaf met resultaten. Hier geef je voor drie zelf gemaakte invoerbestanden met $5 \leq n \leq 20$ het resultaat: de minimale reiskosten en de daarbij behorende tankstrategie. Vermeld bij de tankstrategie ook op welke stations je een joker hebt ingezet.

Aanvullingen / tips / vragen

Eventuele verdere aanvullingen of tips bij de programmeeropdracht komen op de website van het vak

`https://liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

Daar is ook een subpagina met onder andere een template voor het LaTeX-verslag. De behaalde cijfers komen te zijner tijd in Brightspace te staan.

Heb je vragen over de opdracht, dan kun je die uiteraard stellen tijdens de practicumbijeenkomsten van het vak. Je kunt ook emailen naar algoritmiek@liacs.leidenuniv.nl

In te leveren

Via Brightspace:

- je programma (alle .cc/.h bestanden en Makefile voor Linux bij LIACS) met de drie invoerbestanden uit het verslag,
- en een PDF van je verslag

samen in één .zip, .tgz of .tar.gz bestand. Vermeld overal duidelijk de namen van de makers.
Uiterste (!) inleverdatum:
Voor maximale punten: vrijdag 23 mei 2025, 23.59 uur.
Met aftrek van 1 punt: woensdag 4 juni 2025, 23.59 uur.
Met aftrek van 2 punten: dinsdag 10 juni 2025, 23.59 uur.

Om via Brightspace in te leveren, moet je eerst een nieuw groepje vormen voor deze opdracht (ook als je geen programmeerpartner hebt). Vervolgens kun je namens dat groepje je oplossing inleveren.

Normering: werking 5 punten; commentaar en layout 1 punt; modulaire opbouw en OOP 1 punt; verslag 3 punten.

Het is niet toegestaan om je opdracht te laten maken door een large language model als ChatGPT of DeepSeek. Bij opvallende inzendingen kunnen de makers worden uitgenodigd om hun oplossing toe te lichten.