Resultaten uit het verleden...
Derde programmeeropdracht Algoritmiek, voorjaar 2013

"Achteraf is het gemakkelijk," hoor je wel eens, als het gaat over beleggen in aandelen, "dan weet je precies wanneer je welke aandelen had moeten kopen of verkopen." Bij deze opdracht zullen we ontdekken of dit achteraf inderdaad zo gemakkelijk is, zeker als je de beslissingen over kopen en verkopen door een computerprogramma wilt laten nemen.

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. 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 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 ($1 \leq i \leq n$) 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 rente over het bedrag in kas op dag $t \geq 0$, noemen we rente(t). Als de rente bijvoorbeeld 2% is, is rente$(t) = 1.02$. [1] De belegger krijgt in dat geval op dag $(t + 1)$ 2% rente over het bedrag dat hij aan het eind van dag t in kas had.

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.

[1] Inderdaad, rente(t) is niet het rentepercentage zelf, maar de bijbehorende rente-factor.

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.

C++-programma

Je moet bij deze programmeeropdracht een C++-programma schrijven dat het volgende doet:

- Het programma vraagt de gebruiker om de naam van een tekstbestand op te geven. Vervolgens leest het programma uit dit tekstbestand alle parameters, koersen en rentepercentages in.
- Dan berekent het met **bottom-up** dynamisch programmeren de waarde van bedrag$(t_w, 0)$. Maak hierbij gebruik van de analyse hierboven.
- Bedenk hoe je uit een getal a de bits afleidt die aangeven welke aandelen i de belegger bezit.
- Ten slotte wordt de waarde van bedrag$(t_w, 0)$ op het scherm gezet, afgerond op twee decimalen. Ook de transacties (koop en verkoop van aandelen) die tot dit resultaat hebben geleid, worden op het scherm gezet.
- Hoewel de uitvoer uiteindelijk in twee decimalen is, hoef je de bedragen tussendoor niet af te ronden.
- Eisen aan het programma: boven elke functie moet een commmentaarblokje komen met daarin een (zeer) korte beschrijving van wat de functie doet. Noem daarin ook 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 ook op de layout (consequent inspringen) en op het overige commentaar bij de programmacode (alleen zinvol en kort commentaar).
- Het programma moet (ook) onder Linux werken.
- Voor een precieze specificatie van invoer en uitvoer, zie verderop.

In het verslag

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 bitstring 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 programma bereken je met dynamisch programmeren de waarde van bedrag$(t_w, 0)$. Beredeneer wat de tijdscomplexiteit (in orde van grootte, dus grote O) van je programma 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 (ook 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, en afdrukt wat de transacties zijn die hebben geleid tot het antwoord bedrag$(t_w, 0)$.

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 float erop: B_0, met $B_0 \geq 0.00$.
- $t_w + 1$ regels, corresponderend met respectievelijk dag 0, dag 1, ..., dag t_w. Elke regel bevat n floats, 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 float erop: achtereenvolgens: rente(0), rente(1), ..., rente$(t_w - 1)$. Voor elke t geldt $1.00 \leq$ rente$(t) \leq 2.00$.

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 float B_0, met:

```
fin >> n;
fin >> B0;
```

Uitvoer

De uitvoer moet naar het scherm geschreven worden. Allereerst een regel met alleen het eindantwoord, dus de waarde van bedrag$(t_w, 0)$, afgerond op twee decimalen. Vervolgens $t_w + 1$ regels (een regel voor elke dag, in de juiste volgorde) van de vorm:

```
Dag 0: koop 1, koop 4
Dag 1: verkoop 1, koop 3, verkoop 4
...
```

Het verslag

Het verslag moet getypt zijn in LaTeX, en moet een introductie bevatten, de probleemstelling, een analyse van het probleem met een beschrijving van de oplossingsstrategie met dynamisch programmeren, en een hoofdstukje met resultaten.

Bij de analyse met oplossingstrategie beantwoord je ook alle vragen die hierboven in de paragraaf 'In het verslag' worden gesteld.

In het hoofdstukje met resultaten geef je voor drie zelfgemaakte, representatieve invoerbestanden de uitvoer van je programma: bedrag$(t_w, 0)$ (afgerond op twee decimalen), en de daarbij behorende transacties per dag.

Voor u beschikbaar...

Om je te helpen bij deze opdracht worden op de website van het vak,

http://www.liacs.nl/∼graaf/ALGO/

twee bestanden en een eenvoudig programma gezet. De bestanden bevatten respectievelijk een voorbeeldinstantie en de waarde van bedrag$(t_w, 0)$ bij die instantie.

Het programma kun je gebruiken om zelf een random instantie te genereren. Behalve de gewenste waardes van t_w, n en B_0 moet je hiertoe grenzen opgeven waarbinnen de koersen kunnen liggen, en grenzen waarbinnen de rente-factoren kunnen liggen.

Eventuele verdere aanvullingen of tips bij de programmeeropdracht komen ook op de hierboven vermelde website te staan.

In te leveren

Het programma en de drie invoerbestanden uit het hoofdstukje resultaten per e-mail sturen naar: graaf@liacs.nl. Het verslag (inclusief listing) moet op papier worden ingeleverd en in de daartoe bestemde doos met opschrift Algoritmiek in de postkamer van Informatica (kamer 156) worden gedeponeerd. Vermeld overal duidelijk de namen van de makers.
Uiterste inleverdatum: vrijdag 17 mei 2013. Voor elke week te laat inleveren gaat er een punt van het cijfer af.

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

Op de hierboven vermelde website komen te zijner tijd de behaalde cijfers te staan.