NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Eksamensoppgave i Algoritmer og datastrukturer, IDATT2101

Eksamensdato: 30. november 2024

Eksamenstid (fra–til): 09:00–13:00

Tillatte hjelpemiddel: ett A4-ark med notater

Faglig kontakt under eksamen: Helge Hafting

Tlf.: 924 386 56

Annen informasjon: Løsningsforslag i blått

Målform/språk: bokmål

Antall sider (uten forside): 6

Antall sider vedlegg: 0

Informasjon om trykking av eksamensoppgave			
Originalen er:			
1-sidig	☐	**2-sidig**	☒
sort/hvit	☐	**farger**	☒
		Flervalgskjema?	☐

Kontrollert av

. . . .
Dato Sign

Oppgave 1 **20%**

Analyser disse programmene, skriv kjøretiden med asymptotisk notasjon. Bruk Θ om mulig. Om ikke, bruk O og Ω. Alle parametre er større enn eller lik 0.

```
int prog_a(int a, int b) {
    int sum = 0;
    for (int i=0; i<a ; ++i) {
       for (int j=0; j<b; ++j) {
          sum += i*j;
       }
    }
    for (int k=0; k < b; ++k) {
       sum += k;
    }
    return sum;
}
```

```
int prog_b(int q, int r, int p) {
    int sum = 0;
    for (int i=0; i<q ; i += r) {
       sum += sqrt(i*r);
    }
}
```

```
int prog_c(int n, int [] tab) {
    int sum = 0;
    if (n > 0) {
       sum += 3 * prog_c(n/3, tab);
       sum += 3 * prog_c(n/2, tab);
       sum += 3 * prog_c(n/3, tab);
       for (int i=0; i<n; ++i) {
          sum += tab[i];
       }
    }
    return sum;
}
```

```
double prog_d(int n, float x) {
    if (n == 0) return 0.0;
    else return x * n;
}
```

```
int prog_e(int a, int b, int c) {
    int sum = 1;
    for (int i = 1; i < a; ++i) {
        sum += i;
        if (sum > c) return b;
    }
    return sum;
}
```

a: $T(a,b) \in \Theta(ab + a + b)$. Alternativt:

$\Theta(ab)$ når $a > 0$ og $b > 0$

$\Theta(a)$ når $b = 0$

$\Theta(b)$ når $a = 0$

b: $\Theta(q/r)$ d: $\Theta(1)$ e: $\Omega(1)$, $O(a)$

c: Trykkfeil tok denne oppgaven utenfor pensum. Man kan gjøre en antagelse, og få en løsbar oppgave:

c1: Anta det var ment prog_c(n/3, tab) overalt. Mestermetoden gir oss da $T(n) \in \Theta(n \log n)$. Kjøretiden kan helt klart ikke bli bedre enn dette; dette utelukker f.eks. lineær kjøretid.

c2: Anta det var ment prog_c(n/2, tab) overalt. Mestermetoden gir oss da $T(n) \in \Theta\left(n^{\log_2 3}\right) \approx \Theta\left(n^{1,58}\right)$

c3: Andre matematiske metoder kan gi n opphøyd i andre eksponenter. I snitt deler vi opp i 8/3. Setter vi det inn i mestermetoden, får vi $T(n) \in \Theta\left(n^{\log_{8/3} 3}\right) \approx \Theta\left(n^{1,12}\right)$

d: $T(n) \in \Theta(1)$

e: $T(n) \in O(a), \Omega(1)$

Oppgave 2 **20%**

Jeg trenger å lagre folkeregisteret for Norge i en hashtabell. Det er ca. 5 millioner personer, vi tar høyde for at det kan bli 6 millioner med tiden. Personer skal kunne slåes opp på navn. Jeg ønsker å bruke dobbel hashing.

a) For å hashe navn, må de konverteres til tall på noe vis. Foreslå hvordan det kan gjøres i dette tilfellet.

b) Foreslå en størrelse på hashtabellen, og passende hashfunksjoner. Begrunn valgene du gjør. (Det er ikke nødvendig å regne seg frem til bestemte primtall eller toerpotenser. Om du trenger «neste primtall etter 5000» kan du definere p lik neste primtall etter 5000, og deretter bruke p i svaret ditt. Tilsvarende for toerpotenser.)

a) Navn kan konverteres til tall med utgangspunkt i bokstavene. Deres unicodeverdier er tall. En veid sum blir et tall som kan brukes. Her bør vi vekte slik at alle tegn har effekt (ingen vektes med 0). Videre bør de vektes ulikt, så «Leif-Per» og «Per-Leif» ikke kolliderer. Til slutt må vi ha nok spredning, så de veide summene generelt blir større enn størrelsen på hashtabellen.

   En enkel og grei måte er å ta første tegn inn i en sum. Så ganger vi summen med 7, og legger til neste tegn. Slik fortsetter vi, gange med 7 og legge til neste, til alle tegnene er brukt opp. Vi får store tall, og ulik vekting av bokstavene.

Det fins mange andre brukbare metoder også.

b) Vi ønsker plass til 6 mill. Generelt er det lurt med et overhead på ca. 20%, så tabellstørrelsen bør være ca. 7,2 mill. To løsninger basert på pensum:

1. Første løsning, basert på restdivisjon

   La p være første primtall etter 7,2 mill. Tabellstørrelsen blir p. Hashfunksjonene blir $h_1(k) = k \mod p$, $h_2(k) = k \mod (p-1) + 1$ Eventuelt kan man bruke $h_2(k) = k \mod 7$ eller et lignende lavt tall, i håp om å dra nytte av caching når kollisjonskjeden ikke spres så mye. h_2 bør ha mer spredning enn forventet antall kollisjoner per innsetting.

   Begrunnelser: h_1 sprer over hele tabellen. h_2 får ingen felles faktorer med tabellstørrelsen, h_2 gir mindre tall enn primtallet p. h_2 blir heller ikke 0. Tall som kolliderer i h_1, behøver ikke kollidere i h_2 også.

2. Andre løsning, multiplikativ hashfunksjon

   La t være første toerpotens etter 7,2 mill. (2^{23}). Tabellstørrelsen blir t. La $A = \frac{\sqrt{5}-1}{2}$. Tabellstørrelsen blir t. Hashfunksjonene blir $h_1(k) = \lfloor t\,(kA - \lfloor kA \rfloor) \rfloor$, $h_2\,(k) = (2k+1) \mod t$. Om en vil dra mer nytte av caching, kan en lavere toerpotens enn t brukes i h_2, så lenge h_2 har mer spredning enn forventet antall kollisjoner. Med en toerpotens som tabellstørrelse, kan funksjonene implementeres med raske heltallsberegninger.

   Begrunnelser: h_1 sprer over hele tabellen. h_2 lager oddetall, som ikke har felles faktorer med en toerpotens. Tall som kolliderer i h_1, behøver ikke kollidere i h_2 også.

Oppgave 3 **20%**

a) ALT-algoritmen er en videreutvikling av Dijkstras algoritme. Gjør rede for hvordan den virker.

Se notatet om ALT som de fikk utdelt.

ALT: A*, Landemerker, Triangelulikheten.

A*: prioritere noder med summen av avstand fra startnode, og estimert avstand til mål. (Viktig at estimatet ikke blir for stort, men så nær virkelig avstand som mulig. Da holder søket seg nær den optimale ruta, og blir fortere ferdig.)

Landemerkene gir oss måter å estimere avstander. Vi velger en håndfull steder som landemerker, og gjør en preprosessering: Avstand fra hvert landemerke til hver node beregnes. Også fra hver node til hvert landemerke. Preprosessering gjøres bare en gang, altså ikke for hvert søk. Avstandene kan finnes med Dijkstras algoritme.

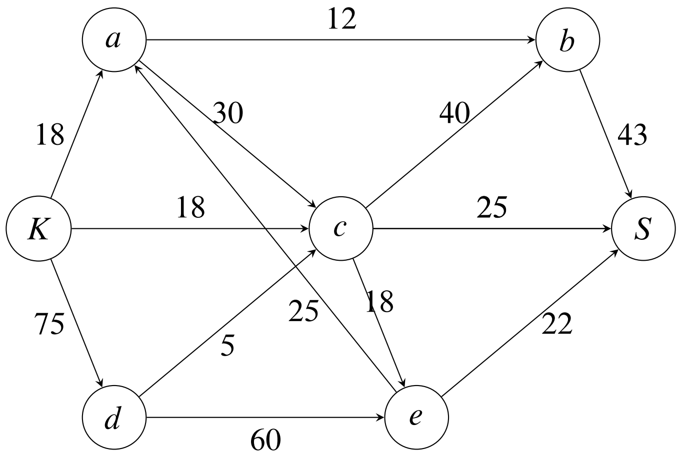
For å fungere godt, bør landemerker ligge bortenfor målet, eller før startpunktet. Men vi kan søke etter hvilke som helst ruter i alle retninger, så landemerkene bør derfor ligge spredt rundt kanten av kartet.

Når algoritmen kjøres, kan vi estimere avstand fra en node til mål som enten Landemerke til mål minus Landemerke til n, eller n til Landemerke minus mål til Landemerke.

Disse beregningene gjøres for alle landemerkene. De går fort, siden alle distanser som involverer et landemerke, er beregnet fra før under preprosesseringen. Man får ikke gode tall fra alle landemerkene, men man bruker enkelt og greit det høyeste av alle estimatene. Triangelulikheten garanterer at estimatene ikke er for høye.

Oppgave 4 **20%**

Bruk denne grafen.



a) Finn maksimal flyt fra K til S. Nytt flytøkende veier, og skriv opp hver vei og hvor mye flyt du legger til langs veien.

Mulige flytøkende veier:

```
Maksimum flyt fra K til S med Edmond-Karp
Økning : Flytøkende vei
    18   K c S
     5   K d c S
    22   K d e S
     2   K a c S
    12   K a b S
     4   K a c b S
    24   K d e a c b S
Maksimal flyt ble 87
```

Det er mulig å ha andre flytøkende veier enn dette, men summen må bli 87.

b) Finn og skriv opp alle grafens sterkt sammenhengende komponenter, eller forklar hvorfor det ikke er mulig.

5 komponenter: K, d, a+c+e, b, S

c) Se bort fra retningen på kantene, og finn et minimalt spenntre for grafen. Skriv opp hvilke kanter som blir med, og total vekt på spenntreet.

Ka:18, ab:12, Kc:18, cd:5, ce:18, eS:22. Samlet vekt 93.

d) Sorter grafen topologisk, eller forklar hvorfor det ikke er mulig.

Umulig på grunn av løkka *acea*.

Oppgave 5 — 20%

a) Forklar kort hvordan dual pivot quicksort virker.

Algoritmen velger to delingstall / pivots.

Quicksort deler tabellen i tre; en del med små nøkler mindre enn minste pivot, en del med store tall større enn største pivot, og en del med nøkler som ligger mellom pivots. Nøkler som står feil, byttes rundt så de blir liggende i riktig del.

Når dette er gjort, har vi en deltabell med små nøkler, en deltabell med middels nøkler, og en med store. De tre delene har rett rekkefølge i forhold til hverandre, men kan ha uorden internt. Delene sorteres rekursivt med dual pivot quicksort, og deles videre opp til hver deltabell har størrelse 1. Da er hele tabellen sortert.

Hvis de to pivots er like, må alle nøkler i midterste intervall også være like, og da trenger ikke dette midterste intervallet sorteres rekursivt.

b) Fortell kort hvordan heapsort virker

Først gjøres tallene om til en max-heap (lag_heap() som bruker fiks_heap() på alle indre noder. fiks_heap() bytter en node med den største barnenoden, hvis barnenoden er større. Deretter rekursiv fiks_heap() på plassen den ble byttet til, i fall den skal lenger ned.) Når dette er ferdig, har tabellen blitt en max-heap.

Deretter fjernes et og ett tall med hent_maks(), og legges i slutten av tabellen (etter heap-strukturen.) Slik gjøres sorteringen in-place.

Når nest siste tall er plassert slik, er tabellen sortert i stigende orden.

En max-heap er et binærtre der hvert element er større eller lik sine barn. I tabellform ligger rota først, deretter dens barn, deretter deres barn osv.

c) Sammenlign heapsort og quicksort. Hva er deres gode og dårlige sider?

quick: generelt raskest, men har et sjeldent n^2-tilfelle som kan være plagsomt om det skulle inntreffe.

heapsort: Garantert kjøretid på $n \log n$, men en høyere konstant faktor gjør den tregere enn quicksort i de aller fleste tilfeller. Likevel god å ha, om man trenger garantier for kjøretiden.