CHALMERS | GÖTEBORGS UNIVERSITET

Institutionen för data- och informationsteknik

TENTAMEN

KURSNAMN	**Algoritmer och datastrukturer, 7.5p**
PROGRAM	**TIDAL-2, TKIEK-2** **LP4 – 2020**
KURSBETECKNING	**LET375**
EXAMINATOR	**Peter Ljunglöf**
TID FÖR TENTAMEN	**Torsdag 2020–06–04, 14:00**
HJÄLPMEDEL	**Se slutet på nästa sida**
ANSVARIG LÄRARE	**Pelle Evensen, se** **https://chalmers.instructure.com/courses/9440/pages/instruktioner-for-e-examination** **för hur Pelle kontaktas under tentamen.**
DATUM FÖR ANSLAG	**Senast 2020–07–02.**
ÖVRIG INFORMATION	**Betygsgränser: Se nästa sida**

TENTAMEN

Algoritmer och datastrukturer

- Skriv rent dina svar. Oläsliga svar *rättas ej!*
- Om du lämnar flera olika svar på *samma uppgift* i *samma fil* eller olika filer (olika filnamn där det inte tydligt framgår vilken som är den sista) med svar på samma uppgift kan det bli så att svaret inte bedöms.
- Oprecisa eller alltför generella (vaga) svar ger inga poäng. Konkretisera och/eller ge exempel. Det är aldrig någon risk att vara övertydlig!
- Programkod skall skrivas i Java 5, eller senare version, och vara indenterad och renskriven.
- Onödigt komplicerade lösningar ger poängavdrag.
- Givna deklarationer, parameterlistor etc. *får inte ändras* om det inte uttryckligen är en del av uppgiften. Fråga i oklara fall!
- Läs igenom tentamenstesen och förbered eventuella frågor.
- Beskrivning av tillåtna/otillåtna hjälpmedel samt hur lärare kontaktas under tentamen finns beskrivet på kurshemsidan: https://chalmers.instructure.com/courses/9440/pages/instruktioner-for-e-examination

Tentamenspoäng och betygsgränser

Tesen innehåller 6 grundläggande och 3 avancerade frågor. Varje fråga kan ge maximalt 2 poäng.

Maxpoäng är följaktligen 18p (12p grundläggande + 6p avancerade). Avancerade poäng kan inte tillgodoräknas som grundläggande. Följande tabell används för betyg:

Betyg	Grundläggande poäng	Avancerade poäng
3	≥ 8	–
4	≥ 9	≥ 2
5	≥ 10	≥ 4

Lycka till!

Uppgift 1, grundläggande: Komplexitet & korrekthet

Betrakta följande funktioner:

```
/**
 * @pre s != null && m != null
 * @param s the sequence to do the matching in.
 * @param p the pattern to match.
 * @return The index to the first occurence of p in s.
 * @return -1 if p does not occur in s.
 */
public static int match1(final int[] s, final int[] p) {
    for (int i = 0; i <= s.length - p.length; i++) {
        int j;
        for (j = 0; j < p.length; j++) {
            if (s[i + j] != p[j]) {
                break;
            }
        }
        if (j == p.length) {
            return i;
        }
    }
    return -1;
}
```

```
// More cleverer version?
// Example: Match {1, 2} in {1, 3, 1, 2, 3} =>
// after finding the first 3, skips ahead to the second 1, then finds {1, 2} at 2.
/** Supposedly a faster drop-in replacement for match1. */
public static int match2(final int[] s, final int[] p) {
    for (int i = 0; i <= s.length - p.length; i++) {
        int j;
        for (j = 0; j < p.length; j++) {
            if (s[i + j] != p[j]) {
                i += j; // Mismatch, skip ahead.
                break;
            }
        }
        if (j == p.length) {
            return i;
        }
    }
    return -1;
}
```

(A) Ge en så snäv värstafallskomplexitet för `match1()` som möjligt. Använd *s* för längden på arrayen som söks i och *p* för längden på arrayen som söks efter. Motivera ditt svar.

(B) Visa på ett fall där `match2()` inte ger samma svar som `match1()`. (`match2()` ger alltså felaktiga svar ibland.)

Uppgift 2, grundläggande: Hashtabeller & funktioner

Rita den hashtabell med open hashing (linear probing) du får med följande värden för uppgift 2:
https://chalmers.instructure.com/courses/9440/pages/probleminstanser

`X:Y` är nyckeln `X` som här hashar till värdet `Y`.

Hashtabellens storlek är *m*. *Istället för att vid kollision hoppa fram 1 steg* så ska du gå *k* steg fram.

Ditt svar ska innehålla följande:

(A) *m*, *k* och nycklarna samt deras hashvärden (i samma ordning som de ges på websidan).

Din tabell, där du markerat vilka element som kolliderat och hur probningen ser ut för att hantera kollisionerna.
Markera varje probning med `X-a` på rätt ställe i tabellen där `X` är nyckeln och `a` är antalet probningar för nyckeln så långt. Där det blir en träff, ringa in `X-a`.
Exempel: Om `A` ska sättas in och första probningen är en miss, skriv `A-1` för första probningen (på rätt tabellindex), sen `A-2`, o.s.v.

Uppgift 3, grundläggande: Prioritetsköer

Antag att du har ett program som skapar en prioritetskö och sedan gör ett antal `add()` och `removeMin()`.

Vi vet inte i vilken ordning programmet utför sina operationer men vi vet att programmet kommer att utföra sju `add()` och sju `removeMin()`.

`add()`-operationerna sker i den ordning som fås genom att följa länken för uppgift 3 här (men det kan alltså ske `removeMin()`-operationer mellan `add()`-operationerna):

https://chalmers.instructure.com/courses/9440/pages/probleminstanser

Varje gång programmet anropar `removeMin()` så skriver det ut det borttagna värdet.

Ordningen på add-operationerna måste ovillkorligen behållas!

Ditt svar ska innehålla följande:

(A) Siffrorna du fick för `add()`-operationerna.

(B) De tre sekvenserna a), b) och c)

(C) För varje sekvens, a), b) och c):
Om sekvensen kan skrivas ut, ge en följd av av `add()` och `removeMin()`-operationer som ger utskriften.
Om sekvensen inte kan skrivas ut, förklara varför utskriften inte kan uppstå.

Uppgift 4, grundläggande: Grafer

Rita en viktad, oriktad graf med följande egenskaper:

(1) Den har exakt 7 noder, A, B, C, D, E, F, & G.

(2) Varje nod har minst tre kanter till andra noder än sig själv.

(3) Alla kanter har vikt sådan att $0 < \text{vikt} \leq 20$.

(4) Den kortaste vägen A till G har vikten R, R fås genom att följa länken till uppgift 4 här: https://chalmers.instructure.com/courses/9440/pages/probleminstanser

(5) Det minimala spännande trädet för grafen har vikt 10 + R.

(6) Alla noder ingår i det spännande trädet.

Ditt svar ska innehålla följande:

(A) Ditt R som du fått genom att följa länken ovan.

(B) Vikten hos kortaste vägen mellan A och G.

(C) Vikten för det minimala spännande trädet.

(D) Grafen, där alla kanters *vikter tydligt syns* och det minimala spännande trädet är inritat, med avvikande, eller tjockare linjer. Förvissa dig om att din graf är lätt att läsa!

(E) Den kortaste vägen som en lista av noder, A → ... → G.

Tips: Det är valfritt att visa hur du gått tillväga för att skapa ditt MST och din kortaste väg.

Uppgift 5, grundläggande: Binära sökträd

Börja med att ta fram ett binärt träd genom att följa länken till uppgift 5 här:
https://chalmers.instructure.com/courses/9440/pages/probleminstanser

(A) Visa 3 olika insättningssekvenser som alla hade kunnat skapa trädet i fråga.

(B) Visa 1 insättningssekvens som *inte* hade kunnat skapa trädet. Första elementet i ditt motexempel måste vara samma element som rotnoden i ditt träd. Om trädet t.ex. har ”P” som rot måste din sekvens alltså lyda ”P ...”

(C) Visa hur ditt träd ser ut efter att du tagit bort det element som utgör rotnoden. Förklara hur du går tillväga och rita det nya trädet.

Uppgift 6, grundläggande: Sortering

Hämta en lista med 6 tal från länken till uppgift 6 här:

https://chalmers.instructure.com/courses/9440/pages/probleminstanser

Givet quicksort-implementationen nedan, visa hur du kan ordna dina element så att värsta-fallet operationer ($\sim n^2 / 2$) uppnås. Till pivot-element väljs alltså alltid det mittersta elementet, avrundat nedåt.
Detta är samma partitionering som i kursboken och kallas ”Hoare partition scheme – Wikipedialänk”.

```
partition(a, start, end) {
    p = a[(start + end) / 2]
    i = start
    j = end
    while (true)
        while (a[i] < p) i = i + 1
        while (a[j] > p) j = j - 1
        if (i ≥ j)
            break
        else
            swap(a, i, j) // Swaps elements at index i and j in a.
            i = i + 1
            j = j - 1

    return j
```

```
sort(a, start, end)
    if(start < end)
        p = partition(a, start, end)
        sort(a, start, p);
        sort(a, p + 1, end);
```

Illustration

Om vi *istället hade valt det första elementet* som pivotelement och vår lista är `{A B C D E F}` så skulle vi kunna uppnå värsta-fallet genom följande ordning:

```
FEDCBA -> EDCBA F
EDCBA -> DCBA E F
DCBA -> CBA D E F
CBA -> BA C D E F
BA -> A B C D E F
```

Ditt svar ska innehålla följande:

(A) Din lista med tal.

(B) Din lista med tal ordnade så att quicksort med mittersta elementet som pivot-strategi kommer att partitionera så obalanserat som möjligt.

(C) Varje steg i partitioneringen (se illustration ovan).

Uppgift 7, avancerad: Komplexitet

Betrakta följande funktion:

```
// Antag att 0 < m < n
verySlow(n, m)
    if n ≤ m
        return 1
    else
        return verySlow(n - 1, m) + verySlow(n - m, m)
```

Som funktionen är formulerad blir den väldigt långsam att beräkna[1].

För 1 poäng:

Formulera om funktionen `verySlow()` så att du istället får funktionen `ratherFast()`.
Tidskomplexiteten för `ratherFast()` ska som sämst vara $O(n)$.
För alla $0 < m < n$ så ska `ratherFast(n, m)` ge samma värde som `verySlow(n, m)`.

För 2 poäng:

Samma som för 1 poäng men med ytterligare bivillkor att din formulering av `ratherFast()` som mest använder $O(m)$ minne, inklusive anropsstack.

1 $O((1+\varepsilon)^n) \leq O(f(n,m)) \leq O(2^n)$, för $0 < \varepsilon$.

Uppgift 8, avancerad: Felsökning

Ett stort program, *K*, har helt plötsligt börjat krascha för vissa indata.
Indata till programmet kan beskrivas som $X = \{x_1, x_2, ..., x_n\}$.

Beskriv en algoritm, för att snabbast möjligt isolera en minimal delsekvens, $Y = \{x_a, ..., x_b\}$, sådan att *K(Y)* kraschar programmet. Vi kallar denna sekvens ”MKS” – ”minimal krasch-sekvens”.

Vi får också göra antagandet att varje sekvens som har en MKS som delsekvens får *K* att krascha.

Om det finns flera olika MKS som får *K* att krascha så är det valfritt vilken av dem som väljs, men den måste fortfarande vara minimal (kortast möjliga längd), se exempel nedan.

Om en given sekvens *X* inte kan få *K* att krascha så ska ditt program svara $a = -1$, $b = -1$.

Värt att notera är att vi i förväg inte känner till kraschsekvensen, bara att den kan ingå som delsekvens i en viss sekvens *X*.

Exempel:

X	MKS	*a*	*b*
{13, 12, 11, 12, 20}	{12, 11}	1	2
{11, 11, 11, 11, 11}	{11, 11, 11}	0, 1 eller 2	*a* + 2
{11, 12, 13, 14, 15}	{11, 12, 13, 14}	0	3
{1, 2, 3, 4}	{4, 3}	–1	–1

K(X) går i linjär tid i antalet element i *X*.

Beskriv en algoritm (tydlig pseudokod) som *löser problemet korrekt* samt beskriv algoritmens komplexitet.

För 1 poäng:

Värstafallskomplexiteten får inte vara högre än $O(n^2)$

För 2 poäng:

Värstafallskomplexiteten får inte vara högre än $O(n \log n)$.

Uppgift 9, avancerad:

Vad gör denna metod?

```
// G is a directed graph.
mystery(G)
    P ← Empty list
    N ← The set of nodes without incoming edges in G
    V ← Empty set, to track of visited edges.

    while not empty(N)
        a ← deleteRandomNode(N) // Remove arbitrary node from N.
        Push a to the end of P
        for every non-visited edge e from a to b
            Mark e as visited.
            if b has no unvisited incoming edges left then
                add b to N

    if size(P) < number of nodes in G
        // No solution.
        return NIL
    else
        return P

end mystery
```

Applicera algoritmen på grafen du får från uppgift 9 på problemsidan:
https://chalmers.instructure.com/courses/9440/pages/probleminstanser

Ditt svar ska innehålla följande:

(A) En bild av din graf G.

(B) En beskrivning av P, N och V genom varje steg i algoritmen (skriv bara ut förändringar).

(C) Algoritmens värstafallskomplexitet som en funktion av antalet noder och kanter i G.
För full poäng krävs det att du tydligt beskriver vilka antaganden du gör (och varför) med avseende på komplexitet för relevanta operationer på G, N och V.
Exempel:
”Vi väljer en dynamisk arraylista för att implementera P. Att lägga ett element till slutet kan göras i [amorterad] konstant tid.”
– Operationer på P behöver du alltså inte ta med i din beskrivning utan antag $O(1)$.