Antagningsprov Programmeringskollo 2024

Programmering är kul men Programmeringskollo 2024 har tyvärr ett begränsat antal platser. För att hålla en jämn nivå på deltagarna har vi därför förberett ett antal uppgifter. För att du ska trivas på lägret är det viktigt att nivån på dina inskickade lösningar motsvarar din egen förmåga. Lös uppgifterna utan hjälp av vare sig föräldrar, kompisar, syskon, chatGPT etc. Om det visar sig att du har fått hjälp med uppgifterna eller samarbetat med någon så får du inte åka på kollot.

Vi tar in de elever med bäst resultat på programmeringsprovet nedan, men om man placerat sig på top 15 i Sverigefinalen i programmeringsolympiaden (PO) är man direktkvalificerad till kollot!

Skicka in lösningarna till **antagningsprov@programmeringskollo.se** senast den **21 april 2024**. Du behöver inte lämna in lösningar på alla uppgifter, det är inte meningen att alla ska lösa allt. Om du har frågor, så skickar du dem till adressen ovan.

Alla dina lösningar ska skickas in samtidigt och vara bilagor till din e-post, observera att vissa epostklienter blockerar script som bilagor, men om du döper om filen till .txt brukar det gå igenom. Koden ska gå att läsa och förstå, ta gärna med kommentarer. Filerna du skickar får inte överstiga 30 Mb i storlek. I ämnesraden skriver du ”Programmeringsprov”, samt ditt förnamn och efternamn och årskurs just nu, till exempel: **”Programmeringsprov Anna Svensson åk 8”** eller **”Programmeringsprov Abraham Swift åk 2 gymnasiet”**.

Om du kommer på att du vill ändra något så kan du skicka alla lösningar på nytt igen med samma ämnesrad. Vi kommer då bara kolla på det senast inskickade mejlet, då ignoreras alla tidigare mejl.

Ju bättre du förklarar dina lösningar, desto större chans har du att komma med på Programmeringskollo 2024! Skriv ner dina tankar även om du inte har löst hela uppgiften, delpoäng kan vara avgörande.

Besked om antagning/reservplats skickas av oss senast den **1 maj**. I slutet av maj meddelar vi också antagningsbesked till reserver. Kom ihåg att anmäla dig som sökande på `http://programmeringskollo.se`! Detta gör du senast den **21 april**.

Misströsta inte om vi inte har möjlighet att ta in just dig till årets kollo. Försök gärna igen nästa år!

Programmering

1. AI-Praktikantens dilemma (sortering)

I en framtid där artificiell intelligens har blivit integrerad i alla aspekter av livet, har du fått i uppgift att hjälpa en AI-praktikant vid namn Aiden att organisera en omfattande databas med forskningsartiklar om AI. Aiden har samlat titlarna på tusentals artiklar men inser att för att underlätta forskarnas arbete, behöver artiklarna sorteras i alfabetisk ordning.

Skriv en funktion som tar emot en lista med titlar på forskningsartiklar som input och returnerar en ny lista där titlarna är ordnade i alfabetisk ordning.

Varje artikel har också en ”relevansscore” baserad på antalet gånger AI nämns i artikeln. Din funktion ska först sortera artiklarna i alfabetisk ordning och sedan, som en sekundär sortering, efter deras relevansscore i fallande ordning där artiklar med samma titel jämförs.

Input

En lista av tuples, där varje tuple innehåller en sträng (artikelns titel) och ett heltal (relevansscoren).

Output

En lista av tuples sorterad först efter titel (i alfabetisk ordning) och sedan efter relevansscore (i fallande ordning för artiklar med samma titel).

Exempel

Input:

```
[('Zebraer och AI', 10),
('AI i vardagen', 5),
('Zebraer och AI', 15),
('AI i vardagen', 20)]
```

Output:

```
[('AI i vardagen', 20),
('AI i vardagen', 5),
('Zebraer och AI', 15),
('Zebraer och AI', 10)]
```

2. Robotens räddningsuppdrag (BFS / DFS)

I en nära framtid har AI-drivna robotar blivit en integrerad del av räddningstjänsten. Ett uppdrag kräver att en sådan robot navigerar genom en komplex labyrint för att rädda en forskare som har blivit instängd i en av dess många kammare.

Labyrinten kan representeras som en tvådimensionell matris där varje element kan vara en vägg (representerad som ’#’), en gång (representerad som ’ ’), robotens startposition (representerad som ’S’), eller forskarens position (representerad som ’F’).

Skriv en funktion som tar emot en tvådimensionell matris som representerar labyrinten och returnerar den kortaste vägen som roboten måste ta för att nå fram till

forskaren. Du kan anta att roboten endast kan röra sig upp, ner, vänster eller höger (ingen diagonalförflyttning är tillåten). Om det inte finns någon möjlig väg, bör din funktion returnera en indikation på detta.

Input

En tvådimensionell lista som representerar labyrinten.

Output

En lista av koordinater (tuples) som representerar den kortaste vägen från 'S' till 'F', inklusive start- och slutpunkterna. Om ingen väg finns, returnera en lämplig indikation på detta.

Exempel

```
[
''# ###'',
''#   #'',
''# #  '',
''#S# F'',
''#####''
]
```

En möjlig lösning (output) kan vara: `[(3,1), (2,1), (1,1), (1,2), (1,3), (2,3), (2,4), (2,5)]`, där varje tuple representerar en position i matrisen som roboten måste passera för att nå forskaren.

3. Robotens effektivitetsmaximering (girig algoritm)

I framtidens fabrik, där du är ansvarig, finns en särskild robot som är designad för att bygga avancerade komponenter till superdatorer. För att hålla produktionen så effektiv som möjligt, vill du se till att roboten är i drift så mycket som möjligt under en arbetsdag. Varje uppgift roboten kan utföra har en bestämd starttid och sluttid. Målet är att schemalägga uppgifter på ett sätt som maximerar den totala tiden roboten arbetar.

Eftersom roboten är den enda i sitt slag i fabriken och inte kan utföra mer än en uppgift åt gången, behöver du välja uppgifter på ett sådant sätt att den sammanlagda tiden som roboten är aktiv blir så stor som möjligt, utan att uppgifternas tider överlappar (däremot kan en uppgift sluta och en ny påbörjas på samma tidpunkt). Roboten kan inte heller avbryta en uppgift mitt i, utan den måste pågå från start till slut.

Skriv en funktion som tar emot en lista av uppgifter, där varje uppgift har en starttid och en sluttid, och returnerar den maximala tiden roboten kan vara aktiv genom att utföra en optimal uppsättning av icke-överlappande uppgifter.

Input

En lista av tuples där varje tuple representerar en uppgift i form av `(starttid, sluttid)`. Starttid är alltid mindre än sluttid.

Output

Ett heltal som representerar den maximala tiden roboten kan vara aktiv.

Exempel

Anta att du har följande uppgifter:
`[(1, 4), (3, 5), (0, 6), (5, 7), (8, 9)]`
En optimal uppsättning av uppgifter som maximerar den tid roboten är aktiv kan vara `(0, 6), (8, 9)`, vilket ger en total aktiv tid på 7 timmar.

4. Navigera i språkmodellens sannolikhetslandskap (grafalgoritm)

I denna uppgift är målet att navigera genom en graf som representerar sannolikheter mellan olika ord i en språkmodell för att hitta den mest sannolika sekvensen av ord från ett startord till ett målord. Varje ord representeras som en nod, och varje kant mellan två ord (noder) har en vikt som representerar sannolikheten för att det ena ordet följer på det andra i språkmodellens genererade text.

Använd en modifierad version av Dijkstra's algoritm för att hitta den mest sannolika sekvensen av ord från ett startord till ett slutord, baserat på sannolikheter mellan orden.

Input

- En graf representerad som en matris av sannolikheter (givna av heltal mellan 0 och 100), där varje rad och kolumn motsvarar ett specifikt ord och cellernas värden representerar sannolikheten för att övergå från ordet i raden till ordet i kolumnen.
- Ett startord.
- Ett slutord.

Input

Den mest sannolika sekvensen av ord från start till slut.

Exempel

Anta att du har en enklare graf där noderna (orden) är ”jag”, ”gillar”, ”dig”, ”nej”, och ”ja”. Sannolikhetsmatrisen och ordindexeringen kan se ut så här:

	jag	**gillar**	**dig**	**nej**	**ja**
jag	0	50	0	20	30
gillar	10	10	70	10	0
dig	0	0	0	10	90
nej	0	0	30	0	70
ja	0	0	0	0	0

Om du startar från ”jag” och målet är att nå ”ja”, skulle den mest sannolika sekvensen av ord kunna vara ”jag” -> ”gillar” -> ”dig” -> ”ja”, vilket representerar en möjlig meningsfull sekvens genom språkmodellens sannolikhetslandskap.

5. AI-drivna säkerhetskamerors synfält (geometriuppgift)

I en framtid där AI och maskininlärning revolutionerat säkerhetssystem, används avancerade AI-drivna säkerhetskameror för att övervaka och skydda viktiga anläggningar. Dessa kameror har förmågan att panorera (rotera horisontellt) för att täcka ett stort område. Varje kamera har ett specifikt synfält som beror på dess position, vinkel och zoomnivå.

Utveckla ett system som beräknar överlappningen i synfältet mellan två eller flera säkerhetskameror i en anläggning. Systemet ska kunna identifiera områden som är väl övervakade (övervakade av 2 eller fler kameror) och områden som utgör säkerhetsrisker (inga kameror som täcker området).

Input

- Lista över kameror, där varje kamera är representerad av sin position i ett tvådimensionellt plan (x, y), dess riktning (i grader), och synvinkel (i grader).
- Det område som ska övervakas, representerat som en rektangel i det tvådimensionella planet, givet av heltalskoordinaterna på sina fyra hörn.

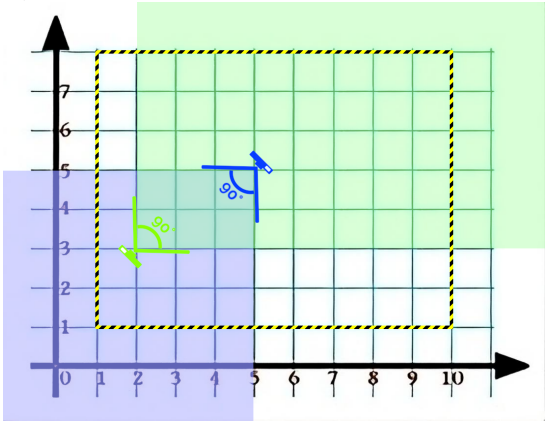
Output

- Hur stort område som är väl övervakat, det vill säga hur stor area av rektangeln som ses av två eller fler kameror.
- Hur stora säkerhetsrisker som finns, det vill säga hur stor area av den angivna rektangeln som inte bevakas av någon kamera.

Exempel

Anta att du har två kameror och ett rektangulärt område som ska övervakas. De två kamerorna har följande egenskaper:
- Kamera 1: Position (2, 3), riktning 45 grader, synvinkel 90 grader.
- Kamera 2: Position (5, 5), riktning 225 grader, synvinkel 90 grader.

Området som ska övervakas är en rektangel med hörn i (1, 1), (1, 8), (10, 8), (10, 1).



Det innebär att 6 a.e. är väl övervakade och 13 a.e. är säkerhetsrisker.

6. Bonus: Realtids ansiktsigenkänning och emotionsanalys (praktisk uppgift)

I denna uppgift ska du utveckla ett program som använder din webbkamera för att kontinuerligt läsa in videoströmmen i realtid. Programmet ska kunna identifiera ansikten i videobilden och sedan analysera och klassificera den upptäckta personens emotion baserat på ansiktsuttrycket.

- Använd OpenCV för att få tillgång till webbkamerans live-feed.
- Implementera ansiktsigenkänning för att identifiera när ett ansikte är synligt i videofeeden.
- Tillämpa en enkel form av emotionsanalys för att klassificera ansiktsuttrycket i kategorier såsom glad, ledsen, förvånad, arg etc. Detta kan göras genom förutbestämda heuristiker eller en *trivial* maskininlärningsmodell om tillräckligt med data och förkunskaper finns tillgängliga.
- Visa resultatet av emotionsanalysen på skärmen i realtid tillsammans med videofeeden, till exempel genom att rita en ram runt varje identifierat ansikte och annotera ramen med den uppskattade emotionen.

Tips:

- Undersök och använd befintliga OpenCV-funktioner för ansiktsigenkänning, som Haar cascades eller DNN-baserade metoder.
- Utforska enkla klassificeringsmetoder för emotionsanalys, möjligen med förtränade modeller.