Laboration 4
Nålen i höstacken

Uppgifterna syftar främst till att öva och examinera lärandemålen 1, 2, 5 och 6

Man brukar säga “det är som att leta efter en nål i en höstack” när man står inför en till synes hopplös uppgift att leta efter någonting. En outtröttlig dator kan dock finna en nål i en höstack - förr eller senare. Med tillräckligt sluga algoritmer kan man dessutom se till att det går fort.

I denna laboration kommer ordet ”höstack” syfta till en datamängd (t.ex array, träd, lista) och ”nål” syfter till ett element man är intresserad av att hitta.

Ni ska utvärdera hur söktiden för olika metoder skiljer sig åt då man låter höstackens storlek variera. För att ni ska få tider som är tillräckligt långa för att vara mätbara måste ni ha rejält många strån i stacken (ett antal miljoner för den största storleken) och låta storleken variera inom ett spann. Ni behöver nog ha minst 4 olika storlekar för att kunna dra några slutsatser.

- För att slumpen inte ska påverka mätningen måste ni upprepa mätningen med nålen inslängd på olika ställen i höstacken.
- Ett alternativ till att söka i enormt stora datamängder för att få tillräcklig noggrannhet på mätningarna, kan vara att upprepa varje mätning tillräckligt många gånger. Hur kan ni åstadkomma det? Kan detta i sig påverka mätningarna i betydande grad?
- Hitta lite information om hur komplexiteten för de olika sökmetoderna varierar. Kan ni förvänta er att detta leder till några problem med era mätningar?

En viss variation i mätningarna kan uppstå om man byter dator mellan körningarna, åtminstone om det är stor skillnad mellan datorernas prestanda. Ni bör därför notera resultaten för de olika metoderna från körningar på samma dator (eller åtminstone datorer med hyfsat lika prestanda, det borde inte göra stor skillnad om ni använder en annan dator i samma labbsal).

För att höstacken ska fungera för de olika metoderna måste ni representera den med unika värden. Varför? Hur kan ni lösa det? De måste också ha någon form av slumpfördelning.
Tips: Skapa ett fält för varje höstack (med olika många element i) där elementen är slumpade och unika. Använd sedan dessa fält för att skapa de andra datatyperna som sökningen ska köras på.

Tiden ni mäter ska bara inkludera själva sökningen. Hur stor noggrannhet behöver man mäta med? Ta reda på vilken kod som behövs för att ni ska kunna få ut exekveringstiden.

Sökningen ska ske på följande sätt:

a) Sekventiell sökning.

Detta är den enklaste sökningen. Lagra elementen i en array, och loopa igenom den tills att det sökta elementet är hittat eller tills hela arrayen är genomsökt.

b) “Optimerad” sökning.

Detta är fortfarande sekventiell sökning i en array. Denna gång sätter ni in en ”extra” nål sist i arrayen (dvs, det datat ni söker). Detta gör att ni kan (och ska) ändra villkoren för loopens terminerande. Ni behöver alltså inte längre kontrollera om arrayen är slut eller om ni sökt igenom

alla element eftersom ni garanterat kommer att hitta det (om ingen annanstans så finns det sist). Lägg in resultatet från denna och a)-uppgiften i samma diagram.

c) Byt algoritm

I många tillämpningar sker sökning betydligt oftare än insättning. T.ex. kan det vara så att man skapar höstacken initialt (dvs sätter in elementen i någon datasamling) och i just det initiala skedet spelar exekveringstiden inte någon roll. Det kan t.ex. ske vid systemstart. Därefter är sökning den helt dominerande operationen, insättning och borttagning av element sker förhållandevis sällan. I denna typ av tillämpningar "lönar" det sig alltså att "investera" i en "dyr" datalagring som görs initialt, om man därigenom uppnår att de efterföljande sökningarna blir "billiga".

I sådana fall kan ett binärt sökträd vara ett bra alternativ, speciellt om man inte vet något om storleken på datamängden. I denna deluppgift ska ni använda ert sorterade binära träd (som också är ett sökträd).

Repetera körningen med samma storlekar på höstacken som i de två tidigare deluppgifterna, och lägg in i ett diagram. Jämför med de tidigare körningarna.

d) En till algoritm

Med samma motiveringar som i föregående deluppgift kan man tänka sig att ta kostnaden för att bygga upp en hashtabell i utbyte mot en efterföljande effektiv uppslagning. Hashtabeller kommer bäst till sin rätt då man i förväg har ett hum om hur mycket data som ska lagras. Låt oss anta en sådan situation. Hur stor bör tabellen vara relativt mängden data som man tror ska läggas in? Ni kan med fördel använda "chained hash table", och då kan det kanske vara läge att tänka på återanvändning.

Leta upp en hashfunktion som verkar bra. Det finns en del skrivet på nätet. Ange referens som kommentar, och skriv även en motivering till ditt val. Många hashfunktioner baserar sig på strängar. Om dina nålar t.ex. är av någon heltalstyp så blir ju inte det riktigt samma sak. Går det att använda strängbaserade funktioner ändå på något sätt om nycklarna utgörs av heltal?

Ni ska förstås göra er hashtabell som en ADT.

Lägg återigen in era körningar i samma diagram. Ni kommer Kanske att behöva lägga in resultaten från binärt träd och hashtabell i ett eget diagram för att det ska vara möjligt att jämföra dem inbördes…

Diskutera era rön med labbassen. Vilka åtgärder var mest värda att investera i om man tar hänsyn till långsiktiga effekter?

Redovisning: Visa och diskutera era resultat för assistenten, diskutera också de val ni gjort i implementationen.
Redovisa förslagsvis laborationstillfälle 6.