Programmeeropdracht 3 — Beurs
Algoritmiek, voorjaar 2021

Inleiding

De spaarrente is historisch laag. Als je een hoog saldo op je rekening hebt, moet je zelfs rente betalen. Logisch dus, dat mensen overstappen op aandelen. Maar ja, hoeveel valt daar te verdienen? Die vraag valt natuurlijk lastig te beantwoorden, omdat niemand precies weet wat de beurskoersen en de rentepercentages in de toekomst gaan doen. Maar stel nu dat je dat allemaal wel zou weten, zou je dan precies kunnen berekenen hoeveel je kunt verdienen? Het antwoord hierop is ja, en in deze opdracht gaan we dat doen met behulp van dynamisch programmeren.

Onze beurswereld

We stellen ons een belegger voor die op dag 0 een bepaald bedrag tot zijn beschikking heeft. Op elke dag kan hij beslissen om aandelen te kopen of te verkopen. Dat gebeurt dan tegen de koersen van de aandelen op die dag. Voor elke koop of verkoop betaalt de belegger een percentage aan provisie. Om geen al te grote risico's te nemen, wil de belegger niet te veel aandelen van hetzelfde bedrijf hebben. Sterker nog: van elk bedrijf wil hij hoogstens één aandeel bezitten.

In de praktijk zal de belegger meestal ook een bedrag in kas (dus niet in aandelen) hebben. Dit bedrag mag niet negatief worden, wat betekent dat de belegger niet altijd alles kan kopen wat hij zou willen. Over het bedrag in kas ontvangt (of betaalt) hij rente, en die rente varieert per dag. Na verloop van tijd wil de belegger een wereldreis gaan maken. Hij verkoopt dan al zijn aandelen. Zijn beleggingsdoel is om bij het begin van zijn wereldreis, na verkoop van al zijn aandelen, zo veel mogelijk geld in kas te hebben.

Notatie

Het bedrag waarmee de belegger begint, noemen we b_0. Het aantal bedrijven waarin hij kan beleggen, noteren we met n, en er geldt $1 \leq n \leq 8$. Een aandeel van bedrijf i ($0 \leq i < n$) heeft op dag $t \geq 0$ een koers van koers(t, i), zowel voor kopen als voor verkopen. Deze koers is vast, en varieert dus niet over de dag. De provisie die de bank rekent voor koop of verkoop van een aandeel bedraag een vast percentage p. Bijvoorbeeld, als $p = 1$, dan betaal je bij aankoop van een aandeel 1% meer dan de koers, terwijl je bij verkoop van een aandeel 1% minder dan de koers ontvangt. De rente over het bedrag in kas op dag $t \geq 0$, noemen we rente(t). Als de rente bijvoorbeeld 2% is, en je eindigt dag t met een bedrag b, dan begin je de dag $t + 1$ met $1.02 * b$.

Omdat de belegger op elk moment hoogstens één aandeel van elk bedrijf bezit, kun je zijn aandelenbezit op een bepaalde dag weergeven met een bitstring a van lengte n, ofwel: een getal tussen 0 en $2^n - 1$. Een bit 1 correspondeert dan met het bezit van het betreffende aandeel, en een bit 0 met het niet-bezit.

Het *maximale* bedrag in kas op dag $t \geq 0$ (om precies te zijn: aan het eind van dag t, na de transacties van die dag) bij een aandelenbezit/bitstring a noteren we met bedrag(t, a). We hebben dus bedrag$(0, 0) = b_0$, waarbij we de tweede component 0 als bitstring interpreteren. De dag waarop de belegger al zijn aandelen wil verkopen om een wereldreis te maken noemen we t_w. Doel is nu om bedrag$(t_w, 0)$ te bepalen, en om te bepalen hoe de belegger dit bedrag bereikt, d.w.z.: op welke dagen hij welke aandelen moet kopen of verkopen.

Analyse

Voor een bitstring/getal a bereken je de waarde van bedrag$(0, a)$ ($t = 0$ dus) rechtstreeks uit b_0 en de koersen van dag 0.

Voor een dag $t \geq 1$ en een bitstring/getal a kun je de waarde van bedrag(t, a) berekenen uit de mogelijke waardes van bedrag$(t - 1, b)$ voor alle bitstrings b die mogelijk waren op dag $t - 1$ (corresponderend met een niet-negatief bedrag in kas). Je moet jezelf hierbij afvragen: hoeveel rente verdien ik tussen dag $t - 1$ en dag t, hoeveel geld heb ik nodig om op dag t de aandelen te kopen die wel in a zitten, maar niet in b, en hoeveel geld krijg ik als ik op dag t de aandelen verkoop die niet meer in a zitten, terwijl ze wel in b zitten.

Voor u beschikbaar

Op de website van het vak

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

is een skeletprogramma beschikbaar, waarin een klasse `Beurs` wordt gedefinieerd, die door het hoofdprogramma gebruikt wordt om een instantie van dit probleem in te lezen en op te lossen. Het programma wordt onder Linux gecompileerd met het commando `make`. Vervolgens kun je het met het commando `./Beurs` runnen. Nadat je een instantie hebt ingelezen, kun je in een menu kiezen op welke wijze je de optimale oplossing wilt bepalen.

Bij het skeletprogramma zit ook een programma waarmee je zelf randominstanties kunt genereren: `genereerinstantie.cc`. Je kunt dit onder Linux compileren met het commando `make -f MakefileGenereer`, waarna je het kunt aanroepen met `./Genereer`. Behalve de gewenste waardes van t_w, n, p en b_0 vraagt het programma je om grenzen op te geven waarbinnen de koersen kunnen liggen, en grenzen waarbinnen de rentepercentages kunnen liggen.

Opdracht

Bedoeling is om de `TODO`'s in `beurs.cc` en `beurs.h` uit te voeren. Desgewenst kun je ook de standaardfuncties uit de bestanden `standaard.h` en `standaard.cc` weer gebruiken. Een extra standaardfunctie die al beschikbaar is, is `schrijfDouble`. Desgewenst kun je zelf standaardfuncties toevoegen.

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.

In `beurs.h` staat gespecificeerd wat er in elke functie moet gebeuren. In de functie `bepaalMaxBedragBU` moet **met bottom-up dynamisch programmeren** het maximale bedrag worden berekend dat onze belegger aan het eind van dag t_w (dus aan het begin van zijn wereldreis) tot zijn beschikking kan hebben. In dit geval moet een tabel `bedrag` *iteratief* ingevuld worden, waarna je daar aan het eind de waarde van bedrag$(t_w, 0)$ uit kunt aflezen. Ook de lijst van transacties om tot dit bedrag te komen, moet geretourneerd worden. **Als je bij functie `bepaalMaxBedragBU` geen gebruik maakt van dynamisch programmeren, zal je inzending nooit voldoende zijn, hoe goed de rest van je inzending ook is.**

In de functie `bepaalMaxBedragRec` moet bedrag$(t_w, 0)$ ook berekend worden, maar dan recursief. Afhankelijk van parameter `memo` moet dit wel (`memo` is `true`) of niet (`memo` is `false`) met opslag en hergebruik van berekende deelresultaten zijn. In het eerste geval spreken we van top-down dynamisch programmeren, in het tweede geval van een rechtstreeks recursieve oplossing.

Invoer

Een invoerbestand voor deze programmeeropdracht is als volgt opgebouwd:

- Een regel met twee integers erop, gescheiden door een spatie: t_w en n, met $1 \leq t_w \leq 100$ en $1 \leq n \leq 8$.
- Een regel met één double erop: p, met $0.0 \leq p \leq 50.0$.
- Een regel met één double erop: $b_0 \geq 0.00$.
- $t_w + 1$ regels, corresponderend met respectievelijk dag 0, dag 1, ..., dag t_w. Elke regel bevat n doubles, gescheiden door spaties: de koersen van de n aandelen op die dag. Voor elke koers k geldt: $0.00 < k \leq 100.00$.
- t_w regels, elk met één double erop: achtereenvolgens: rente(0), rente(1), ..., rente$(t_w - 1)$. Voor elke t geldt $-10.00 \leq$ rente$(t) \leq 100.00$. In het slechtste geval is er dus een negatieve spaarrente van 10%, in het gunstigste geval een positieve spaarrente van 100%.

Bij het skeletprogramma zit ook een voorbeeld van een invoerbestand: `beurs1.txt`. Als het goed is, herken je daarin het hierboven beschreven formaat. Op de website zullen we de waarde van bedrag$(t_w, 0)$ bij deze instantie bekendmaken.

Wellicht ten overvloede: als je in C++ `ifstream fin` gebruikt voor je tekstbestand, dan kun je heel eenvoudig een getal inlezen, b.v. een integer n of een double p, met:

```
fin >> n;
fin >> p;
```

Enkele tips bij het programmeren

- Kijk in het framework waar allemaal `TODO` staat, voor met name de functies die geïmplementeerd moeten worden.
- Implementeer vervolgens enkele eenvoudige functies, zoals `leesIn` en `drukAfInvoer`.
- Bedenk hoe je uit een integer a de bits afleidt die aangeven welke aandelen i de belegger bezit.
- In alle drie de varianten van de berekening zul je gebruik moeten maken van een recursieve formulering voor de waarde van bedrag(t, a). Maak, om die te bepalen, gebruik van de analyse hierboven.
- Hoewel je de resulterende doubles uiteindelijk in twee decimalen op het scherm mag zetten, moet je de bedragen en percentages tussendoor niet afronden.

Let op: het is niet toegestaan om de headers van de gegeven public memberfuncties in `beurs.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 0.25 punt aftrek opleveren.

Algemene opmerkingen

- Maak / behoud een verstandige klassenstructuur.
- Je klassen mogen alleen `public` membervariabelen en methoden hebben, die buiten de klasse bekend moeten zijn. Andere membervariabelen en methoden 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, bijvoorbeeld de constantes die in het skeletprogramma al gedefinieerd zijn.
- 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, set, unordered_set, cstdlib, climits en ctime. Wil je een andere headerfile gebruiken, vraag de docent. De headerfile algorithm mag in ieder geval niet gebruikt worden.
- Boven elke functie moet een commmentaarblokje 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 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 opdracht.
- Een paragraaf met een analyse van het probleem en een beschrijving van de oplossingsstrategie met dynamisch programmeren. Dit houdt in ieder geval het volgende in:
  - Geef een recursieve formulering voor de waarde van bedrag(t, a), en licht deze toe. Met deze formulering leg je vast
    - hoe je voor bitstrings a de waarde van bedrag$(0, a)$ kunt berekenen (voor $t = 0$ dus), en
    - hoe je voor $t \geq 1$ en bitstrings a de waarde van bedrag(t, a) kunt berekenen uit de mogelijke waardes van bedrag$(t - 1, b)$ voor bitstrings b.

    Wees precies in je formulering en je toelichting.
  - In je functie `bepaalMaxBedragBU` bereken je met bottom-up dynamisch programmeren de waarde van bedrag$(t_w, 0)$. Beredeneer wat de tijdscomplexiteit (grote O) van deze functie is, uitgedrukt in t_w en n.
  - Bij dynamisch programmeren maak je gebruik van een tabel met reeds berekende waardes. Leg intuïtief uit hoeveel geheugenruimte (qua grote O) deze tabel in beslag neemt, opnieuw uitgedrukt in t_w en n. Kunnen we hierop bezuinigen bij bottom-up dynamisch programmeren? Licht je antwoord toe.
  - Leg ook uit hoe je bijhoudt en/of achteraf in de tabel opzoekt wat de transacties zijn die hebben geleid tot het antwoord bedrag$(t_w, 0)$.
- Een paragraaf met resultaten. Hierin geef je voor drie zelfgemaakte, representatieve invoerbestanden:
  - het bedrag bedrag$(t_w, 0)$ (afgerond op twee decimalen), en voor elk van de drie varianten van de berekening (bottom-up DP, top-down DP, rechtstreeks recursief) hoeveel tijd de berekening vergde.

  Probeer hierbij minstens één invoerbestand te behandelen waarvoor de rechtstreeks recursieve berekening veel meer tijd kostte dan de berekeningen met dynamisch programmeren, maar nog wel minder dan vijf minuten.
- Een 'appendix' met je complete programma (alle .cc/.h bestanden).

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

Vorm in Brightspace een groepje voor deze opdracht, en upload:

- je programma (alle .cc/.h bestanden en Makefile voor Linux bij LIACS),
- de drie tekstbestanden waarvoor je resultaten in het verslag hebt opgenomen
- en een PDF van je verslag (inclusief het programma!)

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: vrijdag 28 mei 2021, 23.59 uur.

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