Examen Algoritmen en Datastructuren III

Naam : .....

Lees de hele oefening zorgvuldig voordat je begint ze op te lossen!
Als je niet goed verstaat wat de vraag of taak is, vraag het aan de lesgever!
Voor oefeningen die fout verstaan zijn, kunnen geen punten gegeven worden!

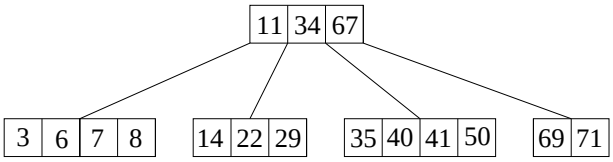
Schrijf leesbaar. Oplossingen die niet leesbaar zijn, kunnen ook niet beoordeeld worden.

Als technieken toegepast moeten worden, toon altijd voldoende tussenstappen om te kunnen zien wat er gebeurt en dat de technieken goed verstaan zijn.

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. **B-trees en B+-trees (2 pt)**

   - Voeg de sleutels 9 en 42 toe aan de volgende B-tree met grootte 4.



- Geldt de volgende uitspraak voor elke B+-tree:
  Een sleutel s die in een interne top van een B+-tree staat, staat ook in een blad.
  Geef uitleg.

- Als je in een B+-tree voor een interne top vaststelt dat die gelijk is aan de sleutel die je wil opzoeken dan kan je zonder nog te moeten vergelijken de positie in het blad opzoeken waar de sleutel zou moeten staan.

  Beschrijf hoe je dat doet en geef uitleg – gebaseerd op de definitie van een B+-tree – waarom jouw manier van doen juist is.

2. **String matching (4 pt)**

- Voor het algoritme van Rabin-Karp is een hashfunctie nodig die aan een string een hashwaarde toekent. Welke eis is belangrijk voor deze hashfunctie die niet geeist moet worden voor hashfuncties van tabellen? Geef uitleg.

- Pas het algoritme van Knuth-Morris-Pratt toe om de string `barbara` in `barbaren_barbecuen_barbara` te zoeken. Toon daarbij niet alleen voor het zoeken maar ook voor het opstellen van de verschuivingstabel voldoende tussenstappen.

- In de les hebben jullie het shift-AND algoritme gezien en in de oefeningen de shift-OR versie van het algoritme. Wat is het verschil? Welke versie zou je kiezen als je een bijzonder efficiënt programma wil schrijven. Geef uitleg! (Een antwoord zonder juist uitleg telt niet mee.)

- Pas het shift-AND algoritme toe om de string `fiets` in `fiere_fietser` te zoeken.

3. **Compressie (3 pt)**

    - Comprimeer de tekst `barbarenbarbara` met LZW.

- De volgende code is het resultaat van LZ77 waarbij de codering zoals in de les en een sliding window van grootte 8 werd gebruikt:
  `(0,0,")(0,0,g)(0,0,i)(0,0,s)(4,1,e)(0,0,n)(0,0,_)(3,2,_)(0,0,m)(5,2,")(1,7,EOF)`
  Decodeer deze code.

- In de les werd gezegd dat je het resultaat van LZ77 ook nog met het Huffman algoritme kan coderen waarbij je zelfs meer dan één Huffman boom gebruikt. Geef uitleg waarom dat mogelijk is en wat het voordeel is.

- In de les hebben wij gezien dat als je arbitraire bestanden zonder eisen aan de structuur bekijkt de theoretische bovengrens voor het aantal bestanden dat je kan comprimeren ongeveer $1/255$-de is. Stel nu dat wij alleen kijken naar bestanden met n bytes die het resultaat van een Burrows-Wheeler transformatie zijn. Daarbij tellen wij alleen de bytes die de tekst beschrijven en niet de eerste bytes die de positie van het eerste teken beschrijven. Wat is een goede bovengrens voor het aantal van deze bestanden dat gecomprimeerd kan worden. Geef uitleg.

4. **Hashing (2 pt)**

   - Voeg de sleutels met de volgende binaire hashwaarden in de gegeven volgorde toe aan een extensible hashing tabel met 2 records per emmer. In het begin heeft de tabel maar twee emmers en gebruik je maar 1 bit van de hashwaarden. Omdat het hier – net zoals in de les – voldoende is de hashwaarden in te vullen worden de echte sleutels niet gegeven.

     De in te vullen hashwaarden zijn
     `00011`, `00100`, `01111`, `10000`, `10111`, `00101`, `11111`.

- Geef één belangrijk voordeel van linear hashing in vergelijking met extensible hashing en één belangrijk voordeel van extensible hashing in vergelijking met linear hashing.

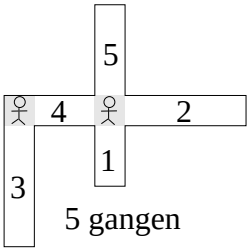
- Geef voor extensible hashing de geamortiseerde kost van een reeks van n toevoegbewerkingen in het slechtste geval (dus als de hashfunctie niet goed is) en als de hashfunctie wel goed is (je mag bijvoorbeeld stellen dat het verschil tussen het minimaal en maximaal aantal sleutels met de eerste bits een index in de pointerarray ten hoogste een factor van 3 is). Geef uitleg, maar een echt bewijs dat jouw antwoorden juist zijn is niet vereist.

5. **Metaheuristieken (4 pt)**

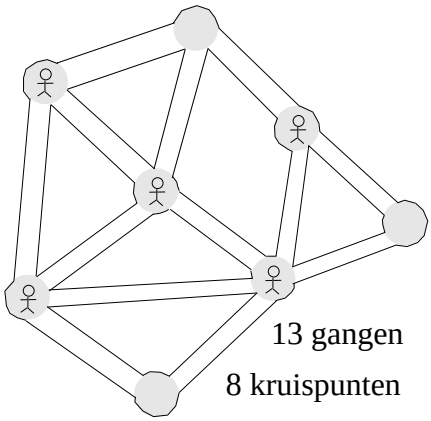
In dit deel is het niet de bedoeling bijzonder efficiënte algoritmen te ontwikkelen. Het is voldoende alle parameters en constructies te beschrijven en het basisalgoritme te geven om aan te tonen dat de metaheuristieken goed verstaan zijn.

Het probleem waarvoor heuristieken ontwikkeld moeten worden, is een probleem dat jullie al uit DA2 kennen:

Probleem: In een museum zijn er verschillende gangen die bewaakt moeten worden. De gangen zijn allemaal recht en relatief kort zodat iemand die op het einde van een gang staat de hele gang kan zien en snel kan bereiken. Het eindpunt van een gang kan tegelijk het eindpunt van andere gangen zijn – dat noemen wij dan een kruispunt. Wij stellen dat er ten minste 2 gangen zijn en dat elke gang aan ten minste één kruispunt grenst.



5 gangen
2 kruispunten
4 eindpunten die
geen kruispunten zijn



13 gangen
8 kruispunten

De taak is nu een manier te vinden om bewakers op de eindpunten van de gangen te plaatsen zodat het aantal bewakers zo klein mogelijk is, maar elke gang door ten minste één bewaker gezien en snel bereikt kan worden.

- Beschrijf een op Variable Neighbourhood Descent gebaseerd algoritme voor dit probleem. Drie buurfuncties zijn daarbij voldoende.

- Beschrijf een genetisch algoritme voor dit probleem.

6. **Verdeelde algoritmen (1 pt)**

   De pseudocode op de volgende pagina beschrijft een verdeeld algoritme voor het museumprobleem van de vorige oefening. Het wordt verdeeld in 100 delen die gekenmerkt zijn door de waarden $0, \ldots, 99$ voor de variabele `mijn_deel`. Jammer genoeg zit er een fout in de pseudocode. Beschrijf precies wat de fout is en wat het effect van de fout is.

Wij beschrijven het probleem als een graaf $G = (V,E)$ waarbij de toppen de eindpunten van de gangen zijn en de gangen de bogen. Wij schrijven altijd `|M|` voor het aantal elementen in een verzameling `M`. De namen van de toppen zijn $1,\ldots,|V|$. Wij hebben de waarde van de variabele `splitlevel`> 1 op voorhand vastgelegd. De variabele `best` wordt globaal voor alle delen op $|V|$ geïnitialiseerd en als `best` door één deelprogramma gewijzigd wordt, hebben alle delen de nieuwe waarde. Het resultaat is wat na afloop van `museum(0)`,`museum(1)`,...,`museum(99)` in `best` staat.

```
 int look_ahead(bew_toppen,bew_gangen)
// geeft een benedengrens terug voor het aantal toppen
// dat nog nodig is
{ return ((|E|-|bew_gangen|)/grootste_graad(G)); }
```

```
 void recursie(bew_toppen,bew_gangen,top,mijn_deel)
 // hier wordt beslist of top bewaakt wordt
 // de parameters worden als kopie doorgegeven
 {
   if (top=splitlevel)
    { counter++;
      if ((counter mod 100) != mijn_deel) return;
    }

   if (top=|V|+1) // over alle toppen beslist
     { if ((|bew_gangen| =|E|) && (|bew_toppen|<best))
                                        best=|bew_toppen|;
       return;
     }

   if (|bew_toppen|+look_ahead(bew_toppen,bew_gangen)>=best)
        return; // kan niet meer beter worden

   // eerst top niet toevoegen
   recursie(bew_toppen,bew_gangen,top+1,mijn_deel);
   // dan wel toevoegen:
   bew_toppen =  bew_toppen ∪ {top}
   bew_gangen =  bew_gangen ∪ {bogen die top bevatten}
   recursie(bew_toppen,bew_gangen,top+1,mijn_deel);
   return;
 }
```

```
 void museum(mijn_deel)
 { bew_toppen=leeg; // de verzameling van bewaakte eindpunten
   bew_gangen=leeg; // de verzameling van bewaakte gangen

   counter=-1; // een globale variabele maar voor elk
               // deelprogramma apart

   recursie(bew_toppen,bew_gangen,1,mijn_deel);
   return;
  }
```

NOG NIET OMDRAAIEN !