Utkast til eksamenssett, TDT4125, våren 2007

Antall studenter:	40
Målformer:	Bokmål, Nynorsk, Engelsk
Eksamensdato:	6. juni
Leveringsfrist til adm.:	30. mai

Oppgave 1

Anta at vi vil finne en tilnærmet løsning på optimaliseringsversjonen av TSP i en komplett, urettet, metrisk vektet graf G (dvs. kantvektenene følger trekantulikheten, $w(i, k) \leq w(i, j) + w(j, k)$ for alle noder i, j, k). Betrakt følgende heuristiske algoritme A:

> Start med en vilkårlig node, og i hvert trinn, gå videre til den nærmeste ubesøkte noden.

Merk: Den turen vi er ute etter starter og slutter i to ulike noder (dvs. vi er her ikke ute etter en sykel).

La T være turen (stien) som produseres av A (fra den første noden til den siste). La $v_1 \ldots v_n$ være nodene i G, ordnet synkende etter kostnaden på ut-kanten noden har i turen T, bortsett fra v_n, som er den siste noden i T (som altså ikke har noen ut-kant). La c_i være kostnaden til T-kanten ut fra v_i.

a. Betrakt nodene $v_1 \ldots v_i$, som altså har de i dyreste ut-kantene i T. Vis at for enhver kant $v_k v_l$ i G, der $k, l < i$, har vi at $w(k, l) \geq c_i$.
Løsning: Anta at $w(k, l) < c_i$. Hvis v_k var den første som ble besøkt av v_k og v_l så ville A ha valgt $v_k v_l$ fremfor kanten ut fra v_k i T, som har en kostnad $c_k \geq c_i$, siden v_l ville være ledig, og kanten dit ville være billigere.

b. Vis at algoritmen A er en $O(\log n)$-approksimasjon, der n er antall noder.
Hint: Vis at $c_i \leq \mathrm{OPT}/i$.

Løsning: En TSP-tur for $v_1 \ldots v_i$ vil ha en kostnad på maksimalt OPT (pga. metrisiteten), og siden den består av i kanter må minst én av kantene ha en kostnad på maksimalt OPT/i. Siden kostnaden til denne kanten (som alle de andre under betraktning) er større eller lik c_i (jfr. **a**) så følger det at $c_i \leq \mathrm{OPT}/i$. Den totale kostnaden til T vil altså være øvre begrenset av
$\mathrm{OPT} \cdot H_{n-1} \in O(\log n)$.

Oppgave 2

Nerde-Nils sier til deg: «Jeg har funnet et sorteringsnettverk med n inputs ($n > 2$), som for n gitte unike heltall, sorterer alle unntatt 1 av de $n!$ permutasjonene korrekt.»

a. Tror du på ham? Begrunn svaret med et eksempel på at det er mulig, eller bevis at det aldri kan være tilfelle.

Løsning: Nerde-Nils lyver. Vi antar at vi har et sorteringsnettverk med n (> 2) inputs som sorterer kun den ene sekvensen $a_1 \ldots a_n$ feil. For at et

sorteringsnettverk skal sortere mer enn én permutasjon riktig må det finnes minst én komparator. I vårt tilfelle er vi avhengige av at mer enn én permutasjon sorteres riktig fordi $3!-1 = 5$. Når det finnes minst én komparator må det også finnes en komparator på nivå 1. Vi antar så at denne komparatoren kobler sammen inputene a_i og a_j i den sekvensen som sorteres feil. Vi vet at outputen fra denne komparatoren alltid vil være $\min(a_i, a_j)$ på den øverste linjen, og $\max(a_i, a_j)$ på den nederste. Dermed vet vi at hvis sekvensen $a_1 \ldots a_i \ldots a_j \ldots a_n$ sorteres feil, så vil også sekvensen der a_i og a_j bytter plass sorteres feil, og vi har fått en selvmotsigelse.

En annen måte å se dette på er ved hjelp av 0-1-prinsippet. For at (minst) én permutasjone skulle ha blitt sortert galt må minst én 0-1-sekvens ha blitt sortert galt. Ved å velge en *cutoff*-verdi k, og la alle verdier under (over) dette mappe til 0 (1) vil vi alltid (ved å velge riktig k) kunne få minst to permutasjoner som mapper til den 0-1-sekvensen som sorteres feil.

Merknad: Teknisk sett burde nok Nerde-Nils uansett ha omtalt nettverket som et *sammenligningsnettverk*, siden et sorteringsnettverk per definisjon skal sortere alle inputs. Besvarelser som *kun* har påpekt dette har fått uttelling, men ikke full uttelling.

Oppgave 3

Betrakt følgende program:

for $i = 1 \ldots n$
    $R[i] \leftarrow$ sum fra $j = 1$ til n { 1 **if** $A[j] < A[i]$ **or** $(A[j] = A[i]$ **and** $j < i)$,
    0 **otherwise** }
for $i = 1 \ldots n$
    $B[R[i]] = A[i]$

a. Hva gjør programmet?

Løsning: Programmet sorterer tabellen A, og resultatet skrives til B vha. en såkalt *rank-sort*.

b. Hvilken kjøretid ville programmet hatt på en sekvensiell maskin?

Løsning: $O(n^2)$, der n er lengden på tabellen A.

c. Skisser en mest mulig effektiv realisering av programmet på en parallell CREW maskin.
Merk: Det er her snakk om effektiv kjøretid. Du står fritt til å bruke så mange prosessorer du vil.

d. Hvor mange prosessorer krever realiseringen din, og hva blir kjøretiden?

Løsning (c og d): Hver av summene i første løkke utføres av n prosessorer på hver vha. *parallel prefix*. Det fører til at første løkke utføres på $O(\log n)$ tid med $O(n^2)$ prosessorer. Siste løkke utføres av n prosessorer i konstant tid.

Kjøretid: $O(\log n)$
Prosessorer: $O(n^2)$

Oppgave 4

a. Let s be a string of length $n > 0$, and let $0 < q \leq n$. Let a q-gram denote a substring of s of length q, i.e., the substring $s[i{:}(i + q)]$ is a q-gram of s for some starting position i. We say that the q-gram covers the characters $\{i, \ldots, i + q\}$. In the following, for simplicity let's ignore the boundary cases, i.e., we can assume that s wraps around and that substring computations are done modulo n. Now, assume that we uniformly and independently sample, with replacement, the starting positions for m q-grams. Let K denote the number of characters in s covered by at least one of the sampled q-grams. What is $E[K]$, i.e., the expected value of K?

Solution: For a single sample, the probability that a character is covered is q/n. So the probability that it is not covered is $(1 - q/n)$. For m independent samples we therefore have that the probability that a character is not covered by any of the samples is $(1 - q/n)^m$. So the probability that a character is covered by at least one sample is $(1 - (1 - q/n)^m)$. Hence, $E[K] = n \cdot (1 - (1 - q/n)^m)$.

b. In **a** we made the simplifying assumption that substring computations were done modulo n. Does this assumption result in the computed $E[K]$ being an over- or underestimate? Explain.

Solution: For the at most $(q - 1)$ first and the at most $(q - 1)$ last characters, the probability for being covered is lower than for the "middle section" of s. Hence the computed $E[K]$ is an overestimate. Our assumption makes the probability of covering of a q-gram uniform for all starting positions, whereas for the "head" and "tail" of s it's really a staircase function. As n increases relative to q, the estimated $E[K]$ becomes more correct.

c. The edit distance $d(s, t)$ between two strings s and t is the minimum number of primitive edit operations required to transform s into t. Let the deletion of a character, insertion of a character, substition of a character and the transposition/swapping of two neighbouring characters define the set of primitive edit operations. Assume that different primitive edit operations may have different costs. Let $|s|$ and $|t|$ denote the lengths of s and t, respectively. What is the recursive formulation of d, that, when evaluated with arguments $|s|$ and $|t|$, yields the edit distance between s and t?

Solution: Let $|s|$ and $|t|$ denote the lengths of s and t, respectively. Then the edit distance between s and t is $ed(|s|, |t|)$, where ed is defined as follows:

$$
\begin{aligned}
ed(i, \mathrm{j})) = \min\{&ed(i, j - 1) + \mathrm{I}(i, j),\\
&ed(i - 1, j) + \mathrm{D}(i, j),\\
&ed(i - 1, j - 1) + S(i, j),\\
&ed(i - 2, j - 2) + T(i, j)\}
\end{aligned}
$$

where we have the boundary cases $ed(0, 0) = 0$ and $ed(i, j) = \infty$ if $i < 0$ or $j < 0$. Here, I, D, S and T denote the costs for insertions, deletions, substitutions and transpositions, respectively. Typically, unit edit costs are assumed, although this can be fully parametrized with, e.g., extensive lookup tables:

$I(i, j) = 1$ for all i, j.

$D(i, j) = 1$ for all i, j.

$S(i, j) = 0$ if $s[i] = t[j]$

$= 1$ otherwise

$T(i, j) = 1$ if $s[i-1] = t[j]$ and $s[i] = t[j-1]$

$= \infty$ otherwise

d. What are the computational complexities for the following three alternative evaluations of the formula in **c**? Use Ω for (i) and O for (ii) and (iii). (Answers that are not necessarily as tight as possible may still be acceptable for (i).)

(i) A brute force recursive evaluation.

(ii) Using a dynamic programming approach.

(iii) Using a dynamic programming approach, but assuming unit edit costs and exploiting bit-parellism.

Solution:

(i) $\Omega(2^{\min\{|s|,|t|\}})$ (Mer presise grenser, som f.eks. med grunntall 3, godtas også.)

(ii) $O(|s| \cdot |t|)$

(iii) $O(|s| \cdot |t| / w)$, where w is the word-size of the machine.

Oppgave 5

I databransjen er det gyldne økonomiske tider, noe som også til en viss grad kommer de ansatte til gode. Lånekassen er derimot ikke mer gavmilde enn vanlig, og du som datastudent begynner å se deg om etter andre ting å gjøre enn å sose rundt på Gløshaugen. Den økonomiske kynismen har nådd deg også, og du vil sørge for at du utelukkende gjør ting som er positive for din egen livslønn. Det finnes imidlertid en stor mengde aktiviteter å velge mellom, og det blir viktig å velge de mest fordelaktige.

Etter lange undersøkelser kommer du fram til en rekke aktiviteter de nærmeste 20 årene, og du vet når de starter og når de slutter. Du lager også et estimat for hvor mye den gitte aktiviteten vil påvirke din livslønn, i kroner. Dessverre er det slik at flere av aktivitetene foregår samtidig (dvs. overlapper helt eller delvis), og du vil finne det utvalget som gir en så høy livslønn som mulig.

a. Gitt en liste av aktiviteter med (st_i, sl_i, c_i), der st_i er starttidspunktet, sl_i er sluttidspunktet, og c_i er den forventede avkastningen for aktivitet i, skisser en algoritme som finner den optimale positive effekt et utvalg av disse aktivitetene kan ha på din livslønn (dvs. det subsettet av ikke overlappende aktiviteter som har størst avkastningssum). st_i og sl_i er kodet som millisekunder siden midnatt 1. januar 1970, og du kan anta at $st_i < sl_i$ for alle aktiviteter.

Løsning: La n være antall akvitiveter. Sorter aktivitetene etter økende sluttidspunkt.

$a \leftarrow$ **new** int[n+1]

$a[0] \leftarrow 0$
for $i \leftarrow 1 \ldots n$
  binærsøk etter siste k slik at $sl_k \leq st_i$.
  Hvis det ikke finnes, returner 0.
  $a[i] \leftarrow \max(a[i-1], c_i + a[k])$
$a[n]$ inneholder svaret

Binærsøket fungerer fordi man har sortert på stigende slutttidspunkt på forhånd.

b. Angi kjøretid og minneforbruk for algoritmen du ga i **a** i O-notasjon.

Løsning: Kjøretiden er $O(n \log n)$. Det er pga. sortering, og binærsøk i siste forløkke. Minneforbruket er lineært, $O(n)$.

Mulige alternativer:

1. Samme tankegang, men med linært søk istedenfor binærsøk. Gir forholdsvis god uttelling.

2. Lage en array som er like stor som avstanden i tid mellom første startede aktivitet, og siste avsluttede. Det blir lite hensiktsmessig, siden tidsoppløsningen er svært stor (og kjøretiden vil her uansett være pseudopolynomisk/eksponensiell). Vil gi noe uttelling.

Korrekt kjøretid og minneforbruk på en lite tilfredsstillende løsning (som for eksempel en *brute force*-løsning) vil gi lite uttelling.

Oppgave 6

Du står overfor et spesielt indekseringsproblem, der domenet er partisjonert i tre mengder, O, P og Q. Anta at disse representerer data-*objekter* (som det søkes blant), *pivot*er og *queries*, respektivt. Avstandsfunksjonen d er definert over objektpar fra mengden $Q \times P \cup P \times O \cup Q \times O$. Dvs. hvis $o \in O$, $p \in P$ og $q, q' \in Q$ er $d(q, p)$, $d(p, o)$ og $d(q, o)$ definert, mens for eksempel $d(o, p)$ eller $d(q, q')$ er udefinerte.

For alle slike objekter o, p og q gjelder trekantulikheten i følgende tre utgaver:

$d(q, p) \leq d(q, o) + d(p, o)$ (*)
$d(p, o) \leq d(q, o) + d(q, p)$
$d(q, o) \leq d(q, p) + d(p, o)$

a. Diskuter kort hvordan eksisterende indekseringsmetoder kan brukes på dette avstandsrommet.

Løsning: De ulike bevisene for Lemma 4.1–4.4 i pensum bruker bare de definerte avstandene, så metodene som baserer seg på disse (og det er jo de aller fleste) kan brukes direkte. I beviset til f.eks. Lemma 4.2 brukes d(o, p) i stedet for $d(p, o)$, men beviset blir helt ekvivalent. Man vil få problemer med metoder som f.eks. AESA og SA-trær, der objekter figurerer både som dataobjekter og pivoter.

Anta nå at den første av de oppgitte trekantulikhetene (merket *) ikke gjelder.

b. Vis kort hvordan en svakere form for pivotering (dvs. med en struktur som ligner LAESA) kan brukes i dette nye avstandsrommet.

Løsning: Vi kan bruke «halvparten» av underestimatet basert på pivoter:

$d(p, o) \leq d(q, o) + d(q, p)$ gir oss $d(p, o) - d(q, p) \leq d(q, o)$.

c. Diskuter kort bruken av hyperplanpartisjonering i dette nye avstandsrommet.

Løsning: Hyperplanpartisjonering baserer seg på lemma 4.4 som igjen baserer seg på den første trekantulikheten. Denne formen for partisjonering er altså ikke mulig.

d. Diskuter kort bruken av ballpartisjonering som brukt i VP-trær i dette nye avstandsrommet.

Løsning: VP-partisjonering baserer seg på snitt både med regionen *innenfor* en ball og regionen *utenfor*. Den bruker altså *begge* halvdeler av underestimatet i Lemma 4.2, og trenger derfor den første trekantulikheten. VP-partisjonering kan altså *ikke* brukes i dette avstandsrommet.