Examen Algoritmen en Datastructuren III

Naam : .....

Stellingen uit de les mogen natuurlijk altijd gebruikt worden zonder dat het bewijs opnieuw gegeven moet worden (behalve in gevallen waar het expliciet anders staat)!

1. Vraag bij het project (Deze vraag wordt samen met het project en de practica gequoteerd op 4 punten.)

   (a) Beschrijf in voldoende detail wat het volgende MPI-programma precies doet. Leg ook bondig uit waarvoor de variabele `mtype` wordt gebruikt. Is het in dit korte programma noodzakelijk om met twee verschillende waarden voor `mtype` te werken? Verklaar.

```
#include <stdio.h>
#include "mpi.h"
#define NRA 62
#define NCA 15
#define NCB 7

MPI_Status status;
main(int argc, char **argv) {

int numtasks, taskid, numworkers, source, dest, nbytes, mtype, intsize,
    dbsize, rows, averow, extra, offset, i, j, k, count;
double a[NRA][NCA], b[NCA][NCB], c[NRA][NCB];

intsize = sizeof(int);
dbsize = sizeof(double);

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
numworkers = numtasks-1;

if (taskid == 0) {
  for (i=0; i<NRA; i++)
    for (j=0; j<NCA; j++)
      a[i][j]= i+j;
  for (i=0; i<NCA; i++)
    for (j=0; j<NCB; j++)
      b[i][j]= i*j;

  averow = NRA/numworkers;
  extra = NRA%numworkers;
```

```
  offset = 0;
  mtype = 1;
  for (dest=1; dest<=numworkers; dest++) {
    rows = (dest <= extra) ? averow+1 : averow;
    MPI_Send(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
    MPI_Send(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
    count = rows*NCA;
    MPI_Send(&a[offset][0], count, MPI_DOUBLE, dest, mtype,
MPI_COMM_WORLD);
    count = NCA*NCB;
    MPI_Send(&b, count, MPI_DOUBLE, dest, mtype, MPI_COMM_WORLD);

    offset = offset + rows;
  }

  mtype = 2;
  for (i=1; i<=numworkers; i++) {
    source = i;
    MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD,
             &status);
    MPI_Recv(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
    count = rows*NCB;
    MPI_Recv(&c[offset][0], count, MPI_DOUBLE, source, mtype,
              MPI_COMM_WORLD, &status);
  }

  for (i=0; i<NRA; i++) {
    printf("\n");
    for (j=0; j<NCB; j++)
      printf("%6.2f   ", c[i][j]);
  }

  printf ("\n");
}

if (taskid > 0) {
  mtype = 1;
  source = 0;
  MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
  MPI_Recv(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
  count = rows*NCA;
  MPI_Recv(&a, count, MPI_DOUBLE, source, mtype, MPI_COMM_WORLD,
&status);
  count = NCA*NCB;
  MPI_Recv(&b, count, MPI_DOUBLE, source, mtype, MPI_COMM_WORLD,
&status);
  for (k=0; k<NCB; k++)
    for (i=0; i<rows; i++) {
      c[i][k] = 0.0;
      for (j=0; j<NCA; j++)
        c[i][k] = c[i][k] + a[i][j] * b[j][k];
      }
```

```
  mtype = 2;
  MPI_Send(&offset, 1, MPI_INT, 0, mtype, MPI_COMM_WORLD);
  MPI_Send(&rows, 1, MPI_INT, 0, mtype, MPI_COMM_WORLD);
  MPI_Send(&c, rows*NCB, MPI_DOUBLE, 0, mtype, MPI_COMM_WORLD);

  }
  MPI_Finalize();
}
```

(b) In het project werd gevraagd om een algoritme te implementeren voor het berekenen van het product van een $n \times n$ matrix A en een vector x met n elementen op **een ring** van p processoren $P_0, P_1, \ldots, P_{p-1}$. Leg uit hoe je de matrix A en vector x over de verschillende processoren verdeelde en hoe je tenslotte de deelresultaten combineerde tot de eindoplossing. Antwoord voldoende nauwkeurig.

2. **Parallelle algoritmen: communicatie-algoritmen (2 pt.)**

   Beschouw een rooster (met wrap-around) van $n = p2$ processoren, genummerd van 0 tot $n-1$. Een **circulaire** q**-verschuiving** is een bewerking waarbij elke processor i een boodschap moet sturen naar processor $(i+q) \bmod p$. Geef een algoritme voor deze bewerking in het rooster met wrap-around. Bespreek ook de totale communicatietijd van het algoritme.

3. **String-matching (2 pt.)**

(a) Zij gegeven een patroon P van lengte m en een tekst T van lengte n. We hebben gezien dat het brute-kracht-algoritme voor het controleren of P in T voorkomt, tijdscomplexiteit $O(m(n-m+1))$ heeft. Geef een voorbeeld van een patroon en een tekst waarvoor het algoritme ook effectief $\Theta(m(n-m+1))$ vergelijkingen doet.

(b) Geef een voorbeeld waarbij de occurrence-heuristiek in het algoritme van Boyer-Moore het patroon P over $|P|-1$ posities naar links zou verschuiven (m.a.w. een negatieve verschuiving).

(c) Kan de occurrence-heuristiek ook leiden tot een verschuiving over nul posities?

4. **(2 pt)**

   Codeer de tekst `geen_eensteensmuur`. Toon voldoende tussenstappen en details om te kunnen zien wat je doet.

   - Gebruik Huffman codering. Construeer de code en codeer de tekst.

- Gebruik LZW. Toon aan hoe het woordenboek wordt opgebouwd en codeer de tekst. Voor de lettertekens moet je de code niet als getal geven – voor bv. de code van het letterteken `a` is de uitdrukking `code(a)` voldoende.

- Decodeer de volgende met LZ77 gecodeerde tekst. De sliding window heeft grootte 10 en de voorstelling van de posities en lengten is zo als in de les. Toon voldoende tussenstappen en details om te kunnen zien wat je doet.

  `(0,0,t)(0,0,o)(2,1,n)(0,0,_)(0,0,B)(2,3,e)(4,2,d)(7,1,_)(0,0,l)(1,3,EOF)`

5. **(2 pt)**

   - Voeg de sleutels $10, 20, 30, 40, 25, 5, 15, 1, 2, 7, 9, 11, 6, 8$ in deze volgorde aan een B+-tree met grootte 4 toe. Toon voldoende tussenstappen en details om te kunnen zien wat je doet maar het is niet noodzakelijk de stappen te tonen waar een sleutel alleen maar op een vrije plaats in een blad geschreven moet worden.

- Ontwikkel een formule voor de maximale diepte van een B+-tree met s records. Let op: je kan niet gewoon de formule voor B-trees uit de les nemen omdat de interne sleutels in een B+-tree niet naar records leiden! Maar je mag de formule wel gebruiken als je dat helpt!

(3 pt)

Een lesrooster moet opgesteld worden. Er zijn vakken $v_1, \ldots, v_n$ die gegeven moeten worden, maar sommige vakken mogen niet tegelijk gegeven worden omdat ze ofwel door dezelfde leraar gegeven moeten worden ofwel dezelfde studenten eraan moeten deelnemen. Het doel is een mogelijk lesrooster van minimale totale lengte te vinden, dus aan elke les een termijn uit een verzameling $t_1, \ldots, t_k$ toe te kennen zodat k minimaal is en nooit twee lessen dezelfde termijn krijgen die niet samen gegeven mogen worden.

- Een gretig algoritme zou b.v. als volgt werken:
  - Ken l_1 termijn t_1 toe.
  - Herhaal tot alle lessen termijnen hebben: als aan $l_1, \ldots, l_i$ al termijnen toegekend zijn, ken aan l_{i+1} de termijn toe met de kleinste index waar het nog gegeven kan worden (dat kan ook een nieuwe termijn zijn).

  Geef een voorbeeld dat toont dat dit algoritme niet altijd een optimale oplossing vindt.

- Geef een variable neighbourhood algoritme voor dit probleem met ten minste 3 buurfuncties. Geef alle definities en parameters die nodig zijn om het algoritme volledig te beschrijven maar het is niet nodig dat ze ook zo gekozen zijn dat het algoritme bijzonder goed werkt. Het algoritme moet alleen maar *in principe* werken en het optimum kunnen vinden (ook al zou dat veel te lang duren en te veel pogingen vragen).

  De bedoeling is gewoon om aan te tonen dat het principe verstaan is.

6. **(1 pt)**

   Stel voor de volgende oefening:

   a.) De publieke sleutel voor RSA encryptie is een heel groot toevallig gegenereerd priemgetal.

   b.) Je hebt een programma dat voor een gegeven IP-adres heel snel één toevallige publieke sleutel op deze computer kan vinden (als er één is). Stel voor deze oefening dat op elke IP-adres precies één publieke sleutel voor RSA encryptie staat.

   c.) Je hebt een lijst van alle bereikbare computers met IP-adressen.

   Nu wil je weten wat de grootste publieke sleutel op één van deze machines is – maar de lijst is veel te lang om alle machines te testen. De bedoeling is dus met een gegeven aantal pogingen een machine met een zo groot mogelijk sleutel te vinden.

   Welk algoritme zou je hiervoor toepassen? Het is **niet** noodzakelijk het algoritme met alle details te beschrijven. Het is voldoende te zeggen welk algoritme je zou toepassen en argumenten te geven waarom dat volgens jou de beste keuze is.

7. **(2 pt)**

Stel dat je een rij van getallen moet sorteren.

- Wanneer pas je het geziene *extern sorteren* algoritme toe en wanneer kies je voor *mergesort* of *quicksort*?

- Beschrijf het extern sorteren algoritme precies. Zeg ook op welke manier je beslist in hoeveel delen de rij wordt gesplitst en hoe je *grootste elementen kiest*.

- Wat is de asymptotische complexiteit van *extern sorteren*, *mergesort* of *quicksort*? (De $O()$-notatie is voldoende.) Welk effect heeft de grootte van het geheugen daarbij?

8. **(2 pt)**

- Hoe duur is een reeks van n toevoegbewerkingen aan een linear hashing tabel in het slechtste geval? De $O()$ notatie is voldoende maar geef uitleg waarom jouw antwoord juist is.

- Stel dat je een *goede* hash-functie hebt die voor een gegeven inputreeks een gelijkmatige verdeling van de hashwaarden geeft – ook als je niet alle bits gebruikt maar voor een arbitrair k alleen de eerste k bits of de laatste k bits.
  - Verwacht je voor extensible hashing dat in alle emmers ongeveer even veel sleutels zitten? Geef uitleg.

  - Verwacht je voor linear hashing dat in alle emmers ongeveer even veel sleutels zitten? Geef uitleg.

- Voeg sleutels 3, 19, 44, 34, 66 en 2 in de gegeven volgorde toe aan een linear hashing tabel met laadfactor maximaal 0.6 waarbij je in het begin maar 1 (lege) emmer gebruikt. Er mogen twee sleutels in één emmer zitten. Het is hier voldoende de hash-waarden in de tabel te schrijven in plaats van de sleutels. Toon voldoende tussenstappen om te zien wat er gebeurt en schrijf altijd expliciet welke emmer herverdeeld moet worden. De hash-waarden zijn (binair):

  $h(3) = 0$ dus als binair getal zonder leidende nullen 0

  $h(19) = 60$ dus binair 111100

  $h(44) = 22$ dus binair 10110

  $h(34) = 63$ dus binair 111111

  $h(66) = 42$ dus binair 101010

  $h(2) = 44$ dus binair 101100

NOG NIET OMDRAAIEN !