29 juli Programmering, Pepper - Pass 2

1.

Fortsätt på klassen med komplexa tal och skapa metoder så att man kan addera, multiplicera, subtrahera, skriva ut talen, hämta ut real-/imaginärdel, hämta talet på rektangulär och polär form, jämföra olika tal mm. Kolla så funktionerna även fungerar med vanliga pythontal (typ att addera med x = 8.6).

Tips på funktioner att använda: __add__, __mul__ , __sub__ , __div__, __abs__ , __str__, __eq__, __ge__, __lt__ ....

Skapa en klass ”reell” som ärver av komplexa tal och sätter imaginärdelen till 0. Den ska kunna användas både med andra reella tal, men också objekt av typen komplex och av vanliga pythontal (typ x = 8.3). Adderar man med ett komplex tal blir svaret naturligtvis komplext och inte reellt.

Vill du kan du skapa barn till reell också som bara hanterar rationella tal, då får du spara täljaren och nämnaren som varsitt heltal för att undvika avrundningsfel i python. Se till att den klassen fungerar korrekt tillsammans med sina föräldrar (reell och komplex).

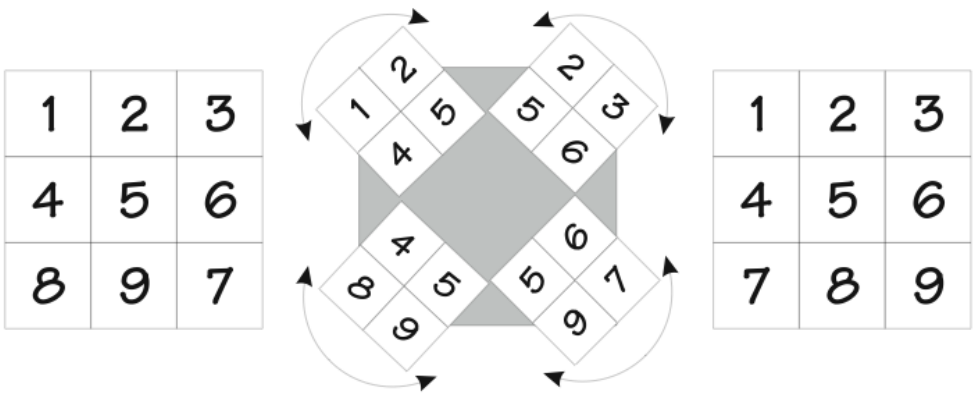
2.

Labyrintkoden som vi nyss har gått igenom är rätt enkel att skriva, och löser oftast uppgiften inom rimlig tid. Men det finns specialfall som gör att DFS kommer ta rätt lång tid. Som tur är finns det en lösning på problemet som snabbar upp koden med nästan en faktor 100! Dynamisk programmering! Det handlar om att hela tiden spara den bästa lösningen som nått fram till ett speciellt tillstånd i spelet. Ibland kan det vara svårt att veta hur man ska spara en lösning i ett visst tillstånd, men just i labyrinten är det enkelt. Det är bara att spara minsta antalet drag för att nå fram till en viss ruta, i rutan själv.

Modifiera er DFS lösning så att den sparar sina tillstånd i labyrinten. Dvs. ändra på besokt-arrayen från en boolsk array till en array med heltal. Uppdatera överallt där du använder dig av besokt variabeln.

Koden borde bli mycket snabbare om ni gjort rätt.

3.



Figur 1: Till vänster ser vi en av många startmöjligheter. I mitten visas hur de fyra 2 × 2-kvadraterna kan roteras. Till höger visas målet.

I flera av Nokias mobiltelefoner finns detta förströelsespel eller pussel. Pusslet går ut på att från ett givet utgångsläge, som till exempel till vänster i figur 1, nå fram till ordning och reda som till höger. I varje ställning finns åtta möjliga drag. Varje liten 2×2-kvadrat kan vridas 90° medurs eller moturs i taget. Om man skulle göra fyra drag i följd med samma kvadrat i samma riktning skulle man vara tillbaka vid utgångsställningen.

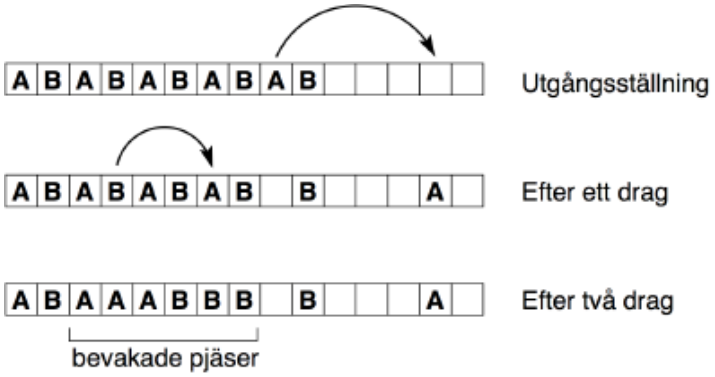
Vi har undersökt att det aldrig behövs fler än 11 drag för att nå målet från vilken utgångsställning som helst. Din uppgift är att skriva ett program som tar emot en ställning och som bestämmer hur många drag den minst kräver för att nå målet. Ställningen (avläst rad för rad) ges till programmet som en sträng innehållande varje siffra i intervallet 1...9 exakt en gång.

Körningsexempel:

```
Ställning ? 123456897
Denna ställning kräver 6 drag
```

4.

Senet är ett uråldrigt egyptiskt brädspel som hittats i otaliga gravar. Reglerna är okända men en av de vanligaste rekonstruktionerna beskrivs här. Spelplanen kan betraktas som en serie med rutor där vi i denna uppgift endast bryr oss om de första 15. De två spelarna, som vi kallar A och B, har vardera 5 pjäser som vid spelets början alltid står uppställda enligt översta schemat i figur 2. Spelarna turas om att göra drag, A har första draget. I varje drag kastar spelaren några stickor, vilka tillsammans utgör en slags tärning som kan anta värdena 1, 2, 3, 4 eller 5. Därefter väljer spelaren en av sina pjäser och flyttar den framåt med så många steg som tärningen visade. För att vara ett godkänt drag måste pjäsen sluta på en ruta som antingen är tom eller innehåller en obevakad motståndarpjäs. I det senare fallet flyttas motståndarpjäsen tillbaka till den ruta varifrån den flyttade pjäsen kom, så att pjäserna i praktiken byter plats. En pjäs är obevakad om det inte finns en likadan pjäs i någon av de två angränsande rutorna.



Figur 2: Utgångsställningen samt ställningarna efter dragen i exemplet nedan.

Du ska skriva ett program som givet pjäsernas nuvarande positioner bestämmer det minimala antalet drag N som har spelats, samt en möjlig sådan dragsekvens.

Programmet ska fråga efter en sträng med längd 15, där varje tecken beskriver en ruta på spelplanen: A för en pjäs tillhörande spelare A, B för en pjäs tillhörande spelare

B och . (punkt) för en tom ruta. Det finns alltid fem A:n, fem B:n och fem punkter i strängen.

Programmet ska skriva ut den kortaste dragsekvensen som leder från utgångsställningen till den inmatade ställningen. Närmare bestämt ska programmet skriva ut en rad med N tal i intervallet 1..5, där varje tal anger hur många steg som har tagits i respektive drag (första talet indikerar första draget etc.). Vilken pjäs som flyttats i varje drag behöver inte anges. Om det finns flera sekvenser med minimal längd kan programmet ange vilken som helst av dem. I samtliga testfall är $N \leq 6$. Notera att N kan vara udda, d.v.s. A kan ha gjort ett drag mer än B.

Körningsexempel:

```
Ställning ? ABAAABBB.B...A.
Dragsekvens: 5 3
```

5.

A .-	B -...	C -.-.	D -..	E .	F ..-.
G --.	H	I ..	J .---	K -.-	L .-..
M --	N -.	O ---	P .--.	Q --.-	R .-.
S ...	T -	U ..-	V ...-	W .--	X -..-
Y -.--	Z --..				

Figur 3: Tabell över morsekoderna

Tabellen ovan visar Morsealfabetet. Punkter och streck används för att beteckna korta respektive långa signaler. Normalt lämnas morsemeddelanden med ett uppehåll mellan varje bokstav, till exempel –. . - som står för ordet GET. Men av en obestämd anledning kommer det just nu in meddelanden i en oavbruten följd av långa och korta signaler, alltså utan uppehåll mellan bokstäverna. GET-signalen blir då –..- och kan därför stå för många andra bokstavssekvenser, t.ex. MU. Skriv ett program som tar emot en följd av streck och punkter, och som sedan skriver ut varje möjlig kombination av bokstäver som svarar mot den givna indatasträngen.

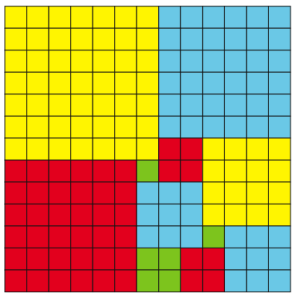
Indata På första raden står signalsträngen, som består av högst 15 tecken.

Utdata Programmet ska skriva ut alla möjliga bokstavssekvenser som svarar mot den givna indatasträngen. Sekvenserna ska skrivas ut i alfabetisk ordning med en per rad. Endast versalerna A-Z får förekomma.

Körningsexempel:

```
Morsekod? --..-
Möjliga sekvenser:
GA, GET, MEA, MEET, MIT, MU, TDT, TNA, TNET, TTEA, TTEET, TTIT, TTU, TX, ZT,
Morsekod? ..-
EA, EET, IT, U,
```

6.



Figur 4

I figur 4 ser du en kvadrat (13×13) som består av 11 mindre kvadrater, som också är det minsta antal ”småkvadrater” som krävs för att skapa en kvadrat av denna storlek.

Skriv ett program som tar emot den stora kvadratens sida S , där $2 \leq S \leq 17$, och bestämmer det minsta antal mindre kvadrater som behövs för att lägga pusslet.

Körningsexempel:

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

7. På mitt jobb finns en colaautomat där en burk Coca-Cola kostar 8 kr. Maskinen har ett myntinkast som accepterar 1 kr, 5 kr och 10 kronorsmynt. Efter att tillräckligt många mynt har stoppats in kan jag trycka på Cola-knappen och få ut en Cola och eventuell växel. Växeln ges alltid tillbaka med det minsta antalet möjliga mynt. Denna procedur upprepas tills jag köpt så många Cola-burkar som jag vill ha. Jag kan bara köpa en burk åt gången. Om jag t.ex. skulle stoppa in 16 kr och trycka på Cola-knappen skulle jag få ut en burk och 8 kronor i växel (en 5-krona och 3 enkronor).

Givet hur många burkar jag vill köpa, samt hur många mynt jag har av varje valör, vad är det minsta antalet mynt jag behöver stoppa in i maskinen för att köpa så många burkar som jag vill ha? Jag kan ta upp mynt som kommer ut som växel. Du kan utgå ifrån att maskinen har obegränsat med mynt i växel.

Programmet ska, i tur och ordning, fråga efter hur många Cola-burkar jag vill köpa (mellan 1 och 40), antalet 1-kronors mynt jag har (mellan 0 och 100), antalet 5-kronors mynt (mellan 0 och 20), och antalet 10-kronors mynt (mellan 0 och 10). Programmet ska skriva ut det minsta antal mynt jag behöver stoppa i myntinkastet. Du kan utgå ifrån att jag har tillräckligt med pengar för att köpa alla Cola-burkar jag vill ha.

Körningsexempel:

```
Antal burkar ? 2
Antal 1-kronor ? 2
Antal 5-kronor ? 1
Antal 10-kronor ? 1
Du behöver stoppa in 5 mynt.

Antal burkar ? 3
Antal 1-kronor ? 20
Antal 5-kronor ? 1
Antal 10-kronor ? 1
Du behöver stoppa in 12 mynt.
```