सैद्धांतिक कंप्यूटर विज्ञान में करोड़ों रूपए के इनामी प्रश्न

रणवीर सिंह
संगणक विज्ञानं एवं अभियांत्रिकी विभाग
भारतीय प्रौद्योगिकी संस्थान इंदौर

संक्षेप: कंप्यूटर के बिना वर्तमान दुनिया अकल्पनीय है। हम सभी अपने दैनिक जीवन में विभिन्न कंप्यूटिंग उपकरणों का उपयोग करते हैं। लेकिन क्या कंप्यूटर हमारे सभी सवालों का समाधान दे सकता है? दूसरा महत्वपूर्ण प्रश्न यह है कि क्या इसके समाधान सदैव कारगर होंगे? इस लेख में हम कुछ सरल उदाहरणों के माध्यम से कंप्यूटर द्वारा हमारी गणना की कुछ संभावित सीमाओं को देखेंगे।

प्रस्तावना: यदि हम बहुत निचले स्तर पर देखें तो कंप्यूटर हमारे लिए केवल गुणा, भाग, जोड़, घटाव, डेटा स्थानांतरण जैसे बुनियादी कार्य करता है। इन बुनियादी परिचालनों (operations) का एक क्रम जो एक सीमित समय में परिणाम देता है उसे एल्गोरिदम (algorithm) कहा जाता है। यहाँ `सीमित' समय बहुत महत्वपूर्ण है। एक एल्गोरिदम द्वारा उठाए गए बुनियादी चरणों की संख्या को उसकी समय-जटिलता (time-complexity) कहा जाता है। एक उदाहरण लेते हैं।

दो n×n आव्यूह (Matrices) A, B का गुणन: यदि हम अपने स्कूल में सीखी गई तकनीक का उपयोग करके एक एल्गोरिदम बनाते हैं, तो A×B की गणना करने में कितना समय लगेगा? A×B की किसी भी संख्या की गणना करने के लिए हमें A की एक पंक्ति और B के एक स्तंभ को चुनना होगा। A की पंक्ति और B के स्तंभ में संबंधित संख्याओं को गुणा किया जाता है और फिर हम उन्हें जोड़ते हैं। तो A×B की एक संख्या की गणना करने के लिए n गुना, n-1 जोड़ की आवश्यकता होती है। चूँकि A×B में n^2 संख्याएँ होंगी, इसलिए आवश्यक इन परिचालनों की कुल संख्या $n^2 \times (n+n-1)$ है, जो लगभग $2n^3$ है। n^3 के सामने स्थिरांक 2 को हम अनदेखा करते हैं, और हम कहते हैं कि मैट्रिक्स गुणन n^3 का समय क्रम (order of n^3 time) लेता है, और हम इसे $O(n^3)$ द्वारा निरूपित करते हैं। गणितज्ञ और कंप्यूटर वैज्ञानिक कम समय-जटिलता वाले एल्गोरिदम के लिए प्रयास करते रहते हैं। वर्तमान में आव्यूह गुणन के लिए सर्वोत्तम एल्गोरिदम की समय जटिलता $O(n^{2.36})$ है।

अब एक उदाहरण आपको हल करना है। मान लीजिए आपके पास n ताश के पत्ते हैं। उन्हें बढ़ते या घटते क्रम में क्रमबद्ध करने के लिए एक एल्गोरिदम विकसित करें। (आप इसे $O(n^2)$ में आसानी से कर सकते हैं। कैसे?। सबसे अच्छा ज्ञात एल्गोरिदम O(n log n) समय लेता है।)

मान लीजिए किसी प्रश्नं के इनपुट की लंबाई या आकार n है। इस प्रश्न के लिए जो एल्गोरिदम कुछ स्थिर c के लिए $O(n^c)$ समय लेता है, हम उन्हें बहुपद समय (polynomial time) एल्गोरिदम कहते हैं। ये एल्गोरिदम आमतौर पर कुशल (efficient) होते हैं और इसलिए वांछनीय होते हैं। जो एल्गोरिदम $O(2^n)$ समय लेते हैं वे n बढ़ने पर बहुत ज्यादा समय लेते हैं , हम उन्हें घातीय (exponential) एल्गोरिदम कहते हैं। मान लीजिए n=100, संख्या 2^{100} बहुत बड़ी है, संभवतः इस ब्रह्मांड में रेत के कणों से भी अधिक। ऐसे एल्गोरिदम व्यावहारिक रूप से कभी भी किसी सुपर कंप्यूटर पर उत्तर देने में सक्षम नहीं होंगे।

दुर्भाग्य से (असल में दिलचस्प बात है) ऐसे कई प्रश्न हैं जिनके लिए हम बहुपद समय एल्गोरिदम विकसित करने में सक्षम नहीं हैं। गणितज्ञ और कंप्यूटर वैज्ञानिक वर्षों से इस पर प्रयास कर रहे हैं। सन 2000 में क्ले मैथमेटिक्स इंस्टीट्यूट ने इन प्रश्नों के लिए बहुपद समय एल्गोरिदम देने वाले के लिए 1 मिलियन अमेरिकी डॉलर का पुरस्कार देने का वादा किया था । और आश्चर्य की बात यह है कि अगर इन सवालों में से किसी के लिए भी बहुपद अल्गोरिथम मिलती तो बाकी सारे ऐसे सवालों के लिए भी बहुपद अल्गोरिथम मिल जायेगी। (हालाँकि इस आश्चर्यजनक तथ्य की चर्चा हम किसी और लेख में करेंगे।)

अभी थोड़ा इस चर्चा पे विराम लगाते हैं और पूछते हैं कि क्या हमारे प्रत्येक प्रश्न के लिए एल्गोरिदम विकसित करना संभव है? बहुपद समय या घातांकीय समय के बारे में फिलहाल भूल जाइए। दुर्भाग्य से (वास्तव में फिर से एक दिलचस्प बात) ऐसे प्रश्न हैं जिनके लिए हम कभी भी कोई एल्गोरिदम विकसित नहीं कर पाएंगे। चाहे हमें कितना भी बड़ा सुपर कंप्यूटर मिल जाए, चाहे हमारे पास कितना भी समय हो, चाहे हम गणित में कितने ही महान क्यों न हो जाएं। ऐसे प्रश्नों को हम अनिर्णीत (undecidable) प्रश्न कहते हैं। कोई भी कंप्यूटिंग प्रक्रिया इन प्रश्नों पर निर्णय नहीं ले सकती कि समाधान सही हैं या नहीं। यह भी हो सकता है कि यह प्रक्रिया कम्प्यूटर पर सदैव चलती रहे और कभी उत्तर न दे। वास्तव में अनिर्णीत प्रश्नों की संख्या अनिर्णीत प्रश्नों की संख्या से अधिक है। इससे पता चलता है कि बुनियादी तौर पर कंप्यूटर की शक्ति और क्षमता सीमित है।

सबसे पहले, कंप्यूटर कभी भी हमारी सभी समस्याओं का समाधान नहीं कर सकता है, दूसरे, कंप्यूटर जिन समस्याओं को हल कर सकता है उनमें से कई समस्याएं हैं जिन्हें कंप्यूटर संभवतः कुशलता से हल नहीं कर सकता है।
लेख के बाकी भाग में हम उपरोक्त चर्चा से संबंधित कुछ दिलचस्प उदाहरण देंगे। एक उदाहरण एक अनिर्णीत प्रश्न का होगा, जिसे हम PCP प्रश्न से जानते है, इसके लिए कोई एल्गोरिदम नहीं बना सकता है। शेष पाँच प्रश्न ऐसे हैं जिनके लिए अब तक कोई बहुपद समय एल्गोरिदम नहीं है, और यदि कोई इसे विकसित करता है तो उसे एक मिलियन डॉलर का पुरस्कार दिया जाएगा। एक ख़ास बात है कि इन प्रश्नों में अगर हल्का सा बदलाव किया जाए तो इन्हें हल करना बहुत आसान है।

1. PCP प्रश्न:

एमिल पोस्ट, 1946

मान लीजिए कि हमारे पास तीन अलग-अलग अक्षर a, b, c और तख्तियों का एक संग्रह है। प्रत्येक तख्ती के ऊपर और नीचे इन अक्षरों का उपयोग करके कुछ श्रंखलायें (string) लिखी गयी हैं। हमें तख्तियों को इस तरह व्यवस्थित करना है कि ऊपर लिखी गई श्रंखला नीचे लिखी गई श्रंखला के समान हो। हम किसी भी तख्ती का कितनी भी बार उपयोग कर सकते हैं। उदाहरण के लिए नीचे कुछ तख्तियां दी हुयी हैं।

$$\left[\frac{bc}{ca}\right] \quad \left[\frac{a}{ab}\right] \quad \left[\frac{ca}{a}\right] \quad \left[\frac{abc}{c}\right]$$

1 2 3 4

इन तख्तियों को नीचे इस प्रकार व्यवस्थित किया गया है कि ऊपर वाली श्रंखला नीचे वाली श्रंखला के समान है।

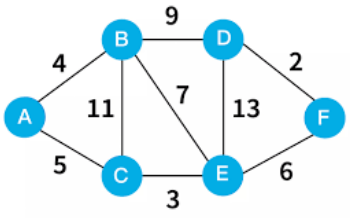
$$\left[\frac{a}{ab}\right] \quad \left[\frac{bc}{ca}\right] \quad \left[\frac{a}{ab}\right] \quad \left[\frac{abc}{c}\right]$$

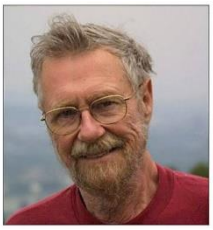
2 1 2 4

(यहाँ तख्ती नंबर 2 का प्रयोग दो बार हुआ है।) सामान्य तौर पर दिए गए अक्षरों और तख्तियों के लिए यह एक अनिर्णीत समस्या है। हम इसके लिए कोई एल्गोरिदम नहीं बना सकते।

2. किसी शहर से सबसे छोटा रास्ता:

मान लीजिए हमारे पास किसी देश या राज्य का सड़क नेटवर्क है। हम किसी शहर A से किसी शहर F तक जाना चाहते हैं। A से F तक का सबसे छोटा रास्ता क्या होगा। उदाहरण नीचे दिया गया है। शहरों को बिंदुओं (vertex) द्वारा और सड़कों को रेखाओं (edge) द्वारा दर्शाया जाता है, और रेखाओं पर संख्याएँ दूरियाँ बताती हैं। हम ऐसी संरचना को ग्राफ़ कहते हैं।





वर्ष 1956 में डिज्स्कट्रा ने इसके लिए एक बहुत ही सरल और तेज़ एल्गोरिदम दी। दिज्स्क्ट्रा एल्गोरिथम देखे गए बिंदुओं का एक समुच्चय (set) और न देखे गए बिंदुओं का एक समुच्चय बनाए रखता है। यह प्रक्रिया स्रोत बिंदु पर शुरू होती है और पुनरावृत्त रूप से स्रोत से सबसे छोटी अस्थायी दूरी के साथ न देखे गए बिंदु का चयन करती है इसके बाद यह इस बिंदु के पड़ोसियों का दौरा करती है और यदि कोई छोटा रास्ता मिलता है तो उनकी अस्थायी दूरी को अद्यतन (update) करती है। यह प्रक्रिया तब तक जारी रहती है जब तक कि गंतव्य बिंदु तक नहीं पहुंच जाता, या सभी बिंदुओं का दौरा नहीं कर लिया जाता।
अब सवाल बदलते हैं। मान लीजिए कि n शहर हैं, और हमें एक शहर A से दूसरे शहर F तक का सबसे लंबा रास्ता खोजना है। हम इसे कितनी कुशलता से हासिल कर सकते हैं? दुर्भाग्य से यहां कोई कुशल एल्गोरिदम नजर नहीं आता। (सोचें कि डिज्स्क्स्ट्रा एल्गोरिदम में सबसे छोटी दूरी को सबसे लंबी दूरी से बदलने से इस प्रश्न का सही उत्तर क्यों नहीं मिल सकता है।) एक बहुत ही स्पष्ट तरीका A से F तक प्रत्येक पथ की जांच करना है और फिर सबसे बड़े पथ का चयन करना है। लेकिन ये

बहुत ही अप्रभावी तरीका है. मान लीजिए कि हम जिस भी मध्यवर्ती शहर में पहुँचते हैं, और आगे दो ही विकल्प हैं। ऐसे नेटवर्क में भी लगभग 2^n संभावनाएँ हैं। यहां तक कि केवल n=100 के लिए भी, यह एक बहुत बड़ी संख्या है जैसा कि हमने पहले प्रस्तावना में देखा है। अब तक n शहरों के नेटवर्क में A से F तक सबसे लंबा रास्ता खोजने के लिए कोई बहुपद समय एल्गोरिदम नहीं है। यदि आप इस प्रश्न के लिए एक बहुपद समय एल्गोरिदम डिज़ाइन कर सकते हैं तो आपको दस लाख डॉलर मिलेंगे।

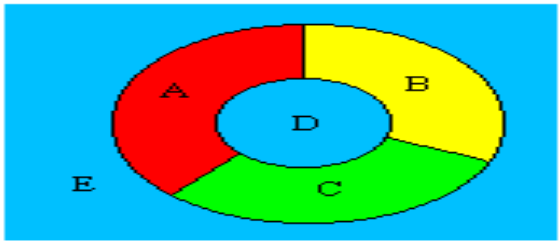
3. क्या हम प्रत्येक मानचित्र को सिर्फ 4 रंगों से रंग सकते हैं? (4-color theorem)

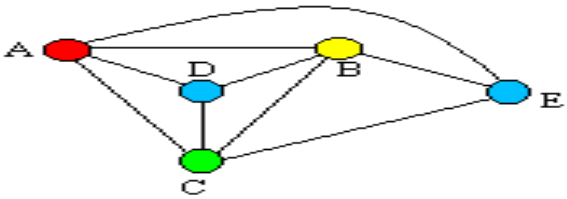
आइए कोई भी मानचित्र लें। हमारा कार्य मानचित्र के क्षेत्रों को रंगना है ताकि सीमा साझा करने वाले क्षेत्रों को समान रंग न मिले। कम से कम कितने रंगों में हम इस लक्ष्य को प्राप्त कर सकते हैं? आश्चर्य की बात है कि किसी भी मानचित्र को रंगीन करने के लिए 4 रंग पर्याप्त हैं। इस खूबसूरत तथ्य के पीछे एक महान इतिहास है।

अनुमानतः पहली बार इस प्रश्न को 23 अक्टूबर 1852 को फ्रांसिस गुथरी द्वारा प्रस्तावित किया गया थ। उन्होंने देखा की इंग्लैंड के मानचित्र को रंगने के लिए केवल चार अलग-अलग रंगों की आवश्यकता थी। उस समय, गुथरी के भाई, फ्रेडरिक, यूनिवर्सिटी कॉलेज लंदन में ऑगस्टस डी मॉर्गन के छात्र थे। फ्रेडरिक इस प्रश्न को डी मॉर्गन के पास ले गए कि क्या हर मानचित्र को इस तरह रंगने के लिए चार रंग काफी हैं ? वर्ष 1852 में विलियम हैमिल्टन को लिखे एक पत्र में डी मॉर्गन ने इस प्रश्न का उल्लेख किया था, बाद में डी मॉर्गन ने 1860 में द एथेनेयम पत्रिका में भी इस प्रश्न का उल्लेख किया।

1879 में अल्फ्रेड केम्पे ने इस तथ्य का एक प्रस्तावित प्रमाण दिया, जिसे व्यापक रूप से प्रशंसित किया गया था; उनको 1889 में फेलो ऑफ़ रॉयल सोसाइटी चुना गया और बाद में लंदन मैथमेटिकल सोसाइटी का अध्यक्ष भी चुना गया। लेकिन 1890 को प्रपर्सी हेवुड ने इस प्रमाण को गलत सिद्ध कर दिया। हालांकि इसे गलत सिद्ध करते करते उन्होंने ये सिद्ध कर दिया कि हर मानचित्र को इस तरह रंगने के लिए पांच रंग काफी है। लेकिन क्या चार रंग काफी है? यह प्रश्न अभी भी चुनौती दे रहा था। अंततः लगभग 90 वर्षों के बाद 21 जून 1976 को केनेथ अपेल और वोल्फगैंग हेकेन ने इस प्रमेय को सिद्ध किया। एक

चिलचस्प बात यह है की यह कंप्यूटर की मदद से किया गया पहला प्रमाण था। आइए अब ग्राफ़ का उपयोग करके इस प्रश्न को समझें।



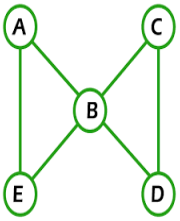


चित्र 1: बाएँ- एक नमूना मानचित्र दिया गया है, दाएँ- संबंधित ग्राफ़ दिया गया है। यह ग्राफ एक समतलीय ग्राफ है।

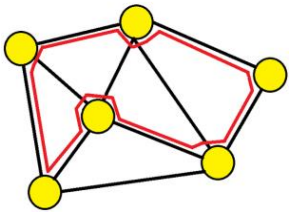
एक मानचित्र को देखते हुए, आइए हम क्षेत्रों को बिंदुओं द्वारा निरूपित करें, और यदि संबंधित क्षेत्र एक सीमा साझा करते हैं तो किन्हीं दो बिंदुओं पर एक रेखा होगी (चित्र 1 को देखें)। इस ग्राफ़ का गुण यह है कि हम इसे इस प्रकार खींच सकते हैं कि इसकी कोई भी रेखा प्रतिच्छेद न करे। ऐसे ग्राफ को हम समतलीय ग्राफ (planar graph) कहते हैं। 4-रंग प्रमेय के अनुसार चार रंग एक समतल ग्राफ के किसी भी बिंदु को रंगने के लिए पर्याप्त होते हैं, ताकि किसी भी दो आसन्न (adjacent) बिंदुओं का रंग समान न हो।
लेकिन क्या हम इसे गैर-तलीय ग्राफ़ (non-planar graph, वे ग्राफ़ जहां हम लाइन क्रॉसिंग से बच नहीं सकते) के लिए कह सकते हैं? किसी भी गैर-तलीय ग्राफ़ को रंगने के लिए न्यूनतम कितने रंगों की आवश्यकता होती है? यह एक चुनौतीपूर्ण प्रश्न है। केवल यह निर्धारित करना कि क्या किसी दिए गए ग्राफ़ को तीन रंगों से रंगा जा सकता है, एक मिलियन डॉलर का प्रश्न है।

4. यूलर दौरा बनाम हैमिल्टनियन दौरा

आइए अब हम एक ग्राफ़ (शहर-नेटवर्क) में दो प्रसिद्ध दौरों पर चर्चा करें। ग्राफ़ में एक दौरा ऐसी यात्रा है जो एक ही बिंदु पर शुरू और समाप्त होती है। एक ऐसा दौरा जहां हम प्रत्येक लाइन (सड़क) पर ठीक एक बार जाते हैं, इसे हम यूलर टूर कहते हैं। 1736 में कोनिग्सबर्ग के प्रसिद्ध सात पुलों की समस्या को हल करते समय पहली बार इस दौरे की चर्चा की गई थी। यही से ही ग्राफ थ्योरी की भी शुरुआत हुयी। दूसरी ओर एक ऐसा दौरा जहां हम प्रत्येक बिंदु (शहर) पर ठीक एक बार जाते हैं, इसे हम हैमिल्टनियन दौरा कहते हैं। मान कहते हैं। एक उदाहरण।



यूलेरियन दौरा A-B-C-D-B-E-A



हैमिल्टनियन दौरे को लाल रंग में दिखाया गया है।

तो आप क्या सोचते हैं कि कौन सी यात्रा आसानी से ढूंढी जा सकती है? या दोनों को ढूंढना कठिन है। आइए सबसे पहले एक यूलर दौरा देखें। यह देखना कठिन नहीं है कि यदि प्रत्येक बिंदु पर रेखाओं की संख्या सम हो तो ग्राफ़ में यूलर टूर होगा। वास्तव में हम बहुपद-समय में एक यूलर टूर पा सकते हैं। लेकिन हैमिल्टनियन दौरे के बारे में क्या? अभी तक इसके लिए कोई बहुपद-समय एल्गोरिथ्म नहीं है। एक बार फिर यदि आप हैमिल्टनियन दौरे को खोजने के लिए बहुपद-समय एल्गोरिदम ढूंढते हैं तो आपको एक मिलियन डॉलर का पुरस्कार मिलेगा।

5. उपसमुच्चय योग प्रश्न (Subset sum question)

इस उदाहरण को बताना तो सबसे ज्यादा सरल है। मान लीजिए कि पूर्णांकों (धनात्मक या ऋणात्मक) का एक समुच्चय (set) S है। क्या S में कोई उपसमुच्चय (subset) मौजूद है जिसके पूर्णांकों का योग 0 है? एक उदाहरण। माना
S= {4, 8, -2 ,11, -3, 1, 21, 99}, तो उपसमुच्चय {4, -2, -3, 1} जिसके पूर्णांकों का योग 0 है। लेकिन आम तौर पर किसी S के किये ऐसा उपसमुच्चय निकालना चुनौतीपूर्ण है।ध्यान दें कि यदि S के सभी उपसमुच्चयों की जाँच करते रहें, तो इसमें घातीय समय लगेगा क्योंकि 2^n उपसमुच्चय हैं। इस प्रश्न के लिए बहुपद समय एल्गोरिदम ढूँढना एक मिलियन डॉलर का प्रश्न है।

6. 2-SAT बनाम 3-SAT


लेविन कुक

हमारा अंतिम मिलियन डॉलर का उदाहरण बूलियन फॉर्मूला पर आधारित है। इसे हम SAT के नाम से जानते हैं। SAT प्रश्न 1971 में स्टीफन कुक और लियोनिद लेविन द्वारा दिया गया था। यह पहला प्रश्न था जिसके लिए बहुपद समय एल्गोरिथ्म संभव नहीं लगता है। हम जानते हैं कि बूलियन सूत्र में बूलियन चर (variable) होते हैं, और AND (∩), OR (U), NOT (-) संचालन (operation) होते हैं। हमारा कार्य चर को TRUE या FALSE मान निर्दिष्ट करना है ताकि बूलियन सूत्र TRUE हो। जब भी यह संभव होता है, हम कहते हैं कि बूलियन सूत्र सन्तुष्टियोग्य (satisfiable) है; अन्यथा असंतुष्टियोग्य (unsatisfiable). हम दो विशेष प्रकार के बूलियन सूत्र लेंगे।

3-SAT: यह फॉर्मूला खंडों का एक सीमित सेट है, जहां प्रत्येक खंड तीन चर का है, जिनके बीच OR संचालन है। किन्हीं दो खंडों के बीच में AND संचालन होता है। एक उदाहरण।

$$(\bar{X}_1 \cup X_2 \cup X_3) \cap (X_1 \cup \bar{X}_2 \cup X_3) \cap (X_1 \cup X_2 \cup X_3)$$

2-SAT: यह फॉर्मूला खंडों का एक सीमित सेट है, जहां प्रत्येक खंड तीन चर का है, जिनके बीच OR संचालन है। किन्हीं दो खंडों के बीच में AND संचालन होता है। एक उदाहरण।

$$(X_1 \cup \bar{X}_2) \cap (\bar{X}_1 \cup \bar{X}_2) \cap (X_1 \cup X_2)$$

n चर पर किसी भी 2-सैट सूत्र की संतुष्टि को बहुपद समय में जांचा जा सकता है। दुर्भाग्य से n चर पर किसी भी 3-सैट सूत्र की संतुष्टि की जांच करने के लिए कोई बहुपद समय एल्गोरिथ्म नहीं है। यदि आपने **3-SAT** के लिए एक बहुपद समय एल्गोरिदम विकसित किया है तो आपको मिलियन डॉलर मिलेंगे।

निष्कर्ष: हमारे कंप्यूटर में अपार शक्ति और संभावनाएं हैं, इसलिए हमारा वर्तमान और भविष्य काफी हद तक उन पर निर्भर है। लेकिन हमारे कंप्यूटर की सीमाएँ हैं। ऐसे प्रश्न हैं जिनके समाधान में हम अपने कंप्यूटर का उपयोग नहीं कर सकते। इसके अलावा ऐसे प्रश्न भी हैं जिनके समाधान के लिए कंप्यूटर का उपयोग किया जा सकता है लेकिन हम व्यावहारिक रूप से कुशल एल्गोरिदम की उम्मीद नहीं कर सकते हैं। अगर इन सवालों में से किसी के लिए भी बहुपद अल्गोरिथम मिलती तो बाकी सारे ऐसे सवालों के लिए भी बहुपद अल्गोरिथम मिल जायेगी। इस तथ्य की बात हम अगले अंक में करेंगे।

उद्धरण (References)

1. **PCP प्रश्न:** http://www.cs.columbia.edu/~aho/cs3261/Lectures/L17-PCP.html
2. **किसी शहर से सबसे छोटा रास्ता:** https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e
3. **क्या हम प्रत्येक मानचित्र को सिर्फ 4 रंगों से रंग सकते हैं?** https://mathworld.wolfram.com/FourColorTheorem.html#:~:text=The%20four%2Dcolor%20theorem%20states,conjectured%20the%20theorem%20in%201852.
4. **यूलर दौरा बनाम हैमिल्टनियन दौरा :** https://stackoverflow.com/questions/3269013/difference-between-hamiltonian-path-and-euler-path
5. **उपसमुच्चय योग प्रश्न**: https://archive.org/details/algorithmdesign0000klei
6. 3-SAT **बनाम** 2-SAT: https://cs.stackexchange.com/questions/40390/the-relation-between-2sat-and-3sat
7. P **बनाम NP:** https://www.claymath.org/wp-content/uploads/2022/06/pvsnp.pdf