Complexiteit woordzoeken

Laat W een string van lengte M zijn en laat T een string van lengte N zijn met $N \geq M \geq 1$ zijn. We gaan er vanuit dat de letters in W zijn opgeslagen in posities $0 \ldots M-1$ en dat de letters in T zijn opgeslagen in posities $0 \ldots N-1$.

We willen het aantal voorkomens van het woord W in de tekst T tellen. Hiervoor kunnen we het volgende algoritme gebruiken:

```
teller = 0;
for i=0 to N-M do  // i is mogelijke beginpositie van W in T
{ j=0;
  OK=true;
  while (j<M  AND  OK)
  { if (W[j] == T[i+j]) then
      j ++;
    else
      OK = false;
  }
  if (OK) then
    teller ++;
}
```

a Hoeveel iteraties heeft de while-lus maximaal (in het slechtste geval dus) binnen één iteratie van de for-lus?

b Druk de tijdscomplexiteit van het hele algoritme (voor het slechtste geval) uit in N en M.

c Geef een voorbeeld van twee strings W en T waarvoor de hoeveelheid werk in dit algoritme maximaal is (een voorbeeld van het slechtste geval dus).

Correctheid en complexiteit machtsverheffen

In opgave 5.15 worden verschillende algoritmes voorgesteld voor machtsverheffen: gegeven twee gehele getallen m en n, met $m \geq 1$ en $n \geq 0$, bereken m^n. Van twee van deze algoritmes gaan we de complexiteit analyseren

i Algoritme Pwr1:

```
PW = 1;
for i=1 to n do
  PW = PW * m;
```

iii Algoritme Pwr3:

```
PW = 1;
B = m;
E = n;
while (E != 0) do
{ if (E is even) then
  { B = B*B;
    E = E/2;
  }
  else   // E is oneven
  { PW = PW*B;
    E = E-1;
  }
}
```

a Wat is de tijdscomplexiteit van Pwr1?

b Algoritme Pwr3 ziet er 'magisch' uit. Toch is het correct.

- Toon aan dat de volgende bewering een invariant voor de while-lus is:
  $B^E * PW = m^n$
- Geef een mogelijke convergent voor de while-lus.

c Hoeveel iteraties (afhankelijk van n) heeft de while-lus in Pwr3 minimaal (in het beste geval) en maximaal (in het slechtste geval)?

d Wat is de tijdscomplexiteit van Pwr3?

Complexiteit heapsort

Ervanuitgaande

- dat er $N \geq 1$ getallen in het array A zijn opgeslagen, op posities $1 \ldots N$

kunnen we heapsort op de volgende manier implementeren:

```
  // maak een heap van het array A
for i=2 to N do
{   // laat A[i] omhoog borrelen
  j = i;
  while (j>=2  AND  A[j/2]>A[j])
  { verwissel A[j/2] en A[j];
    j = j/2;
  }
}

  // haal de getallen in oplopende volgorde uit de heap
TmpN = N;
for i=1 to N do
{ write A[1];
  A[1] = A[TmpN];
  TmpN --;

    // laat A[1] (de oude A[TmpN] dus) omlaag borrelen
  j = 1;
  OK = false;
  while (j<=(TmpN/2)  AND  !OK)
  { k = index van kleinste kind van A[j];
      // (k=2j of k=2j+1)
    if (A[j]>A[k])
    { verwissel A[j] en A[k];
      j = k;
    }
    else
      OK = true;
  }
}
```

a Hoeveel iteraties (afhankelijk van de waarde van i) heeft de while-lus in de eerste for-lus maximaal (in het slechtste geval dus) binnen één iteratie van de for-lus?

b Wat is de totale tijdscomplexiteit van de eerste for-lus (voor het slechtste geval)?

c Geef een voorbeeld van een rij getallen in A met $N = 10$ getallen, waarvoor de hoeveelheid werk in de eerste for-lus maximaal is (een voorbeeld van het slechtste geval dus).

d Hoeveel iteraties (afhankelijk van de waarde van TmpN) heeft de while-lus in de tweede for-lus maximaal (in het slechtste geval dus) binnen één iteratie van de for-lus?

e Wat is de totale tijdscomplexiteit van de tweede for-lus (voor het slechtste geval)?

f Geef een voorbeeld van een heap in A met $N = 10$ getallen, waarvoor de hoeveelheid werk in de tweede for-lus maximaal is (een voorbeeld van het slechtste geval dus).

Correctheid en complexiteit kortste pad met dynamisch programmeren

Op blz. 89–91 van het boek wordt een algoritme beschreven voor het berekenen van het kortste pad in een gerichte acyclische graaf. Dit algoritme maakt gebruik van dynamisch programmeren.
Ervanuitgaande

- dat de graaf $N \geq 1$ knopen bevat
- dat de buren van een knoop X eenvoudig af te lopen zijn met behulp van de volgende constructie:

```
for alle buren Y van X do
{
}
```

- dat het gewicht van de tak van knoop X naar knoop Y te verkrijgen valt met behulp van de functie Gewicht(X,Y)
- dat de knopen in de graaf al geordend zijn in een array Sort (op posities $1 \ldots N$), zó dat er alleen takken van links naar rechts lopen. Meer precies: als er een tak is van knoop X naar knoop Y, dan is $X = \text{Sort}[i]$ en $Y = \text{Sort}[j]$, met $1 \leq i < j \leq N$.

kwamen we eerder al tot de volgende pseudo-code implementatie:

```
L [ Sort[N] ] = 0;
for i=N-1 downto 1 do
{ X = Sort[i];
  Min = MaxInt;
  for alle buren Y van X do
  { if (Gewicht(X,Y) + L[Y] < Min)
      Min = Gewicht(X,Y) + L[Y];
  }
  L[X] = Min;
}
write (L[ Sort[1] ]);
```

a Wat kan er mis gaan met dit algoritme als er knopen X zijn vanwaaruit de laatste knoop Sort[N] helemaal niet te bereiken is?

Neem aan

- dat de laatste knoop Sort[N] vanuit iedere knoop X te bereiken is,
- dat de graaf M takken bevat.

b Druk de tijdscomplexiteit van het algoritme uit in N (het aantal knopen) en/of M (het aantal takken).

c Wat is het verband tussen N en M, en heeft dit gevolgen voor je antwoord bij onderbeel **b**?