Programmeeropdracht 2 — Woordsompuzzel
Algoritmiek, voorjaar 2025

Inleiding

In het voorjaar heeft Sinterklaas niet zo veel te doen. Hij gebruikt die periode vooral om inspiratie op te doen voor de volgende, drukke decembermaand. Zo bezocht hij een college Algoritmiek, waarin een woordsompuzzel van de vorm `DONALD + GERALD = ROBERT` werd besproken. Hij zag meteen een kans: zulke puzzels zouden wel eens heel populair bij de (kleine en grotere) kinderen kunnen worden. Zou hij in december geen boekjes met zulke puzzels kunnen gaan schenken?

Er was wel een probleem: niet elke combinatie van drie woorden vormt een geschikte puzzel: sommige combinaties hebben geen enkele oplossing, sommige combinaties hebben juist meer dan één oplossing. Bijvoorbeeld: de puzzel `SINT+PIET=FEEST` heeft precies één oplossing, de puzzel `SINT+PIET=RUZIE` heeft meer dan één oplossing.

Kinderen willen alleen puzzels hebben met een unieke oplossing. Maar ja, hoe construeer je zulke puzzels? Kun jij Sinterklaas helpen? Schrijf een C++programma dat o.a. voor een gegeven puzzel van drie woorden bepaalt hoeveel oplossingen er zijn, en dat voor twee gegeven woorden bepaalt hoeveel puzzels met een unieke oplossing er te maken zijn met die twee woorden.

Specificatie

In 'gewone' woordsompuzzels wordt gerekend in het decimale stelsel, waar het grondtal voor het opbouwen van getallen 10 is. Dan kun je dus maximaal tien letters gebruiken in je puzzel, zoals ook in de voorbeelden hierboven is gebeurd. Sinterklaas wil graag zo veel mogelijk mensen een passend cadeau kunnen geven, en wil daarom puzzels van verschillende moeilijkheidsgraden. Daarom kan bij zijn puzzels het grondtal g varieren van 2 t/m 26. Zo komt bijvoorbeeld het 'getal' 3-12-5 bij $g = 14$ overeen met het decimale getal $3 * 14^2 + 12 * 14^1 + 5 * 14^0 = 3 * 196 + 12 * 14 + 5 * 1 = 761$.

Een geschikte puzzel is een puzzel die bestaat uit drie niet-lege woorden van allemaal hoofdletters, waarvoor precies één oplossing bestaat bij het opgegeven grondtal g. Een oplossing is een toekenning van 'cijfers' uit de verzameling $\{0, 1, 2, \ldots, g-1\}$ aan de letters in de puzzel, zodat de som klopt. Verschillende cijfers moeten aan verschillende letters worden toegekend, en omgekeerd. Eventueel, als er minder letters in de puzzel dan mogelijke cijfers zijn, blijven er cijfers ongebruikt. De getallen in de puzzels mogen geen onnodige voorloopnullen (*leading zeroes*) hebben: een getal als 0172 mag dus niet ontstaan. Het enige getal dat met een 0 mag beginnen, is het getal 0 zelf.

Je programma moet dus kunnen bepalen of drie strings `nwWoord0`, `nwWoord1` en `nwWoord2` een geschikte puzzel `nwWoord0 + nwWoord1 = nwWoord2` vormen. In het bijzonder moet het kunnen bepalen hoeveel verschillende oplossingen er zijn bij een puzzel. Het is a priori niet gezegd dat de strings alleen hoofdletters bevatten, laat staan *hoogstens g verschillende* hoofdletters. Dat moet allemaal ook gecontroleerd worden.

Om als gebruiker een beetje te kunnen spelen met de puzzel, moet het ook mogelijk zijn om bij een puzzel waardes toe te kennen aan één of meer letters. Of juist om letters waaraan eerder een waarde is toegekend weer vrij te geven. Wanneer je bij een puzzel waarin sommige letters al een waarde hebben naar een oplossing gaat zoeken, moet die oplossing consistent zijn met de reeds toegekende waardes.

Daarnaast moet je programma voor twee gegeven strings `nwWoord0` en `nwWoord1` bij een grondtal g bepalen hoeveel geschikte puzzels `nwWoord0 + nwWoord1 = nwWoord2` daarbij te construeren zijn. Hierbij moet je dubbele puzzels zien te voorkomen, zie de uitleg over de functie `construeerPuzzels` in de paragraaf 'Voor u beschikbaar' hieronder.

Voor u beschikbaar

Op de website van het vak

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

is een skeletprogramma beschikbaar, waarin een klasse `WoordSomPuzzel` wordt gedefinieerd, die door het hoofdprogramma gebruikt wordt om o.a. een woordsompuzzel op te lossen, en woordsompuzzels te construeren.

Het programma wordt onder Linux gecompileerd met het commando `make`. Vervolgens kun je het met het commando `./WoordSomPuzzel` runnen. Je krijgt dan een menu aangeboden, met de keuze om een puzzel op te lossen of puzzels te construeren. Bedoeling is om de `TODO`'s in woordsompuzzel.cc en woordsompuzzel.h uit te voeren.

De meeste functies die je moet implementeren worden toegelicht in het skeletprogramma, speciaal in `woordsompuzzel.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.
Van een aantal functies volgt nog een nadere toelichting:

- `WoordSomPuzzel::zoekOplossingen (...)`

  Dit is de functie die verantwoordelijk is voor het backtracking element van de opdracht. Zonder deze functie zal je oplossing voor de opdracht **nooit** voldoende worden, hoe goed de rest ook is.

  Deze functie moet het aantal oplossingen bepalen van de puzzel `nwWoord0 + nwWoord1 = nwWoord2` bij grondtal g. Dit moet met behulp van backtracking gebeuren. Aan de letters van de drie woorden moeten cijfers uit $\{0, 1, 2, \ldots, g-1\}$ worden toegekend, **te beginnen bij de achterste letters**.

  Zodra de achterste letters van `nwWoord0` en `nwWoord1` een waarde hebben, kun je die namelijk optellen, en daarmee kun je bepalen wat de achterste letters van `nwWoord2` moeten zijn. Als daarbij iets niet klopt, hoef je niet meer verder te zoeken bij de huidige, gedeeltelijke toekenning.

  Zoals gezegd mogen oplossingen geen onnodige voorloopnullen bevatten. In de puzzel `DONALD + GERALD = ROBERT` mogen de `D`, `G` en `R` dus geen 0 worden. Daar moet je bij het toekennen van cijfers aan de letters al direct rekening mee houden.

  Soms kun je aan de lengtes van de drie woorden al zien dat er geen oplossing bestaat. In dat geval moet de constructor van klasse `WoordSomPuzzel` al vaststellen dat de puzzel niet geldig is.

  De functie `zoekOplossingen` moet ook het aantal deeloplossingen bepalen dat bij het zoeken naar oplossingen wordt bekeken. `Je mag er niet vanuit gaan` dat de parameter

`deeloplossingen` bij de eerste aanroep al gelijk is aan 0. Een elegante manier om dit voor elkaar te krijgen is door `zoekOplossingen` als *wrapper-functie* te gebruiken, met een recursieve hulpfunctie waar het echte rekenwerk gebeurt.

- `WoordSomPuzzel::construeerPuzzels (...)`

  Deze functie moet bepalen hoeveel verschillende puzzels met een unieke oplossing met grondtal g er te construeren zijn, van de vorm `nwWoord0 + nwWoord1 = ...`, waarbij je op de `...` een derde woord moet invullen. Dit aantal zul je waarschijnlijk met brute force moeten bepalen. Dat kan (in ieder geval) op twee manieren:

  1. Ken op alle mogelijke manieren cijfers toe aan de letters van `nwWoord0` en `nwWoord1`. Kijk wat de som is, en of de resulterende puzzel een unieke oplossing heeft.
  2. Construeer alle mogelijke woorden voor het derde woord, en kijk of de resulterende puzzel een unieke oplossing heeft.

  Implementeer een van de mogelijke manieren.

  Het is de bedoeling om dubbele puzzels te voorkomen. Bijvoorbeeld, bij `DONALD + GERALD = ...` bevatten de twee gegeven woorden al acht verschillende letters. Als derde woord kun je `ROBERT` hebben, wat betekent dat de letters `B` en `T` zijn toegevoegd, van links naar rechts. Voor hetzelfde geld zou je ook `ROTERB` als derde woord kunnen kiezen, of `ROBERC`. Dit beschouwen we als equivalente puzzels, die je maar één keer moet tellen. Je kunt hier voor zorgen door de letters die je toevoegt aan de puzzel in een vaste volgorde te kiezen.

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 karakter en een getal 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 (gedeeltelijke) oplossingen van de puzzel representeert. Dat wil zeggen: hoe je bijhoudt welke waardes al aan welke letters zijn toegewezen. Implementeer vervolgens achtereenvolgens de constructor en betrekkelijk eenvoudige memberfuncties als `drukAfPuzzel`, `kenWaardeToe` en `maakLetterVrij`. De functie `zoekOplossingen` die daarna aan de beurt komt, kun je ten slotte gebruiken bij de implementatie van de functie `construeerPuzzels`.
- Het lijkt handig om bij het zoeken van oplossingen van een woordsompuzzel een recursieve hulpfunctie te gebruiken, die als parameter een letter/positie in een van de woorden meekrijgt: de letter waaraan als eerstvolgende letter een cijfer moet worden toegekend. Bij het construeren van puzzels kun je iets dergelijks doen.
- Je mag, net als in het skeletprogramma al gebeurt, aannemen dat de gebruiker een string invoert als daarom gevraagd wordt. Er moet echter wel worden gecontroleerd of de strings alleen hoofdletters (hoogstens g verschillende) bevat, en of de lengtes van de strings een oplossing sowieso onmogelijk maken.

- Bij de memberfuncties van klasse `WoordSomPuzzel` moet je steeds eerst controleren of er een geldige puzzel is. Het is handig om daarvoor een boolean membervariabele te gebruiken die in de constructor een waarde krijgt.
- Het kan handig zijn om verschillende karakters rechtstreeks te laten corresponderen met verschillende indices in een array, bijvoorbeeld als je wilt bijhouden welk cijfer is toegekend aan welk karakter: "het cijfer dat aan de letter 'T' is toegekend, is 0." Bedenk dan dat je eenvoudig karakters naar getallen kunt converteren en omgekeerd. Voor een hoofdletter `kar` en een integer `i` kun je bijvoorbeeld schrijven:

  ```
  i = kar - 'A';
  ```

  Het resultaat is dat `i` het nummer van `kar` in het alfabet wordt. Als `kar` gelijk is aan `T` (de twintigste letter in het alfabet), dan maakt dit `i` gelijk aan $20 - 1 = 19$. Omgekeerd kun je bijvoorbeeld

  ```
  kar = (char)('A' + i);
  ```

  schrijven om bij het getal `i` de corresponderende hoofdletter `kar` te bepalen.

Let op: het is niet toegestaan om de headers van de gegeven public memberfuncties in `woordsompuzzel.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 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 bij memberfuncties ook aan wat deze 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 de puzzel en de opdracht.
- Een paragraaf waarin je uitlegt
  - hoe je, voor de functie `zoekOplossingen`, oplossingen voor een woordsompuzzel stap voor stap opbouwt,
  - hoe je daarbij controleert of de huidige gedeeltelijke toekenning van cijfers aan de letters (mogelijk) nog uitgebreid kan worden tot een complete toekenning,
  - hoe je, voor de functie `construeerPuzzels`, verschillende puzzels construeert bij twee gegeven strings,
  - en hoe je daarbij dubbele puzzels voorkomt.
- Een paragraaf met antwoorden op de volgende vragen over specifieke puzzels:
  - Beredeneer wat de verschillende oplossingen zijn voor de puzzel `A + B = B` en grondtal $g \geq 2$. Hoeveel oplossingen zijn er dus?
  - Beredeneer wat de verschillende oplossingen zijn voor de puzzel `A + B = CD` en $g = 10$, wanneer je eist dat `A < B`. Hoeveel oplossingen van de puzzel `A + B = CD` zijn er dus in totaal bij dit grondtal?
  - Hoeveel oplossingen kent de puzzel `STOOM + BOOT = PAKJE` volgens je programma bij grondtal $g = 10$?
  - Hoeveel oplossingen kent de puzzel `SINT + PIET = RUZIE` volgens je programma bij grondtal $g = 9$, $g = 10$, $g = 11$, $g = 20$ en $g = 26$? En hoeveel oplossingen als je bij $g = 26$ waardes $N = 3$ en $P = 21$ toekent? Hoe lang (milliseconden, seconden, minuten...) had je programma nodig om deze aantallen te bepalen, en hoeveel deeloplossingen bekeek het daarbij? Zet al deze resultaten in een overzichtelijke tabel.
  - Hoeveel puzzels met een unieke oplossing zijn er volgens je programma te construeren, van de volgende vormen:
    * `SINT + PIET = ...`
    * `KLAAS + PAARD = ...`
    * `MIJTER + MANTEL = ...`

    In alle gevallen met grondtal $g = 10$. Geef bij alle drie de vormen ook aan hoe lang (milliseconden, seconden, minuten...) je programma nodig had om het aantal puzzels te bepalen. Als je programma meer dan vijftien minuten nodig heeft voor een tweetal woorden, mag je de constructie afbreken. Zet ook al deze resultaten in een overzichtelijke tabel.

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)
- 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: dinsdag 29 april 2025, 23.59 uur. Je kunt maximaal twee weken te laat inleveren, maar dan gaat er wel per (deel van een) week een punt van het cijfer af. Zie de website voor de exacte regeling.

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.