Obligatorisk oppgave 2 i INF 4130, høsten 2009

Leveringsfrist 23. oktober

Generelt for alle oppgavene

Samme reglement gjelder som for obligatorisk oppgave 1. Det kan komme presiseringer og forandringer i oppgaveteksten underveis, så følg med på beskjeder.

Husk at det ikke bare er programkode som skal leveres! Det er like viktig å svare skikkelig på drøftingsspørsmål som å levere god kode.

Vi legger opp til at programmene deres vil testes automatisk mot en rekke datasett når de mottas. Det er derfor viktig at output-formatet er identisk (tegn for tegn, også blanke) med formatet som er gitt i eksempelet på hver oppgave. Vi anbefaler derfor sterkt at en sjekker svaret programmet lager nøye med testdataene som følger med hver oppgave. Spør gjerne gruppelærer dersom en er usikker på dette.

Det er lagt ut en side med ytterligere leveringsanvisning for obliger. Den ligger under "Oppgaver" på kurssiden. Det blir typisk utvidet etter hvert, så følg med!

Det skal her i oblig 2 leveres to programmer (ett i oppgave 1 og ett i oppgave 2) og disse skal ha navn nøyaktig "Oppgave1" og "Oppgave2" (på den måten det blir mest naturlig innenfor språket en bruker, for eksempel skal klassen med main( )-metoden hete "OppgaveX" i Java og skriptfila skal hete "OppgaveX.py" i Python).

Alle programmer skal ta input-fil som sin første parameter og output-fil som sin andre parameter. Under vil vi bare referere til input og output, da er det disse vi mener. Regler for programmeringsspråk er som på forrige oblig. De som fikk godkjent et annet språk for oblig 1 kan bruke dette på denne obligen også.

Se oppgave 1 i oblig 1 angående tips til I/O-håndtering i Java.

Oppgave 1 (Venstrevridde og korte heaper)

Venstrevridde heaper (*leftist heaps*) holder orden på nullsti-lengder (*null path lengths*) og snur om på barnenoder hvis nullsti-lengden i høyre barn av en node er større enn nullsti-lengden i venstre barn. Dette for å sikre at den høyre stien i trærne våre forblir nogenlunde kort, slik at merging av heaper (og andre operasjoner implementert via merge) går raskt. Vi skal i oppgavene 1a, 1b og 1c tenke ut fra venstrevridde heaper med tradisjonell implementasjon, men i oppgave 1d skal vi lage en vri på implementasjonen.

a) Tegn treet vi får ved å sette inn følgende elementer i en vanlig, tom venstrevridd heap: 5, 3, 4, 1, 2 (altså en hvor vi snur om på barna på vanlig måte). Tegn også opp alle mellomstadiene. Som vanlig betyr lav key høy prioritet. Vi tar

altså utgangspunkt i implementasjonen til Mark Allen Weiss som ligger på kurssiden:
`www.uio.no/studier/emner/matnat/ifi/INF4130/h09/obliger/Kildekode.xml`

b) Finnes det en rekkefølge (m.h.t. prioritet) slik at om vi setter elementer inn i denne rekkefølge i en vanlig, tom venstrevridd heap (uten å ta noen ut), så vil vi med jevne mellomrom (når antall elementer er $2^n - 1$) få et fullt balansert tre? Forklar.

c) Finnes det en (annen) rekkefølge som, igjen for en vanlig en venstrevridd heap, gir oss en lang (venstre)sti uten forgreininger? Forklar hvorfor dette er en heldig situasjon for venstrevridde heaper.

d) Når vi skal implementere venstrevridde heaper er det imidlertid ikke nødvendig å bytte om på barna, så lenge vi husker hvilket av dem som har kortest nullsti-lengde. (Man kan tenke seg situasjoner hvor man ikke ønsker å vri på trærne, men vil beholde strukturen slik den var.) Implementér en variant av venstrevridde heaper (vi nøyer oss med merge( ), insert( ) og deleteMin( )) som ikke baserer seg på å rotere subtrær, men som isteden, på passelig måte, husker hvilket barn som har kortest nullsti-lengde. For entydighetens skyld tenker vi oss at et ikke-eksisterende barn har nullsti-lengde lik -1 (minus en), og at vi velger det venstre barnet dersom venstre og høyre barn har lik nullsti-lengde. Implementasjonen til Mark Allen Weiss kan passelig brukes som utgangspunkt.

Input

Programmet i 1d skal lese input fra en angitt fil, denne angis til programmet ved oppstart, sammen med navnet for output-fil. Filen inneholder to linjer med tall som skal settes inn i to heaper (alt heltall, med blanke mellom), som så skal merges.

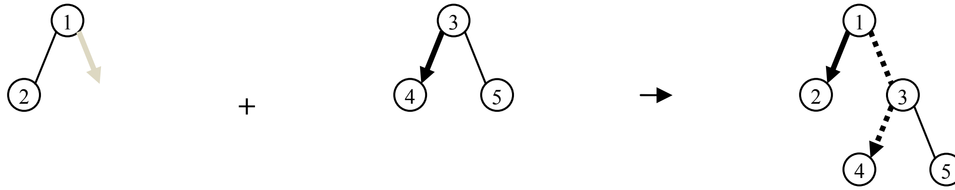
Output

Treet en slik merging resulterer i skal skrives ut på infix form, deretter skal det kommet en bindestrek, så skal stien som fremkommer ved å starte i roten og hele tiden å gå til barnet med lavest nullsti-lengde skrives ut.

Eksempel-input 1	**Eksempel-output 1**
`2 1` `3`	`2 1 3-1 2`

Eksempel-input 2	**Eksempel-output 2**
`2 1` `3 4 5`	`2 1 4 3 5-1 2`

Figur 1 viser hvordan heapene i eksempel 2 over merges. Et lite sett med testdata ligger på kurssidene (filene “Obl2-Opg1-Test[1-3].txt”).



Figur 1: To heaper h_1 og h_2 merges til h_3. De korte stiene som fremkommer ved å hele tiden gå til barnet med lavest nullsti-lengde er markert med heltrukne fete piler. Stien vi merget langs (de to korte stiene fra h_1 og h_2 spleiset sammen) er i h_3 stiplet.

Oppgave 2 (A-søk)*

Oppgaven er å skrive et program for løsing av "15-spillet", som også kommer som 8-spillet og generelt ($N \times N-1$)-spillet (på et $N \times N$-brett). Dette er diskutert i læreboka som "8-puzzle game" på side 717. Programmet skal lages generelt for $N \times N$-brett (men du kan i implementasjonen forutsette at $N \leq 10$). En tilstand for spillet med $N = 3$ kan vises slik:

```
1 2 3
0 4 5
7 8 6
```

Her angir 0 det tomme feltet. Programmet skal da finne frem til en måte å flytte brikkene (et antall "trekk") slik at man kommer til måltilstanden:

```
1 2 3
4 5 6
7 8 0
```

(og tilsvarende for $N \times N$-brett). Et lovlig trekk er å la den tomme posisjonen (0-en) "bytte" posisjon med en av de (maksimalt 4) nabobrikkene. Et slikt trekk angis med hvordan *den tomme posisjonen* flytter seg, med V, H, O, N (for Venstre, Høyre, Opp og Ned). En løsning på problemet over er derved: HHN, og denne skal skrives ut på skjermen om programmet finner en løsning. Merk at løsningen man finner skal være optimal, i den forstand at det ikke finnes noen løsning som har færre trekk. Programmet skal bruke A*-søking, og kan passelig bruke en rett fram Manhattan-heuristikk (se oppgave 23.7).

Programmet bør kunne løse alle 8-spill-problemer på rimelig tid, og noe mer vil ikke bli forlangt. Det er jo imidlertid morsomt å se om det også kan løse enklere 15-spill-problemer (slike som kan løses i få trekk), og det er morsomt om dere legger ved i leveringa en kommentar om hva programmet klarer, og på hvilken tid.

Input

Programmet i skal lese input fra en angitt fil, denne angis til programmet ved oppstart, sammen med navnet for output-fil. Første linje av input-fila skal inneholde verdien av N, og startilstanden er så angitt på N linjer.

Output

Første linje i output-fila skal være antall trekk løsningen er på, denne verdien er entydig bestemt for hver oppgave. Andre linje skal være en sekvens av O, N, V og H, som angir løsningen slik som angitt over (på noen oppgaver er det flere mulige svar her). Tredje til femte linje skal angi hvor mange tilstander en måtte innom etc. (disse verdiene er ikke entydig bestemt og vil variere bl.a. avhengig av heuristikk). Nærmere bestemt skal de angi henholdsvis:

- Antall tilstander en besøkte, inklusive starttilstanden og sluttilstanden (altså omtrent antall kall til extractMin i prioritetskøen, litt avhengig av implementasjonen).

- Antall ganger det skjer at en oppdager en ny tilstand (= antall insert i prioritetskøen av en tilstand som ikke ligger i køen fra før).

- Antall ganger en modifiserer kosten til en allerede oppdaget (men ikke besøkt) tilstand (= antall decreaseKey/re-insert i prioritetskøen).

Eksempel-input	**Eksempel-output**

`3`
`1 2 3`
`0 4 5`
`7 8 6`
	`3`
	`HHN`
	`4`
	`8`
	`0`

Implementasjonstips

Hver tilstand må representeres på en eller annen måte. Det enkleste er å bruke objekter av en egen klasse State med en byte[]-array, og gjør gjerne det (det går fint siden $N \leq 10$). Det finnes også mer plassbesparende representasjoner, f.eks. som ett eller flere "lange" tall, eller ved å la hver byte representere to posisjoner på brettet. Men disse vil jo bruke mer tid.

A*-søk krever en del spesialiserte datastrukturer, og en trenger i alle fall en prioritetskø til å lagre køen, og en effektiv oppslagsstruktur (hashmap, søketre, el.l.) til å lagre allerede besøkte noder. Du kan her bruke en passende innebygd struktur eller et valgfritt open source-bibliotek.

Se `http://www.uio.no/studier/emner/matnat/ifi/INF4130/h09/faq.html` for flere tips til hvordan oppgaven kan implementeres.

Oppgave 3 (Algoritmer og avgjørbarhet)

NB: Stoffet til denne oppgaven vil bli gjennomgått 8/10. Merk at det kan komme presiseringer og ekstrainformasjon til oppgaven (som eventuelt blir annonsert med en beskjed). Følg med!

Du skal her avgjøre om følgende problemer er avgjørbare eller ikke. Et “skritt” i programutføringen er en passelig basal enhet som klart kan utføres på endelig tid. Det kan f.eks. være det å “hente” en operand, det å utføre en aritmetisk eller logisk operasjon, eller å utføre et assignment, og skrittet må være slik definert at input eller output av ett tegn regnes som ett skritt. Et skritt her skal grovt sett tilsvare et steg for en Turing-maskin. For å løse oppgavene under må du altså enten vise at om det aktuelle problemet kunne løses så kan også stoppeproblemet løses (som jo er en motsigelse, slik at problemet er uavgjørbart), eller du må komme opp med en algoritme som løser det.

a) Instans: Gitt et program
   Spørsmål: Vil dette programmet, for ethvert input, alltid skriver ut tegnet A?

b) Instans: Gitt et program
   Spørsmål: Vil dette programmet skrive ut en A for input “abc”?

c) Instans: Gitt et program
   Spørsmål: Vil dette programmet, for ethvert input, alltid skrive ut en A etter mindre enn 50 skritt?

Leveringsanvisning

Det skal kun leveres kildekode og ett enkelt PDF-dokument som besvarelse. Kommentarer til besvarelsen som du ønsker å gi kan du legge på starten av PDF-dokumentet. Dersom du er klar over en svakhet i algoritmen som slår ut på et spesifikt tilfelle kan du også (hvis du vil, dvs. ønsker tilbakemelding om hvor feilen ligger) legge ved et datasett som demonstrerer dette og henvise til det i besvarelsen. Merk: Koden du leverer skal kunne kjøre på de vanlige Ifi-maskinene.

- Legg kildekoden (så mange filer du vil, men ikke spre koden over flere mapper) og et dokument kalt “oblig1.pdf” i en mappe kalt: `oblig2_<ditt brukernavn>.`

- Pakk mappa i en tar.gz-fil eller zip-fil med samme navn. For eksempel `tar cvzf oblig1_kjetimh.tar.gz oblig2_kjetimh`

- Send fila til: `inf4130-1@ifi.uio.no`.

Dersom du ønsker kan du eventuelt legge ved et plain text-dokument i stedet for PDF, kalt “oblig1.txt” (men det kan jo konverteres med f.eks. kommandoen a2ps eller a2pdf).

[slutt]