Lösningsförslag till tentamen i ID1020, P1 2014

Tisdagen 2014-10-28 kl 9.00–13.00

Examinator: Johan Karlander, Jim Dowling

Tentamen består av två delar. Del I innehåller tio uppgifter för betyget E. Del II, som inte är obligatorisk, innehåller fyra uppgifter för betygen A-D. Maximal skrivtid är fyra timmar.

Varje lösning kommer att bedömas med 0-10 poäng.

Kravet för godkänt (E) är 50 poäng på del I. Du måste bli godkänd på del I för att kunna erhålla högre betyg. Poängen från del I förs inte vidare till del II.

Under förutsättning att du är godkänd på del I gäller följande betygsgränser för del II:

- Betyg D: 5-10 poäng
- Betyg C: 11-20 poäng
- Betyg B: 21-30 poäng
- Betyg A: 31-40 poäng

Uppgifter Del 1

Följande uppgifter avser betyget E. Krav på *fullständiga lösningar med motiveringar* gäller för samtliga uppgifter.

1. Skriv en rekursiv implementation av power-funktionen nedan antingen i Java eller i pseudokod. Sedan, i Java, jämför minneskomplexiteten för power-funktionen nedan och din rekursiva lösningen.

```
double power(double x, unsigned n) {
  double r = 1;
  while (n != 0) {
     r = x * r;
     --n;
  }
  return r;
}
```

Lösningen

```
double power(double x, int n) {
  if (n == 0) {
    return 1;
  return x * power(x, n-1);
}
```

Rekursiva lösningen i Java har O(n) minneskomplexitet eftersom Java saknar svansrekursion och loop-baserad lösningen har O(1) minneskomplexitet.

2. Ge tidskomplexiteten för följande javakodstycken:

```
for (int i = 0; i < n; i++)
    sum++;
```

(a)

```
for(int i = 1; i < n; i = i * 2)
    sum++;
```

(b)

```
for (int i = 0; i < n; i++)
    for ( int j = 0; j < n * n; j++)
        for (int k = 0; k < j; k++)
            sum++;
```

(c)

Vad är tidskomplexiteten (worst-case time complexity) för att hitta ett värde i en sorterade array med hjälp av följande javakodstycke:

```
public static int binarySearch(int[] a, int key)   {
     int lo = 0, hi = a.length-1;
     while (lo <= hi) {
         int mid = lo + (hi - lo) / 2;
         if (key < a[mid]) {
             hi = mid - 1;
         }
         else if (key > a[mid]) {
             lo = mid + 1;
         }
         else {
             return mid;
         }
     }
    return -1;
 }
```

Lösningen

(a) - O(n)

(b) - O(nlog(n))

(c) - $O(n^5)$

binarySearch - O(log(n))

3. Vi har en sammanhängande, oriktad graf utan loopar och parallella kanter. Den har viktade kanter. Vi får veta att den har a kanter med vikt 1 och b kanter med vikt 2. Utgående från denna information, avgör vad den största respektive minsta vikt som ett minimalt spännande träd kan ha.

Lösningen

Grafen innehåller alltså $a+b=e$ kanter. Vilket är det minsta antal kanter som ett MST kan ha? Det minsta antal noder n som grafen kan ha ges av ekvationen $e = \binom{n}{2}$. Kalla detta värde för n_0. Då innehåller ett spännande träd $n_0 - 1$ kanter. Om $n_0 - 1 \leq a$ så blir den minsta vikten $n_0 - 1$, annars blir den $a + 2(n_0 - 1)$. Det störtsta antal kanter som ett MST kan ha är $a + b$. Den största vikten är då $a + 2b$.

4. Avgör, med motivering om följande är sant eller inte: Om vi har en riktad graf G med exakt två sammanhängande komponenter kan man alltid lägga till två nya, riktade, kanter så att hela grafen blir starkt sammanhängande.

Lösningen

Påståendet är sant. Antag att komponeterna är K_1 och K_2. Antag att noden $a \in K_1$ och noden $b \in K_2$. Då kan vi lägga till två riktade kanter (a, b) och (b, a). Eftersom varje nod i K_1 kan nås både med framåt- och bakåtriktade stigar från a och motsvarande gäller för b och K_2 så kommer det att gå riktade stigar mellan alla par av noder. Grafen är alltså starkt sammanhängande.

5. **Socialt nätverks-konnektivitet**. Givet ett socialt nätverk med N medlemmar och en loggfil som innehåller M tidstämplar, där varje tidstämpel är tiden då ett par medlemmar blev *vänner*, designa en algoritm som fastställer den tidigaste tidpunkt då alla medlemmar är förbundna (dvs., varje medlem är en vän av en vän .. av en vän). Du kan anta att loggfilen är sorterad efter tidstämplarna och att vänskap är en ekvivalensrelation (equivalence relation). Körtiden för algoritmen ska vara $Mlog(N)$ i tidskomplexitet (worst-case time complexity) eller bättre.

   Lösningen

   Socialt nätverksgrafen kan modelleras som en NxN matris och sedan blir problemet dynamisk konnektivitet. Union-find algoritmen kan användas för att lösa problemet, men man ska lägga till en virtuell-top och en virtuell-botten till matrisen. Vänskap är en union operation. Find implementerar operations som kollar om all medlemmar är förbundna. Tidskomplexiteten kommer att vara högst $Mlog(N)$.

6. Beskriv det exakta innehållet i arrayen av hashtabellen vid följande inmatning av <nyckel, hashvärde> par. Inmatningen sker i den givna ordningen. Beskriv både för listkedjande (separate chaining) och linjär sondering (linear probing) hashtabeller. Arrayen har storlek 8. A:1 B:4, C:2, D:2, E:0, F:4, G:3, H:7

   Lösningen

   Se: sista sidan.

7. Mergesort har för mycket *overhead* för små subarrayer. Beskriv en förbättring av mergesort som minskar förväntade körtiden för små subarrayer. Man kan också optimera mergesort om array:n redan är sorterad. Hur gör man det? Om du använder mer optimeringar, hur låg kan körtiden för mergesort bli om man kör mergesort på en array där alla element har samma värde?

   Lösningen

   Använd insertionsort för små subarrayer med en cutoff (t.ex. av 10 poster). Om största posten i första hälften är lika med eller mindre än den minsta posten i den andra hälften, då kan man hoppa över sort steget. Om alla element har samma värden, kan körtiden optimeras till O(n).

8. a) Vad är djupet för ett perfekt balanserat binärt träd med N nycklar?

   b) Vilken datastruktur skulle du använda för att lägga upp en symboltabell från en lista av N nycklar? Antag att efter skapandet så kommer symboltabellen inte att förandras, d.v.s. den skall bara läsas.

   c) Vad är tidkomplexiteten för skapandet av symboltabellen i ditt val i b) ovan?

   d) Vad för slags datastruktur skulle du använda för att mellanlagra en kontinuerlig ström av indata? Elementen i strömmen är av formen <Belopp, Info> där Belopp är ett godtyckligt heltal, och Info en text. Syftet med lagring är att kunna samtidigt som inmatning sker också kunna kontinuerlig plocka ut de största elementen för vidare bearbetning.

   e) Vilken datastruktur skulle du använda för implementera en symboltabell på en Internetserver under förutsättning att nycklarna bestäms av klienterna (t.ex. användarnamn)?

   Lösningen

a) lg N b) t.ex. röd-svart binärt sökträd (eller bara ett binärt sökträd). c) N lg N d) Max-orienterad prioritetskö e) T.ex röd-svart binärt sökträd men ej en hashtabell

9. Låt G vara en sammanhängande, oriktad graf utan loopar och parallella kanter. Vi antar att den har n noder.En nod s är markerad som startnod. Vi vill numrera noderna $1, 2, 3, ..., n$ så att varje nod får ett unikt nummer och så att det för varje nod u finns en stig med noder av växande index som går från s till u. Beskriv en algoritm som konstruerar en sådan numrering.

   Lösningen

   Man inser att en sådan ordning och motsvarande stigar ges av det träd som genereras av en DFS-sökning som startar i s. Vi kan då ta koden för en implementation av DFS och lägga till en counter som initieras till 1 och som sedan används för att indexera varje nybesökt nod. Vid varje besök ökas counterns värde med 1.

10. Ange tidskomplexiteten i följande fall (strunta i den konstanta faktorn). Ge både genomsnittsfallet (expected case) och värsta alternativet (worst case). a) Skapa ett röd-svart balanserat sökträd från en lista av N nycklar b) Besvara en rankningförfrågan (rank) i ett binärt sökträd av storlek N c) En läsning (get) i en listkejdande hashtabell. Tabellen innehåller N nycklar och arrayen är av storleken ca. 4N. d) En inläggning i en oordnad arraysymbol-tabell av storlek N. e) En inläggning i en ordnad arraysymbol-tabell av storlek N.

    Lösningen

    a) Förväntat och värsta: N lg N b) Förväntat: lg N Värsta: N (d.v.s. maximalt obalanserad träd) c) Förväntat: 1 Värsta: N (då alla nycklar has samma hashvärde) d) Förväntat och värsta: 1 e) Förväntat och värsta: N

Uppgifter Del 2

Följande uppgifter avser betygen A-D. Krav på *fullständiga lösningar med motiveringar* gäller för samtliga uppgifter.

1. Låt oss anta att vi har en oriktad graf G. Vi kan anta att den är sammanhängande. Vi säger att en kant är en *bro* (engelska: bridge) om grafen blir osammanhängande om kanten tas bort. Det betyder att en kant är en bro om och endast om den inte ingår i någon cykel. Du ska konstruera en algoritm som givet G hittar alla broar i G. Din algoritm ska ha så liten tidskomplexitet som möjligt. För full poäng krävs att den går i tid $O(|E|)$. Beskriv algoritmen med lämplig kod. Motivera varför den är korrekt och ge en komplexitetsanalys.

   Lösningen

   Ett sätt att göra det på är att använda en modifierad variant av DFS. Till att börja med gör vi som i uppgift 1.9 och numrerar noderna i den ordning de besöks. vi kan kalla numreringen för $nummer(u)$ för alla u. Vi kan sedan skapa en mängd *cykelkant* som ska innehålla kanter och som från början är tom. Vi använder också en variabel *start*. Vi gör nu följande tillägg till DFS: Om vi befinner oss i u så undersöker vi alla grannar till u. Om v är en granne *som redan är besökt* så sätter vi $start(u) = min(start, v)$, När man sedan bakåt-trackar görs följande: Om vi återvänder till en nod w och $start \leq w$ så

lägger vi till den senast använda kanten i *cykelkant*. När vi är klara kommer Mängden av broar att vara $E - cykelkant$. Som lätt ses är tidskomplexiteten $O(|E|)$.

2. Skriv funktionen *LikaSummanSubsträng()* som tar en sträng s som argument, där s bara består av heltal som är större än noll. Funktionen ska returerna längden av den längsta sammanhängande (contiguous) substräng i s, som är sådan att längden av substrängen är $2 * N$ (max längden av strängen är 63) och summan av de N heltal som är längst till vänster är lika med summan av de N heltal som är längst till höger. Om det inte finns någon sådan sträng ska funktionen returnera 0.

   Du får godkänt för en lösning med $O(n^3)$ som tidskomplexitet, men för att få toppbetyg ska du ge en lösning med tidskomplexitet bättre än $O(n^3)$.

   Testfall 1 Input: **345453** Output: 6

   Förklaringen: 3 + 4 + 5 = 4 + 5 + 3

   Längden av längsta substrängen = 6 där summan av första hälften = summa av andra hälften

   Testfall 2 Input: 010**1357245**00 Output: 8

   Förklaringen: 1 + 3 + 5 + 7 = 2 + 4 + 5 + 5

   Längden av längsta substrängen = 8 där summan av första hälften = summa av andra hälften

   Lösningen

   En enkel lösning som har $O(n^3)$ är

```
for i = 2 to n
    for j = i-1 to 0
       summera elementen j to i och summera elementen i to 2*i-j.
       Test om summerna ar lika.
```

   Det här kodstycket i python iterarer genom alla möjligheter i nedstigande ordningen av längden och den cachar delsummerna för att undvika återberäkningen av delsubsträngar. Den löser problmet i O(N*(N-2)).

```
def get_equal_sum_substring(s):
    s = map(int, s)
    S = [0]
    sum = 0
    for i in s:
        sum += i
        S.append(sum)

    def sub_sum(low, high):
        return S[high] - S[low]

    # Set slen to be provisional length of half of
    # the substring. We start with the highest possible
```

```
    # value of slen so that we can return as soon as
    # find and equal-sum substring.
    slen = len(s) // 2
    while slen > 0:
        for i in range(len(s)-2*slen):
            lsum = sub_sum(i, i+slen)
            rsum = sub_sum(i+slen, i+slen*2)
            if lsum == rsum:
                return 2 * slen
        slen -= 1
    return 0
```

3. Förklara tidskomplexitetsbegreppen Big Theta, Big Omega, och Big Oh (stora ordo).

   Vilken sorteringsalgoritmen har tidskomplexitet $O(N * logN)$?

   Stabilitet är en viktig egenskap hos sorteringsalgoritmer. Ge ett exempel på en stabil sortering och en sortering som inte är stabil. Ange en sorteringsalgoritm som ger en stabil sortering och en annan som inte ger en stabil sortering.

   Designa en algoritm som hittar elementet i mitten (mittpunkten) av en länkad lista med bara en genomsökning av listan (dvs. du får inte läsa från början till slutet av listan och återvända till mittpunkten). Minneskomplexiteten ska vara $O(1)$.

   Lösningen

   Se föreläsningar för tidskomplexitetsbegreppen. Sorteringsalgoritmen är mergesort. Insertion sort och mergesort är stabila. Selection sort och shell sort är inte stabila. Mittpunktalgoritmen: För att hitta elementen i mitten behöver man två pekare. Den första pekare inkrementeras med en nod i listan åt gången, medan den andra pekare inkrementeras med två noder i listan åt gången. När den andra pekare når slutet av listen, kommer den första pekaren att peka till mittpunkten i listan.

4. Du har tillgång till två stora textfiler, en med Shakespeares samlade verk och en med Miltons samlade verk. Antag att både innehåller N ord. Du skall skriva ett program för att ta reda på följande:

   (a) Antalet ord som Shakespeare använder minst en gång men som Milton inte använder alls.

   (b) De M mest frekventa ord som Shakespeare använder men inte Milton använder.

   (c) Antalet ord som Milton använder minst en gång men som Shakespeare inte använder alls.

   (d) De M mest frekventa ord som Milton använder men som Shakespeare inte använder.

   Du skall också ange tidskomplexitet i termer av åtminstone följande faktorer (du får förfina din analys med andra faktorer)

   (a) N: ordlängden i textfilerna

   (b) U: antalet unika ord i Shakespearefilen

För full poäng krävs lägsta möjliga tidskomplexitet. Vi antar att texten inte innehåller egennamn och andra icke-ord (d.v.s. texten har förfilteras). Använd något API som förekommer i kursboken. Var noga med att ange de datastrukturer och algoritmer som din lösning bygger på.

Lösningen

Vi get svaret i pseudokod.

```
BST shakeTree = new BST<String,Integer>();
%% (t.ex. r\"{o}d-svart bin\"{a}rt s\"{o}ktr\"{a}d)
BST miltTree = new BST<String,Integer>();
for all words w in Shakespeare {
  Integer v=shakeTree.get(w);
  if v==NULL then
     shakeTree.put(w,1)
  else
     shakeTree.put(w,v+1);}   %% antar att put skriver \"{o}ver
}
```

Efter detta har vi ett träd med orden som finns i Shakespeare (med ordet som nyckel och antalet förekomster som värde). Beräkningskomplexitet av denna fas är NlgU. Slingan körs N gånger och i varje steg görs en get och en put. Både operationer är lg U (trädet innehåller som mest U ord).

```
for all words w in Milton {
  Integer v=shakeTree.get(w);

  % hittat ett ord som Milton men inte Shakespeare anv\"{a}nder
  if v==NULL then
    Integer v2=miltTree.get(w)
    if v2==NULL then
       milTree.put(w,1)
    else
       miltTree.put(w,v2+1)
  else
    shakeTree.delete (w);
```

Vi arbetar här med 2 träd. Shakespeare trädet krymper där vi tar bort alla ord som Milton också använder. Milton trädet växer när vi hittar ord i Miltons text som Shakespare inte använder. Beräkningkomplexitet är också Nlg U.

Svar A = shakeTree.size(); Svar C = miltTree.size();

För att svara på b) och d) kan man plocka ut elementen och sortera dem med avseende på antalet förekomster. Detta är SlgS där S är antalet ord som den ena författaren men inte den andra använder (t.ex.. Dette är forsumbart i jämförelse med först fasen för att S är mindre än U - antagligen mycket mindre. En lösning är att använda sig av en max-orienterade prioritetskö.

```
Queue q=shakeTree.keys();
MaxPQ<KeyValuePair>pq = new MaxPQ<KeyValuePair>(shakeTree.size())
w = q.dequeue();
while (w != NULL) {
 pq.insert(new KeyValuePair(shakeTree.get(w),w);
 w=q.dequeue();
}
```

Svar B

```
for(i=0,i++,i<M)
  printPair(pq.max());
```

Vi antog att vi hade tillgång till classen KeyValuePair där *compareTo* som jamför värden finns. Vi visar lösningen till b) d.v.s för Shakespeare.

Lösningen till 1.6 på nästa sida: