Kapitel 14

Tentamenspreparering

Uppgift 1 – Tribonacci

Som en utveckling av *Fibonacci talföljd* presenterar vi här *Tribonacci talföljd*, som definieras genom

$$t_n = t_{n-1} + t_{n-2} + t_{n-3}$$

där $t_0 = 0$, $t_1 = 0$ och $t_2 = 1$.

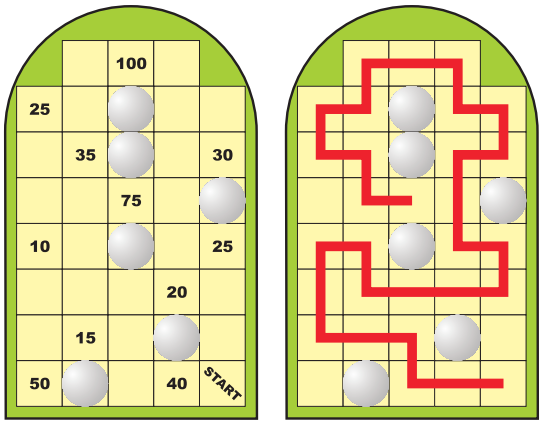
Skriv ett program som listar de 39 första talen i Tribonacci talföljd, numrerade från 0 till 38. Inget, utom de tre första talen, får finnas 'hårdkodat' i programmet.

Med enkel rekursion kommer programmet att kunna lista de 30 till 32 första talen inom en sekund. Därefter ökar exekveringstiden dramatiskt. För större tal krävs en speciell idé i programmet, som kommer att skriva ut samtliga önskade tal på mindre än en sekund.

För ett program som listar de 30 första talen på cirka en sekund får du 1 **poäng**. För ett program som listar alla önskade tal på cirka 1 sekund får du 2 **poäng**. Körningsexempel med några tal utskrivna:

```
t(0) =0
t(1) =0
t(2) =1
...
t(16)=3136
...
t(28)=4700770
...
t(38)=2082876103
```

Uppgift 2 – Flipperspel



Figur 14.1: *Till vänster ett flipperspel med hinder och poäng. Till höger den optimala sättet att rulla kulan i detta exempel.*

Här gäller det att samla så många poäng som möjligt genom att låta kulan rulla på ett listigt sätt genom flipperspelet.

Spelets plan består alltid av 8×5 rutor. Spelet startas genom att placera en kula längst ned i högra hörnet. Kulan kan sedan flyttas, till en annan gul ruta i någon av de fyra riktningarna: *upp, ned, vänster* eller *höger*. Detta under förutsättning att rutan inte är markerad med ett runt hinder eller att kulan redan besökt denna ruta. En del rutor har markerats med tal som anger hur många poäng man får, när kulan besöker denna ruta.

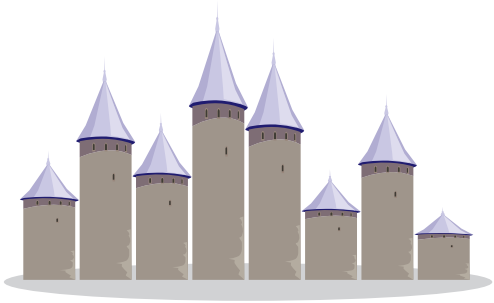
Skriv ett program som bestämmer den största poäng man kan få för given spelplan. På filen `flipper.txt` beskrivs spelplanen enligt

- Filen består av 8 rader med 5 tal på varje.
- Talet 0 innebär *tom gul ruta*
- Talet -1 innebär att rutan innehåller ett hinder.
- Andra tal, alla positiva, anger rutans värde i poäng

En testkörning:

```
Maxpoäng = 375
```

Uppgift 3 – Torn



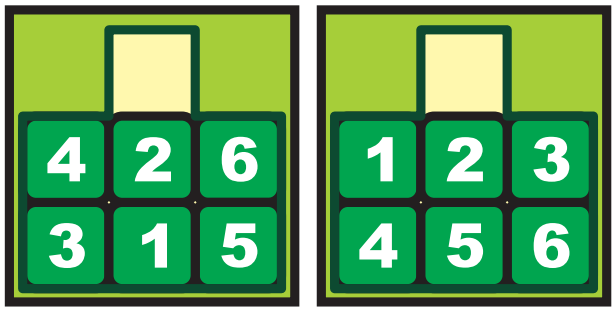
Figur 14.2:

I figur 14.2 ser vi 8 torn, alla med olika höjd. Ställer man sig *till vänster* om tornen i exemplet och betraktar raden, ser man 3 torn. De övriga är skymda av högre torn. Ställer man sig *till höger* ser man 4 torn.

Skriv ett program som tar emot uppgift om det *totala antalet* torn $t, 1 \leq t \leq 11$ i raden, samt hur många torn man ser från höger respektive vänster sida och som bestämmer hur många *arrangemang* av de t tornen det finns, som översensstämmer med dessa data. En testkörning:

```
Antal torn       ? 8
Ses från vänster ? 3
Ses från höger   ? 4
Det finns 1750 möjligheter att ställa upp tornen
```

Uppgift 4 – Pussel



Figur 14.3:

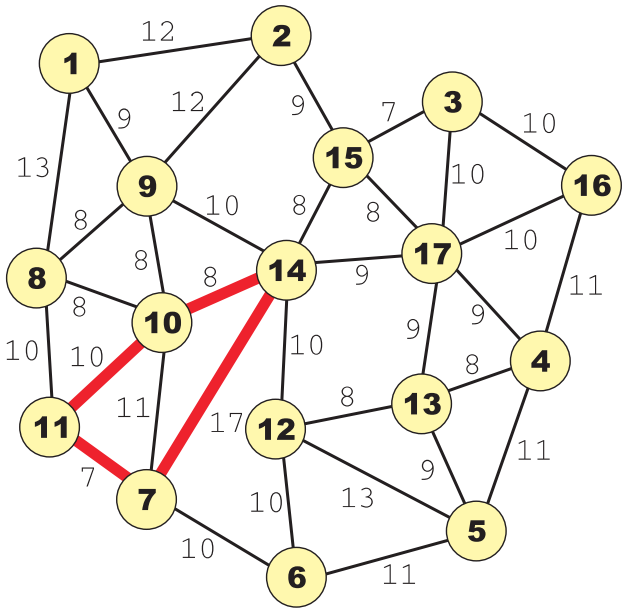
Till vänster i figur 14.3 ser du ett pussel med 6 brickor numrerade $1 \ldots 6$. Dessutom finns från början, i övre raden, en tom plats (gul) som rymmer en bricka. I exemplet kan brickan med nummer 2 skjutas till den tomma platsen. I nästa drag kan brickorna med nummer $4, 1, 6, 2$ skjutas till den nu uppkomna tomma platsen.

Pusslets mål är att, utifrån en given placering av de 6 brickorna ordna dem, som till höger i figuren, med så få drag som möjligt.

Skriv ett program som läser in aktuell placering från filen `pussel.txt` och bestämmer det minsta antalet drag som krävs för att nå målet. Filen innehåller två rader med tre tal på varje. Körningsexempel:

```
Antal drag = 10
```

Uppgift 5 – Maratonlopp



Figur 14.4: *Kartan över området som ska arrangera ett maratonlopp*

I det område där man planerar ett maratonlopp finns ett antal byar, se kartan i figur 14.4. Förutom de numrerade byarna finns vägar som förbinder byarna. På kartan anges längden, i kilometer, hos dessa vägar. Man önskar nu ta reda på hur många möjligheter det finns att lägga ut en maratonbana på 42 km (återstående 195 meter kommer att ordnas med ett extra varav runt torget). Banan ska starta och sluta i angiven by.

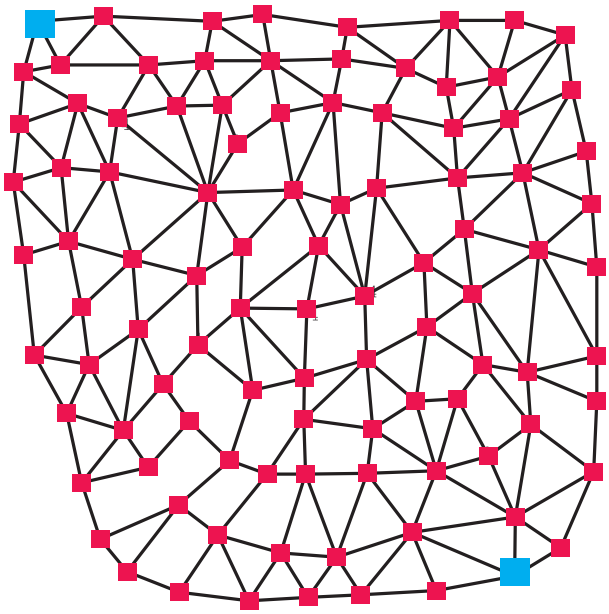
Skriv ett program som från filen `maraton.txt` läser in information om kartan, tar emot numret på en by och bestämmer hur många olika martonbanor man kan ordna med start och mål i denna by.

Första raden i filen anger antalet byar $b, 3 \leq b \leq 100$. Andra raden innehåller antalet vägar $v, 3 \leq v \leq 500$. Därefter följer v rader med tre tal på varje: *från by nummer, till by nummer* och *vägens längd*.

```
I vilken stad ska loppet starta ? 10
Det finns 2 olika banor
```

Observera att det alltid finns ett jämnt antal banor, eftersom en bana kan löpas antingen med- eller moturs.

Uppgift 6 – Jan Ersa och Per Persa



Figur 14.5:

Inledningen av en dikt av Gustaf Fröding:

> Jan Ersa ägde Nackabyn,
> Per Persa ägde Backabyn
> i By i Västra Ed.
> Jan Ersa,
> Per Persa,
> de höllo aldrig fred.

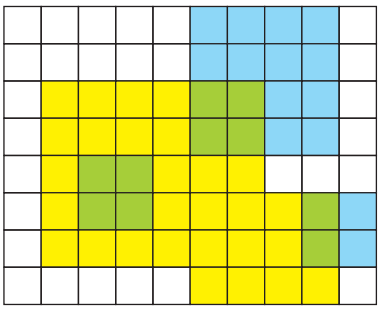
berättar om Jan och Per som inte trivdes tillsammans. När de därför, på äldre dagar, flyttade in till staden såg de till att de hamnade så långt från varandra som möjligt. Kartan i figur 14.5 visar stadens alla hus och vägar. Vi ser också de blå husen där Jan Ersa och Per Persa numera bor. Dessa två hus är de hus i staden som ligger längst ifrån varandra.

Skriv ett program som läser in information om kartan från filen `karta.txt` och som sedan bestämmer mellan vilka två hus det kortaste avståndet är som längst, samt detta avstånd. Filen inleds med ett tal $h, 2 \leq h \leq 100$, som anger antalet hus i staden. På nästa rad återfinns ett tal som anger antalet vägar $v, 2 \leq v \leq 500$. Därefter följer v rader med tre tal på varje: *från hus nummer, till hus nummer* och *vägens längd* (givet i 100-tals meter).

Ett körningsexempel:

```
Mellan hus 1 och 17 är längsta kortaste avståndet 5300 m
```

Uppgift 1 – Färgblandning



Figur 14.6:

Som bekant blir det *grönt* när man blandar *blått* och *gult*. Skriv ett program som 'målar' blå och gula rektanglar i första kvadranten, begränsad av $(0 \leq x, y \leq 24)$, och skriver ut tavlan där de delar av en eller flera gula rektanglar som helt eller delvis täcks av en eller flera blåa rektanglar blir gröna.

På filen `rektanglar.txt` finns information om rektanglarnas utsträckning och färg. Här ett exempel:

```
5
1 1 6  7 1
2 2 4  4 2
2 5 8  9 4
1 5 3  9 0
2 8 3 10 1
```

Förklaring: Det är 5 rektanglar som ska målas. På det efterföljande 5 raderna beskrivs dessa. Första talet anger färgen (1=gult, 2=blått). De två efterföljande talen anger (x_1, y_1) för rektangelns *övre vänstra hörn*. De två sista talen anger (x_2, y_2) för rektangelns *nedre högra hörn*. Jämför med figur 14.6.

Ditt program ska för exemplet ge utskriften

```
.....2222.
.....2222.
.11113322.
.11113322.
.133111...
.133111132
.111111132
.....1111.
```

Du använder alltså *punkt* för vit färg (ingen täckning), 3 för grönt, 1 för gult och 2 för blått.

Uppgift 2 – Minsta triangeln

På filen `punkter.txt` finns koordinater till ett antal punkter i första kvadranten $0 \leq x, y \leq 100$. Filen inleds med ett tal $n, 3 \leq n \leq 100$, som anger antalet punkter. Därefter följer n rader med två tal på varje, x- respektive y-koordinat för en punkt. Skriv ett program som bestämmer arean till den minsta triangel som kan bildas, där hörnen är tre av de givna punkterna.

För att beräkna arean till en triangel där hörnen ligger i punkterna (x_1, y_1), (x_2, y_2) och (x_3, y_3) kan man använda formeln:

$$\text{Arean} = \left| \frac{1}{2}(x_1 y_2 - x_2 y_1) + \frac{1}{2}(x_2 y_3 - x_3 y_2) + \frac{1}{2}(x_3 y_1 - x_1 y_3) \right|$$

Ett körningsexempel:

```
Den minsta triangeln har arean 6
```

Uppgift 3 – Bilda heltalskvadrater

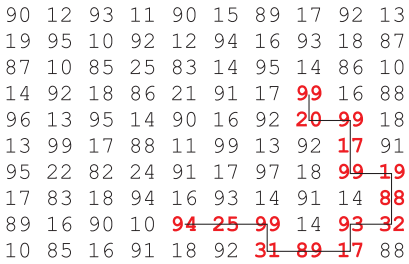
Vi utgår från siffrorna $1 \ldots 9$, som ska användas precis en gång var, för att bilda 9-siffriga heltal. Det finns $9!$ sådana tal.

Några av dessa har egenskapen att de kan 'huggas' av i *tre* delar så att varje del blir en heltalskvadrat. Några exempel

$$\begin{array}{rcll} 817569324 & \rightarrow & 81|7569|324 & = 9^2, 87^2, 18^2 \\ 125673984 & \rightarrow & 1|256|73984 & = 1^2, 16^2, 272^2 \\ 361529784 & \rightarrow & 361|529|784 & = 19^2, 23^2, 28^2 \end{array}$$

Skriv ett program som tar reda på hur många av de $9!$ sifferföljderna som har denna egenskap.

Uppgift 4 – Pathfinder



Figur 14.7:

Här gäller det att finna den största möjliga summan av 15 tal som ligger i en följd i den givna 10×10 stora tabellen. Följande regler gäller.

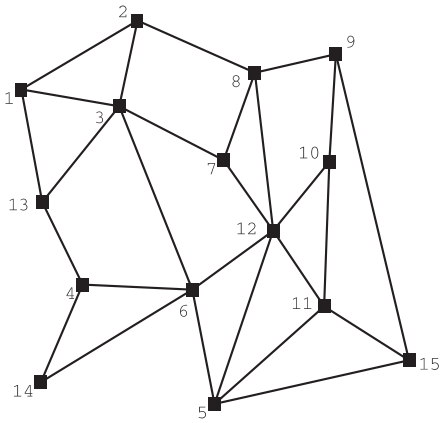
- Följden får starta i vilken ruta som helst
- Ett tal få endast användas en gång
- Från ett tal tar man sig till nästa genom gå en rad uppåt eller nedåt eller gå en kolumn åt vänster eller höger.

Skriv ett program som finner denna största möjliga summa för de tal som finns på filen `tabell.txt`, fördelade på 10 rader med 10 tal på varje. Ett körningsexempel

```
Största summan är 921
```

Det kan finnas flera följder som ger samma maximala summa.

Uppgift 5 – Kraschade maskiner



Figur 14.8:

Figur 14.8 visar ett datornätverk med 15 datorer. Vi frågar oss hur många av datorerna från nummer 2 till 14 som måste gå sönder, för att ingen kontakt ska finnas mellan maskin 1 och maskin 15 (dessa två går aldrig sönder!)

Filen `maskiner.txt` inleds med ett tal $n, 3 \leq n \leq 25$ som anger hur många maskiner det finns i nätverket. Det är då alltid mellan maskinerna 1 och n, som kontakten ska kontrolleras. På nästa rad anges antalet förbindelser m. Därefter följer m rader, med två tal på varje, som anger mellan vilka maskiner det finns direkt förbindelse.

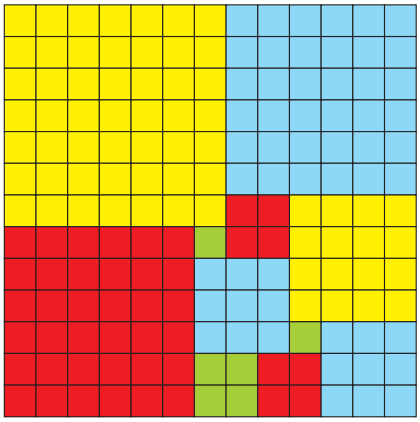
Skriv ett program som tar reda på det minsta antalet maskiner som måste krascha för att ingen kontakt ska finnas mellan maskin 1 och n.

Ett körningsexempel:

```
Det minsta antalet är 3
```

Observera att det ofta finns många olika kombinationer av kraschade maskiner som leder till detta minimum. Både kombinationen $5, 9, 11$ och $2, 3, 13$ av trasiga maskiner orsakar avbrott mellan 1 och 15.

Uppgift 6 – Kvadratpussel



Figur 14.9:

I figur 14.9 ser du en kvadrat (13×13) som är sammansatt av 11 mindre kvadrater, som också är det minsta antal 'småkvadrater' man behöver för att skapa en stor kvadrat av denna storlek.

Skriv ett program som tar emot den stora kvadratens sida $s, 2 \leq s \leq 19$ och som bestämmer det *minsta* antal mindre kvadrater som behövs för att lägga pusslet.

Ett körningsexempel:

```
Kvadratens sida ? 13
Det behövs minst 11 kvadrater
```

14.0.1 Lösningstips

Här följer några tips för hur problemen i de två tentorna kan lösas.

14.1 Tentamen 071220

Problem 1 – Tribonacci

Enklast är förstås att skapa en loop, där de centrala satserna är

```
t3=t0+t1+t2;
t0=t1;
t1=t2;
t2=t3;
```

Men att använda sig av en tabell och *memoization* går också utmärkt.

14.1.1 Problem 2 – Flipperspel

Ett problem för *backtracking*. Deklarera globalt själva bordet med en ram av förbjudna rutor, som 10×7. Anropa funktionen `solve` som ska lösa problemet med `solve(8,5,0);`, med parametrarna i tur och ordning *startrad, startkolumn* och *antal poäng hittills*.

Kontrollera om uppnådd poäng är maximal hittills. Markera aktuell ruta som besökt. Försök att förflytta kulan i de fyra riktningarna från aktuell ruta. Återställ rutans värde efter anropet i bästa backtracking-stil. Programmet tar upp ungefär 35 satser.

14.1.2 Problem 3 – Torn

Antag att de `t` tornens höjder är `1...t`. Generera de `t`! möjliga permutationerna av tornen genom backtracking. Kontrollera varje gång en permutation är fullbordad hur många gånger ”rekordet” slås då man söker det största tornet från index `0` och uppåt. Samma kontroll då man startar i index `t - 1` med avtagande index.

14.1.3 Problem 4 – Pussel

Även i detta problem kan det vara lämpligt att ha en ring av förbjudna rutor runt den 3×3 stora spelplanen. Eftersom vi inte känner till det största antal drag som kan förekomma är det lämpligast att använda *bredden först*. Men efter en del experimenterande kanske man kan bestämma sig för ett värde för maximalt djup i rekursionsträdet, även om detta kan vara lite osäkert.

När vi utför ett drag tömmer vi en ruta och den som tidigare var tom får ett värde. Då vi använder djupet först måste vi alltså spara och återställa två värden.

Om vi undviker att direkt flytta tillbaka en bricka till rutan den kom ifrån tjänar vi in mycket. Endast hälften av $6! = 720$ möjliga startkonfigurationerna kan lösas.

Vi behöver också en funktion som kontrollerar om målet har uppfyllts. Om så är fallet testar vi om antalet drag är nytt rekord.

14.1.4 Problem 5 – Maratonlopp

Vi läser in alla vägar till en avståndstabell deklarerad `avst[101][101]`. Eftersom byarna är numrerade `1` till `100`. Talet `0` i denna tabell innebär att ingen väg finns mellan byarna. Vi passar också på att lägga in samma avstånd i `avst[i][j]` som i `avst[j][1]`.

Vi anropar från `main` sedan den rekursiva funktionen med alla `b` byarna som start. Genom avståndstabellen får vi fram vilka grannbyar vi kan åka till. I arrayen `used` bokar vi av de byar vi besökt. Om vi råkar få en total sträcka på `42` räknar vi upp antalet möjligeter. Så fort vi överskrider `42` avbryter vi rekursionen.

14.1.5 Problem 6

Vi startar med att bygga en avståndstabell, precis som i förra uppgiften. Här är det bra om man känner till *Dijkstra* eller kanske ännu hellre *Floyd-Warshall*. Om vi bestämmer oss för att använda *Dijkstra* anropar vi en funktion från `main` för alla kombinationer av 'start'- och 'stopp'-hus. Funktionen hittar du i kompendiet.

14.2 Tentamen 080115

14.2.1 Problem 1 – Färgblandning

Ett problem som skulle kunna ha förekommit i *Programmering grundkurs*. Deklarera `int canvasblue[25][25]={0}` och `int canvasyellow[25][25]={0}`för att måla på. Läs in data för en rektangel i taget. Måla rektangeln på rätt canvas. Skriv ut slutresultatet genom att jämföra värdet på båda dukarna.

14.2.2 Problem 2

Läs in alla punkter till `koord[100][3]`. Skapa en trippelloop som plockar en punkt var från tabellen. Bestäm arean och jämför med tidigare minimum.

14.2.3 Problem 3

Bygg genom backtracking upp alla permutationer av de nio siffrorna. För varje permutation ska vi sätta in två 'streck'. Bilda talen mellan strecken och kolla om de är heltalskvadrater. Du vet väl att man skriver

```
if(sqrt(s)==(int)sqrt(s))
```

för att kolla om s är en heltalskvadrat.

14.2.4 Problem 4

En uppgift för backtracking. Läs in talen till en matris `tab[10][10]` eller `tab[12][12]` om du väljer att lägga en förbjuden ram runt om. Från `main` anropas en funktion `solve(rad,kol,0)` för alla tänkbara starttal. Från varje ruta väljer man de möjliga fortsättningarna. Kan maximalt vara 3 utom i första steget då det är 4. Rekursionsträdet har $< 4 \cdot 3^9 = 78732$ noder, som ska utföras för de 100 startvärdena.

14.2.5 Problem 5

Läs in alla förbindelser till `int tab[26][26]`. 1 i denna tabell betyder att direkt förbindelse finns. 0 att ingen förbindelse finns. Här ska vi låta alla kombinationer av maskiner vara trasi-

ga utom maskinerna `1` och `n`. Deklarera en array `trasig[26]`. Generera i tur och ordning de kombinationer där en maskin är trasig. Testa om man då kan ta sig från `1` till `n`. Om det är möjligt att komma fram för alla kombinationer där `t` maskiner är trasiga genererar man alla kombinationer där $t + 1$ maskiner är trasiga. Så fort man hittar en kombination där man inte kan komma fram har man svaret.

Man använder backtracking för att finna samtliga kombinationer med `t` trasiga maskiner. Algoritmen finns i kompendiet.

Man använder backtracking för att ta sig från `1` till `n` och bokar av i `tab` genom att sätta in `0`.

14.2.6 Problem 6

Vi startar med att deklarera en `int platta[n][n]={0}` på vilket vi ska lägga pusslet. En funktion för backtracking startar med att ta reda på första tomma rutan. Den letar radvis. Om det inte finns någon tom ruta är pusslet lagt. Vi jämför då antalet bitar vi använt med det hittills minsta antalet och uppdaterar eventuellt rekord. Annars tar vi reda på den största tänkbara kvadrat vi kan lägga in utifrån den tomma rutan vi funnit.

När vi nu känner den största tänkbara känner vi också alla mindre kvadrater ned till den 1×1 som kan placeras här.

I tur och ordning placerar vi in dem på plattan och börjar med den största tänkbara. Vi anropar funktionen rekursivt. När vi kommer tillbaka tar vi förstås bort den kvadrat vi just placerat och sätter istället in en mindre.

Här kan det vara viktigt att klippa i trädet när antalet insatta kvadrater blir större än det minsta kända antalet.