Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Filip Strömbäck och Magnus Nielsen

22 augusti 2023

TDDI16
Datastrukturer och algoritmer
Datortentamen (DAT1)
2023-08-22, 8–12

Examinator:	Filip Strömbäck
Jour:	Filip Strömbäck (telefon 013-28 27 52)
Antal uppgifter:	6
Max poäng:	40 poäng
Preliminära gränser:	Betyg 5 = 35p, 4 = 27p, 3 = 20p.
Hjälpmedel:	Inga hjälpmedel tillåtna!

VÄNLIGEN IAKTTAG FÖLJANDE

- Observera att betygsgränserna kan komma att justeras, i samtliga kurser.
- Vid frågor om tidskomplexitet, svara alltid på den form som är mest relevant.
- Skriv dina lösningar i valfri textredigerare, exempelvis LibreOffice, Emacs, eller liknande. Strukturera din text så att det enkelt går att se vad som svarar på vilka deluppgifter.
- Lösningar till olika problem skall skrivas i en egen fil. Skriv inte två lösningar i samma fil. Delproblem får dela fil.
- Om inte annat framgår ska indexering av arrayer/listor börja från 0.
- Skicka in som lösning till rätt problem i tentaklienten, och döp filen till ett passande namn (exempelvis uppg1.txt/uppg1.odt).
- **MOTIVERA DINA SVAR ORDENTLIGT:** avsaknad av, eller otillräckliga, förklaringar resulterar i poängavdrag. **Även felaktiga svar kan ge poäng** om de är korrekt motiverade.
- Om ett problem medger flera olika lösningar, t.ex. algoritmer med olika tidskomplexitet, ger endast optimala lösningar maximalt antal poäng.
- SE TILL ATT DINA LÖSNINGAR/SVAR ÄR LÄSLIGA. Oläsliga lösningar beaktas ej.

Lycka till!

1. **Hashtabeller** (5 p)

Vi har en hashtabell med linjär adressering och några element instoppade. Hashfunktionen är $h(x) = x$ mod size (arrayindex står under arrayen för tydlighets skull).

Null	11	22	31	21	41	4	17	24	Null
0	1	2	3	4	5	6	7	8	9

(a) I vilken ordning har elementen stoppats in? Det finns flera lösningar. Ge fyra korrekta lösningar för maxpoäng. (2)

(b) Om vi tar bort 11 från den delvis fyllda hashtabellen ovan (remove / delete), hur kommer tabellen se ut? Det finns flera olika lösningar. Ge två olika lösningar som inte hashar om hela tabellen utom möjligen i värsta fallet, och förklara dem, för maxpoäng. (2)

(c) Ytterligare fyra element sätts in i hashtabellen ovan (exempelvis 58, 12, 29 och 81). Beskriv kort (2–3 meningar) hur implementationen gör för att få plats med dessa element. Hashtabellen ska fortfarande använda linjär adressering. (1)

2. **Sorteringsalgoritmer** (4 p)

Svaren behöver ej motiveras.

Efter **ett fåtal** iterationer av några olika sorteringsalgoritmer på den osorterade arrayen *Original* i tabellen nedan (iteration i bemärkelse fullständig körning av inre loop, alternativt rekursivt anrop) har vi resultaten i *1, 2, 3* och *4*.

Original:	63	79	32	96	0	58	46	55	57	83	51	47	41	93	29
1:	32	63	79	96	0	58	46	55	57	83	51	47	41	93	29
2:	0	29	32	96	63	58	46	55	57	83	51	47	41	93	79
3:	79	63	58	57	51	47	46	55	32	0	41	29	83	93	96
4:	0	29	32	41	63	58	46	55	57	47	51	79	96	93	83
Sorterad:	0	29	32	41	46	47	51	55	57	58	63	79	83	93	96

Matcha delvis sorterad array mot en algoritm. Felaktig matchning ger minuspoäng, dock kan uppgiften ej ge total minuspoäng. Utelämnat svar ger ej minuspoäng. För full poäng krävs 4 korrekta svar.

(a) Quicksort, elementet längst till höger i partitionen används som pivot

(b) Heapsort

(c) Insertionsort

(d) Selectionsort

3. **Kulstötning** (10 p)

En av dina klasskamrater har på senare tid blivit intresserad av kompetetiv kul(l)stötning. Som den del i detta finns en stor önskan av att kunna jämföra sig med andra kul(l)stötare. Din klasskamrat vill därför ha ett program med en datastruktur som har följande två operationer:

`create(x)` Skapa och initiera datastrukturen. x är en array med nuvarande rekord för kulstötare. Varje element representerar en kulstötares bästa stöt. Varje element består av ett par (l, n) där l är längden av rekordstöten i form av ett heltal mätt i centimeter, och n är namnet på kulstötaren.

`find_closest(d)` Hittar det närmsta kastet i datastrukturen, givet längden på en stötning från din kompis (d). Funktionen ska returnera ett par (l, n) som representerar det närmsta rekordet. Likt i indata är l är längden på stöten, och n är namnet på kulstötaren.

Efter lite funderande så kommer du fram till följande möjliga implementationer av datastrukturen:

1: Lagra alla par i en hashtabell. Längden är nyckeln i hashtabellen, och värdet är namnet på stötaren. `create` kan då implementeras genom att helt enkelt sätta in alla par i hashtabellen. `find_closest` kan sedan implementeras genom att leta först efter stötar med en skillnad av 0 cm i avstånd, sedan en skillnad på 1 cm i avstånd, och så vidare tills ett resultat hittas.

   Alltså: för heltal n från 0 och uppåt så letar vi efter kasten $d-n$ och $d+n$ tills vi hittar en match.

2: Lagra alla par i en array. `create` kan då implementeras genom att helt enkelt kopiera parametern till en ny array. `find_closest` implementeras sedan genom at iterera igenom arrayen och hålla koll på skillnaden mellan d och det element som varit närmast hittills. När hela arrayen har traverserats har vi då den närmaste stöten.

3: Lagra alla par i en sorterad array. `create` kopierar då arrayen, och sorterar den sedan med hjälp av quicksort. `find_closest` implementeras sedan genom att först söka fram den stöt som är närmast större än d med hjälp av binärsökning. Sedan returneras det element som hittades, eller det precis innan, beroende på vilket som var närmast.

Svara på följande frågor med avseende på de tre alternativen ovan:

(a) Vilken tidskomplexitet har vart och ett av alternativen 1–3 ovan för att skapa datastrukturen (dvs. `create`)? Uttryck tidskomplexiteten i storleken av indata (n). Beskriv kort ditt resonemang. (3)

(b) Vilken tidskomplexitet har vart och ett av alternativen 1–3 ovan för att hitta den närmsta stöten (dvs. `find_closest`)? Uttryck tidskomplexiteten i storleken av datamängden (n). Beskriv kort ditt resonemang. (3)

   Tips: Vid behov kan du även använda m för längden på den längsta stöten i cm.

(c) Vilken av alternativen 1–3 är bäst i fallet då programmet körs en längre tid (dvs. man startar programmet en gång, sen gör många frågor)? Är samma alternativ bäst om man startar programmet, och sedan bara ställer en fråga? Motivera ditt svar. (2)

(d) Ett av alternativen 1–3 ovan tappar bort resultat i vissa fall. Vilket av alternativen gör detta, och vad är problemet? (2)

4. **Algoritmer och tidskomplexitet** (6 p)

Visa kort hur du kommer fram till tidskomplexiteten på följande uppgifter.

(a) Beräkna tidskomplexiten med avseende på parametern n för följande funktion: (1)

```
int f(int n) {
  int result = 0;
  for (int i = 1; i < n; i = i * 2) {
    result *= i;
  }
  return result;
}
```

(b) Beräkna tidskomplexiten med avseende på parametern n för följande funktion: (1)

```
int fn(int n) {
  int result = 0;
  for (int i = 1; i < n; i++) {
    result += i;
  }
  result += f(n);
  return result;
}
```

(c) Beräkna tidskomplexiten med avseende på parametrarna n och x för följande funktion: (2)

```
int fun(int n, int x) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < x*n; j++) {
      sum += i;
    }
  }
  return sum;
}
```

(d) Beräkna tidskomplexiten med avseende på storleken av x för följande funktion. Antag att `LinkedList` är en dubbellänkad lista. Metoden `first` hämtar första elementet. Metoden `insertFirst` sätter in ett element i början av listan. Metoden `find` letar efter ett värde i listan och returnerar dess index (eller -1 om elementet inte finns). Tänk på bästa- och värstafall i din analys av funktionen `function`. Beskriv också kortfattat hur du kom fram till ditt svar. (2)

```
int function(LinkedList<int> x) {
  int oldFirst = x.first();
  if (oldFirst > 200) {
    x.insertFirst(10);
  }
  return x.find(10);
}
```

5. **Ett par fiender** (5 p)

Du har blivit inhyrd som konsult på det lokala mobilspelsföretaget. De håller på att utveckla nästa stora PTW-spel (Pay To Win). Det som gör just det här spelet unikt är att man alltid möter fiender i par. För att balansera spelet så ska spelet generera ett par fiender som matchar spelarens styrka precis lagom. Detta så att spelaren måste spendera maximalt med virtuell valuta för att kunna besegra fienderna. Det är dock viktigt att fienderna inte är för bra så att spelaren ger upp (inkomsten från spelaren riskerar därmed att gå mot noll...).

I spelet så finns en stor lista av fiender (det finns tusentals fiender, alla med en helt egen nyans av färgen grön för variation). Mest intressant är att varje fiende har ett värde som beskriver dess Sensible Monster Skill (SMS). I varje runda av spelet så måste spelet hitta två fiender, vars summerade SMS-poäng är så stor som möjligt, men inte större än den angivna Desired Player Skill (DPS)-poängen.

Din uppgift som konsult är att implementera en lämplig algoritm för att hitta dessa par av fiender. I och med att detta är en central del av spelet är det viktigt att lösningen är så effektiv som möjligt! För att göra detta finns två funktioner som anropas av resten av spelet:

`initialize()` Denna funktion anropas när spelet startas. Här finns en möjlighet att skapa eventuella datastrukturer som din lösning behöver, samt populera den med fiender. Innan funktionen körs så har alla fiender i spelet lästs in i en global variabel. Funktionen kan fritt läsa från (men inte modifiera) listan som lagras i denna globala variabel för att initiera de datastrukturer som behövs för att göra `find_pair` så effektiv som möjligt.

`find_pair(DPS)` Denna funktion anropas för att hitta ett par (=2) fiender vars summerade SMS-poäng är så stor som möjligt, utan att vara större än den inskickade DPS-poängen.

Notera: I och med att `find_pair(DPS)` kommer att köras ofta är det viktigt att denna gör så lite arbete som möjligt. Om det går att göra delar av arbetet i `initialize()` är det alltså att föredra.

(a) Beskriv hur du kan implementera `initialize()` och `find_pair(DPS)` ovan. (3)

Beskriv din algoritm, gärna i punktform eller med pseudokod. Tänk på att ha med hur du lagrar data och vilka datastrukturer din algoritm använder, och hur de initieras i `initialize()` om så behövs. Du behöver inte detaljbeskriva sådant som finns i standardbiblioteket.

(b) Vilken tidskomplexitet har din implementation av `initialize()`, uttryckt i antal fiender i systemet (n)? (1)

(c) Vilken tidskomplexitet har din implementation av `find_pair(DPS)`, uttryckt i antal fiender i systemet (n)? (1)

6. **Banala banverifikationer** (10 p)

När du löste problemet i uppgift 5 så blev företaget väldigt imponerade med dina kunskaper, och gav dig en svårare uppgift. En del av spelet innehåller en 2-dimensionell värld där spelaren kan vandra mellan olika platser (på olika platser så har fiender olika utseenden, inte bara olika nyanser av grönt). Utvecklarna av spelet har dock ett problem. Med jämna mellanrum råkar de som designar världen blockera vägar mellan platser, vilket har gjort det omöjligt att ta sig till vissa delar av världen. Mer än en gång har inte detta upptäckts i tid, så företaget ber nu dig att bygga ett verktyg som snabbt kan kontrollera ifall problemet har uppstått.

Världen består alltså av ett 2-dimensionellt rutnät. Varje ruta kan innehålla tre saker: en vägg, en plats (där man kan hitta fiender), eller så kan den vara tom. Allt utom väggar är passerbart av spelaren.

Uppgiften är alltså att se om det går att ta sig från platsen där spelaren startar till alla andra platser. Världen är lagrad som en 2-dimensionell array. Varje punkt i arrayen är märkt med 0 om rutan är tom, -1 om rutan är en plats, och 2 om den är en vägg. Koordinaterna till startplatsen skickas tillsammans med antalet platser som parametrar till den funktion du ska skriva för att kontrollera framkomligheten.

(a) Beskriv hur du kan implementera en funktion som kontrollerar att det går att ta sig fram till alla platser i spelet. Tänk på att din lösning ska vara så effektiv som möjligt, så att de världsdesigners som arbetar med världen inte behöver vänta i onödan! (4)

Beskriv din algoritm, gärna i punktform eller med pseudokod. Tänk på att ha med hur du lagrar data och vilka datastrukturer din algoritm använder, och hur de initieras i `initialize()` om så behövs. Du behöver inte detaljbeskriva sådant som finns i standardbiblioteket.

(b) Vilken tidskomplexitet har din implementation i (a), uttryckt i antal rutor i världen (n)? (1)

(c) Efter lite testande inser företaget att det inte räcker att kontrollera framkomlighet från startplatsen. I vissa fall har de råkat blockera den snabbaste vägen till en plats, så att spelaren måste ta en stor omväg runt hela världen för att komma dit. Därför vill de nu också verifiera att vägen mellan startplatsen och alla platser i världen är tillräckligt kort! (4)

Utöver detta har de också implementerat olika terränger i spelet. Varje ruta innehåller fortfarande -1 för att representera platser. Väggar representeras nu av $+\infty$, och alla tal $0 < t < \infty$ representerar hur lång tid (i sekunder) det tar för spelaren att passera rutan.

Företaget har nu bett dig att revidera din algoritm så att den 1) tar hänsyn till terrängen, och 2) beräknar hur lång tid det tar för spelaren att komma från startplatsen till alla andra platser, så att de kan se om de har råkat förlänga vägen till dem för mycket.

Beskriv din algoritm, gärna i punktform eller med pseudokod. Tänk på att ha med hur du lagrar data och vilka datastrukturer din algoritm använder, och hur de initieras i `initialize()` om så behövs. Du behöver inte detaljbeskriva sådant som finns i standardbiblioteket.

(d) Vilken tidskomplexitet har din implementation i (c), uttryckt i antal rutor i världen (n)? (1)