TECHNISCHE UNIVERSITEIT DELFT
Faculteit Elektrotechniek, Wiskunde en Informatica

TU Delft

Tentamen IN2505-A en TI2300 Algoritmiek

11 april 2012, 14.00-17.00

- Het gebruik van boek, aantekeningen of rekenmachine tijdens dit tentamen is ***niet*** toegestaan.
- Schrijf tenminste op je eerste antwoordvel en op het formulier voor de multiple choice de vakcode van het vak waarvoor je tentamen wil doen: IN2505-A (oud) of TI2300 (nieuw).
- Dit tentamen heeft 15 meerkeuzevragen goed voor 3/10 van je cijfer en 4 open vragen in totaal goed voor 6/10 van je cijfer (onderverdeeld in 30 punten). Je krijgt sowieso 1/10 kado. Merk op dat je eindcijfer voor dit vak ook voor $\frac{1}{3}$ uit het onafgeronde gemiddelde van het practicum bestaat (mits beide ≥ 5.8).
- Als je voor de huiswerkopgaven van dit collegejaar gemiddeld minstens een 5.8 hebt, dan wordt je cijfer voor dit tentamen met 1.0 verhoogd (maar nooit hoger dan 10.0).
- Wat betreft de meerkeuzevragen:
  - Er is voor iedere vraag telkens maar één goed antwoord mogelijk.
  - Alle vragen tellen even zwaar, maar bij de puntentoekenning wordt wel gokkanscorrectie toegepast. Als je geen enkel antwoord geeft, wordt dit altijd fout gerekend.
  - Schrijf je antwoorden eerst op kladpapier en neem ze later over op het antwoordformulier.
  - Vul je studienummer zowel met cijfers in als met streepjes/blokjes.
  - Vul het antwoordformulier bij voorkeur in met pen (geen rode pen) of potlood (goed zwart maken); gebruik geen doorhalingen.
  - Probeer in totaal niet meer dan een uur aan de meerkeuzevragen te besteden.
- Voor de open vragen heb je ongeveer twee uur nodig:
  - Geef antwoord in correct Nederlands of Engels en schrijf leesbaar (gebruik eerst kladpapier).
  - Geef geen irrelevante informatie. Dit kan leiden tot puntenaftrek. Het aangegeven maximaal aantal regels is van toepassing op (getypte) regels en wordt dus aangepast aan de grootte van je handschrift en het gebruik van witruimte.
  - Als er wordt gevraagd om een efficiënt algoritme, geef dan het meest efficiënte algoritme dat je kunt bedenken. Een langzamer algoritme kan minder punten opleveren.
  - Als er wordt gevraagd om pseudocode, dan mag je daarin algoritmen uit het boek aanroepen, tenzij anders vermeld.
  - Voordat je je antwoorden inlevert, controleer of op ieder blaadje je naam en studienummer staat en geef de vakcode en het aantal ingeleverde bladen voor de open vragen aan op (tenminste) de eerste pagina.
- De tentamenstof bestaat uit Chapters 4–7 uit Algorithm Design van Kleinberg en Tardos (behalve secties 4.4, 4.9*, 5.6, 6.8–6.10*, 7.4*, en 7.13*), en bovendien de notities over de Master Method en over bewijstechnieken (Proving guide).
- Totaal aantal pagina's (zonder dit voorblad): 6.

Meerkeuzevragen

1. Voor je volgende project moet je van n taken de looptijd meten. Voor ieder van de taken $i \in \{1, \ldots, n\}$ ben je naar verwachting t_i minuten bezig met het implementeren en daarna heeft die taak naar verwachting p_i minuten nodig om uitgevoerd te worden op een van de vele pc's op de Drebbelweg (eventueel parallel).

   In welke volgorde moet je de taken implementeren om zo snel mogelijk met het project klaar te zijn?

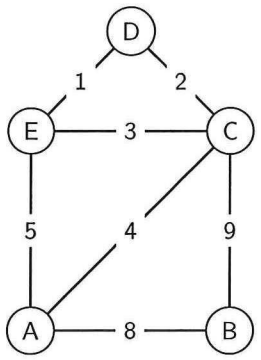
   A. Aflopend op t_i
   B. Oplopend op t_i
   C. Aflopend op p_i
   D. Oplopend op p_i

2. Beschouw de volgende twee beweringen over de Huffman codering.

   1. In een Huffman codering heeft een letter met een hogere frequentie nooit een langere codering dan een letter met een lagere frequentie.
   2. Een Huffman codering kan altijd gerepresenteerd worden door een *gebalanceerde* prefix boom.

   A. Beide beweringen zijn correct.
   B. Alleen bewering 1 is correct.
   C. Alleen bewering 2 is correct.
   D. Geen van beide beweringen is correct.

3. Beschouw de graaf $G = (V, E)$ in Figuur 1. Wat weten we over een minimale opspannende boom (MST) van deze graaf?



Figuur 1: Graaf G

   A. De kanten (C,E) en (A,E) komen in iedere minimale opspannende boom voor.
   B. De kanten (C,E) en (A,E) komen in geen enkele minimale opspannende boom voor.
   C. De kant (C,E) komt in iedere minimale opspannende boom voor en de kant (A,E) in geen enkele.
   D. De kant (A,E) komt in iedere minimale opspannende boom voor en de kant (C,E) in geen enkele.

4. Gegeven een verzameling punten en afstanden tussen die punten, hoe kun je efficiënt een clustering van k clusters vinden waarbij de kleinste afstand tussen punten van verschillende clusters zo groot mogelijk is (max spacing)?
   A. Kies een lijn L en cluster de punten links van L en rechts van L in $\lfloor \frac{k}{2} \rfloor$ en $\lceil \frac{k}{2} \rceil$ door twee recursieve aanroepen.
   B. Sorteer de paren punten oplopend op hun onderlinge afstand en stop punten in die volgorde bij elkaar in een groep totdat er precies k groepen zijn.
   C. Bouw een minimale opspannende boom (minimum spanning tree) van de punten en knip deze op vanuit de wortel totdat er precies k deelbomen zijn.
   D. Sorteer de paren punten oplopend op hun onderlinge afstand en kies recursief een opdeling waarbij de som van de kwadraten van de onderlinge afstanden van punten binnen een cluster minimaal is.

5. Het algoritme (van Karatsuba en Ofman) om in minder dan kwadratische tijd twee integers te vermenigvuldigen van n bits is gebaseerd op een herschrijving van de vermenigvuldiging en een recursieve aanroep van de vermenigvuldiging van integers bestaande uit minder bits. Wat is de krapste bovengrens op de looptijd van dit algoritme? (Hint: gebruik de Master methode.)
   A. $O(n \log n)$
   B. $O(n^2 \log n)$
   C. $O(n^{\log_2 3})$
   D. $O(n^{\log_3 2})$

6. Laat een recursief algoritme gegeven zijn met een looptijd beschreven door de recurrente betrekking $T(n) = T(n-1) + n^2$. Wat is de krapste bovengrens op de looptijd van dit algoritme?
   A. $O(n^2)$
   B. $O(n^2 \log n)$
   C. $O(n^3)$
   D. $O(2^n)$

7. Gegeven een lijn L, leveren de twee recursieve aanroepen voor het bepalen van de twee dichtstbijzijnde punten in het platte vlak (het Closest-Pair-Of-Points algoritme) de kleinste afstand tussen twee punten ofwel links ofwel rechts van L op. Het minimum hiervan noemen we δ. Wat gebeurt er vervolgens voor ieder punt p binnen een afstand δ van L om efficiënt de twee dichtstbijzijnde punten in het *hele* vlak te vinden?
   A. Het algoritme berekent de kleinste afstand van p tot een constant aantal punten niet verder dan δ van L, in volgorde van y coördinaat.
   B. Het algoritme berekent de kleinste afstand van p tot alle punten met een afstand van minder dan δ van p, in volgorde van y coördinaat.
   C. Het algoritme berekent de kleinste afstand van p tot alle punten aan de andere kant van L en niet verder dan δ van L, in volgorde van y coördinaat.
   D. Het algoritme sorteert alle punten niet verder dan δ van L op y coordinaat en berekent dan de kleinste afstand van p tot de eerste v̈an de gesorteerde lijst.

8. Gegeven is een verzamelingen van n open intervallen (starttijd, eindtijd) met unieke eindtijden in $\mathbb{R}^+$ en voor ieder interval i een waarde v_i. Voor ieder interval i definiëren we de $p(i)$, de predecessor van i, als het laatste interval dat nog eindigt voor i begint. Met welke van de volgende formules voor OPT kun je (door middel van dynamisch programmeren) efficiënt bepalen wat de hoogste totaalwaarde (som van de waarden) is van een deelverzameling niet-overlappende intervallen?
   A. $\mathrm{OPT}(j) = \max_{1 \le i \le j}(\ \mathrm{OPT}(p(j)) + v_i\ )$
   B. $\mathrm{OPT}(j) = \max(\ \mathrm{OPT}(j-1), v_j + \mathrm{OPT}(p(j))\ )$
   C. $\mathrm{OPT}(j) = \max(\ v_j + \mathrm{OPT}(j-1), \mathrm{OPT}(p(j))\ )$
   D. $\mathrm{OPT}(i,j) = \max(\ \mathrm{OPT}(i-1,j), v_i + \mathrm{OPT}(i-1,p(i))\ )$

9. Beschouw het volgende algoritme voor het bepalen van de maximale som niet groter dan W van een deelverzameling van de getallen $\{w_1, w_2, \ldots, w_m\}$ (Subset Sum).

```
1  for j ← 1 to n do
2      M[j, 0] ← 0
3  end
4  for w ← 0 to W do
5      M[0, w] ← 0
6  end
7  for j ← 1 to n do
8      for w ← 1 to W do
9          if w_j ≤ w then
10             M[j, w] ← ...
11         end
12         else
13             M[j, w] ← M[j − 1, w]
14         end
15     end
16 end
17 print M[n, W]
```

Wat is een correcte invulling van regel 10?

A. $M[j, w] \leftarrow M[j-1, w-1]$
B. $M[j, w] \leftarrow M[j-1, w-w_j]$
C. $M[j, w] \leftarrow \max(M[j, w-1],\ w_j + M[j-1, w-w_j])$
D. $M[j, w] \leftarrow \max(M[j-1, w],\ w_j + M[j-1, w-w_j])$

10. Het probleem van het vinden van de ruimtelijke structuur van een RNA-molecuul (RNA Secondary Structure) is als volgt: gegeven een string basen $B = b_1 \ldots b_n$ vind een zo groot mogelijke verzameling S van paren (i, j) waarbij $i, j \in \{1, \ldots, n\}$, die aan de volgende voorwaarden voldoen:

    1. Geen scherpe bochten: als $(i, j) \in S$ dan $|j - i| > 4$.
    2. Ieder paar verwijst ofwel naar een A en een U, ofwel naar een C en een G.
    3. Geen enkele base b_i voor $i \in \{1, \ldots, n\}$ komt in meer dan één paar voor.
    4. Geen kruisingen: Als (i, j) en (k, l) twee paren in S zijn, dan mag het niet zo zijn dat $i < k < j < l$.

    De oplossing voor dit probleem met behulp van dynamisch programmeren slaat de resultaten van deelproblemen op in een tabel. Schrijf de recursieve functie die door dit algoritme gebruikt wordt op je kladpapier. Beantwoord dan de volgende vraag: wat is de krapste bovengrens op de looptijd van dit algoritme?

    A. $O(n \log n)$
    B. $O(n^2)$
    C. $O(n^2 \log n)$
    D. $O(n^3)$

11. Het probleem van het vinden van de beste alignment van twee strings is als volgt: gegeven strings X en Y van lengte m en n, match posities van X met posities van Y zodanig dat de boete voor mismatches in een alignment geminimaliseerd wordt. De boete voor mismatches in een alignment $M = \{(i, j)\}$ waarbij $i \in \{1, \ldots, m\}$ en $j \in \{1, \ldots, n\}$ wordt als volgt berekend:

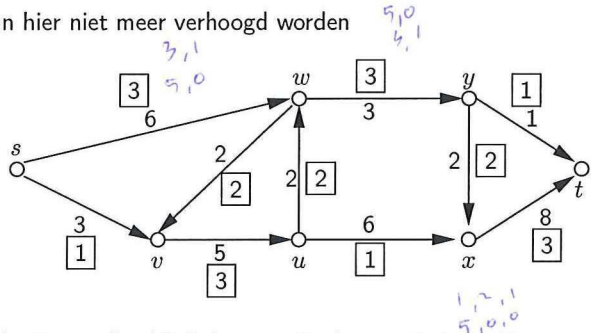
    1. een boete $\delta \geq 0$ voor iedere positie $i \in \{1, \ldots, m\}$ of $j \in \{1, \ldots, n\}$ die niet in M voorkomt, plus
    2. een boete $\alpha_{x_i y_j} \geq 0$ voor iedere $(i, j) \in M$ waarvoor $x_i \neq y_j$.

De standaardoplossing voor dit probleem gebruikt $\Theta(mn)$ geheugen. Er is ook een oplossing waarin het geheugengebruik lineair is in de grootte van de invoer. Van welke technieken wordt daarbij gebruik gemaakt?

A. Greedy en dynamisch programmeren
B. Stroomnetwerken (flow networks) en dynamisch programmeren
C. Verdeel en heers (divide and conquer) en dynamisch programmeren
D. Alle bovenstaande techniqueken

12. Laat een stroom (flow) f en een s-t snede (A, B) gegeven zijn. Welke bewering is correct?
A. De waarde van de stroom f is nooit meer dan de capaciteit van de snede (A, B).
B. Als f de maximale stroom is, dan is de waarde van f gelijk aan de capaciteit van de snede (A, B).
C. Als (A, B) een minimale snede (min-cut) is dan is de waarde van f gelijk aan de capaciteit van de snede (A, B).
D. Als (A, B) geen minimale snede is, dan bestaat er een stroomverhogend pad (augmenting path) van s naar t.

13. Bekijk het stroomnetwerk in figuur 2 en teken de restgraaf (residual graph). Met hoeveel kan de stroom hier nog verhoogd worden door het vinden van stroomverhogende paden in de restgraaf?
A. met 2
B. met 4
C. met 5
D. de stroom kan hier niet meer verhoogd worden



Figuur 2: Een stroomnetwerk. De getallen bij de kanten zijn de capaciteiten. De stroom op een kant is weergegeven in een vierkantje.

14. Bekijk nogmaals het stroomnetwerk in figuur 2 en je getekende restgraaf. Wat is een minimale snede?
A. $(\{s\}, \{u, v, w, x, y, t\})$
B. $(\{s, u, v, w\}, \{x, y, t\})$
C. $(\{s, v, w, y\}, \{u, x, t\})$
D. $(\{s, u, v, w, y\}, \{x, t\})$

15. Van het algoritme van Ford-Fulkerson bestaat ook een variant die een stuk efficiënter is door het gebruik van een schaling (scaling) techniek. Welk van de volgende beschrijvingen geeft het beste het idee van deze techniek weer?
A. Verlaag in iedere iteratie de restcapaciteit van de kanten met Δ en halveer Δ na iedere iteratie.
B. Verlaag in iedere iteratie de restcapaciteit van de kanten met Δ en verdubbel Δ na iedere iteratie.
C. Bekijk in iedere iteratie alleen kanten met een restcapaciteit van minimaal Δ en halveer Δ na iedere iteratie.
D. Bekijk in iedere iteratie alleen kanten met een restcapaciteit van minimaal Δ en verdubbel Δ na iedere iteratie.

Open vragen

1. Geef een asymptotische boven- en ondergrens voor $T(n)$ in ieder van de volgende recurrente betrekkingen. Maak je grenzen zo krap (tight) mogelijk en geef aan hoe je aan je antwoord komt (in maximaal 5 regels per recurrente betrekking).
   (a) (3 punten) $T(n) = 4T(n/2) + n^3 \log n$
   (b) (3 punten) $T(n) = 9T(n/3) + n^2$

2. Je hebt afgesproken om in de vakantie met een paar vrienden door de bergen trekken. Om geen tent mee te hoeven sjouwen spreken jullie af om in berghutten langs de route te overnachten. Je doel is om zo min mogelijk berghutten te reserveren, maar toch nooit meer dan 20km per dag te hoeven lopen. De lokatie van iedere berghut i langs de route ($1 \leq i \leq n$) is gedefinieerd door de afstand x_i in km vanaf het begin van de route. Het is gegeven dat iedere twee direct opeenvolgende berghutten nooit meer dan 20km van elkaar vandaan liggen.
   (a) (4 punten) Gegeven een serie berghutten $x_1, \ldots, x_n$ gesorteerd op hun afstand vanaf het begin van de route, geef pseudocode voor een iteratief $O(n)$ greedy algoritme dat de kortste serie $b_1, \ldots, b_k$ van te reserveren berghutten bepaalt zodanig dat er tussen iedere twee na elkaar geboekte berghutten b_j en b_{j+1} (voor $0 \leq j < k$) hooguit 20km afstand zit. Neem aan dat er ook een berghut $b_0 = x_0 = 0$ aan het begin van de route zit waar jullie sowieso willen overnachten (in maximaal 10 regels).
   (b) (3 punten) Je vrienden willen er echt zeker van zijn dat je programma correct is. Geef daarom eerst een bewijs met inductie ("greedy stays ahead") voor de volgende stelling (in maximaal 15 regels):
   Laat de optimale serie berghutten $o_1, \ldots, o_m$ gegeven zijn. Voor iedere $j = 1, \ldots, m$ geldt dat de j-de berghut geselecteerd door jouw algoritme minstens net zo ver lopen is als de optimale keuze, dus $b_j \geq o_j$.
   (c) (1 punt) Beargumenteer tenslotte hoe uit de bovenstaande propositie volgt dat je algoritme de kleinste serie berghutten geeft (in maximaal 5 regels).

3. Een tekst bestaat uit een serie van n woorden. Voor ieder woord i is het aantal karakters w_i gegeven. De breedte van een pagina (fixed-width font) is L karakters (inclusief witruimte). We bekijken het probleem van het verdelen van de serie woorden over verschillende regels zonder de woorden af te breken. Als eerste wordt voor iedere mogelijke regel, aangegeven door de index i van het eerste en j van het laatste woord van die regel, de slack $S_{i,j}$ berekend door het volgende algoritme.

```
for i ← 1...n do
    ℓ = −1 // Geen spatie aan het begin van de regel
    for j ← i...n do
        ℓ ← ℓ + w_j + 1 // Voeg woord en spatie toe
        if ℓ > L then S_{i,j} ← ∞ else S_{i,j} ← L − ℓ
    end
end
```

   De boete van een verdeling van de serie woorden over regels van ten hoogste L karakters definiëren we als de som van de kwadraten van de slack over deze regels. Let op dat ook de laatste regel meedoet in de verdeling en in deze boete.
   (a) (4 punten) Geef een recursieve formule voor het bepalen van de laagste mogelijke boete voor een verdeling van de woorden over regels. Geef ook aan met welk(e) argument(en) de eerste aanroep gedaan wordt.
   (b) (4 punten) Geef pseudocode voor een efficiënt iteratief algoritme op basis van dynamisch programmeren en de functie van de vorige vraag. Je mag alleen gebruik maken van opdrachten (statements) die standaard in een taal als Java beschikbaar zijn. Je mag er wel van uitgaan dat S_{ij} al berekend is voor iedere $i \leq j$.
   (c) (1 punt) Geef de krapste bovengrens op de looptijd van je algoritme uit de vorige vraag.

4. Om in een digitale foto voorgrond van achtergrond te onderscheiden, wordt het volgende model gebruikt. Voor iedere pixel i is een waarschijnlijkheid gegeven dat deze bij de voorgrond hoort, a_i, en dat deze bij de achtergrond hoort, b_i. Verder wordt voor iedere twee pixels i en j die naast of boven elkaar liggen een boete p_{ij} gegeven als een van beide tot de voorgrond en de ander tot de achtergrond wordt gerekend. De vraag is om een partitie (A, B) van de pixels te maken zodanig dat de kwaliteit $q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{i \text{ boven/naast } j,\ |A \cap \{i,j\}|=1} p_{ij}$ zo hoog mogelijk is.

   (a) (4 punten) Modelleer dit probleem door middel van een stroomnetwerk (flow network) zodat uit de minimale snede afgeleid kan worden welke pixels het beste bij de voorgrond en welke bij de achtergrond kunnen worden ingedeeld.

   - Beschrijf wat de knopen zijn,
   - wat de kanten zijn,
   - wat de capaciteiten op de kanten zijn en
   - hoe je de partitie van de pixels kunt maken (in maximaal 8 regels).
   - Maak ook een schets van een eenvoudig voorbeeld.

   (b) (3 punten) Bewijs dat de minimale snede (A, B) de partitie is met maximale kwaliteit $q(A, B)$ (in maximaal 5 regels).

Einde tentamenopgaven