Dodatak

Primjena računara

Sadržaj

Praktičan primjer pretrage igra riječi ... 4

A* Pathfinding za Početnike ... 4

Uvod: Područje pretrage ... 5

Početak traženja ... 5

Utrošak staze ... 6

Nastavak traženja ... 7

Rezime A* metode ... 11

Računska i računarska teorija složenosti ... 12

Algoritmi ... 12

Podjela algoritama prema paradigmi programiranja ... 12

Podjela prema složenosti ... 13

Podjela prema izračunljivosti ... 14

Koliko je neki algoritam brz? ... 14

Računska teorija složenosti ... 15

Problemi odluke ... 16

Računski resursi ... 16

Klase složenosti ... 17

Neukrotivost ... 17

Računska i računarska teorija složenosti ... 18

Uvod ... 18

Otvorena pitanja ... 18

P-problemi ... 18

NP-problemi ... 20

P = NP pitanje ... 21

Nepotpuni problemi u *NP* ... 21

NP* = *co*-*NP ... 21

Imitacija dijaloga ... 22

Rješavanje svih varijanti problema ... 22

Ekstrapolacija ... 22

Učenje ... 23

Mogućnost učenja ... 23

Osnove teorije vjerovatnoće i kombinatorike 24
Uvod 24
Prostor uzoraka 25
Aksiomski koncept vjerovatnoće 25
Ishod i vjerovatnoća presjeka i unija događaja 26
Bajesova teorema 27
Principi kombinatorike 27
Permutacije 28
Varijacije 28
Kombinacije 29
Raspodjele vjerovatnoće 29
Binomna raspodjela 29
Primjer 1. 30
Primjer 2. 30
Puasonova raspodjela 30
Uslovna vjerovatnoća 31

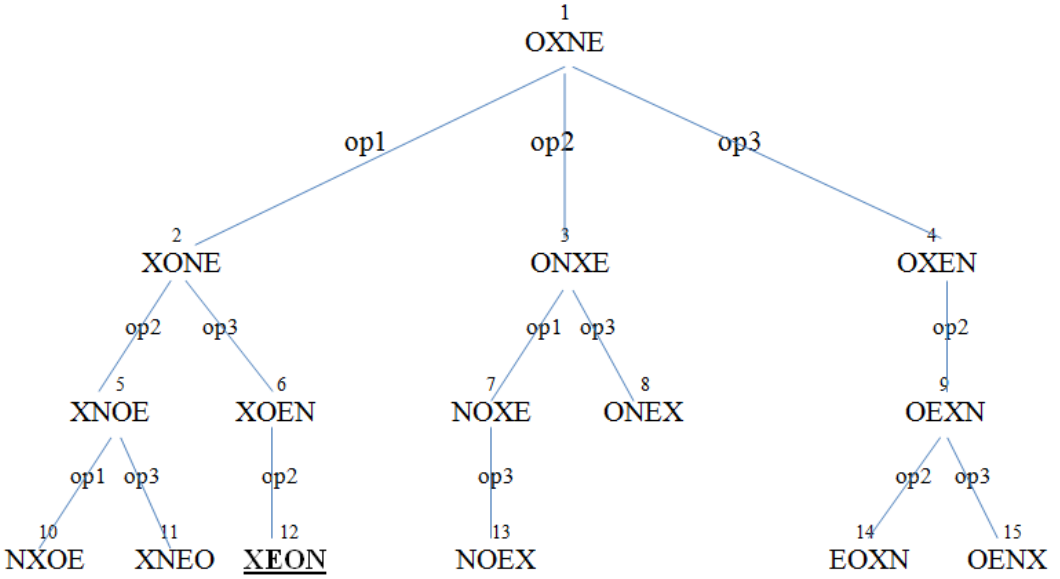
Praktičan primjer pretrage igra riječi

Dragoslav Čikarić SEMINARSKI RAD SINTAKSNA PRETRAGA PROSTORA STANJA "IGRA REČI"

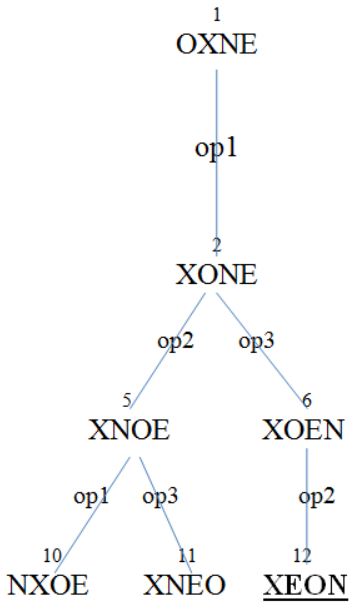
Slaganje slova:

Uzet je slučajni niz slova iz kojeg je potrebno formirati konkretnu reč.
Početno stanje je: OXNE
Ciljno stanje je: XEON
Operatori:
Zadatak koristi tri operatora u ovom primjeru:
Op1: zameni 1,2
Op2: zameni 2,3
Op3: zameni 3,4

Pretraga u širinu:



Pretraga u dubinu:



A* Pathfinding za Početnike

od Patrick Lester (Update-ovano 18. Jula, 2005)

Preveo i adaptirao sa engleskog jezika: Aleksandar Božinović. Sva pitanja možete poslati na e-mail adresu alexbozinovic@yahoo.com

---

Ovaj članak ne pokušava da bude definitivan rad na subjekat (naslov) teme. umjesto toga, članak opisuje osnove i priprema vas za sljedeći korak, da biste razumeli i drugu materiju, a koja je usko povezana sa ovom. Na dnu strane ovog članka postavljeni su linkovi ka nekim od tih već pomenutih „nastavaka".

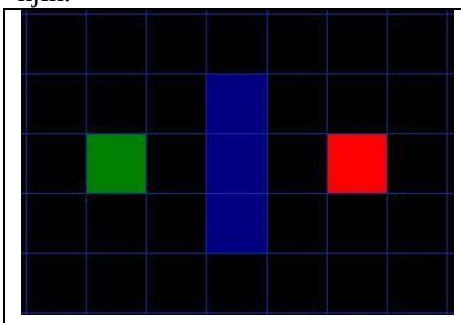
Ovaj članak nije predodređen samo za jedan programski jezik. Napravljen je tako da se adaptira bilo kom programskom jeziku. Kao što i očekujete, u ovom članku naći ćete i link ka programu koji je primjer, na kraju ovog članka. Primjerak sadrži dvije verzije: jedna je u C++-u, a jedna u Blitz Basic-u. Takođe sadrži i executable's (egzekutabilne) fajlove, za slučaj da želite da vidite A* u akciji.

Malo smo se udaljili od suštine. Hajde da počnemo, naravno, otpočetka...

Uvod: Područje pretrage

Pretpostavimo da imamo nekoga ko želi da ide od tačke A do tačke B.

Pretpostavimo da se zid nalazi između ovih tačaka. Ovo je ilustrovano ispod, zelenom je obeležena početna tačka A, crvenom završna tačka B, a plavom su obojeni kvadrati koji predstavljaju zid između njih.



[Figure 1]

Prva stvar koju treba primetiti je da smo podjelili naše područje pretrage na kradratnu mrežu.

Pojednostavljenje područja pretrage je prvi korak u pronalaženju staze (pathfinding). Ovaj metod smanjuje naše područje pretrage na jednostavne dvije dvodimenzionalne matrice. Svaka stavka u matrici predstavlja jedan kvadrat u mreži i sadrži status, to jest obeležava kvadrat kao prohodan ili neprohodan. Staza se nalazi otkrivanjem kojim putem treba ići (preko kojih kvadrata treba preći) da bi se stiglo od tačke A do tačke B.

Jednom kad se staza nađe, naša osoba (koju smo ranije zamislili) se pomera od centra jednog kvadrata do centra sljedećeg i tako dalje, sve dok ne stigne do željenog mesta.

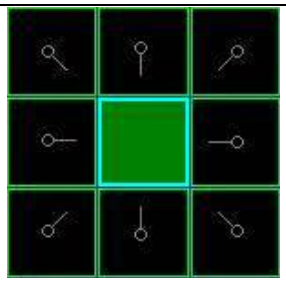
Ovi centri (kvadrata) se nazivaju „čvorovi". Kad čitate o pronalaženju staze (pathfinding) negde drugde, često ćete videti da ljudi raspravljaju o čvorovima. Zašto ih ne zvati samo kvadratima? Zato što je moguće da tu oblast, na kojoj tražite stazu, podjelite i na druge oblike. Mogu biti podjeljeni na pravougaonike, šestougaonike, trouglove, ili zaista, bilo koji oblik. Što se tiče čvorova, oni mogu biti postavljeni bilo gdje po tom obliku, može u centru, duž ivica, ili bilo gdje drugde. Međutim, mi koristimo ovaj sistem jer je najjednostavniji.

Početak traženja

Kada smo pojednostavili područje pretrage na određeni broj čvorova, kao što smo uradili to iznad, sljedeći korak je pokretanje pretrage da bismo našli najkraći put (stazu). Uradićemo ovo počevši od tačke A, proverimo susedne kvadrate i generalno tražimo sve dok ne nađemo metu.

Počinjemo pretragu tako što radimo sljedeće:

1. Počinjemo kod početne tačke A i dodajemo je na „otvorenu listu" kvadrata koja će se posmatrati. Otvorena lista je ista kao i shopping lista (lista za kupovinu). Trenutno je jedna stavka na listi, ali ćemo imati još stavki kasnije. Otvorena lista sadrži kvadrate koji se mogu naći duž staze koju ćemo odabrati, ali možda i neće. U suštini, ovo je lista kvadrata koji treba da se provere.
2. Usmerićemo pogled ka dostupnim, to jest, prohodnim susednim kvadratima početnog kvadrata, ignorišući kvadrate sa vodom, zidovima i drugim neprohodnim terenima. Dodaćemo ih na otvorenu listu (dodajemo samo prohodne kvadrate!). Za svaku od ovih kvadrata (na otvorenoj listi), sačuvaćemo tačku A kao njegov (kvadratni) roditelj. Ovaj izraz „kvadratni roditelj" ili „roditelj kvadrata" biće veoma važan kada budemo pratili našu stazu. Biće detaljnije objašnjen kasnije.
3. Ispustićemo početni kvadrat A sa naše otvorene liste i dodati ga u „zatvorenu listu" kvadrata za koje nema potrebe da ponovo pogledate za sada.

 *[Figure 2]*	U ovom momentu objašnjavanja slijedi ilustracija. Na njoj, zeleni kvadrat u sredini je početni kvadrat. On je uokviren plavom bojom koja označava da je kvadrat dodat na zatvorenu listu. Svi susedni kvadrati su sada na otvorenoj listi kvadrata koji treba da se provere i oni su uokvireni zelenom bojom. Svaki ima i zeleni pokazivač koji pokazuje nazad na svog roditelja, koji je, u ovom slučaju, početni kvadrat. Sada, odabiramo jedan od susednih kvadrata sa otvorene liste i, manje-više, ponavljamo prethodni proces, kao što je opisano ispod. Pitanje je sad: „Koji kvadrat da izaberem?“ Onaj sa najmanjim F utroškom.

Utrošak staze

Ključ, koji određuje koji će kvadrat da se koristi da bi se otkrila staza, određen je sljedećim računanjem:

F = G + H

Gde su:

- G = utrošak pomeranja za kretanje od početne tačke A do datog kvadrata na mreži, prateći određenu stazu da ode tamo.
- H = utrošak pomeranja za kretanje od tog datog kvadrata na mreži do krajnje destinacije, tačke B. Ovo je često referisano kao heuristično (po Vujakliji – heuristički: pronalazački, (analitički ili genetički) *heuristički metod* je put koji vodi ka pronalaženju naučnih istina), malo je zbunjujuće. Mi zaista ne znamo pravu distancu dok ne nađemo stazu jer put može imati mnogo prepreka (zidovi, voda, itd). U ovom tutorijalu vam je dat jedan od načina da se izračuna H, ali postoje i mnogi drugi koje možete naći na netu.

Kao što je opisano iznad, G je utrošak pomeranja za kretanje od početne tačke do određenog kvadrata, idući po stazi koja je stvorena (generisana) da se dođe do tu. U ovom primjeru, dodijelićemo utrošak od 10 za svako horizontalno ili vertikalno kretanje kvadrata, a za dijagonalno kretanje utrošak
će biti 14. Koristićemo ove brojeve jer je stvarna distanca dijagonalnog kretanja jednaka korjenu iz 2 (), što je jednako negde oko 1.414, što predstavlja utrošak dijagonalnog kretanja. Da bi stvari bile jednostavnije, umjesto 1 ćemo koristiti 10, a umjesto 1.414 ćemo uzeti 14. Otprilike je srazmerno, a ujedno smo se oslobodili decimala, dakle pojednostavili smo stvari.

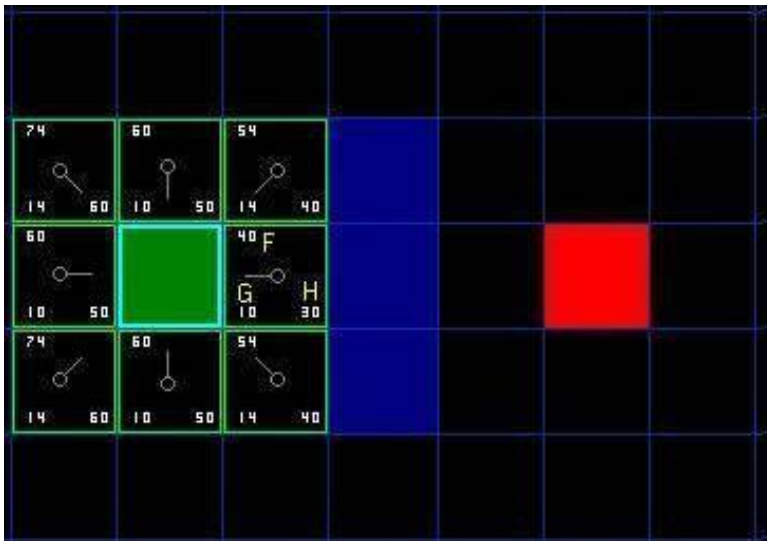
Pošto računamo G utrošak duž staze do datog kvadrata, način za određivanje G utroška tog kvadrata je da se uzme G utrošak od njegovog roditelja, pa se onda doda 10 ili 14 u zavisnosti da li se pokrenuo dijagonalno ili ortogonalno (nedijagonalno) od tog kvadrata roditelja.Potreba za ovom metodom biće očigledna malo kasnije na ovom primjeru.

H utrošak može biti procjenjen na razne načine. Metod koji ovde koristimo se zove Menhetenov metod (Manhattan), gdje računate totalni broj kvadrata kojim se pokreće horizontalno i vertikalno da bi se dostigla meta (željeni kvadrat) od kvadrata u kojem se trenutno nalazi, ignorišući dijagonalne pokrete, i ignorišući bilo koje prepreke koju mogu biti na putu. Onda pomnožite ovaj broj sa 10 (to jest, utroškom jednog horizontalnog ili vertikalnog kretanja), te tako dobijate H utrošak. Podsećam vas da se prilikom računanja ide samo horizontalno i vertikalno, to jest ortogonalno.

Čitajući ovaj opis, nagađate da je heuristički način gruba procjena preostalog rastojanja između trenutnog kvadrata i mete. To realno nije tako. Ustvari, mi pokušavamo da procjenimo preostalo rastojanje duž staze (koje je često duže). Što je naša procjena bliža stvarnoj razdaljini, to je i algoritam brži. Ako precenimo ovu razdaljinu, kako god, nije garantovano da ćemo dobiti najkraću stazu. U takvim slučajevima, imamo nešto što se zove „neprihvatljiva heuristika“.

Tehnički, u ovom primjeru, Menhetenov metod je nedopustiv jer precenjuje preostalo rastojanje. Mi ćemo se ipak služiti ovom metodom jer je mnogo lakše da se razume i zadovoljava naš cilj, ali i zato jer je to samo blago (dakle, do određene mere) precenjivanje. U retkim slučajevima, kada rezultujuća staza nije nakraća moguća, biće blizu najkraće (malo duža od najkraće).

Utrošak F se izračunava sabiranjem G i H utroška (F = G + H). Ovi rezultati prvog koraka naše pretrage mogu se videti na ilustraciji ispod. F, G i H utrošci su napisani u svakom kvadratu. Kao što je označeno na prvom desnom kvadratu (od početnog zelenog), F je otštampano u gornjem levom uglu, G je u donjem levom uglu, a H je otštampan u donjem desnom uglu.



[Figure 3]

Bacite pogled na neke od ovih kvadrata. Na desnom kvadratu sa slovima unutar, G = 10. Ovo je zato što je samo jedan kvadrat od početnog u horizontalnom pravcu. Kvadrat iznad, zatim ispod, kao i onaj s leve strane početnog, svi oni imaju G koje je jednako 10. Dijagonalni kvadrati imaju G koji je jednak 14.

H utrošak jer izračunat procjenom Menhetenove distance (rastojanja) do crvenog kvadrata, pomerajući se samo horizontalno i vertikalno, ignorišući zid koji je na putu. Koristeći ovu metodu, kvadrat s desne strane početnog kvadrata je udaljen 3 kvadrata od crvenog kvadrata, stoga mu je H utrošak jednak 30. Kvadrat iznad početnog kvadrata je 4 kvadrata udaljen od crvenog (zapamtite, samo se pomera horizontalno i vertikalno), te za H utrošak ima 40. Možete i sami da vidite kako su ostali H-ovi izračunati za druge kvadrate.

F utrošak, za svaki, da ponovimo, je ‚jednostavno, jednak zbiru G i H.

Nastavak traženja

Da bismo nastavili traženje, jednostavno ćemo izabrati kvadrat sa najnižim F-om od svih kvadrata sa otvorene liste.

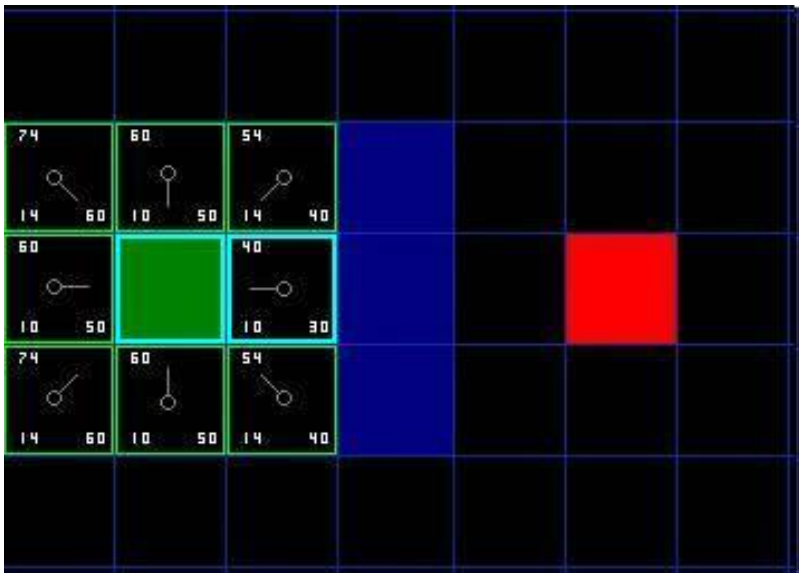
Onda ćemo uraditi sljedeće sa odabranim kvadratom:

4) Ispustićemo ga sa otvorene liste i dodajemo ga na zatvorenu listu.

5) Onda ćemo da proverimo sve susedne kvadrate tog odabranog kvadrata (prilikom čega ignorišemo kvadrate koji su na zatvorenoj listi ili su neprohodni, to jest, tereni sa zidom ili vodom...) i dodati ih na otvorenu listu, ako već nisu na njoj. Načinićemo odabran kvadrat roditeljom novih kvadrata.

6) Ako je neki susedni kvadrat već na otvorenoj listi, proverimo da li je staza od ovog kvadrata (ovo nije gramatička greška jer se ne radi o pripadnosti staze kvadratu, već o početku staze odatle, dakle od...) ka cilju bolja. Drugim rečima, proverimo da li je G utrošak tog kvadrata manji ako koristimo trenutni kvadrat da stignemo tamo. Ako nije, ne činimo ništa. U drugu ruku, ako je G nove staze manji, treba da se promeni roditelj susednog kvadrata na novo odabrani kvadrat (u dijagramu iznad, treba promeniti smer pokazivača ka tački novog selektovanog kvadrata). Najzad, ponovo izračunajte F i G tog kvadrata. Ako ovo izgleda zbunjujuće, videćete to ilustrovano ispod.

Ok, pa, hajde da vidimo kako ovo radi. Od naših 9 inicijalnih (početnih) kvadrata, imamo 8 postavljenih na otvorenu listu, pošto je početni kvadrat prebačen na zatvorenu listu. Od ovih na otvorenoj listi, kvadrat sa najmanjim F-om je sa desne strane od početnog kvadrata sa F = 40, tako da ćemo ga odabrati da bude sljedeći. Uokviren je plavom bojom na sljedećoj ilustraciji.



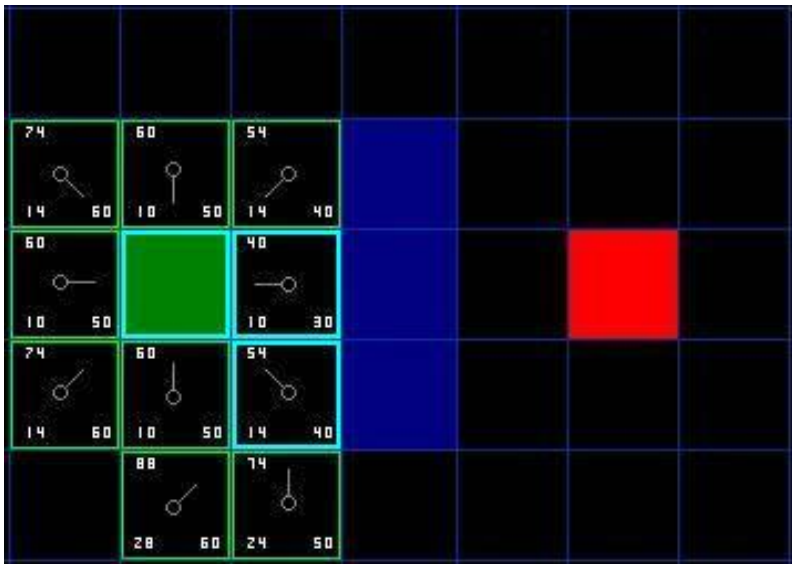
[Figure 4]

Prvo ga spustimo sa otvorene liste i dodajemo ga na zatvorenu listu (zbog toga je uokviren plavom bojom). Onda proverimo susedne kvadrate. Ovi sa desne strane su kvadrati zidovi, pa ih ignorišemo. Onaj sa leve strane je početni kvadrat, a s obzirom da je na zatvorenoj listi, ignorisaćemo i njega takođe. Ostala četiri kvadrata su već na otvorenoj listi, te treba da proverimo da li je staza (počevši) od njih imalo bolja, koristeći G kao tačku reference. Pogledajmo kvadrat desno iznad našeg selektovanog kvadrata. Njegov G je 14. Ako smo umjesto onog puta, prošli kroz ovaj kvadrat da bismo tamo stigli, G bi bio jednak 20 (10, što je G utrošak da se stigne do ovog kvadrata, plus još 10 da se pokrene vertikalno do onog iznad njega). G utrošak 20 je veći od 14, tako da ovo nije bolja staza. To bi imalo smisla ako pogledate na dijagram. Usmerenije je otići do tog kvadrata od početnog kvadrata ako se jednostavno pomeri za jedan kvadrat dijagonalno da dođe tamo, nego da se pomeri horizontalno za jedan kvadrat, pa onda vertikalno za jedan kvadrat.

Kada ponovimo ovaj proces, za sva četiri susedna koja su već na otvorenoj listi, videćemo da nijedna od ovih staza nije poboljšana (to jest bolja od prethodne), tako da nećemo mijenjati išta. Sada, kad smo pregledali sve susedne kvadrate, završili smo sa ovim kvadratom i spremni smo da se pomerimo na sljedeći kvadrat.

Kako idemo kroz otvorenu listu kvadrata, koja za sada ima sedam kvadrata, odabraćemo jedan sa najnižim F-om, odnosno u ovom slučaju, imamo 2 kvadrata sa F-om od 54. Koji ćemo onda da izaberemo? Zaista nije važno. Ako će nam kao razlog odabira biti brzina, onda bi najbrže bilo da se izabere onaj koji je poslednji dodat na otvorenu listu.

Hajde da izaberemo onaj ispod, a sa desne strane od početnog kvadrata, kao što je i prikazano na sljedećoj ilustraciji.

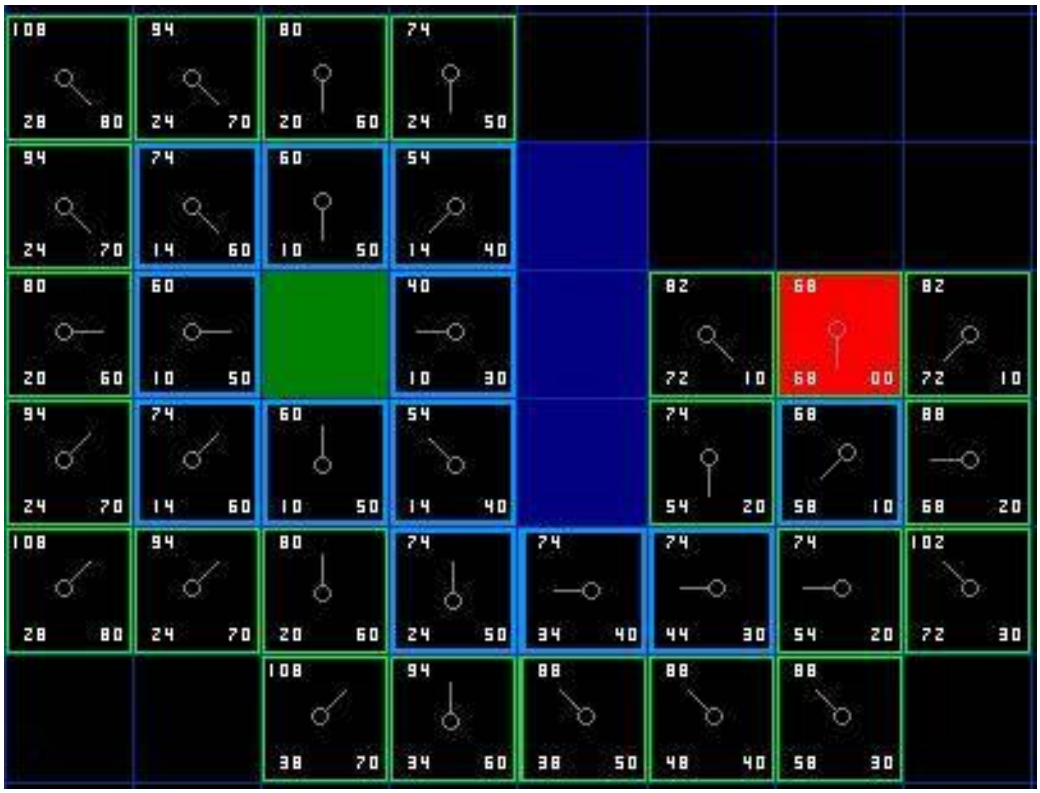


[Figure 5]

Ovog puta, kadaproveravamo susedne kvadrate, nalazimo da li je onaj s desne strane kvadrat zid, pa ga ignorišemo. Isto ide za onaj koji je iznad tog. Takođe ignorišemo kvadrat koji je ispod zida. Zašto? Zato što se ne može doći do tog kvadrata direktno od trenutnog kvadrata (na kom smo sada), bez usecanja kroz ugao obližnjeg zida. Stvarno treba da idete pravo dole, a onda da pređete preko tog kvadrata, zaobilaženjem ugla u procesu. (Primedba: Ovo pravilo o zasecanju uglova je opcionalno (fakultativno, neobavezno. Njegova primjena zavisi od toga kako su čvorovi postavljeni).

To ostavlja 5 drugih kvadrata. Druga dva kvadrata ispod trenutnog kvadrata nisu još uvek na otvorenoj listi, pa ćemo ih dodati, te će trenutni kvadrat postati njihov roditelj. Od tri drugih kvadrata, dva su već na zatvorenoj listi (početni kvadrat i onaj iznad trenutnog su uokvireni plavom bojom na dijagramu), pa ih ignorišemo. Poslednji kvadrat, s leve strane trenutnog kvadrata, proveravamo da bismo Vidjelii da li je njegov G manji ako idemo kroz trenutni kvadrat da stignemo tamo. Izgleda da nije. Gotovi smo i spremni da proverimo sljedeći kvadrat na otvorenoj listi.

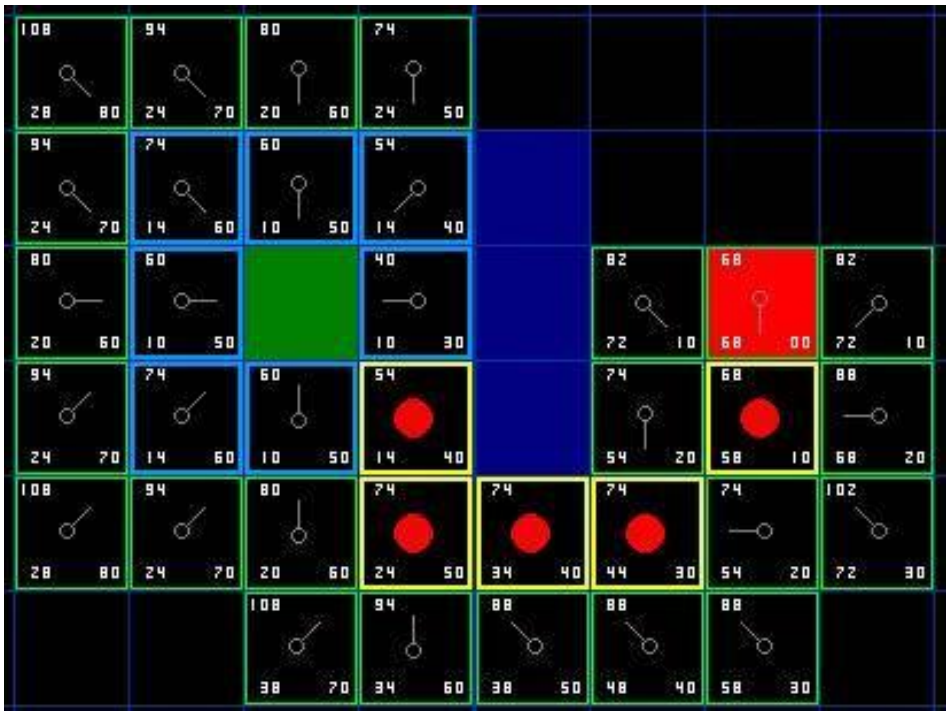
Ponovićemo ovaj proces sve dok ne dodamo završni kvadrat na zatvorenu listu, što je ilustrovano ispod.



[Figure 6]

Opazite da se kvadrat roditelj kvadrata, koji je dva kvadrata ispod početnog kvadrata, promenio od prošle ilustracije. Ranije je imao G utrošak od 28 i pokazivao nazad na kvadrate iznad njega i s desne strane. Sada ima vrednost 20 i pokazuje ka kvadratu koji je iznad njega. Ovo se dešava negde duž puta na našoj pretrazi, gdje se G proverava i promijenjen je u nižu vrednost koristeći novu stazu – tako da je roditelj zamjenjen, a G i F utrošci se ponovo izračunavaju. Iako ova promena ne izgleda kao tako značajna na ovom primjeru, postoje brojne situacije gdje će ovo konstantno proveravanje da napravi razliku u određivanju najbolje staze ka svom cilju.

Kako ćemo onda odrediti stazu? Jednostavno, samo treba da počnemo od crvenog kvadrata i da vršimo kretanje unazad od kvadrata ka njegovom roditelju, prateći strelicu (pokazivač). Ovo će vas na kraju dovesti nazad do početnog kvadrata, i to je vaša staza. Treba da liči na sljedeću ilustraciju. Pomerajući se od početnog kvadrata A do destinacionog kvadrata B je, ustvari, kretanje od centra do centra svakog kvadrata (čvora) na stazi, dok se ne dostigne meta.



[Figure 7]

Rezime A* metode

Ok, sada kad ste prošli kroz objašnjenje, hajde da postavimo korap po korak ovu metodu sve na jedno mesto:

1. Staviti početni kvadrat (ili čvor) na otvorenu listu.
2. Ponoviti sljedeće:

a. Pronaći kvadrat sa najnižim F utroškom na otvorenoj listi. Odnositi se na taj kvadrat kao na trenutni kvadrat.

b. Premestiti trenutni kvadrat na zatvorenu listu.

c. Za svaki od 8 susednih kvadrata ovog kvadrata:

- Ako nije prohodan ili ako je na zatvorenoj listi, ignorišemo ga. U suprotnom, učiniti sljedeće:
- Ako nije na otvorenoj listi, dodajte ga na listu. Načiniti trenutni kvadrat roditeljom ovog kvadrata. Sačuvajte F, G i H utroške kvadrata.
- Ako je već na otvorenoj listi, proveriti da li je staza do ovog kvadrata bolja, koristeći G utrošak kao meru odlučivanja. Niži G utrošak znači da je staza bolja. Ako je tako, načiniti taj kvadrat roditeljom i ponovo mu izračunati G i F utroške. Ako na otvorenoj listi držite F utroške, onda je potrebno da resortirate listu.

d. Zaustaviti se kada:

- se doda završni kvadrat na zatvorenu listu, kada je staza pronađena, ili
- ne pronađe zavrpni kvadrat i kada je otvorena lista prazna. U tom slučaju, nema staze.

3. Sačuvati stazu. Treba ići unazad od završnog kvadrata, ići od svakog kvadrata do njegovog roditelja sve dok se ne dosegne početni kvadrat. To je vaša staza.

Za **bolje razumjevanje pogledjte i youtube video**:
A* Pathfinding Tutorial - Part 1
http://www.youtube.com/watch?v=KNXfSOx4eEE
A* Pathfinding Tutorial - Part 2:
http://www.youtube.com/watch?v=BG7Bc3C-W-k

Računska i računarska teorija složenosti

Najvećim dijelom preuzeto iz članka P=NP? Vedran Šego

Da bismo mogli razmatrati algoritme koji rješavaju pojedine probleme, potrebno je prvo reći šta je to uopšte algoritam. Najkraće rečeno, algoritam je popis jednoznačnih uputa koje treba slijediti da bismo u konačnom vremenu rješili problem iz neke zadane klase problema (ili, uopšte, obavili neki posao). Ovdje je bitno naglasiti:

1. konačnost niza uputa, da bismo algoritam uopšte mogli zapisati;
2. do rješenja treba doći u konačnom vremenu, tj. algoritam se ne smije vječno izvršavati;
3. algoritam rješava probleme iz određene klase problema, dakle, čak i ako ga možemo "zaposliti" s nekim problemom za koji on nije predviđen, ne možemo očekivati da će ga korektno riješiti.

Pogledajmo primjer iz matematike: zadana je jednadžba $ax^2 + bx + c = 0$.

za neke zadane $a,b,c \in R$. Trazi se kompleksni broj $x \in C$ koji zadovoljava zadanu jednadžbu. Jednostavno, ali i pogresno rjesenje ovog problema bi bilo:

Jednadžba $ax^2 + bx + c = 0$, pri čemu su $a,b,c \in R$, u skupu kompleksnih brojeva ima dva rješenja, data formulom $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

Prikazana formula je poznata za rješavanje kvadratnih jednadžbi, no u njoj se kriju zamke koje - kad računamo "na prste" - obično zaobiđemo "u hodu".

No i ta zaobilaženja treba ugraditi u algoritam:

1. Ako je $a = 0$, jednadžba nije kvadratna, a primjena gornje formule rezultirala bi dijeljenjem s nulom. Tada imamo slučajeve:
   (a) Ako je $b = c = 0$, svaki kompleksni broj $x \in \mathbb{C}$ je rješenje zadane jednadžbe.
   (b) Ako je $b = 0$ i $c \neq 0$, zadana jednadžba nema rješenja.
   (c) Ako je $b \neq 0$, zadana jednadžba ima jedinstveno rješenje
   $$x = -\frac{c}{b}.$$
2. Ako je $a \neq 0$ i $b^2 - 4ac = 0$, zadana jednadžba je kvadratna, ali ima jedinstveno rješenje
   $$x = -\frac{b}{2a}.$$
3. Ako je $a \neq 0$ i $b^2 - 4ac \neq 0$, zadana jednadžba je kvadratna i ima dva rješenja, dana formulom
   $$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Tek sada imamo jednoznačnu i tačnu proceduru rješavanja zadane jednadžbe koja će uvijek u konačno mnogo koraka dati ispravno rjesenje zadanog problema.

Podsjećamo:Algoritam je opis za rješavanje nekog problema.

U novije vreme, algoritam je pojam koji se gotovo isključivo vezuje za informatiku i, mada ne postoji jedinstvena opšteprihvaćena definicija, podrazumjeva se da je u pitanju nekako opisana procedura za obavljanje posla.

Podjela algoritama prema paradigmi programiranja

Jedan način razvrstavanja je po metodologiji projektovanja ili primijenjenom obrascu. Postoji izvestan broj različitih obrazaca kako se pristupa realizaciji algoritama. Dalje, svaka od navedenih kategorija sadrži više različitih tipova algoritama. Neki uobičajeno korišćeni obrasci su:

- **Podjeli pa vladaj** algoritmi smanjuju stepen složenosti problema podjelom na dva ili više manjih problema od iste vrste (obično rekurzivno), dok od problema ne ostane toliko mali deo da se može jednostavno rešiti.

- **Dinamičko programiranje**. Kada problem pokazuje optimalnu podstrukturu, u smislu da se optimalno rješenje problema može konstruisati iz optimalnog rješenja potproblema, i preklapanjem potproblema, što znači da se isti potproblem koristi za rješavanje više različitih primjera problema, možemo rešiti brzo koristeći *dinamičko programiranje*, pristup koji izbegava ponovno izračunavanje rješenja koja su već izračunata. Na primjer, najkraći put do cilja iz čvora težinskog grafa može biti nađen koristeći najkraći put do cilja od svih obližnjih čvorova.
- **Pohlepni algoritam**. Algoritam lakomosti je sličan dinamičkom programiranju, ali je razlika u tome što rješenja potproblema ne moraju biti poznata u svakom trenutku. Stoga, pri traženju rješenja je moguće napraviti i 'lakom' izbor onoga što izgleda najbolje u tom trenutku.
- **Linearno programiranje**. Problem se rešava korišćenjem linearnog programiranja kada postoji više linearnih nejednačina a zadatak je naći maksimum (ili minimum) po nekom kriterijumu. Mnogi realni problemi (kao što je maksimiziranje protoka u usmerenom grafu) mogu biti iskazani na ovaj način, a onda rešeni nekim 'generičkim' algoritmom, kao što je Simpleks algoritam.
- **Pretraga i numeracija**. Mnogi problemi (kao što je igranje šaha) mogu biti modelovani kao problemi grafa. Algoritam pretraživanja grafa daje pravila kretanja kroz graf i koristan je baš za ovakve probleme. Ova kategorija obuhvata i algoritme pretraživanja i povratka kroz stablo odlučivanja (bektreking).
- **Heuristički algoritmi** i **algoritmi slučajnosti** ne odgovaraju u potpunosti strogoj definiciji algoritma
    1. Algoritmi slučajnosti prave u nekim situacijama slučajan (ili pseudo slučajan) izbor; za neke probleme se stvarno može dokazati da se do najbržeg rješenja može doći samo uvođenjem izvesnog stepena slučajnosti.
    2. Genetički algoritam pokušava da nađe rješenje problema imitirajući biološki evolucioni proces, koji u nizu slučajnih mutacija daje uzastopne generacije 'rješenja'. Tako računar simulira razmnožavanje i 'preživljavanje najprilagođenijih'. U genetičko programiranje je ovaj pristup proširen na algoritme.
    3. Heuristički algoritmi su takvi algoritmi čija je osnovna namena nalaženje optimalnog rješenja, u stvari približnog rješenja, jer vreme ili memorijski prostor za izvršavanje algoritma za nalaženje tačnog najboljeg rješenja nije praktično moguće. Primjer algoritama koji su ovakvog tipa su za lokalno pretraživanje, tabu pretraživanje ili algoritam simuliranog otpuštanja, vrsta heurističkog algoritma slučajnosti koji varira rješenje problema u slučajnim iznosima. Naziv simulacija otpuštanja aludira na metalurški proces suprotan kaljenju kada se metal greje pa sporo hladi radi otklanjanja defekta u materijalu. Namera slučajnog variranja je traženje što bližeg rješenja opštem optimalnom rješenju, a ne jednostavno lokalno optimalno rješenje. Ideja je da se amplituda slučajne veličine smanjuje kako se približavamo rješenju.

Podjela prema složenosti

U teoriji složenosti, što nije isto što i teorija izračunljivosti, se izučava problematika složenosti, kompleksnosti, algoritma, u smislu zauzimanja resursa, a to su prostor (količina zauzete memorije) i vreme (količina potrošenog vremena procesora). Složenost je funkcija veličine ulaznih podataka. Algoritmi se prave za rješenje opšteg problema, bez obzira na veličinu ulaza, ali sa druge strane razne ulazne veličine izazivaju da programi troše razne količine resursa.

Vremenska složenost algoritma se iskazuje kao broj elementarnih koraka za obavljanje algoritma, što je zavisno od veličine ulaza, a koja može biti izražena u bitovima ili nekim sličnim merilom. Ako kažemo da je algoritam (A) uređivanja niza od n elemenata problem vremenske složenosti n^2, znači da dvostruko veći broj elemenata zahtjeva četiri puta više vremena za uređivanje. Ako je, pak drugi algoritam (B) malo pažljivije napisan i brži je dvostruko, on će raditi dvostruko brže za bilo koju veličinu niza. Međutim, ako se programer namuči i osmisli suštinski drugačiji algoritam (C) za uređivanje, on stvarno može biti reda složenosti $n \cdot log(n)$. Algoritmi (A) i (B) su iste složenosti, jer se u notaciji sa velikim O

obeležavaju sa O(n^2), a u govoru se zovu 'algoritmi kvadratne složenosti', dok je algoritam (C) 'algoritam složenosti *n·log(n)'*. Zaključak: algoritam (C) je najbolji sa stanovišta korišćenja vremena za velike setove ulaznih podataka.
Prostorna složenost se na isti način odnosi na funkciju zavisnosti zauzimanja memorijskog prostora u zavisnosti od veličine ulaza. Dešava se da je pronalaženje algoritma manje vremenske složenosti vezano sa povećanjem prostorne složenosti. Obično se algoritmi dele na one logaritamske složenosti, linearne, kvadratne i uopšteno polinomijalne složenosti, kao i one najzahtjevnije, eksponencijalne složenosti.

Podjela prema izračunljivosti

Algortime je moguće klasifikovati i prema izračunljivosti. Ovo se obično radi tako što se razmatraju klase algoritama što omogućava smanjenje vremenskih i memorijskih resursa koji se koriste u izračunavanju. Na primjer, klasa rekurzivnih algoritama uključuje algoritme za sve funkcije koje se mogu izračunati pomoću Tjurnigove mašine.

Koliko je neki algoritam brz?

Kad se postavi pitanje brzine algoritma, prvo sto nekome padne na pamet obično je "stopericu u ruke i pokrenimo program". Ovakav pristup iz mnogo razloga nije dobar.
Kao prvo, program treba napisati u nekom jeziku. Bitno je naglasiti da implementačija algoritma (u odredenom programskom jeziku, na određenoj arhitekturi računara i sl.) nije isto što i algoritam. Jezici se međusobno razlikuju i po mogučnostima (npr. FORTRAN podržava kompleksne brojeve, dok C i Pascal ne) i po brzini (programi pisani u C-u i Pascalu su opčenito daleko brzi od onih pisanih u C++ i Delphiju). Dapače, mnogi jeziči imaju vise od jednog prevodioča koji se medusobno opet razlikuju (C je posebno poznat po ovome, no slično vrijedi i za Pačal, C++, Perl, SmallTalk i mnoge druge). Zbog svega navedenog, postavlja se pitanje kako usporediti dva algoritma ako imamo njihove programske implementačije napisane u različitim jezicima?
Kao drugo, računari se medusobno razlikuju. Neka imaju brze procesore, memorije, sabirniče; neka imaju više radne memorije ili međumemorije; različita moze biti i programska podrska (npr. operativni sistem); itd. Kako usporediti rezultate dobivene "stopericom" za program koji je testiran na Athlonu na 2.7GHz s 1GB RAM-a pod nekim Linuxom i program testiran na 64-bitnom Pentiumu na 3GHz i s 2GB RAM-a pod Mac OS X-om?
Računari se razvijaju velikom brzinom i novi modeli dolaze na trziste svakodnevno.
Kako bi se izbjegla potreba stalnog biranja idealne platforme za testiranja algoritama (sto bi opet onemogućilo usporedbe starih i novih testova), analiza algoritama bazira se na teorijskom računaru koje je 1937. godine u predstavio Alan Turing. Taj računar zovemo Turingov stroj. Ovdje čemo dati neformalni opis.
Turingov stroj zamišljamo kao računar koji ima beskonačnu traku. Po toj trači čita i pise glava stroja koja se u svakom koraku (izvrsavanju jedne naredbe) moze pomaknuti jedno mjesto lijevo ili desno ili moze ostati na mjestu. Stroj prati i stanje u kojem se nalazi, svojevrstan pandan vrijednos- tima varijabli u programima.
U svakom koraku, Turingov stroj nalazi se u nekom stanju q, te čita s trake (na mjestu na kojem se nalazi glava stroja) znak x. Ovisno o stanju i pročitanom znaku, stroj prelazi u stanje *q'*, na traku zapisuje znak *x',* te se glava vrsi pomak p (za jednom mjesto lijevo ili desno ili ostaje na mjestu). Matematički zapisano, jedan korak je zapravo obično pridruzivanje:

$$(q, x) \mapsto (q', x', p).$$

Kad popisemo sve takve korake, definsali smo Turingov stroj.
Primijetimo: ako su svi parovi (q,x) medusobno različiti, opisani Turingov stroj je zapravo matematička funkcija. To znaCi da ponasanje stroja ovisi isključivo o njegovom stanju i znaku pročitanom na traci. Takve strojeve zovemo deterministički Turingovi strojevi.
Ako se neki od parova (q,x) preslikavaju u vise različitih trojki *(q',x'*,p), opisani skup pravila nije funkcija, nego samo relacija. Za takav stroj kazemo da je nedeterministički. Iako je to naizgled u koliziji sa zahtjevom da koraci algoritma moraju jednoznačno opisivati sto algoritam treba raditi, kod nedeterminističkih Turingovih strojeva pretpostavljamo da će stroj pogoditi ispravni put k rješenju.

Recimo da se stroj nalazi u stanju q, te je s trake pročitao znak x i pravila dopusta ju korake

$$(q, x) \mapsto (q'_i, x'_i, p_i), \quad i = 1, 2, \ldots, n,$$

za neki n > 1. Tada pretpostavljamo da će stroj odabrati upravo takav i da ga prelazak u stanje *qi* uz zapisivanje znaka x'_i i pomak glave *pi* vode prema rješenju.
Zanimljivo je da su nedeterministički Turingovi strojevi ekvivalentni deterministčkima (tj. mogu riješiti iste probleme): dovoljno je napraviti deterministički stroj koji će nekim redom isprobati sve mogućnosti kroz koje bi mogao proći njegov nedeterministički ekvivalent. Naravno, stroj koji isprobava sve mogućnosti obavlja posao puno sporije od onoga koji odmah pogada ispravni put.
Upravo to je bitna razlika izmedu P- i NP-problema.
P- i NP-problemi obično se definišu kao problemi odlučivanja (oni koji daju odgovor na da/ne-pitanje), no u širem kontekstu oni omogućavaju traženja odgovora i na opšta pitanja.

Računska teorija složenosti

Kao grana teorije računanja u računarstvu, **računska teorija složenosti** opisuje skalabilnost algoritama, te inherentnu teškoću u pružanju skalabilnih algoritama za specifične računske probleme.
To jest, teorija odgovara na pitanje, "**Kako se veličina ulaza algoritma povećava, kako se mijenjaju vrijeme izvršavanja i memorijski zahtijevi algoritma**?".
Teorija određuje praktične granice na ono što računari mogu obaviti.

Implikacije teorije su važne vladi i industriji. Brzina i memorijski kapacitet računara uvijek rastu, baš kao i veličine skupova podatka koje treba analizirati. Ako algoritam ne uspije skalirati, tada će velika poboljšanja u računarskoj tehnologiji rezultirati tek u marginalnim povećanjima u praktičnim skupovima podataka.
Algoritmi i problemi su kategorizirani u klase složenosti.
Najvažnije otvoreno pitanje teorije složenosti jest pitanje istovjetnosti klase *P* i klase *NP*, odnosno je li prva podskup druge kao što se općenito smatra.
Daleko od toga da se radi o ezoteričnoj vježbi - ubrzo nakon što je pitanje prvi put postavljeno, shvatilo se da su mnogi važni industrijski problemi u polju operacijskih istraživanja potklasa od *NP* zvana NP-potpuni problemi. NP-potpuni problemi imaju svojstvo da im se rješenja mogu brzo provjeriti, s tim da trenutne metode pronalaženja problema nisu skalabilne. Ako je klasa *NP* veća od *P*, tada ne postoji nijedno skalabilno rješenje za ove probleme. Je li to uistinu tako, je jedno od važnih otvorenih pitanja u računskoj teoriji složenosti.

Teorija se složenosti bavi relativnom računskom teškoćom izračunljivih funkcija. Ovo se razlikuje od teorije izračunljivosti, koja se bavi općenitim pitanjem može li se problem riješiti, bez obzira na zahtijevane resurse.
Jedan "problem" je jedinstven skup povezanih pitanja, pri čemu je svako pitanje string konačne duljine.
Na primjer, problem *FAKTORIZIRAJ* jest: za dani cijeli broj zapisan u binarnom sistemu, vrati sve proste faktore tog broja. Pojedinačno se pitanje zove *instancom*. Na primjer, "daj faktore broja 15" je instanca problema *FAKTORIZIRAJ*.

Vremenska složenost problema je broj koraka potreban da bi se instanca problema riješila kao funkcija veličine ulaza (obično mjerenog u bitovima) koristeći najefikasniji algoritam. Da bi se ovo intuitivno razumjelo, može se razmotriti primjer instance duljine *n* bitova koja može biti riješena u n^2 koraka. U ovom primjeru kažemo da je problem vremenske složenosti n^2. Naravno, točan će broj koraka ovisiti o korištenom stroju. Kako bi se izbjegao taj problem, koristi se veliko O notacija. Ako problem ima vremensku složenost $O(n^2)$ na jednom tipičnom računaru, tada će također imati složenost $O(n^2)$ na većini drugih računara, tako da nam ova notacija omogućava poopćavanje detalja pojedinačnog računara.
Primjer: *Košenje trave ima linearnu vremensku složenost s obzirom da treba dvostruko više vremena kako bi se kosilo dvostruko veće područje. Međutim, binarno pretraživanje rječnika ima svega*

logaritamsku vremensku složenost budući da dvostruko veći rječnik treba biti otvoren tek jedan put više (npr. tačno u sredini - tada se veličina problema smanji za pola).

Prostorna složenost problema je povezan koncept, koji mjeri količinu prostora, ili memorije koju algoritam zahtijeva. Neformalna bi analogija bila količina papira korištenog za skiciranje prilikom rješavanja problema olovkom i papirom. Prostorna se složenost također mjeri velikom O notacijom.
Različita mjera složenosti problema, korisna u nekim slučajevima, jest **složenost sklopa**. Ovo je mjera veličine bulovskog sklopa potrebnog za računanje rješenja problema, u terminima broja logičkih vrata zahtijevanih za izgradnju sklopa. Takva je mjera korisna, na primjer, prilikom izgradnje sklopovskih mikročipova za izračunavanje funkcije, radije nego programske podrške.

Problemi odluke

Veći se dio teorije složenosti bavi problemima odluke. Problem odluke je problem koji uvijek ima odgovor *da* ili *ne*. Teorija složenosti razlikuje probleme koji verificiraju odgovore *da* i one koji verificiraju odgovore *ne*. Problem koji invertira *da* i *ne* odgovore drugog problema se zove komplement tog problema.
Na primjer, poznati problem odluke *IS-PRIME* vraća odgovor *da* kad je dani ulaz prost, a inače *ne* odgovor, dok problem *IS-COMPOSITE* određuje *nije* li dani cijeli broj prost broj (tj. složen broj). Kad *IS-PRIME* vrati *da*, *IS-COMPOSITE* vraća *ne* i obrnuto. Stoga je *IS-COMPOSITE* komplement problema *IS-PRIME*, baš kao što je i *IS-PRIME* komplement problema *IS-COMPOSITE*.
Problemi se odluke često razmatraju budući da proizvoljan problem može uvijek biti sveden na problem odluke. Na primjer, problem *HAS-FACTOR* jest: za dane cijele brojeve *n* i *k* napisane u binarnom sistemu, vrati ima li *n* neke proste faktore manje od *k*. Ako problem *HAS-FACTOR* možemo riješiti korišćenjem određene količine resursa, tada možemo to rješenje iskoristiti za rješavanje problema *FACTORIZE* bez mnogo dodatnih resursa. Ovo se može ostvariti binarnim pretraživanjem na *k* sve dok nije pronađen najmanji faktor od *n*, te potom dijeleći tim faktorom i opetujući postupak sve dok svi prosti faktori nisu nađeni.
Važan rezultat teorije složenosti jest činjenica da bez obzira koliko težak problem može postati (tj. koliko vremenskih i prostornih resursa zahtijeva), uvijek će postojati teži problemi. Za vremensku složenost, ovo se dokazuje teoremom o vremenskoj hijerarhiji. Na sličan način može biti izveden teorem o prostornoj hijerarhiji.

Računski resursi

Teorija složenosti analizira teškoću računskih problema u terminima mnogo različitih računskih resursa. Isti se problem može opisati u terminima potrebnih količina raznih računskih resursa, uključujući vrijeme, prostor, slučajnost, alternaciju i ostale nešto manje intuitivne mjere. Klasa složenosti je skup svih računskih problema koji mogu biti riješeni koristeći određenu količinu određenog računskog resursa.
Jedni od najviše proučavanih računskih resursa su **determinističko vrijeme** (DTIME) i **deterministički prostor** (DSPACE). Ovi resursi predstavljaju količinu *vremena računanja* i *memorijskog prostora* potrebnih determinističkom računaru, poput onih stvarno postojećih. Ovi su resursi od velikog praktičnog interesa, i naširoko proučavani.
Neke je računske probleme lakše analizirati u terminima neobičnijih resursa. Na primjer, nedeterministički Turingov stroj je računski model kojem je dozvoljeno grananje kako bi odjednom ispitao mnogo različitih mogućnosti. Nedeterministički Turingov stroj nema mnogo veze sa stvarim fizičkim načinom na koji želimo izračunavati algoritme, ali njegovo grananje tačno odgovara mnogim matematičkim modelima koje želimo analizirati, tako da je nedeterminističko vrijeme vrlo važan resurs u analiziranju računskih problema.
Mnogi, još neobičniji računski modeli se koriste u teoriji složenosti. Tehnički, bilo koja se mjera složenosti može shvatiti kao računski resurs, i mjere su složenosti vrlo široko definisane Blumovim aksiomima složenosti.

Klase složenosti

Klasa složenosti je skup svih računskih problema koji se mogu riješiti koristeći određenu količinu određenog računskog resursa.
Klasa složenosti *P* je skup svih problema odluke koji mogu biti riješeni deterministčkom Turingovom mašinom u polinomnom vremenu. Ova klasa odgovara intuitivnoj ideji problema koji mogu biti efikasno riješeni u najgorim slučajevima.
Klasa složenosti *NP* je skup problema odluke koji mogu biti riješeni nedeterministomm Turingovom mašinom u polinomnom vremenu. Ova klasa sadrži mnoge probleme koje bi ljudi željeli efikasno riješiti, uključujući problem bulovske ispunjivosti, problem hamiltonovskog puta i problem prekrivanja vrhova. Svi problemi u ovoj klasi imaju svojstvo da im se rješenja mogu efikasno provjeriti.
Mnoge se klase složenosti mogu karakterizirati u terminima matemtičke logike potrebnih da bi se izrazili - ovo se polje zove deskriptivna složenost.

Neukrotivost
Problemi koji su rješivi u teoriji, ali ne mogu biti riješeni u praksi, se zovu *neukrotivima* (engl. *intractable*). Što tačno može biti riješeno "u praksi" je otvoreno za diskusiju, ali općenito su samo problemi koji imaju vremenski polinomna rješenja rješivi za više od najmanjih ulaza. Problemi za koje se zna da su neukrotivi uključuju one koji su EXPTIME-potpuni. Ako NP nije isti kao i P, tada su NP-potpuni problemi također neukrotivi.
Da bi se vidjelo zašto rješenja u eksponencijalnom vremenu nisu uporabljiva u praksi, neka se razmotri problem koji zahtijeva 2^n operacija za rješavanje (n je veličina ulaza). Za relativno male ulaze od n=100, i pretpostavljajući računar koji može obaviti 10^{12} operacija u sekundi, rješenje bi zahtijevalo $4*10^{10}$ godina, mnogo više od trenutne starosti svemira.

Primjedba 7.1. **Ponekad je počeljno da problemi budu teški**.
Pogledajmo problem kriptiranja podataka na kojem se bazira sigurnost raznih sistema. Na primjer, plaćanja preko weba koriste enkripćiju kako bi se spriječile krade podataka o kreditnim kartićama i sl. Očito, da bi sistem funkćionirao, ekrip- ćija i dekripćija moraju biti brzi algoritmi za one koji imaju potrebne podatke (npr. lozinku), a izuzetno spori (dakle “tečki”) za one koji te podatke nemaju.
No, recimo da imamo neki niz znakova L za koji mislimo da bi mogla biti lozinka za dekripciju nekog teksta T. Kako provjeriti da je L ispravna lozinka za T ? Najjednostavnije je pokušati dekriptirati tekst T s nizom znakova L. Ako dobijemo smislene podatke, gotovo je sigurno da je L lozinka. Drugim ri-ječima, problem dekripcije bez lozinke možemo smatrati NP-problemom (ima brzu provjeru i potencijalno sporo traženje). Kad bi ispalo da je P=NP, to bi značilo da postoje i polinomni algoritmi za otkrivanje lozinki, neovisno o primijenjenim metodama enkripcije (dok god se držimo razumne pretpostavke da brzo radimo ako znamo lozinku).

Računska i računarska teorija složenosti

Uvod

Godine 1971., neovisno jedan o drugome, Cook i Leonid definsali su P- probleme kao "one koje je jednostavno riješiti" i NP-probleme kao "one čije je rješenje jednostavno provjeriti". (*Pitanje šta je jednostavno?*)

Do danas nije poznato je li riječ o istoj klasi problema ili postoji problem čije je rjesenje jednostavno provjeriti, ali ga nije jednostavno naći. To pitanje zovemo "P_NP?" problem.

Otvorena pitanja

The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) odabrao je 7 važnih, a dugo vremena neriješenih problema, ponudivši nagradu od po milon dolara za rješenje svakog od njih. Ti problemi zovu se *Milenijski problemi.*
Jedan od tih ptoblema je tzv. "P=NP?" ili "P vs. NP".
Riječ je o problemu iz teorije algoritama, a koji znači "postoje li, za naizgled složene probleme, jednostavni algoritmi koji ih rješavaju?".

Na web stranid CMI-a problem "P=NP?" prikazanje sljedećim slikovitim primjerom:
Pretpostavimo da zelimo organizirati smještaj za grupu od 400 studenata. Prostor je ograničen, te će samo 100 studenata dobiti sobu. Da ne bi bilo prejednostavno, Dekan je dao listu parova "nekompatibilnih" studenata - onih koji ne mogu zajedno biti u domu (tj. ako jedan dobije smještaj, drugi ga ne smije dobiti).
Ovo je tipični NP-potpuni problem, jer je za veC gotovo rjesenje jednostav¬no provjeriti zadovoljava li dane uvjete, ali do samog rjesenja opCenito nije jednostavno doCi.
Naravno, ako je zadana lista parova "lijepa" (npr. skoro prazna ili sadrži skoro sve parove studenata), vodit Ce nas gotovo deterministiCki prema nekom rjesenju. No, opCenito, moze se dogoditi da moramo pokusati sve kombina¬ti^.
Matematičkim rječnikom, to znači da treba pogledati sve 100-člane pod- skupove skupa studenata, te provjeriti koji od njih (ako ijedan!) zadovoljava uvjet "nekompatibilnosti". Iz kombinatorike, poznato je da m-članih pod- skupova n-članog skupa ima

$$\binom{n}{m} = \frac{n!}{(n-m)!\ m!}.$$

U nasem slučaju, traženih podskupova ima

$$\begin{aligned}\binom{400}{100} &= \frac{400!}{(400-100)!100!} \\ &= 300742969389401893510717432 0 \approx 3 \cdot 10^{27}.\end{aligned}$$

Za usporedbu, danas najbrži računar uspjelo je postiči 2331 TFLOPS , odnosno oko 2.33 $X10^{15}$ operacija s pomičnim zarezom u sekundi. Provjera uvjeta za svako od 3 X 10^{27} mogućih rješenja nužno zahtijeva puno više od jedne operacije s pomičnim zarezom, no čak i kad bi nam jedna operacija bila dovoljna, navedenom super-računaru trebalo bi vise od 10^{12} sekundi, odnosno vise od 30000 godina, da rijesi zadani problem.
Opisani način rjesavanja problema naziva se brute force (rjesavanje grubom silom) i često je neizvedivo zbog ogromnog vremena izvršavanja.

P-problemi

Kao P-probleme (ili probleme klase P) definisamo one probleme za koje je moguće definisati

deterministički Turingov stroj koji rjesenje nalazi u poli- nomnom vremenu, sto znaCi da postoji neki polinom p(X) takav da je broj koraka Turingovog stroja uvijek manji od p(X), gdje je X duljina ulaznih podataka zapisanih na traCi Turingovog stroja.
Turingov stroj nam sluzi za opis osnovnih operacija. Jedan korak ovdje jasno označava: jedno čitanje s trake, jedno pisanje na traku, jedan pomak glave stroja i jednu promjenu stanja. Kod analize algoritama Cesto brojimo osnovne računske operacije, što se jednostavno prevede u terminologiju deterministickih Turingovih strojeva.

Primjer.
Zadan je niz brojeva koje treba poredati (sortirati) od najmanjeg prema najvećem.
Očito, općenito je nebitno koje su vrijednosti brojeva koje želimo poredati, ali je itekako bitno koliko ih ima. Označimo duljinu niza s n.
Algoritam "Isprobaj sve mogućnosti". Jedan način sortiranja brojeva je pronaći sve moguće redoslijede tih brojeva i za svakog provjeriti je li dobar. Sama provjera je jednostavna: potrebno je jednom proći kroz ćijeli niz i provjeriti jesu li svaka dva susjedna broja u ispravnom poretku. To se, očito, izvede u n — *1* koraka (prećiznije, usporedbi brojeva), sto je polinom prvog stupnja, pa se provjera izvodi u polinomnom vremenu. Dapaće, posto je polinom prvog stupnja (dakle, linearan), kazemo da je vrijeme izvršavanja linearno.
No, svih mogućnosti ima
n! = nX (n — 1)X (n — 2)X .....X 2 X 1.
Lako se provjeri da za proizvoljni polinom *p(x)* postoji dovoljno veliki n takav da je n! > p(n), sto znaći da takav algoritam ima nadpolinomno vrijeme izvrsavanja.
No, ćinjenića da smo pronasli spori algoritam koji rjesava nas problem, naravno, ne znaći da je sam problem tezak (u smislu da nema polinomno rjesenje), nego samo da je pristup "isprobaj sve mogućnosti" spor.
Algoritam "Usporedi svaka dva". Mozemo usporediti svaka dva broja, te ako su oni u pogresnom redoslijedu, onda ih zamijeniti. Rijeć je o tzv. *selection* sortu koji je jednostavan za implementaciju i analizu slozenosti, no sporiji od ostalih koji su u upotrebi.
Primijetimo da brojeva ima n, a svakog mozemo usporediti (i eventualno zamijeniti) s njih najvise n — *1* (jer nema smisla uspoređivati broj sa samim sobom), pa će broj koraka očito biti manji ili jednak vrijednosti polinoma
p(n)= n(n — 1)= n^2 — n.
Zbog jednostavnosti, prikažimo uzlazni sort (dakle, onaj koji čeli elemente niza poredati od najmanjeg do najvećeg):

```
za sve x[i] min = i
za sve x[j] takve da je i različito od j
ako je x[k] > x[j]
min = j ako je min različito od i
zamijeni vrijednosti x[i] i x[min]
```

Očito je da ovakav algoritam zaista sortira niz. Naime, on pronalazi index *min* takav da je $x_m i_n$ najmanji element podniza x_i, x_{i+1},.... x_n, te ga smješta na mjesto x_i (dakle, ispred svih elemenata koji su veći od njega). Tako ce u prvom prolazu vanjske petlje najmanji element biti postavljen na prvo mjesto u nizu. U drugom prolazu, drugi najmanji će biti postavljen na drugo mjesto, itd.
U stvarnosti, usporedbi i potencijalnih zamjena ima *n(n — 1)/2)*, no taj detalj utješe samo na koeficijente polinoma, ali ne mijenja činjenicu da je riječ o polinomu drugog stupnja. Zamjena brojeva **x1** i **x2** vrsi se u tri koraka:

```
privremeni = x1;
x1 = x2;
x2 = privremeni;
```

Dakle, broj koraka u zamjeni dva broja ne ovisi o n, pa i zamjene utječu samo na koeficijente, ali ne i stupanj polinoma kojim ograničavamo broj koraka.
Konačno, jer je polinom kojim močemo ograničiti broj koraka ovog algoritma drugog stupnja (tj. kvadratni), kačemo da je algoritam kvadratni.

Kao što smo vidjeli u prethodnom primjeru, problem sortiranja brojeva je klase P.
Opisano rješenje problema je daleko od optimalnog, te postoje mnogo efikasniji sortovi, kao i usko specijalizirani sortovi koji donose dodatna ubrzanja (npr. Radix sort koji postize linearnu slozenost, ali

po cijenu ograničavanja veličine svakog elementa niza).

NP-problemi

NP-probleme definišemo slično P-problemima, uz dodatak onog "N" koje znači "nedeterministički". Dakle, NP-problemi (ili problemi klase NP) su oni problemi za koje je moguće definisati nedeterminističke Turingove strojeve koji ih rjesavaju u polinomnom vremenu.
Kad nedetrministički stroj, kad ima "dilemu", ispravno pogada šta će od ponuđenih akcija napraviti.
Kad bismo pokusali implementirati takav stroj, morali bismo isprogramirati isprobavanje svih mogućnosti, pa bi broj koraka jako narastao (u pravilu barem eksponencijalno).

Primjer Problem trgovačkog putnika.
U nekoj državi postoji n gradova. Gradovi A i B povezani su, a udaljenost izmedu njih je d(A, B) (metara, kilometara,... svejedno je; moze biti i vrijeme, ćijena,...).
Traži se najkraći put kojim trgovać moze obići sve gradove, te se vratiti na poćetak, na način da svaki grad posjeti toćno jednom. Drugim riječima, traze se gradovi
g1, g2, . . . , gn
takvi da je
{g1, g2, . . . , gn} = {1, 2, . . . , n}
i da je
d(g1, g2) + d(g2, g3) + • • • + d(gn−1, gn) + d(gn, g1)
najmanje moguće. *Takvi putevi koji obidu svaki vrh tačno jednom te se vrate u polazini vrh grafa zovu se Hamiltonovi ciklusi.*
Slično kao kod sortiranja iz ranijeg primjera, i ovdje možemo pribjeći traženju svih mogućih puteva. Sva argumentacija navedena tamo, vrijedi i ovdje, pa je riječ o izuzetno sporom rješenju.
Ono sto nije poznato je postoji li polinomni algoritam koji će ovaj problem riječiti tačno. Postoje razni algoritmi koji problem rješavaju brzo, ali rječenja su približa ili ograničena na posebne situacije (npr. zadovoljenost nejednakosti trokuta i sl).

Spomenuli smo da se P- i NP-problemi obično bave odlučivanjem. Problem trgovačkog putnika lako je svesti na da/ne-pitalicu: "Za zadani x *el* R+, postoji li put kojim trgovac može putovati tako da pređe udaljenost manju od x?" Ako na to pitanje nađemo odgovor u polinomnom vremenu, onda jednostavnim postavljanjem tog pitanja za nekoliko vrijednosti x_i (ali ne naročito mnogo njih) mozemo pronači i optimalni put.

Primjedba 5.2. Za koliko različitih vrijednosti x_i bismo trebali postaviti pitanje iz prethodnog paragrafa? Primijetimo, odgovor na pitanje je jednak za vrijednosti x_i i xj (xi < xj) ako ne postoji put u grafu duljine *x′* takve da je $x_i < x' < xj$. Drugim riječima, zanimljivi su nam samo oni x_i koji su jednaki duljini nekog puta u grafu. No, skup svih puteva u grafu je sigurno podskup skupa P (E*)*, gdje je E skup svih bridova u grafu, a P (E*)* njegov partitivni skup. Posto je graf potpun (svaka dva grada su povezana), skup E ima n (n — *1)* elemenata, pa njegov partitivni skup ima

$$\operatorname{card} \mathcal{P}(E) = 2^{\frac{n}{2}(n-1)}$$

elemenata, a broj kandidata x_i je sigurno manji od toga. Traženi x_i možemo naći binarnim traženjem. Prvo sortiramo sve vrijednosti x_i uzlazno po veličini, te provjeravamo može li trgovac obići sve gradove tako da prijeđe udaljenost najviše x_k za

$$k = \left\lfloor \frac{n}{2} \right\rfloor$$

(drugim riječima, postavljamo pitanje za srednji x_i).
Ako moče, odbacujemo sve x_i takve da je $i > |\_nJ$ (jer znamo da i za njih moče). U protivnom, odbacujemo one za koje je $i < |\_nJ$ (jer znamo da niti za njih ne moče). Zatim ponavljamo postupak za preostale kandidate. Broj kandidata se u svakom koraku raspolavlja, pa je najveći mogući broj koraka jednak $log_2(card\ P(E))$. Konačno, imamo najveći broj postavljanja pitanja o obilasku uz duljinu najviše x_i (za odabrane x_i):

$$\log_2(\operatorname{card} \mathcal{P}(E)) = \log_2 2^{\frac{n}{2}(n-1)} = \frac{n}{2}(n-1) = \frac{n^2}{2} - \frac{n}{2},$$

što je polinom.
Primijetimo i da je riječ o (gruboj gornjoj ogradi jer skup P*(E)* sadrži mnoge podskupove od *E* koji

nisu kruzni putevi u grafu, pa je stvarni broj postavljanja pitanja puno manji (pravi broj mogućih puteva je
(n — 1)!/2).

P = NP pitanje

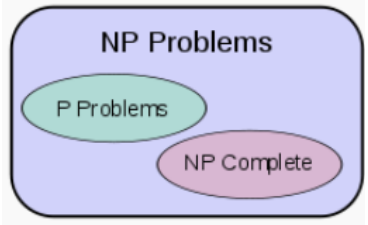
Pitanje istovjetnosti skupova *NP* i *P* (tj. mogu li problemi koji mogu biti riješeni u *nedeterminističkom* polinomnom vremenu riješeni u *determinističkom* polinomnom vremenu) je jedno od najvažnijih otvorenih pitanja u teoretskom računarstvu, s obzirom na široke implikacije koje bi rješenje tog pitanja imalo.[1]. Jedna negativna posljedica je ta da bi se mnogi oblici kriptografije mogli jednostavno "razbiti" i stoga postali beskorisni. Međutim, postojale bi i ogromne pozitivne posljedice, budući da bi mnogi važni problemi imali učinkovita rješenja. Takvi problemi uključuju razne tipove cjelobrojnog programiranja u operacijskim istraživanjima, mnoge probleme u logistici, predviđanju strukture bjelančevina u biologiji, te sposobnosti učinkovitog pronalaženja formalnih dokaza teorema u čistoj matematici korišćenjem računara. *Clay Mathematics Institute* je 2000. obavio da će isplatiti nagradu od USD$ 1 000 000 prvoj osobi koja dokaže rješenje.

Pitanja poput ovih motiviraju koncepte *težine* i *potpunosti*. Skup problema *X* je težak za skup problema *Y* ako svaki problem u *Y* može "lako" biti transformiran u neki problem u *X* koji producira rješenje. Definicija "lakog" je različita u različitim kontekstima. Važan *teški* skup u teoriji složenosti jest NP-težak - skup problema koji nisu nužno sami u *NP*, ali na koje bilo koji *NP* problem može biti sveden u polinomnom vremenu.

Skup *X* je potpun za *Y* ako je težak za *Y*, ali je također i podskup od *Y*. Važan potpun skup u teoriji složenosti je NP-potpun skup. Ovaj skup sadrži "najteže" probleme u NP, u smislu da su to oni koji najizglednije nisu u P. Zbog činjenice da problem *P = NP* ostaje neriješen, svođenje bi problema na poznati NP-potpun problem indiciralo da ne postoji poznato vremenski polinomno rješenje za njega. Slično, budući da svi *NP* problemi mogu biti svedeni na skup, pronalaženje *NP-potpunog* problema koji bi mogao biti riješen u polinomnom vremenu bi značilo *P = NP*.[1]

Nepotpuni problemi u *NP*

Drugo otvoreno pitanje vezano za problem *P = NP* jest postoje li problemi koji su u NP, ali ne i u P, koji nisu NP-potpuni. Drugim riječima, problemi koji trebaju biti riješeni u nedeterminističkom polinomnom vremenu, ali ne mogu biti svedeni na polinomno vrijeme iz drugih nedeterminističkih vremenski polinomnih problema. Jedan takav problem, za koji se zna da je NP ali ne i da je NP-potpun, jest problem izomorfizma grafa - problem koji pokušava odlučiti jesu li dva grafa izomorfna (tj. dijele li ista svojstva).



Dijagram klasa složenosti uz pretpostavku da vrijedi ***P*** ≠ ***NP***.

Postojanje problema izvan klasa i **P** i **NP**-potpun u ovom slučaju je postavio Ladner.[5]
Pokazano je da ako vrijedi *P* ≠ *NP*, da takvi problemi postoje.

NP = co-NP

Gdje je co-NP skup koji sadrži komplementarne probleme (tj. problem sa invertiranim *da*/*ne* odgovorima) NP problema. Vjeruje se da te dvije klase nisu jednake, iako to dosad nije dokazano. Pokazano je da ako ove dvije klase nisu jednake, da tada slijedi da nijedan NP-potpun problem ne može biti u co-NP i nijedan co-NP-potpun problem ne može biti u NP.

Očito, svaki deterministički Turingov stroj je ujedno i specijalna vrsta nedeterminističkog (koji nema niti jednu situaciju u kojoj mora pogadati šta dalje), pa su svi P-problemi ujedno i NP-problemi.

Imitacija dijaloga

Imitacija dijaloga je karakteristika koja potiče od čuvenog engleskog matematičara Tjuringa (*Turing*). On je predložio da se sistem, čovjek ili računar, smatra inteligentnim ako se ne može uočiti, u toku konverzacije, s kim se vodi dijalog, sa čovjekom ili sa računarom.

Tjuringov test je naziv ovog testa inteligencije, za koji su bili razvijeni posebni dijalogni programi. U toku provođenja testa sa bolesnicima, većinu sagovornika je bilo teško ubijediti da "razgovaraju" sa računarom, odgovarali su "da ih on tako dobro razumije". Međutim, vrlo brzo se došlo do zaključka da je ovakav test samo imitacija intelekta i da je jedan od potrebnih, ali ne i dovoljnih uslova za inteligenciju. Dalji razvoj dijalognih programa, kao i razvoj Tjuringovog testa prije tridesetak godina, vezivan je za rješavanje određenih logičkih ili računarskih problema.

Posebno se isticala mogućnost igranja šaha, što je u ne tako davnoj prošlosti zadiralo u domen naučne fantastike. Duži niz godina postoji veći broj programa koji igraju veoma dobar šah već i na kućnom računaru. Time se pokazalo da je područje vještačke inteligencije vrlo ćudljivo područje nauke, neki problemi za koje se smatralo da su nesavladivi postali su jednostavni i lagani, dok su drugi, za koje se mislilo da su jednostavni i lagani, postali praktično nesavladivi.

Rješavanje svih varijanti problema

Rješavanje svih varijanti problema predstavlja slijedeću karakteristiku inteligencije. Ovdje se misli na rješavanje svih varijanti nekog problema, ali ne lošije od čovjeka. Ova karakteristika, nekada veoma popularna, danas se smatra nedovoljnom i nepotpunom. Postavlja se pitanje o kakvom problemu i kakvim varijantama se govori. Probleme ne treba tražiti u području numerike, jer onda o intelektu nema ni govora. Neki autori predlažu da treba preći u mnogo "intelektualnije" područje, muziku.

Rješavanje netrivijalnih zadataka

Odmah na početku se postavlja pitanje šta su trivijalni, a šta netrivijalni zadaci. Uobičajena matematička definicija kaže da je trivijalni zadatak onaj kod koga je način rješavanja poznat. Pri tome nije važno da li rješenje postoji ili ne, odnosno da postavke dovode do apsurda. Iz definicije trivijalnog zadatka proizilazi

definicija netrivijalnog zadatka: to je zadatak kod koga se mora pronaći način rješavanja. Ovdje se ne misli na klasu problema koji su rješavani, odnosno nisu rješavani, nego na onoga ko mora rješiti zadatak, a ne poznaje algoritam rješavanja.

PRIMJER: Množenje dva broja.

Za učenike starijih razreda množenje dva broja je trivijalni zadatak, pošto su učili tablicu množenja, a za prvoškolca ovo je netrivijalni zadatak, jer on ne poznaje način rješavanja zadatka.

Klasični primjer inteligencije dat je u anegdoti iz đačkog života Karl Friedrich Gauss-a. Kao prvoškolac je dobio "nerješiv" zadatak da sabere brojeve od jedan do sto. Gauss je vrlo brzo i elegantno došao do rješenja: uvidio je da je zbir prvog i zadnjeg broja 101, drugog i predzadnjeg takođe 101 i tako sve do zbira zadnjeg para brojeva 50 i 51. Budući da parova ima 50, proizvod 50 puta 101 daje rješenje 5050.

Ekstrapolacija

Ekstrapolacija, odnosno aproksimativno odlučivanje na osnovu niza faktora, je jedna od karakteristika koju je korisno objasniti i preko primjera.

Neka je dan uzrok neke pojave i njegova posljedica: pritisak na prekidač izaziva paljenje svjetla, guranje bureta izaziva njegovo kotrljanje, pojava problema zahtjeva njegovo otklanjanje, itd. Kod trivijalnih problema rješenje je očigledno, što nije slučaj kod zdravstvenih tegoba relativno nedefinisanog oblika ili proizvodnje nekog dijela ili sklopa.

PRIMJER*: *Donošenje odluke o kupovini automobila.

Može se napraviti tabela konkurentnih karakteristika dva ili više automobila različitih proizvođača u istoj klasi i sa približno jednakom cijenom. U tabeli će se naći: potrošnja, komfor, snaga, servis, veličina putnog i prtljažnog prostora,dizajn, itd. Može se napraviti bodna lista, gdje svaka karakteristika ima svoju težinsku vrijednost, čime se dobija sistem jednačina u kojima je upoređivanje karakteristika poznato kao proces identifikacije. Nakon identifikacije treba dobiti rješenje, ali rješenje, praktično nikada, nije jednoznačno i definitivno. Odluka je tada i sama niz odluka, odnosno proces koji nazivamo jednim imenom ekstrapolacija.

Učenje

Učenje je jedna od najvažnijih i najtežih karakteristika spoznaje, i samim tim, inteligencije uopšte. Pri tome se ne misli da je učenje memorisanje činjenica.To bi bilo isto kao kada bi se reklo da je vožnja automobila pritiskanje papučice gasa. Doduše, bez gasa automobil se ne može voziti, ali samo sa gasom to je još besmislenije.

Memorisanje činjenica jeste nužna i neophodna karakteristika učenja, ali daleko od toga da bude i dovoljna. **Aktiviranjem čula** počinje učenje, odnosno uspostavljanje kontakta sa vanjskim svijetom. Kod živih bića ovo je stalan, neprekidan proces, koji počinje sa rađanjem i prestaje sa smrti.

Sposobnost pamćenja je slijedeća karakteristika procesa učenja. Samo prosto memorisanje podataka nije karakteristika inteligencije, jer ukoliko bi to bilo tačno onda i papir i crijep sa klinastim pismom i magnetni medij bi bili inteligentni.

Mogućnost učenja

Mogućnost učenja je veoma bitna karakteristika računara. Današnji računari rade tačno ono što je programom predviđeno. Bez obzira na vanjsku manifestaciju, rad računara se svodi, uglavnom, na korišćenje memorije i dovoljno mnogo pitanja - skretnica tipa "AKO ... → TADA". Odstupanje od programiranog ponašanja znači grešku ili sistema ili programa. Međutim, ako se računaru omogući da uči, u bilo kojem obliku, tada se mogu očekivati optimalni programi, primjereni danom problemu.

Osnove teorije vjerovatnoće i kombinatorike

Zamislite kako neki čovjek 60 puta zaredom baci igraću kocku i da pri tome izbroji koliko je puta dobio
svaki od 6 brojeva na kocki. Kad je završio s bacanjem, on konstauje jedno od ovoga dvoga:
1.Šest brojeva nisu se pojavili jednak broj puta, ***usprkos*** *tome što su svi brojevi imali jednaku vjerojatnoću pojavljivanja.*
2.Šest brojeva nisu se pojavili jednak broj puta, ***zato*** *što su svi brojevi imali jednaku vjerojatnoću pojavljivanja.'*

(Max Woitschach 1973. u knjižici 'Vjerojatnost i slučaj')

ako se slažete s 1. konstatacijom, griješite: praktičmo je nemoguće da se kod većeg broja bacanja svaki broj pojavi jednak broj puta. **Potpuno jednaka šansa ... upravo uzrokuje nejednaku raspodjelu brojeva...**

ako se slažete s 2.konstatacijom, pripadate manjem broju onih kojima su osnove vjerojatnoće poznate.

Dva kockara A i B bacaju novčić, pri čemu jedan od njih, recimo A, igra na „pismo", a drugi na „grb". Broj pojave „pisma" („grba") predstavlja broj poena za igrača A (igrača B). Dobitnički ulog namjenjen je onom koji prvi dođe do unapijed utvrđene sume. Međutim, zbog nekih razloga igra je prekinuta.
Postavlja se pitanje kako podeliti ulog znajući da su u momentu prekida kockaru A bila potrebna 2 poena dodobitne sume, a kockaru B 3 poena.
Ovaj problem, postavljen od jednog poluprofesionalnog kockara, analizirali su i rješili čuveni francuski matematičari Pascal i Fermat. Njihova studija o ovom problemu iz 1654. godine uzima se za početak teorije vjerovatnoće.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a	*a*	*a*	*a*	*b*	*a*	*a*	*b*	*a*	*b*	*b*	*b*	*b*	*b*	*a*	*b*
a	*a*	*a*	*b*	*a*	*a*	*b*	*a*	*b*	*a*	*b*	*b*	*b*	*a*	*b*	*b*
a	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*b*	*b*	*a*	*b*	*a*	*b*	*b*	*b*
a	*b*	*a*	*a*	*a*	*b*	*b*	*b*	*a*	*a*	*a*	*a*	*b*	*b*	*b*	*b*

Rješenje: Očigledno, ne više od četiri bacanja novčića je dovoljno da se igra okonča. Neka a označava opit gdje A pobeđuje (pojava „pisma"), a b opit u kome B pobjeđuje (pojava „grba"). Postoji 16 varijacija od dva slova a i b dužine 4, kao što je prikazano u tabeli. Između 16 mogućih slučajeva, 11 je povoljno za igrača A (slučajevi 1-11, gde se a javlja 2 ili više puta), dok je 5 povoljno za B (slučajevi 12-16, gde se b javlja 3 ili više puta). Prema tome, vjerovatnoća dobitka je 11/16 za kockara A, i 5/16 za kockara B.
Kockari bi trebalo da podjele ulog proporcionalno vjerovatnoćama dobitka, dakle, u odnosu 11:5.

Uvod

Postoji više razloga koji doprinose pojavi interesovanja za vjerovatnoću i razvoju njenih matematičkih osnova.
Prvi je proizišao iz matematičkih problema u igrama na sreću. Švajcarski matematičar Bernuli u XVIII veku je postavio teorijske osnove vjerovatnoće, kao jedne matematičke discipline. Nešto docnije taj razvoj je išao dalje u radovima Laplasa sa njegovim strogo determinističkim pogledom na svijet. Po njemu, vjerovatnoća je sastavni deo nauke o prirodi, kao teorija grešaka, u čijoj osnovi je sistematsko proučavanje sredine i njenog varijabiliteta u ponovljenim merenjima. U razvoju teorije vrejdan je Gausov doprinos sa radovima u oblasti normalnog zakona grešaka itd.
Drugi razlog interesovanja za vjerovatnoću proizišao je iz osiguranja protiv rizika, koje se praktikovalo u trgovini u italijanskim gradovima u periodu renesanse.
Kod programiranja je poznavanje osnova teorije vjerovatnoće je od posebnog interesa za rješavanje problema kada nije poznato algoritamsko rješenje problema, kada odgovarajući algoritam ne postoji.
U tom slučaju se koriste heuristike, pravila za rješavanje kojima se na osnovu prethodnog znanja, iskustva i intuicije sužava i usmjerava područje traganja za rješenjem.
U tom slučaju su i dobijeni zaključci prihvatljivi samo sa izvesnom vjerovatnoćom, pa se sistem koristi za procjenu izvjesnosti pojedinih mogućnosti

Osnov praktične primjene vjerojatnoće je statistika. Statistika je bazirana na matematičkoj teoriji vjerojatnoće, a ona, u svojoj osnovi, na teoriji skupova.Teorija vjerovatnoće je posebno značajna u procjenjivanju: statističkoj inferenciji (**Statistical Inference),** koja počiva na njenim osnovama.
Danas se statistička teorija inferencije primijenjuje u skoro svim naučnim disciplinama. Postala je sastavni deo teorijske fizike (nauka o toploti, kvantna mehanika itd). Preko kvantne teorije nalazi svoje mjesto i u atomistici itd.

Bez obzira na diskusije o računu vjerovatnoće i njegovoj interpretaciji, njegova formalna osnova nije diskutabilna. Ipak, kad se primijenjuje račun vjerovatnoće u statističkim istraživanjima interpretacija

modela i rezultata ne sme da se posmatra kao nešto odvojeno. Subjektivni prilaz računu vjerovatnoće i, na njegovoj osnovi, Bajesova statistika, kako se često naziva, je savremeni trend.

Prostor uzoraka

Osnovni sastojak teorije vjerovatnoće je eksperiment koji se, makar hipotetički, može ponoviti u suštinski identičnim uslovima, a koji, sa svakim ponavljanjem, može voditi ka različitim ishodima. Skup svih mogućih ishoda eksperimenta se naziva **prostor uzorka.**
Skup ishoda, odnosno prostor uzorka, može biti konačan ili beskonačan.
Ako je konačan, znači da se mera izražava na prekidnoj skali, kao na primer broj glasača koji su glasali za određenog kandidata. Druga merenja su na neprekidnoj skali. To su uglavnom fizičke veličine kao što su temperatura, vreme reakcije, marginalni prihod i sl.
Zbog jednostavnosti, matematičari su prvo izučavali eksperimente sa konačnim prostorom uzoraka i to takve kod kojih je vjerovatnoća pojedinačnih ishoda jednaka. U ovom slučaju, vjerovatnoća pojedinačnog ishoda je definisana kao odnos broja poželjnih ishoda i ukupnog broja ishoda.
Slučajni događaj je onaj događaj čiji ishod ne može da se predvidi unapred. Poznat je samo skup ishoda od kojih jedan sigurno mora da bude, a smatra se da je baš taj ishod rezultat slučajnosti.
Događaj je dobro definisani podskup prostora uzorka. Pri tome, podskup može imati proizvoljan broj elemenata, sve dok je manji od broja elemenata u skupu.
Elementarni događaj je podskup prostora uzorka koji ima samo jedan element.
Složeni događaj je podskup prostora uzorka koji ima više od jednog elementa, tj. sastoji se od više od jednog elementarnog događaja.

Aksiomski koncept vjerovatnoće

Postoje najmanje dva uspješna pokušaja da se formalizuje vjerovatnoća, koji su nazvani Kolmogorova formulacija i Coxova formulacija. U oba slučaja zakoni vjerovatnoće su isti, sa malom razlikom u tehničkim detaljima.
Ovdje ćemo dati jedno slično, elementarno, objašnjenje pomoću teorije skupova koje omogućuje nam da se upoznamo osnovama teorije vjerovatnoće preko njenih aksioma. Ovaj koncept uzima u obzir različite definicije vjerovatnoće i na njihovoj osnovi nastoji da pruži jednu manje-više zajedničku logičnu strukturu, koja je nezavisna od bilo koje formalne definicije.

Vjerovatnoća se definiše kao funkcija podskupova u prostoru događaja.
Pri tome se pošlo od dva bitna preduslova:
1) da teorija vjerovatnoće mora da bude potpuno matematička i
2) da je ona u skladu sa empirijskim činjenicama.
Kod datog prostora događaja *S,* vjerovatnoća za svaki podskup *A* u S je neki realni broj.

Vjerovatnoća od *A* piše se *P(A)* i ona počiva na sljedećim aksiomima:

1. $0 \leq P(A) \leq 1$. **Vjerovatnoća je broj između 0 i 1**;
2. $P(\Omega) = 1$. **Zbir vjerovatnoćâ da će se posmatrani događaj dogoditi, i da se on neće dogoditi iznosi 1**;
3. $P(\bar{A}) = 1 - P(A)$. **Vjerovatnoća da će se neka dva događaja dogoditi je jednaka proizvodu vjerovatnoće jednog od njih i vjerovatnoće drugog pri uslovu da se prvi već dogodio**;
4. $P(\emptyset) = 0$. **Vjerovatnoća nemogućeg događaja je 0**;
5. $\sum_{i=1}^{n} P(A_i) = 1$. **U jednom potpunom sistemu događaja A_i je njihov proizvod vjerovatnoće jednak 1.**

Veza bilo kojeg podskupa sa elementima u prostoru događaja naziva se *funkcija vjerovatnoće.*

Čitava teorija vjerovatnoće, posmatrana kao formalan matematički sistem, razrađena je na bazi ovih aksioma.

Skala vjerojatnoće:

$0 \leq P(A) \leq 1$

P(S)=1 (*100)=100%

P(ø)=0

Klasična definicija vjerojatnoće (vjerojatnoća a priori)

Prema *klasičnom ili a priori* konceptu, (Laplas 1812. godine) vjerovatnoća jednog događaja *A* je odnos broja za njega povoljnih događaja *a,* prema broju svih jednako mogućih događaja *n.*

Tu vjerovatnoću pišemo i predstavljamo kao: $P(A) = \frac{a}{n}$.

Vjerojatnost da će se između N međusobno bezavisnih, a jednako vjerojatnih događaja, dogoditi jedan određeni među njima: p=1/N (vjerojatnost a priori)*

Predstavljanje i interpretacija vrijednosti u vjerovatnoći

Vjerovatnoća događaja se predstavlja kao realan broj između 0 i 1.

Nemoguć događaj ima vjerovatnoću 0, a siguran događaj ima vjerovatnoću 1.

U slučaju da je jednaka vjerovatnoća da će se događaji dogoditi, kao i da neće, vjerovatnoća je 0,5.

Statistička definicija vjerovatnoće

Ako se broj ponavljanja eksperimenara izvedenih u istim uslovima povećava u beskonačnost, tada je vjerojatnoća nastupa događaja *A* granična vrijednost relativne frekvencije povoljnog ishoda događaja *A*

$$P(A) = \lim_{n \to \infty} \frac{m}{n}$$

Ishod i vjerovatnoća presjeka i unija događaja

Pretpostavimo da imamo dva događaja: A i B.

Presjek događaja A i B se obeležava sa A∩B, a čine ga ishodi koji pripadaju prostoru uzorka koji je zajednički događajima A i B. Presjek događaja nije prazan ako postoji makar jedan ishod događaja takav da pripada prostoru uzorka događaja A i događaja B. Na taj način presjek događaja A i B odgovara logičkom operatoru „i".

Međusobno isključivi događaji su takvi događaji kod kojih pojava jednog događaja isključuje mogućnost pojave drugih događaja. Presjek međusobno isključivih događaja je prazan skup. Po definiciji, svi elementarni događaji su međusobno isključivi.

Unija događaja A i B se obilježava A U B i odgovara logičkom operatoru „ili". Unija događaja nije prazna ako se dogodio makar jedan ishod događaja A ili B. Formula za izračunavanje broja ishoda unije dva događaja glasi

$$n(A \cup B) = n(A) + n(B) - n(A \cap B)$$

Broj ishoda događaja A je n(A), a broj ishoda događaja B je n(B). Unija obuhvata sve ishode A i sve ishode B, što obuhvata i one ishode koji su u presjeku događaja. **Zbog toga se presjek događaja mora isključiti iz formule za računanje broja ishoda u uniji događaja**, jer bi se u suprotnom ti ishodi brojali dva puta.

Pošto je presjek međusobno isključivih događaja jednak 0, ova formula se za međusobno isključive događaje pojednostavljuje i glasi

$$n(A \cup B) = n(A) + n(B)$$

Kada formulu za uniju događaja A i B podelimo sa brojem mogućih ishoda n(S), dobijamo vjerovatnoću dešavanja unije događaja:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

odnosno, za međusobno isključive događaje

$$P(A \cup B) = P(A) + P(B)$$

Ova formula je poznata kao **pravilo sabiranja** i važi samo za međusobno isključive događaje.

Vjerovatnoća unije događaja A i B ne može biti manja od vjerovatnoća događaja A i B, tj.

$$P(A \cup B) \geq \min(P(A))$$

Bajesova teorema

Engleski svštenik T. Bajes (1702-1761), koji se istovremeno bavio i matematikom, bio je zaokupljen problemom izvođenja zaključaka na osnovu induktivne inferencije. Do tada matematičari su se na osnovu dedukcije, polazeći od date hipoteze na bazi vjerovatnoće a priori, bavili eventualnim posljedicama njene primjene.

Bajes je pošao obmutim putem, došao je je do teoreme pomoću koje se na osnovu posmatranja posljedica utvrduje hipoteza. Teorema je još od prvih dana pobudila interesovanje i bila je predmet kontraverzi koje i danas traju. Njena matematička osnova nije spoma, nego primena u vezi sa utvrđivanjem vjerovatnoće *a priori.*

Ovdje ćemo dati samo formalne osnove teoreme.

Neka su Ai>Az, . . . , Ah . . . ,An međusobno isključivi događaji u prostoru elementarnih dogadaja. Druge medusobno isključive dogadaje označimo sa Bj. Nastupanje jednog od dogadaja B uslovljeno je događajem A,.

Drugim rječima, dogadaj Bj ne može da nastupi prije dogadaja A, tako da oni nisu isključivi.

Uzmimo da su vjerovatnoće P(Ai) i uslovna verovatnoca od B poznati, Bajesovom teoremom se dolazi do vjerovatnoće a posteriori događaja A, nakon nastupanja dogadaja B .

Njen matematički izraz je

$$P(A_1|B) = \frac{P(A_1)P(B|A_1)}{\sum_{i=1}^{n} P(A_i)P(B|A_i)}.$$

Principi kombinatorike

Vidjeli smo kako se na osnovu aksioma definiše apstraktni koncept vjerovatnoće.

Za ilustraciju smo uzeli mali broj elemenata u prostoru događaja koji su služili kao osnova za utvrđivanje vjerovatnoće podskupova. Razumljivo da je to u praksi komplikovanije jer je skup daleko brojniji. Kombinatorika olakšava utvrđivanje veze između skupa i podskupova. To se postiže na osnovu izvesnog sistema pomoću koga se pružaju svi mogući rasporedi i grupacije za podskupove datog broja elemenata. S obzirom na način na koji su elementi razmešteni u podskupovima, **kombinatorika se dijeli na *permutacije, kombinacije i varijacije.***

Kod definisanja ovih oblika kombinatorike ukazaćemo na *princip množenja* na koji se kombinatorika u osnovi oslanja. Jedan eksperiment može da ima m ishoda, drugi «2 itd. Ukupan broj mogućih ishoda u *k* eksperimenata uzetih zajedno je (ni) (n_2) ... - («*).

Tako, kod ogleda sa bacanjem jedne kocke ima *ni = 6* ishoda, kod drugog bacanja ima opet «2 = 6

ishoda itd. Ukupan broj ishoda kod tri uzastopna bacanja je 6 • 6 • 6 = 216.

Permutacije

Ako imamo *n* različitih elemenata jedan od zadataka može da bude broj njihovih mogućih rasporeda ili permutacija. Prema principu množenja, bilo koji elemenat može da zauzme prvo mjsto, drugo mjesto može da zauzme n-1 elemenata
treće *n-2* itd. sve do posljednjeg mesta koje je rezervisano za poslednji neraspoređeni elemenat u datom redosljedu.
U kombinatorici, permutacija je niz koja sadrži tačno jednom svaki element iz nekog konačnog skupa. Permutirati znači zadane elemente na sve moguće načine spajati u skupine tako da svaka skupina **sadrži sve zadane elemente.**
Koncept niza se razlikuje od koncepta skupa po tome što u skupu nije određen redoslijed elemenata, dok su u nizu tačno određeni prvi, drugi, itd. elementi.

Permutacije bez ponavljanja članova skupa $P = n!$ gdje je n broj elemenata skupa koji mogu biti izabrani.

Primjer 1. Koliko se troznamenkastih brojeva (s različitim znamenkama) može napisati od cifri 1, 2, 3?

To je broj permutacija bez ponavljanja od 3 elementa

$$P_3 = 3! = 3 \cdot 2 \cdot 1 = 6$$

Ispišimo ih: 123 132 213 231 312; 321

Primjer 2. Koliko permutacija elemenata skupa $\{2,4,6,8\}$ počinje s 8?

Skupine su oblika 8 _ _ _ , pri čemu na preostalim mjestima možemo napisati bilo koju permutaciju bez ponavljanja od elemenata skupa $\{2,4,6\}$.
Možemo ih ispisati na $P_3 = 3! = 6$ različitih načina.

Permutacije sa ponavljanjem članova skupa

Ako između *n* zadanih elemenata ima k_1 jednakih jedne vrste, k_2 jednakih druge vrste, ..., k_r jednakih *r*-te vrste, govorimo o permutacijama s ponavljanjem

Naravno, za *n* = 0 imamo prazan skup i samo jedan način, pa je 0! = 1.

Kod permutacije je od značaja redosljed elemenata. Kada se iz skupa *n* uzme podskup od *r* različitih elemenata, gdje je njihov redosled bez znaeaja, onda je riječ o *kombinaciji bez ponavljcnja*.

Varijacije

Varijacije su u stvari permutacije na osnovu podskupova od *r* elemenata uzetih u isti mah iz skupa od *n* elemenata.

Varijacije bez ponavljanja članova skupa:

Varijacije *k*-tog razreda od *n* elemenata su načini na koje možemo izabrati *k* elemenata iz skupa od *n* elemenata uz razlikovanje redosljeda izbora elemenata

$$V = n(n-1)(n-2)...(n-r+1) = \frac{n!}{(n-r)!} = \binom{n}{r} r! = Kr!$$

Varijacije sa ponavljanjem članova skupa:

$$Pp = \frac{n!}{r!s!}$$

Kombinacije

Kombinacije bez ponavljanja članova skupa:
Kombinacije *k*-tog razreda od *n* elemenata su načini na koje možemo izabrati *k* elemenata iz skupa od *n* elemenata bez obzira na redosljed izbora.

$$K = \frac{n!}{r!(n-r)!} = \binom{n}{r} = \binom{n}{n-r}$$

Kombinacije sa ponavljanjem članova skupa:

$$Kp = \frac{(n+r-1)!}{r!(n-1)!} = \binom{n+r-1}{r} = \binom{n+r-1}{n-1}$$

Raspodjele vjerovatnoće

Raspodjela vjerovatnoće je funkcija koja dodjeljuje vjerovatnoće elementima nekog skupa. Raspodjela je diskretna ako je taj skup prebrojiv (najčešće podskup skupa prirodnih brojeva), a neprekidna ako je funkcija raspodjele definisana na nekom konačnom ili beskonačnom intervalu skupa realnih brojeva i neprekidna na njemu. Skoro sve raspodjele od praktične važnosti su ili diskretne ili neprekidne.

n	***np***	**Raspodjela**
$n \leq 50$	nebitno	Binomna raspodjela
$n > 50$	$np \leq 10$	Puasonova raspodjela
$n > 50$	$np > 10$	Normalna raspodjela

Binomna raspodjela

Jakov Bernuli (1654-1705).

Binomna raspodjela je distribucija vjerovatnoća diskretnog (prebrojivog skupa elemenata) nezavisnih slučajnih događaja jednakih, konstantnih vjerovatnoća.

Izvodi se niz S_n od *n* nezavisnih eksperimenata, opita. U svakom je vjerovatnoća realizacije, ishoda događaja *A* ista i jednaka $p \in [0, 1]$. Vjerovatnoća da se događaj *A* nije realizovao je $q = 1 - p \in [0, 1]$. Pratimo da li se događaj *A* realizovao, ili nije.
Vjerovatnoća da se tačno $k \in [0, n]$ puta realizovao dati događaj je:

$$P_{n,k;p} = P(S_n = k; p) = \binom{n}{k} p^k q^{n-k}.$$

Naime, prvi faktor "n nad k" je broj kombinacija, na koliko načina možemo iz niza od *n* elemenata izvući *k* potskupova. Drugi faktor je *k*-to struki proizvod vjerovatnoća tih *k* događaja *A*, dok je treći faktor proizvod $n - k$ vjerovatnoća preostalih *n* događaja u nizu.
Vjerovatnoća da se dati događaj u datom nizu realizovao 0, 1, 2, ..., *k* puta je:

$$P(S_n \leq k; p) = \sum_{r=0}^{k} \binom{n}{r} p^r q^{n-r}.$$

Ova posljednja vjerovatnoća, za $k = 0$ je nula, a za $k = n$ je 1.
Naziv "binomna raspodjela" dat je prema vjerovatnoćama $P_{n,k;p}$ koje su članovi binomnog razvoja:

$$(p+q)^n = \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} =$$

$$= q^n + \binom{n}{1} p q^{n-1} + \ldots + \binom{n}{n-1} p^{n-1} q + p^n =$$

$$= \sum_{k=0}^{n} P_{n,k;p} = 1,$$

jer je $p + q = 1$.

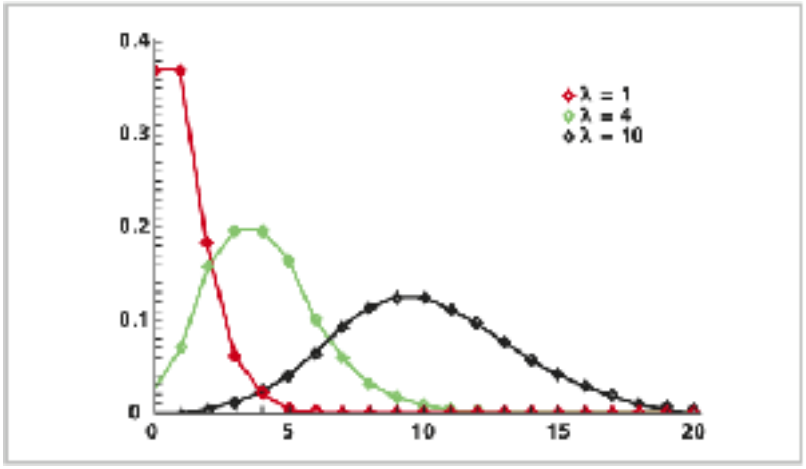
Primjer 1. 	**Bacamo novčić** i padaju Pismo ili Glava ravnopravno. Vjerovatnoća da će pasti Pismo u jednom bacanju je 1/2, jer su jednake šanse za svaki od 2 ishoda: P i G.
Vjerovatnoća da će pasti bar jedno Pismo u dva bacanja je 3/4, jer su jednake šanse za svaki od 4 ishoda: PP, PG, GP i GG. Vjerovatnoća da će Pismo pasti tačno dva puta u tri bacanja je 3/8, jer su jednake šanse za svaki od 8 ishoda: PPP, PPG, PGP, PGG, GPP, GPG, GGP i GGG. Uopšte, postavljamo pitanja kolika je vjerovatnoća da u nizu od n = 1, 2, 3, ... bacanja Pismo padne tačno k = 1, 2, ..., n puta?	
Primjer 2. 	**Bacamo kocku** za igru "Ne ljuti se čovječe" i padaju brojevi 1, 2, ..., 6, ravnopravno. Vjerovatnoća da će pasti šestica u jednom bacanju je p = 1/6, jer su jednake šanse za svaki od šest ishoda.
Vjerovatnoća da će pasti bar jedna šestica u dva bacanja je 11/36, jer su jednaki izgledi za svaki od sljedećih 36 ishoda: 11, 12, ..., 16; 21, 22, ..., 26; ...; 61, 62, ..., 66. Drugim riječima, trebamo ishode oblika 6x, ili x6, gdje na mjestu x može biti bilo koji od pet brojeva koji nije šestica, plus ishod 66. Treći način, 11 ostane ako od svih 36 ishoda izbacimo one oblika xy gdje su na mjestima x, ili y bilo koji od brojeva 1 - 5. Vjerovatnoća da će pasti tačno dvije šestice u tri bacanja kocke je 15/216, jer ima tačno 15 načina da padnu dvije šestice u nizu od $6^3 = 216$ ravnopravnih ishoda. Naime, mogući su 15 slučajeva oblika 66x, 6x6 i x66, gdje na mjestu x može biti bilo koji od pet brojeva koji nije šestica.	

Za velike brojeve n *i* k *je teže izračunavanje pomenutih vjerovatnoća, ali se Binomna raspodjela može aproksimirati Puasonovom, zatim Normalnom. Aproksimacija Puasonovom raspodjelom je utoliko bolja ukoliko je* n *veće (veće od 50) i* p *manje (manje od 0,1). Prema teoremi Muavra-Laplasa, binomna raspodjela teži normalnoj raspodjeli.*

Puasonova raspodjela

Puasonova raspodjela je diskretna raspodjela vjerovatnoća koja se danas najčešće upotrebljava za izračunavanja kod događaja unutar datog intervala vremena. Nekada, ona je bila glavna zamjena za Binomnu raspodjelu kod relatovno malih vjerovatnoća p (manjih od 0.2), kada je broj ponavljanja n veći od 50.

Definicija



Neka je X **slučajna promjenljiva** za koju važi zakon raspodjele

$$P(X = k) = e^{-\lambda}\frac{\lambda^k}{k!}.$$

Ova raspodjela se naziva Puasonova raspodjela (eng. *Poisson Distribution*).
Pošto Puasonova raspodjela zavisi samo od parametra lambda, označavamo je $\wp(\lambda)$.
Na grafu lijevo gore, apscisa je $x = k$, a na na ordinati su vjerovatnoće Puasonove funkcije za tri različita parametra lamda: strmo padajuća crvena kriva, zvonasta zelena sa nižim maksimumom, i najniža crna kriva, redom za $\lambda = 1, 4$ i 10.
Puasonova raspodjela dobro zamjenjuje Binomnu raspodjelu kada je $np = \lambda < 10$ i kada je broj ponavljanja $n > 50$.

Primjer 1.
Vjerovatnoća da je **proizvod defektan** je 0,01. Iz velikog skladišta se uzima 100 proizvoda. Kolika je vjerovatnoća da među tih 100 proizvoda:
a. bude tačno 5 defektnih;
b. broj defektnih nije veći od 10.
Rješenje. Ovdje je $\lambda = np = 100 \cdot 0.01 = 1 < 10$, pa umjesto Binomne možemo koristiti Puasonovu raspodjelu:

$$a.\quad P(X_{100} = 5) = e^{-1}\frac{1^5}{5!} = 0.003;$$
$$b.\quad P(X_{100} \le 10) = \sum_{k=0}^{10} e^{-1}\frac{1^k}{k!} = 1.000.$$

Primjer 2.
Neka je X(t) broj čestica koje emituje radioaktivni izvor za t časova. Pretpostavimo da X(t) ima Puasonovu raspodjelu s parametrom $\lambda = 20t$. Kolika je vjerovatnoća da izvor emituje tačno 5 čestica u nekom 15-minutnom intervalu?
Kako X(t) ima raspodjelu $\wp(20t)$, za t = 0.25 časova imamo da X(0.25) ima raspodjelu $\wp(5)$, pa je:

$$P(X(0.25) = 5) = 0.175.$$

Poopštenje
Značaj Puasonove raspodjele ne iscrpljuje se time što je ona jedna aproksimacija binomne raspodjele. Postoje matematički modeli vezani za "protok događaja" koji su prirodno vezani za Puasonovu raspodjelu, tada kada je raspodjela vjerovatnoća slučajne promjenljive X na datom intervalu $[a, b]$: homogena, nezavisna i separabilna.
Homogenost: raspodjela vjerovatnoća za $X[a, b]$ ne zavisi od položaja intervala $[a, b]$, već samo od njegove dužine $b - a$:

$$P_k(b - a) = P(X[a, b] = k), \quad k = 0, 1, 2, \ldots.$$

Nezavisnost: ako su intervali $[a_1,b_1]$ i $[a_2,b_2]$ disjunktni (nemaju zajedničkih tačaka) onda su događaji $X[a_1,b_1]$ i $X[a_2,b_2]$ nezavisni.
Separabilnost: znači praktičnu nemogućnost da se u istom trenutku dogodi više od jednog događaja:

$$\lim_{\Delta t \to 0} \frac{P(X[t, t + \Delta t] > 1)}{\Delta t} = 0.$$

Pokazuje se da pod ovim pretpostavkama slučajna promjenljiva $X[0,t] = X(t)$ ima Puasonovu raspodjelu, gdje je lambda > 0 fiksiran parametar koji karakteriše intenzitet "protoka događaja".

Uslovna vjerovatnoća

Često smo u prilici da tražimo vjerovatnoću nekog događaja A, posjedujući informaciju o tome da se događaj B realizovao ili pretpostavljajući da će se realizovati.
Definicija:
Verovatnoća *P(A|B),* zove se ***uslovna vjerovatnoća*** dodađaja A pod uslovom B i definiše se sa

$$P(A|B) = \frac{P(AB)}{P(B)}, \text{ za } P(B) > 0.$$

Primjer:
U kesi se nalazi 5 belih i 9 crnih kuglica. Izvlačimo nasumice 2 kuglice, jednu po jednu, bez vraćanja. Kolika je verovatnoća da ćemo iz drugog puta izvući crnu, ako je u prvom izvlačenju izvučena bela kuglca?
Rješenje:
Neka je A događaj izvlačenja crne kuglice, a B vjerovatnoća izvlačenja bele kuglice iz prvog izvlačenja.

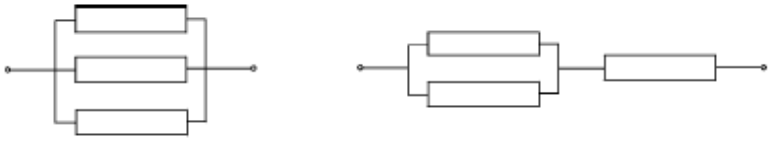
$$P(B)=\frac{5}{14}.$$

$$P(A\cap B)=P(AB)=\frac{45}{V_2^{14}}=\frac{45}{14\cdot 13}=\frac{45}{182}$$

$$P(A\setminus B)=\frac{P(AB)}{P(B)}=\frac{9}{13}$$

Primjer2: Novčić se baca ili do pojave grba ili do tri uzastopne pojave pisma. Pod uslovom da je rezultat prvog bacanja pismo, naći vjerovatnoću da dinar bude bačen 3 puta.
Prilikom bacanja novčića moguće su situacije.
G, PG, PPG, PPP

Kao sledeći primer posmatrajmo dva sistema A i B prikazana na slici 3.2 čije sve komponente imaju istu pouzdanost p. Odredimo koji od njih ima veću pouzdanost.



A B

Slika 3.2 Koji sistem je pouzdaniji?

Sistem A ne radi ako i samo ako ni jedna komponenta ne radi. Verovatnoća da sistem ne radi jednaka je $(1-p_1)(1-p_2)(1-p_2)=(1-p)^3$. Nas interesuje verovatnoća suprotnog događaja (da sistem radi), tako da je pouzdanost sistema A data sa

$$p_A=1-(1-p)^3=3p-3p^2+p^3.$$

Isti rezultat se dobija primenom Teoreme zbira verovatnoća.

Kombinujući rezultate iz prethodnog primera za rednu i paralelnu vezu, nalazimo pouzdanost sistema B kao verovatnoću

$$p_B=(p_1+p_2-p_1p_2)\cdot p_3=(2p-p^2)p=2p^2-p^3 \quad (p\in(0,1)\ .$$

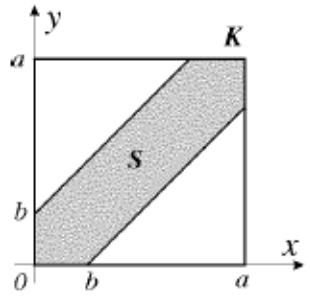
Kako je

$$p_A-p_B=(3p-3p^2+p^3)-(2p^2-p^3)=p(2p^2-5p+3)=2p(1-p)\left(\tfrac{3}{2}-p\right)>0,$$

nalazimo da je $p_A>p_B$, dakle, sistem A je pouzdaniji.

Primer 4.1. Osobe A i B dogovorile su se da se susretnu na određenom mestu između 13^{h} i 13^{h} i a minuta. Prema dogovoru, onaj ko prvi dođe čeka na drugog b minuta. Odrediti verovatnoću da dođe do susreta ako se pretpostavlja da je vreme dolaska za svaku od osoba podjednako verovatno raspoređeno između 13^{h} i 13^{h} i a minuta.

Rešenje: Neka je osoba A došla u 13^{h} i x minuta, a osoba B u 13^{h} i y minuta. Tada je, prema uslovu zadatka, $0 \leq x \leq a$, $0 \leq y \leq a$ (mogući ishodi). Do susreta će doći ako i samo ako pored ovih uslova bude i $|x-y| \leq b$. Dakle, tačka sa koordinatama (x, y) može se nalaziti u kvadratu K stranice a, a do susreta će doći ako se nalazi u osenčenoj oblasti S (povoljni ishodi) koja je ograničena pravama $y = x - b$, $y = x + b$ i stranicama kvadrata (slika 4.1). Odnos ovih površina daje traženu verovatnoću:



Slika 4.1

$$p = \frac{S}{K} = \frac{a^2 - (a-b)^2}{a^2} = \frac{b(2a-b)}{a^2}.$$

Koja je vjerojatnost da ćemo iz skupine od 52 igraće karte izvući karo kralja, srce desetku i tref asa?

Odgovor

Oprez,

to nije 1/52*1/52*1/52

nego 1/52*1/51*1/50 = 1/132 600

jer –pošto smo izvukli jednu kartu, preostalo ih je još 51 itd.

To će se dakle u prosjeku dogoditi 1 put u 132 600 pokušaja.

Banka raspolaže s tri identična kompjuterska sustava: osnovnim i dva rezervna (back-up).

Sustavi rade neovisno, s istom programskom podrškom.

Prema navodu proizvođača, vjerojatnost zastoja sustava u jednom danu zbog hardwerskih poteškoća iznosi 0,01.

Kolika je vjerojatnost da nastane zastoj u radu svih triju sustava u jednom danu?

Odgovor

P= p(A1)*p(A2)*p(A3)=0,013= 0,000001.

Ako na kladionici treba pogoditi rezultate 12 utakmica, od kojih svaka može imati tri ishoda (1, 2, 0), koliki je mogući broj kombinacija?

Kolika je vjerojatnost da ćemo pogoditi?

a) 3*3*3*3*3*3*3*3*3*3*3*3 = 531 441

Dakle, vjerojatnost da ćemo slučajno pogoditi iznosi 1/531 441= 0, 0000019 što praktički znači 2 na milijun.

Ako u restoranu možemo dobiti ručak koji se sastoji od juhe, mesa, priloga i kolača, a može se birati između

4 juhe, 3 vrste mesa, 5 vrsta priloga i 4 vrste kolača,

koliko se kombinacija može sastaviti iz ovog menija?

Iz ovog menija može se sastaviti

4*3*5*4=240 kombinacija.

Koja je vjerojatnost da ćemo u igri pogađanja (npr. loto) od 49 brojeva pogoditi 6 brojeva?

$$\frac{49!}{6!(49-6)!} = \frac{49!}{6!43!} = \frac{49*48*47*46*45*44}{6!} = 13983816$$