Programmeeropdracht 3: Trouwboeket
Algoritmiek voorjaar 2024

Inleiding

Giorgia is bloemen aan het plukken in het veld, verschillende soorten, meerdere bloemen van eenzelfde soort, een heel boeket. Een beetje spannend is het wel, want als ze klaar is met plukken, zal ze haar vriend Geert een bloemensoort laten uitkiezen. Vervolgens zal Giorgia met 'hij houdt van me, hij houdt niet van me' één voor één de bloemen van die soort opzij leggen. Als het bij de laatste bloem van die soort 'hij houdt van me' is (als er een oneven aantal bloemen van die soort was), zullen Geert en Giorgia trouwen. Zo niet, dan gaan ze uit elkaar. Giorgia hoopt op het eerste, dus ze doet haar best om van alle soorten een oneven aantal exemplaren in haar boeket te krijgen. Kun jij haar helpen?

Specificatie

Het veld waarin Giorgia bloemen plukt is rechthoekig van vorm, en bestaat uit h rijen (de hoogte) van elk b vierkante vakken (de breedte), die allemaal even groot zijn. Het veld is hoogstens 100 vakken hoog en hoogstens 100 vakken breed. Giorgia begint in de linker bovenhoek van het veld, op vakje (0,0), en gaat iedere keer óf naar het rechterbuurvak óf naar het benedenbuurvak. Ze verlaat het veld wanneer ze in de rechter benedenhoek, op vakje $(h-1, b-1)$ is aangekomen. In elk vak staat één bloem. De soort waartoe een bloem behoort, wordt aangegeven met een cijfer s met $0 \leq s \leq 7$. In elk vak waar Giorgia komt, plukt ze de betreffende bloem. Het gaat er dus om een route te bepalen van de linker bovenhoek naar de rechter benedenhoek, waarop van zoveel mogelijk bloemen een oneven aantal exemplaren voorkomt. Merk op dat het getal 0 (nul) even is.

Voorbeeld

Beschouw het volgende veld, waarbij de route van Giorgia loopt via de roodgedrukte cijfers.

	kolom											
	0	1	2	3	4	5	6	7	8	9	10	11
rij 0	0	3	6	1	4	7	0	1	2	3	4	5
1	1	4	7	2	5	6	7	0	1	2	3	4
2	2	5	0	3	6	5	6	7	0	1	2	3
3	3	6	1	4	7	4	5	6	7	0	1	2
4	4	7	2	5	0	3	4	5	6	7	0	1
5	5	0	3	6	1	7	5	3	1	7	5	3
6	6	1	4	7	2	6	4	2	0	6	4	2
7	7	2	5	0	3	5	3	1	7	5	3	1
8	0	3	6	1	4	4	2	0	6	4	2	0
9	1	4	7	2	5	3	1	7	5	3	1	7
10	2	5	0	3	6	2	0	6	4	2	0	6

In dit geval zou Giorgia op het volgende boeket komen: drie keer bloem 0, twee keer bloem 1, vier keer bloem 2, twee keer bloem 3, twee keer bloem 4, twee keer bloem 5, drie keer bloem 6, vier keer bloem 7. Dit is geen optimaal boeket.

Opdracht

Het is de bedoeling om een veld met bloemen in te lezen uit een tekstbestand en dan voor dit veld een optimaal boeket voor Giorgia te bepalen, op haar route van linksboven naar rechtsonder. Hiertoe moet je per vak in het veld bijhouden welke aantallen bloemen je van de verschillende soorten al in je boeket kunt hebben, als je op dat vak komt (en de daar aanwezige bloem plukt). De exacte aantallen zijn niet van belang; het is voldoende om van elk soort bloemen bij te houden of je er een even aantal of een oneven aantal van kunt hebben. Acht bits zijn daarvoor voldoende, en acht bits komen overeen met de getallen 0 tot en met 255.

Houd voor elk vak en voor elke mogelijke bitstring (getal tussen 0 en 255) bij of je in het vak een boeket kunt hebben dat met die bitstring overeenkomt. **Gebruik daarvoor een driedimensionaal array `mogelijk` van booleans**, met dimensies hoogte × breedte × 256.

Wanneer b.v. `mogelijk[i][j][154]=true`, betekent dit dat het mogelijk is om in vak (i,j) een boeket te hebben dat met de bitstring 10011010 (binaire representatie van 154) overeenkomt: een oneven aantal bloemen van soort 7 (want bit 7 vanaf rechts, het meest linker bit is 1), een even aantal bloemen van soort 6 (want het hierna volgende bit is 0), een even aantal bloemen van soort 5, een oneven aantal bloemen van soort 4, een oneven aantal bloemen van soort 3, een even aantal bloemen van soort 2, een oneven aantal bloemen van soort 1, en een even aantal bloemen van soort 0. Als `mogelijk[i][j][154]=false`, is zo'n boeket niet mogelijk in vak (i,j).

Bedenk een **recurrente betrekking** waarmee je `mogelijk[i][j][k]` uitdrukt in booleans `mogelijk[i'][j'][k']` voor buurvakjes (i', j'), ofwel vakjes (i', j') die je direct vóór (i, j) kunt bezoeken op je route vanaf vakje (0,0). Uiteindelijk gaat het natuurlijk om de waardes voor vakje $(h-1, b-1)$ rechtsonder. In de paragraaf 'Verslag' gaan we in iets meer detail in op deze recurrente betrekking.

De recurrente betrekking gebruik je vervolgens op drie verschillende manieren om het probleem voor Giorgia op te lossen:

1. Rechtstreeks recursief. Dus met een recursieve functie die rechtstreeks gebaseerd is op de recurrente betrekking, zonder tabel met oplossingen van deelproblemen.

2. Met top-down dynamisch programmeren. Opnieuw met een recursieve functie op basis van de recurrente betrekkingen, maar nu met een tabel met oplossingen van deelproblemen, zodat elk deelprobleem maar één keer hoeft te worden opgelost.

3. Met bottom-up dynamisch programmeren. De tabel met oplossingen van deelproblemen wordt nu niet recursief, maar systematisch iteratief gevuld.

Als je de methode met bottom-up dynamisch programmeren niet (goed genoeg) implementeert, zal je inzending nooit voldoende zijn, hoe goed de rest van je inzending ook is.

Als je niet goed weet wat de drie methodes inhouden, kijk dan nog eens naar de voorbeelden in hoorcollege 8 van dit voorjaar. Voor het berekenen van de Fibonaccigetallen zijn alle drie de methodes uitgewerkt. Zoek de verschillen. Voor andere voorbeelden zijn vaak twee van de drie methodes uitgewerkt.

Je programma moet onder Linux bij LIACS getest zijn en werken.

Invoer

De invoer-tekstbestanden zijn van de volgende vorm:

- een regel met twee getallen h en b, waarvoor zou moeten gelden $1 \leq h \leq 100$ en $1 \leq b \leq 100$: respectievelijk de hoogte en de breedte van het veld,
- h regels met elk b getallen s, die zouden moeten voldoen aan $0 \leq s \leq 7$: de bloemsoorten van links naar rechts in de betreffende rij van het veld. De eerste regel correspondeert met de bovenste rij, de laatste regel met de onderste rij.

De getallen die op een zelfde regel staan, worden steeds gescheiden door een spatie.

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;
```

Voor u beschikbaar

Op de website van het vak

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

is een skeletprogramma beschikbaar, waarin een klasse `Veld` wordt gedefinieerd, die door het hoofdprogramma gebruikt wordt. Het programma wordt onder Linux gecompileerd met het commando `make`. Vervolgens kun je het met het commando `./Boeket` runnen. Je krijgt dan een menu aangeboden, waar je een tekstbestand kunt opgeven met een veld om in te lezen. Hieerna volgt een tweede menu, waar je kunt kiezen om met een van de drie methodes een optimaal boeket voor Giorgia te bepalen. Bedoeling is om de `TODO`'s in de gegeven bestanden `veld.cc` en `veld.h` uit te voeren.

In de bestanden `standaard.h` en `standaard.cc` zijn enkele standaardfuncties uitgewerkt. Die kun je o.a. gebruiken om te testen of een integer tussen bepaalde grenzen ligt, en om te controleren wat het i^{e} bit van achteren is van een getal.

Bij het skeletprogramma zitten ook nog drie andere bestanden:

`veld1.txt`, een invoer-tekstbestand dat met het voorbeeldveld met bloemen hierboven overeenkomt,

`testbits.cc`, een programma om te spelen met bitrepresentaties van getallen, inclusief twee functies `getBit` en `setBit` die ook voorkomen in `standaard.cc/h`.

`genereerveld.cc`, een programma om een random veld met bloemen te genereren, waarbij je waardes voor h en b kunt opgeven.

De functies die je moet implementeren worden toegelicht in het skeletprogramma, met name in `veld.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.
Voor enkele functies geven we hier ook nog een toelichting:

- `bepaalOptimaalBoeketRec (...)` en `bepaalOptimaalBoeketTD (...)`

  Deze functies bepalen recursief een optimaal boeket. In principe zou je hierbij voor elk mogelijk boeket (gerepresenteerd door getallen $k = 0, 1, \ldots, 255$) moeten nagaan of zo'n boeket mogelijk is op vakje $(h-1, b-1)$. Als je echter zeker weet dat je al een optimaal boeket k hebt gevonden, zou je de andere waardes van k kunnen overslaan.

- `bepaalOptimaalBoeketBU (...)`

  Deze functie berekent niet alleen een optimaal boeket, maar bepaalt ook een concrete route door het veld die tot dit optimale boeket leidt.

  De route moet overigens van vakje (0,0) tot en met $(h-1, b-1)$ lopen, ook wanneer je de route in omgekeerde volgorde opbouwt.

Enkele tips bij het programmeren

- Kijk in het framework waar allemaal `TODO` staat, voor met name de functies die geïmplementeerd moeten worden.
- In het skeletprogramma wordt gebruik gemaakt van `vector` en `pair`. Als je niet weet hoe je die gebruikt, zoek het op in de C++ reference. Een handige functie voor het gebruik van pairs is `make_pair`, waarmee je een nieuw pair aanmaakt.
- Implementeer vervolgens enkele eenvoudige functies, zoals de constructor, `leesInVeld`, en `drukAfVeld`.
- Als je bij de vereiste controles in de memberfuncties een fout ontdekt, geef dan ook een passende foutmelding aan de gebruiker.
- In alle drie de methodes om een optimaal boeket te bepalen heb je de recurrente betrekking voor `mogelijk[i][j][k]` nodig. **Bedenk deze recurrente betrekking eerst, voordat je deze drie methodes gaat programmeren.**

Let op: het is niet toegestaan om de headers van de gegeven public memberfuncties in `veld.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, zoals ook in het skeletprogramma al gebeurt. Kijk bijvoorbeeld in bestand `constantes.h`.

- Het werkende programma mag er op het scherm eenvoudig uitzien, maar moet wel duidelijk zijn. De enige te gebruiken headerfiles zijn in principe `iostream`, `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).

- 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 een beschrijving van het probleem dat je voor Giorgia moet oplossen.

- Een paragraaf met een beschrijving en toelichting van de recurrente betrekking die je hebt gebruikt bij de drie methodes. Laat mogelijk$(i, j, s_7, s_6, \ldots, s_1, s_0)$ weergeven of het mogelijk is om in vak (i, j) van het veld een boeket te hebben dat met de bitstring $s_7 s_6 \ldots s_1 s_0$ overeenkomt. [1] Hoe bereken je mogelijk$(i, j, s_7, s_6, \ldots, s_1, s_0)$ dan (uit de waardes in andere vakken)?

  En hoe bepaal je uit de waardes van mogelijk$(i, j, s_7, s_6, \ldots, s_1, s_0)$ het maximale aantal soorten bloemen met een oneven aantal aan het eind van de route?

- Een paragraaf met uitleg hoe je vervolgens een optimale route bepaalt.

- Een paragraaf met een analyse van zowel de tijdcomplexiteit als de ruimtecomplexiteit van het gebruikte algoritme voor bottom-up dynamisch programmeren. Allereerst voor de eigenlijke situatie, waarin er acht soorten bloemen zijn, maar vervolgens ook voor een situatie dat er $n \geq 1$ soorten bloemen zouden zijn,

- Een beschrijving van een (voor Giorgia) zo slecht mogelijk veld met hoogstens 10×10 vakken met bloemen. Dat wil zeggen: een $h \times b$ veld met $1 \leq h \leq 10$ en $1 \leq b \leq 10$, waarin elk van de acht soorten bloemen minstens één keer voorkomt, waarvoor Giorgia (zelfs bij een optimale route) van zo min mogelijk soorten een oneven aantal kan plukken. De exacte waardes van h en b mag je zelf bepalen. Beredeneer ook *waarom* dit een zo slecht mogelijk veld is. Om hoeveel soorten met een oneven aantal gaat het dan?

Dit jaar hoef je in je verslag niet meer een 'appendix' met je complete programma (alle .cc/.h bestanden) op te nemen. We printen het toch niet meer uit.

[1] In je programma wordt dit een array-element `mogelijk[i][j][k]`, waarbij k het getal is dat met de bitstring $s_7 s_6 \ldots s_1 s_0$ gerepresenteerd wordt.

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. Lever geen object-bestanden (.o) of executables in.

Uiterste (!) inleverdatum: dinsdag 28 mei 2024, 23.59 uur.
Met 1 punt aftrek: uiterlijk vrijdag 21 juni, 23.59 uur.
Met 2 punten aftrek: uiterlijk vrijdag 5 juli, 23.59 uur.

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

Normering: werking 5.0 punten; commentaar en layout 1 punt; modulaire opbouw en OOP 1 punt; verslag 3.0 punten.

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