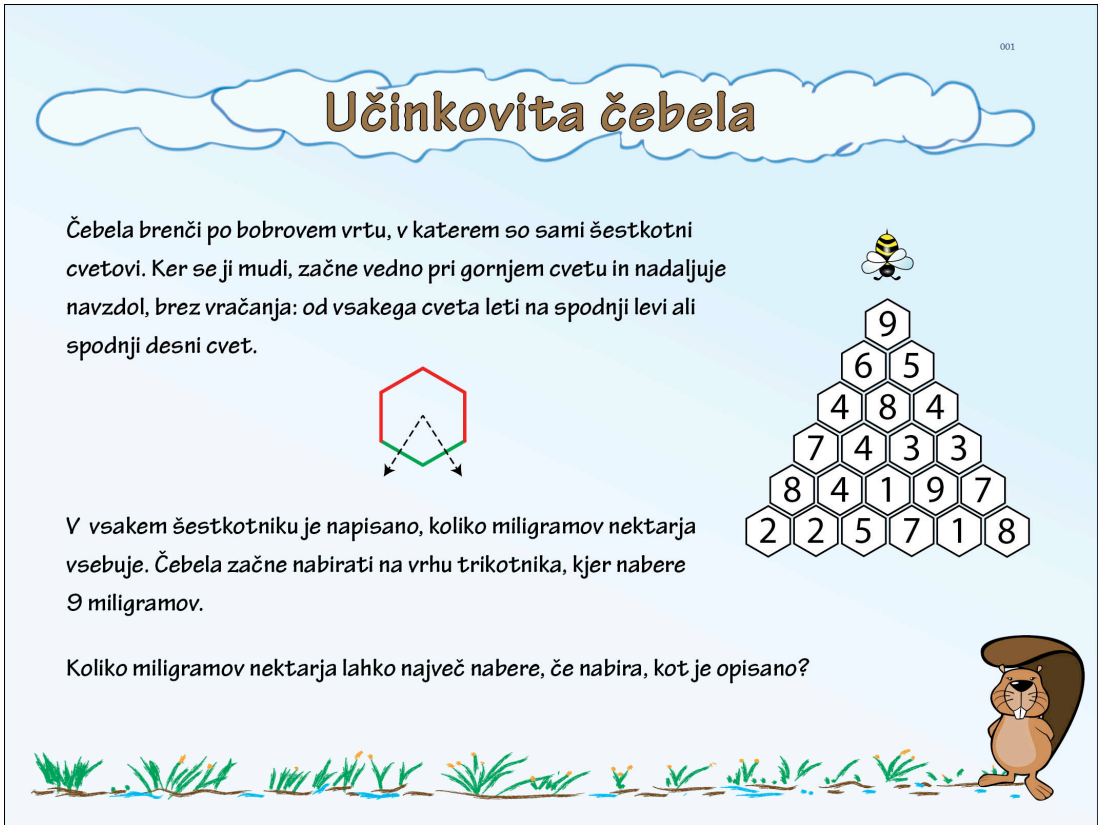
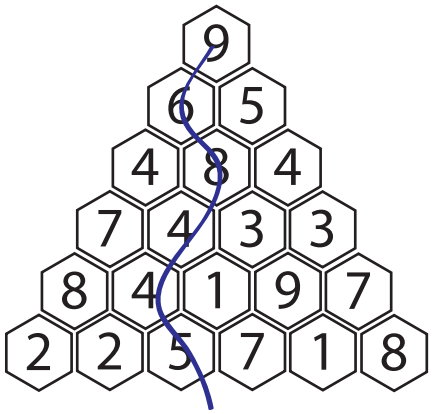
Dinamično programiranje

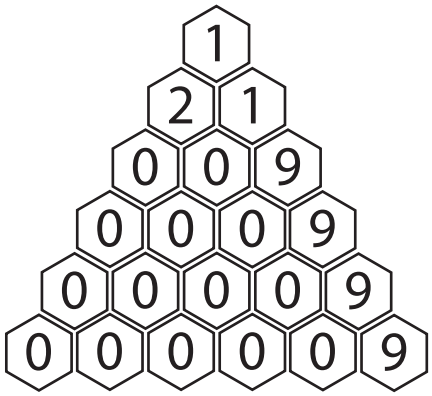


Prva – kar takoj povejmo, da ne preveč dobra – ideja, je požrešna: želva v vsakem koraku zavije k cvetu z več medu.



Rezultat je 36 mg medu. Pesimist pomisli, da je to najbrž tudi največ, kar lahko dobimo. Optimist se strinja rekoč, da je požrešna metoda gotovo dobra metoda.

Najprej razočarajmo optimista: požrešna metoda sicer lahko včasih slučajno da najboljšo rešitev, v splošnem pa ne. O tem nas hitro prepriča spodnja hudobna postavitev cvetov.



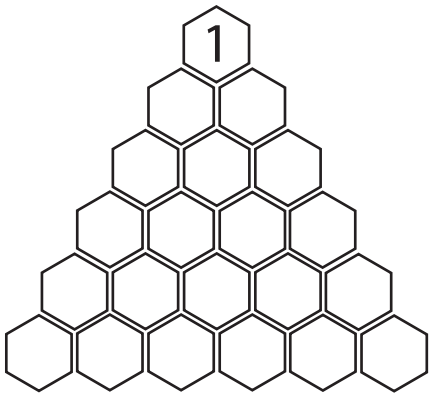
Ko čebela v prvem koraku podleže požrešnosti in odleti na cvet z dvema miligramoma medu, je njena usoda zapečatena: nabrala ne bo ničesar več. Pohlevnost v prvem koraku bi jo vodila prek bogato založenih cvetov na desni.

Tudi otroci, ki rešujejo nalogo s tekmovanja, opazijo, da bi bilo namesto zaključka 4 + 4 + 5 bolj donosno iti po poti 3 + 9 + 7 – ko bi iz cveta z 8 mg zavili na 3 namesto na 4. Tedaj se pojavijo dvomi: je 9 + 6 + 8 + 3 + 8 + 7 največ, kar lahko dobimo, ali pa bi šlo še boljše?

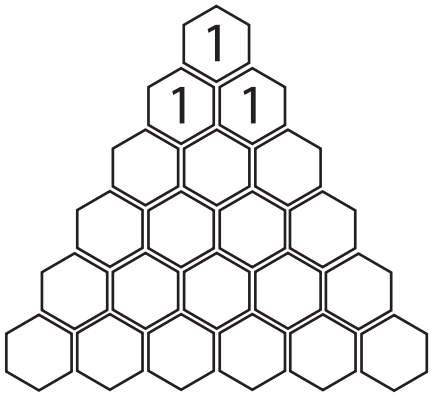
Na prvi pogled je rešitev le ena: preveriti vse možne poti. Nekateri otroci so se res lotili tega podviga; rezultat je odvisen od njihove vztrajnosti in sistematičnosti. Kdor poseduje oboje: ni težav. Nesistematični se bodo izgubili. Nevztrajni pa se bodo sredi dela vprašali: koliko tega me še čaka?

Odgovorimo jim. Na koliko različnih načinov lahko čebela preleti vrt, od gornjega polja do kateregakoli od cvetov v spodnji vrstici? Odgovor na to vprašanje nam sicer ne bo pomagalo rešiti problema (ali pač, speljal nas bo na prave misli), je pa zanimiv, zato ga le poiščimo.

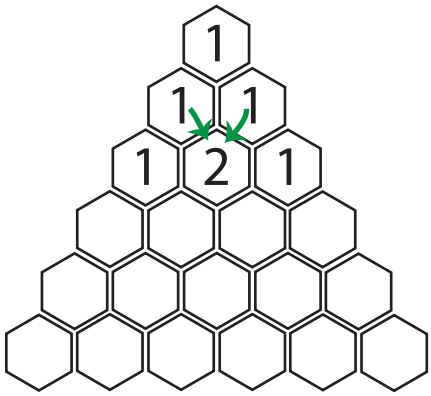
Najprej se vprašajmo nekaj preprostega: na koliko načinov lahko pridemo na prvo polje? Očitno na enega samega – tam pač začnemo.



Na koliko načinov pa pridemo na polji v drugi vrsti? Na vsakega od njiju pridemo na en sam način, namreč s prvega polja.

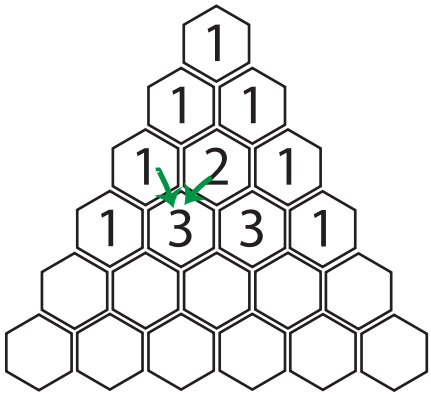


Na koliko načinov pa pridemo do polj v tretji vrsti? Na skrajno levo in skrajno desno polje pridemo na en sam način, iz skrajno levega ali skrajno desnega polja prejšnje vrste. Na srednje pa pač na dva, bodisi z leve bodisi z desne.

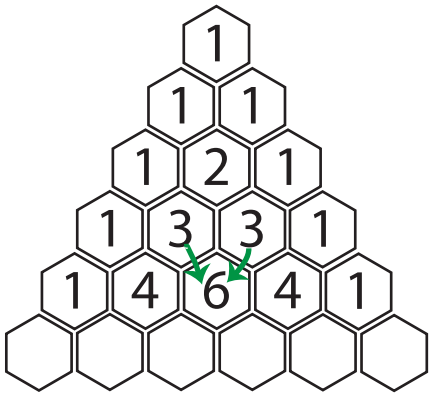


Zdaj pa postane reč zanimivejša, čeprav bo vprašanje enako: na koliko načinov pridemo z začetnega polja do posameznih polj v četrti vrsti? Na skrajni polji še vedno pridemo le iz skrajnih polj. Na drugo polje pa lahko pridemo na tri načine. Takole. Obstaja natančno en način, na katerega pridemo od začetka pa do levega polje tretje vrste; torej obstaja en način, kako priti od začetka prek levega polja tretje vrste do drugega polja četrte. Od začetnega polja do srednjega polja tretje vrste pa pridemo, kot vemo, na dva načina. Torej obstajata dva načina, da pridemo od začetnega polja prek srednjega polja tretje vrste do drugega polja četrte. Skupaj so torej 1 + 2 = 3 načini, da pridemo od začetnega polja do drugega polja četrte vrste.

Tretje polje je podobna zgodba.



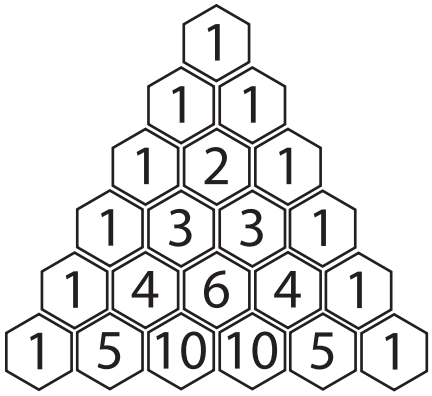
Pa četrta vrsta? Še enkrat ponovimo razmišljanje od prej, a tokrat se osredotočimo na najzanimivejše, srednje polje, tisto, v katerem piše 6.



Vanj lahko pridemo iz drugega ali tretjega polja četrte vrste. Kot že vemo, vodijo natančno tri različne poti od začetnega polja do drugega polja četrte vrste. To pomeni, da obstajajo tri poti od začetnega polja prek drugega polja četrte do srednjega polja pete vrste. Prav tako obstajajo tri poti prek tretjega polja četrte vrste. Skupaj pridemo do srednjega polja pete vrste na 3 + 3 = 6 načinov – tri poti pripeljejo z leve, tri z desne.

Podobno smo naračunali štirico: do drugega polja pete vrste pridemo na en način z leve in na tri načine z desne (ker pač obstajajo tri poti do polja na desni (ali levi, s čebeline perspektive)).

Enako naračunamo še zadnjo vrsto.



Otroci tega ne vedo, mi, ki smo odrasle, zrele osebe, pa vemo, kako se reče tej reči: Pascalov trikotnik.

Zdaj vemo, ena od možnih poti se konča na skrajnem levem polju, pet na drugem, deset na tretjem in tako naprej. Vseh možnih poti je 1 + 5 + 10 + 10 + 5 + 1 = 32.

32? Nekam čudno okrogla številka – za računalnikarja. Hm, če bi imeli eno vrsto manj, bi bilo možnih poti 1 + 4 + 6 + 4 + 1 = 16. Če bi imeli le štiri vrste, bi jih bilo 1 + 3 + 3 + 1 = 8, s tremi jih je 1 + 2 + 1 = 4, z dvema 1 + 1 = 2 in z eno 1 = 1. Nenavadno naključje?

Niti ne. Če vemo, kaj računa Pascalov trikotnik, je jasno, da mora biti tako. Pascalov trikotnik računa binomske koeficiente in z njimi velja:

$$(a+b)^n = \binom{n}{0} a^n b^0 + \binom{n}{1} a^{n-1} b^1 + \binom{n}{2} a^{n-2} b^2 + \cdots \binom{n}{n} a^0 b^n$$

če zvito rečemo $a = b = 1$, izvemo, da je $2^n = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n}$.

V našem primeru je *n* enak 5 (prva vrstica ima številko 0), torej imamo $2^5 = 32$ možnih poti.

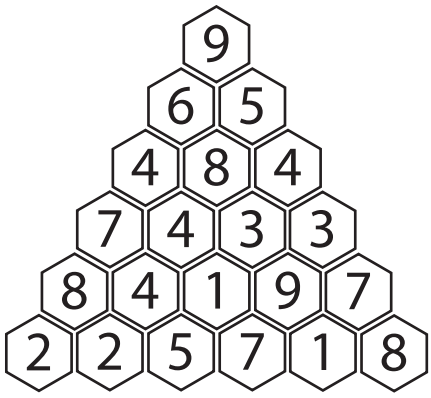
Ah, pa je bil ves ta trud res potreben? Saj gre preprosteje! Čebela se mora petkrat odločiti, kam gre. Vsakič ima dve možnosti. Z nekaj kombinatorike – dovolj preproste, da jo razložimo desetletniku, če debato začnemo z vprašanje, na koliko načinov se lahko obleče, če ima dvoje čevljev, dvoje hlač in dvoje srajc –ugotovimo, da je možnih kombinacij odločitev 2×2×2×2×2=32.

Čemu smo se mučili z vsem tem preštevanjem? Zaradi dveh stvari. Najprej, želimo se prepričati, kako narašča število poti: vsaka dodatna vrstica podvoji število poti. Algoritem, ki pravi "preveri vse možne poti" je neuporabno počasen: čas računanja ne narašča linearno s številom vrstic (dvakrat več vrstic = dvakrat več računanja), niti ne kvadratno s številom vrstic (dvakrat več vrstic = štirikrat več računanja). (Mimogrede se spomnimo, da pri algoritmih urejanja nismo marali tistih s kvadratno časovno zahtevnostjo!) Še več, ne narašča niti s kubom ali četrto potenco števila vrstic: *ena vrstica več = dvakrat več računanja*. Čas izvajanja je sorazmeren 2^n, kjer je *n* število vrstic. Računalnikarji to sorazmerje označimo z $O(2^n)$.

Takšne časovne zahtevnosti so nesprejemljive. Takšnih algoritmov ne maramo, saj delujejo le pri res res res majhnih problemih. Potrebno si bo izmisliti boljšega.

In zdaj pride drugi razlog, zakaj se nam je zdelo koristno preštevati poti: ker bomo na podoben način razmišljali, ko bomo sestavljali algoritem.

Na tem mestu bomo postali nekoliko matemetični, formalni in zateženi. Bralec, ki ga to plaši, lahko ta del mirno preskoči. S seboj povabim druge vas junake: stvar ni tako težka, zato vsaj poskusite. Z ostalimi se ponovno vidimo, ko bomo **začeli razmišljati naprej**.



Najprej se zmenimo za oznake. Polja bomo označevali tako, da bomo zapisali številko vrstice in nato številko polja. Tretje polje v šesti vrstici zapišemo s "koordinatami" (6, 3). Zanimalo nas bo, koliko medu lahko čebela največ nabere na poti do nekega polja (a, b); to število bomo označili kot $m(a, b)$. Če rečemo, recimo, m(3, 1) = 23, bo to pomenilo,

da lahko čebela na poti do (vključno) polja (3, 1) (srednje polje v tretji vrsti) nabere največ 23 mg medu.

Da bomo lažje napisali kakšno splošno formulo, recimo še, da s $q(a, b)$ označimo količino medu v polju (a, b). Tako je $q(6, 3) = 5$.

Za začetek problem rešujmo narobe, ritensko. Izberimo si neko polje, recimo tretje polje v zadnji, šesti vrsti in se vprašajmo, koliko, največ, medu lahko nabere čebela, če bo pot končala v njem, torej, koliko je m(6, 3). Odgovora sicer ne vemo, vemo pa, da čebela pride v (6, 3) bodisi s polja (5, 2) bodisi s (5, 3). Ker jo v (6, 3) čaka 5 mg medu, bo $m(6, 3)$ enak bodisi $m(5, 2) + 5$ bodisi $m(5, 3) + 5$ – toliko, kolikor nabere do enega od teh dveh polj in še 5 zraven.

Zdaj pa pazimo, $m(6, 3)$ nista dve številki, temveč ena sama: $m(6, 3)$ pove, koliko največ medu lahko čebela nabere do (vključno) polja (6, 3). Odgovor na to bi bil lahko preprost: če je m(5, 2) večje od m(5, 3), bo šla čebela, ki hoče na vsak način končati pot v polju (6, 3), raje prek (5, 2), saj bo tako nabrala več. Če je m(5, 3) večje od m(5, 2), pa bo šla do (6, 3) raje prek (5, 3). Nabrala bo torej toliko, kolikor dobi v enem ali drugem od teh dveh polj in še 5 zraven, torej $m(6, 3) = \max(m(5, 2), m(5, 3)) + 5$.

Žal pa smo mogli napisati le "bi bil lahko preprost", to pa zato, ker se nam niti ne sanja, koliko bi utegnila biti m(5, 2) in m(5, 3). A tudi na to vprašanje vemo poiskati odgovor: do (5, 2) pridemo bodisi iz (4, 1) bodisi iz (4, 2) – pač od ondod, od koder se bolj splača. V (5, 2) so 4 mg medu, zato $m(5, 2) = \max(m(4, 1), m(4, 2)) + 4$. A tu spet naletimo na isti problem: koliko medu lahko naberemo do (4, 1) in koliko do (4, 2)? Tudi ta problem rešimo na enak način.

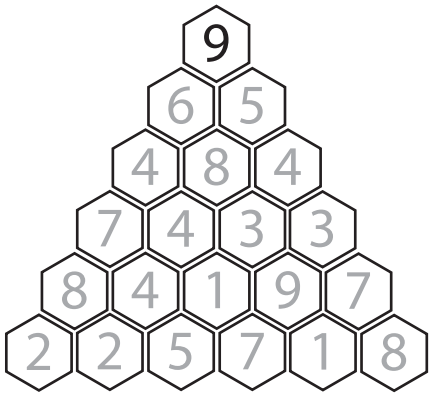
V splošnem velja: $m(a, b) = \max(m(a - 1, b - 1), m(a - 1, b)) + q(a, b)$, oziroma, na robovih, $m(a, 0) = m(a - 1, 0) + q(a, 0)$ ter $m(a, a) = m(a - 1, a - 1) + q(a, a)$.

Problem ritenskega razmišljanja je tule: ko bomo računali, koliko medu lahko naberemo do (5, 2), bomo morali izračunati, koliko ga je mogoče nabrati do (4, 2). Nekoliko kasneje se bomo vprašali, koliko medu je mogoče nabrati do (5, 3), zapisali bomo $m(5, 3) = \max(m(4, 2), m(4, 3)) + 1$ ... in ponovno računali, koliko medu je potrebno nabrati do (4, 2)!

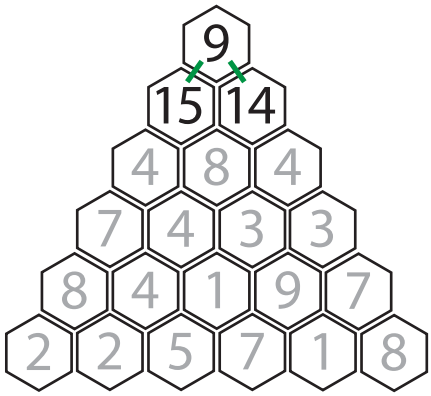
Z malo nespretnosti bomo prišli do algoritma, za katerega z malo spretnosti izračunamo, da bo potrebovali nebodijihtreba 2^n računanj. Temu se izognemo tako, da si vse, kar smo že enkrat izračunali, zapomnimo. Pri računanju nazaj je to nerodno.

Zato raje razmišljajmo naprej. (Mastni tisk je uporabljen, da pritegne vse matematikofobne bralce, za katere je besedilo od tega mesta naprej spet varno.) Razmišljanje naprej bo nenavadno podobno načinu, na katerega smo ravno prejle zabredli v računanje Pascalovega trikotnika.

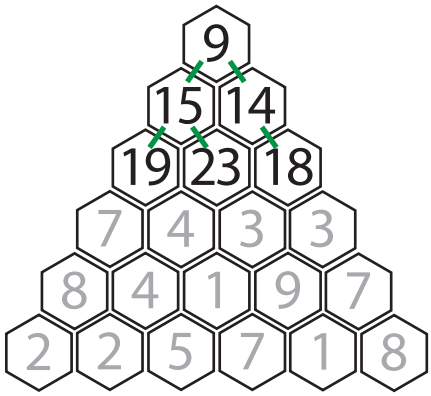
Na poti do (vključno) prvega polja čebela vedno nabere 9 mg medu.



Tudi polji v drugi vrstici ne zahtevata posebnega razmišljanja: do levega nabere 9 + 6 = 15 mg in do desnega 9 + 5 = 14 mg. Številki zapišimo kar v polje, namesto (ali prek, če rešujemo na papir) številk s količino medu.

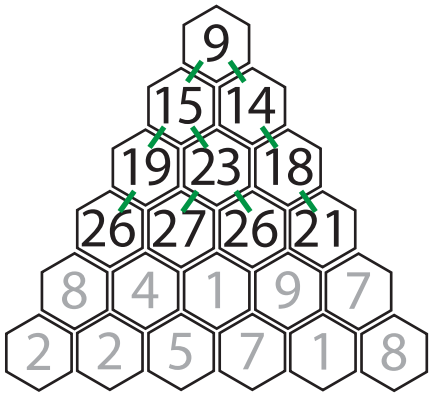


Tretja vrstica je malenkost zanimivejša:

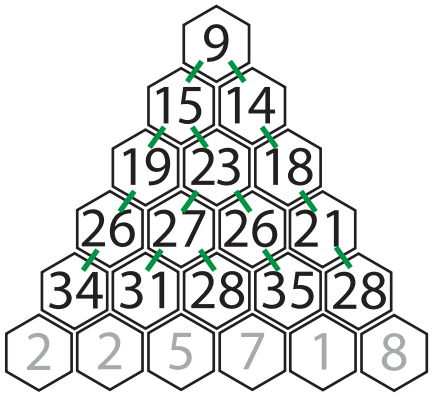


Stranski polji sta trivialni, do srednjega pa se bolj splača priti z leve, s 15. Na ta način čebela nabere 15 + 8 = 23 mg medu; če bi prišla z druge strani, bi ga le 14 + 8 = 22 mg. V polje napišemo 23 in zabeležimo, s katere strani je potrebno prileteti vanj.

V četrti vrstici skrajni polji, kot vedno, nimata dilem. V srednji dve se najbolj splača prileteti s srednjega polja tretje vrstice, saj čebela tako nabere 23 + 4 = 27 in 23 + 3 = 26 mg medu. Številki napišemo v polji in narišemo črtici, s katero označimo, odkod je potrebno priti nanju.

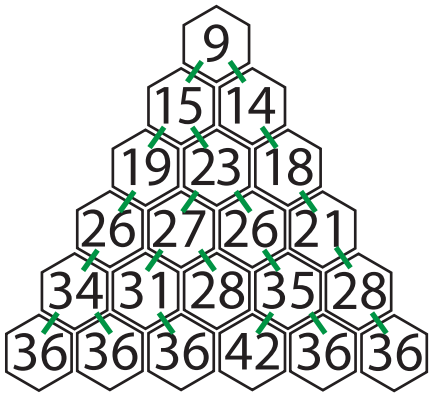


Zdaj preračunamo peto vrstico.

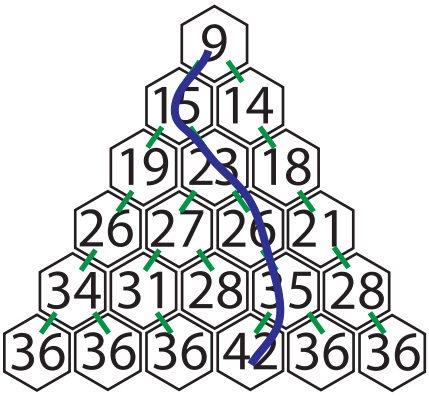


Prvo polje je jasno. V drugega pridemo z desne, s polja s številko 27 (ker je pač 27 več kot 26). Tako čebela namere 27 + 4 = 31 mg, kar napišemo v polje in dodamo črtico. Tudi v tretje polje pride z istega polja iz četrte vrstice; nabrala bo 27 + 1 = 28 mg medu, kar vpišemo in dodamo črtico. V četrto polje prileti z leve (ker je 26 več kot 21), pri čemer dobi 26 + 9 = 35 mg medu; vpišemo in narišemo črtico. Skrajno desno polje je spet trivialno.

Na koncu na enak način izračunamo še zadnjo vrstico.



Če se čebela odloči končati na četrtem cvetu zadnje vrste, lahko nabere (največ) 42 mg medu; če kjerkoli drugje, le 36. Kje pa mora iti, da nabere toliko? Zdaj sledimo poti nazaj: v ta cvet mora priti z desne (35), tja z leve (26), vanj z leve (23) in vanje spet z leve (15), vanj pa, jasno, s prvega cveta.



Koliko računov je potrebno opraviti za n vrstic? Toliko, kolikor je v n vrsticah polj. To pa je 1 + 2 + 3 + 4 + ... + n, kar je, kot že vemo, sorazmerno n^2. Ob urejanju smo nad časovnimi zahtevnostmi, ki so bile sorazmerne kvadratu števila elementov, godrnjali. Tu je to najboljše, na kar moremo upati: vsako polje moramo pač nujno pogledati vsaj enkrat, ne?

Splošno o dinamičnem programiranju

Dinamično programiranje je učinkovit in popularen pristop k snovanju algoritmov. Tule smo si izmislili algoritem za računanje optimalne poti čebele: algoritem je splošen in bi dal pravilen rezultat tudi, če bi bila količina medu v cvetovih drugačna.

Dinamično programiranje je praktično vedno, kadar lahko rešitev naračunamo iz rešitev istega problema, ki so v nekem smislu enostavnejše. Tule "enostavnost" pomeni krajšo pot: kakšna je optimalna pot do nekega cveta izvemo, če poznamo eno ali dve krajši optimalni poti. Tudi tidve naračunamo iz še krajših poti ... dokler ne pridemo do ene same poti, kjer ni več kaj razmišljati (v našem primeru do začetnega polja).

Ali, če razmišljamo naprej: zanima nas rešitev določenega problema (najboljše poti do določenega polja). Namesto, da bi rešili ta problem, rešujemo preprostejše probleme (vedno daljše poti od začetnega polja) in tako "širimo fronto" svojega znanja, dokler ne pridemo do rešitve, ki jo dejansko iščemo (količine nabranega medu v vseh končnih poljih).

Pri reševanju te naloge smo računali polja od spodaj navzgor. Fronto bi lahko peljali tudi drugače, na primer z leve proti desni – najprej bi izračunali maksimalno količino medu do vseh skrajnih desnih polj, nato do vseh drugih polj v vrsticah, pa do tretjih, četrtih... Lahko bi bili celo povsem nesistematičnih, paziti bi morali le, da za vsako polje, ki se ga lotimo računati, že poznamo količino medu v poljih nad njim.

Še več, v to smo včasih prisiljeni. V nalogi s čebelo so bili cvetovi lepo zloženi v trikotnik. Dinamično programiranje pogosto uporabljamo tudi v problemih, ki nimajo tako lepe, jasne, prostorske ponazoritve. Tam niti ne moremo govoriti o "od zgoraj navzdol" ali "z leve na desno", temveč pazimo le, da gremo po vrsti v tem smislu, da vedno poznamo vse, kar je potrebno poznati za izračun vrednosti funkcije pri določenih vrednostih.

Dinamično programiranje se obnese predvsem v problemih, v katerih bi bili sicer prisiljeni večkrat računati eno in isto vrednost funkcije. Ob razmišljanju o čebeli smo

tako opazili, da bi z malo nespretnosti dvakrat računali vrednost v polju (4, 2). Dinamično programiranje nas tega reši.

V kontekstu bobra še svarilo: algoritme, sestavljene s pristopom dinamičnega programiranja (ob grafih bomo namreč spoznali še enega podobnega), je težko izvajati ročno. Naloge na tekmovanjih je zato preprosteje in varneje reševati z drugo metodo, metodo ostrega pogleda, ki pravi, da *bulji, dokler ne vidiš*. Nič pa ni narobe, če otrokom, predvsem starejšim ali bistrejšim, telovadimo možgane tudi s sistematičnim reševanjem, kakršnega smo opisali tule.

Iskanje najkrajše poti

Med dvema točkama na grafu navadno obstaja več možnih poti. Če imajo različne povezave različno ceno, nas pogosto zanima tista, pri kateri je vsota povezav na njej najmanjša. Takšni poti rečemo *najkrajša pot*.

Algoritem, ki ga bomo spoznali, zahteva, da nobena cena ni negativna. Če ni tako, potrebujemo popolnoma drugačen algoritem, ki zahteva tudi veliko več časa (navadno celo preveč, da bi bil praktičen, zato namesto njega uporabljamo približne algoritme). Predpostavimo tudi, da iskana pot obstaja; če ciljno vozlišče sploh ni dosegljivo iz začetnega, bomo pot iskali zaman.

Iskanje najkrajših poti je ena najpopularnejših tipov nalog na tekmovanju Bober. Naloge na to temo: od očitnih, kjer je potrebno poiskati najkrajšo pot na zemljevidu, do prikritih, kot je iskanje najcenejšega ali najpreprostejšega zaporedja del, ki nas pripelje do končnega izdelka.

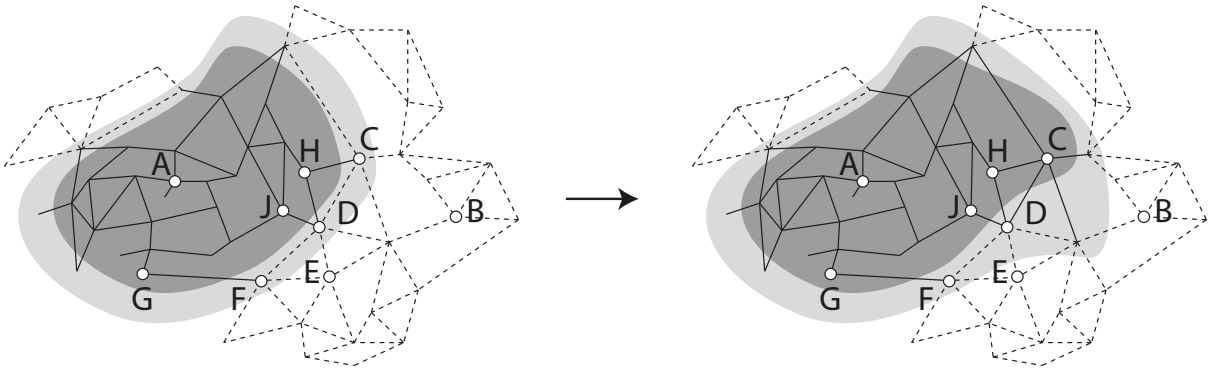
V nalogah, s kakršnimi se spopadajo osnovnošolci na tekmovanjih, je najkrajšo pot navadno mogoče poiskati kar z metodo ostrega pogleda. Sestavljene so namreč tako, da je število poti (razen očitno predolgih, na primer takšnih, v katerih se točke ponavljajo) dovolj majhno, da jih lahko sistematično pregledamo, ali pa se različne poti na enem ali več mestih združijo in lahko nalogo rešujemo tako, da iščemo najkrajše poti po kosih, med posameznimi "ozkimi grli".

Računalnik nima ostrega pogleda, pa tudi ljudje pri dovolj velikih grafih ne moremo več biti prepričani, da smo res preverili vse možnosti. Tedaj uporabimo algoritem, ki je

dobil ime po slavnem nizozemskem računalnikarju Edsgerju W. Dijkstri: Dijsktrin algoritem.

Algoritem ne poišče le najkrajše poti od začetne do ciljne točke, temveč tudi najkrajše poti do množice drugih točk. To počne tako, da začne pri začetni točki in nato postopno širi množico točk, do katerih pozna najkrajšo možno pot.

Točkam, do katerih je že odkrita najkrajša pot, bomo – iz razlogov, ki bodo postali jasni vsak čas – rekli *obiskane točke*. Točke, ki so neposredno povezane z obiskanimi, rečemo *mejne točke*. Algoritem si bomo najprej ogledali na skici, nato še na konkretnem primeru.



Poiskati želimo najkrajšo pot od A do B na gornji sliki. Začnimo na levi strani. Temnejši del kaže obiskane točke. Zanje že vemo, kakšna je najkrajša pot od A do njih – prek katerih točk vodi in koliko je dolga.

Mejne točke so v svetlejšem delu; primeri takšnih točk so C, D in F. Za mejne točke še ne poznamo najkrajše poti. Seveda lahko izračunamo, kako dolga bi bila, recimo, najkrajša pot do F, ki bi vodila prek G: k (že znani) dolžini najkrajše poti do G prištejemo dolžino povezave med G in F. Prav tako lahko za D izračunamo dolžino poti prek H (kot vsoto znane najkrajše poti do H in dolžine povezave med H in D) ter prek J; najkrajša pot do D, ki vodi naposredno iz obiskanega dela, je dolžina krajše od teh dveh možnih poti.

Nihče pa nam ne zagotavlja, da so tako izračunane poti res najkrajše poti do mejnih točk. Najkrajša pot do F ne vodi nujno po neposredni povezavi iz točke G; morda se do F bolj splača iti iz druge mejne točke D, morda pa celo prek mejne točke D in še točke E, o razdalji do katere še ne vemo prav ničesar.

Če razmislimo, pa bomo odkrili, da vsaj za eno mejno točko že poznamo tudi najkrajšo razdaljo do nje. Med mejnimi točkami C, D, F in ostalimi, ki na skici niso označene, poiščemo tisto, do katere vodi najkrajša pot iz ene od obiskanih točk. Recimo, da je to točka C: recimo, da je pot od A do C, ki vodi iz H, krajša od katerekoli druge poti do katerekoli druge mejne točke – krajša, torej, od poti prek H ali J do D, krajša od poti prek G do F, krajša od vseh drugih poti do mejnih točk.

Če je tako je pot do C, ki vodi iz H, tudi najkrajša možna pot do C. Res, ne more biti drugače. Če obstaja kakšna še krajša pot, bi morala voditi prek kake druge mejne točke, recimo D. To pa ne more biti, saj že je pot do D daljša kot pot do C-ja – rekli smo, da je C "najbližja" mejna točka. Do točke C je seveda mogoče priti tudi iz točk, ki niso mejne, a

takšne poti bi bile še daljše, saj moramo tudi do teh točk nekako priti, pridemo pa lahko le prek mejnih točk, ki pa so, spet, lahko le daljše.

Zdaj, ko poznamo najkrajšo pot do C, jo lahko *obiščemo*. Rezultat obiska kaže desni del slike. C smo dodali med obiskane točke in si zapomnili, iz katere obiskane točke smo prišli vanj. Vse točke, s katerimi je C povezana, so postale mejne: zanje zdaj poznamo najkrajšo pot do njih, ki vodi prek C (seveda pa to, kot zdaj vemo, ni nujno tudi najkrajša pot do njih). Za obstoječe mejne točke pa se je najkrajša znana razdalja do njih morda skrajšala: najkrajša znana pot iz obiskanih točk do D morda po novem ne vodi več prek H ali J, temveč prek C.

Korak ponavljamo: spet poiščemo najbližjo mejno točko in jo dodamo med obiskane. Postopek lahko končamo, ko obiščemo B, saj nam je s tem znana najkrajša razdalja do nje. Morebitne preostale mejne točke in točke onstran nas ne zanimajo več.

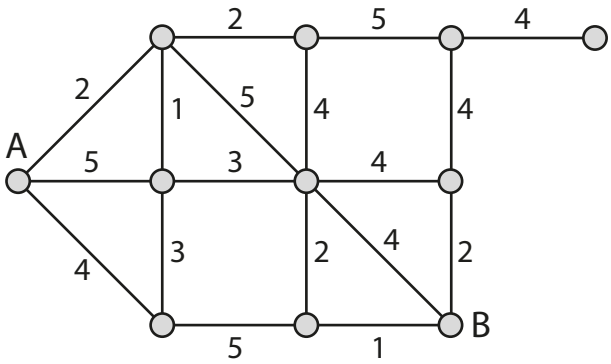
Še enkrat povejmo, ker je pomembno: postopek se ustavi, ko obiščemo ciljno točko in ne že takrat, ko postane mejna. Za mejne točke namreč še ne vemo, ali poznamo najkrajšo pot do njih ali ne. Za obiskane točke pa je najkrajša pot znana.

Če bi ravno hoteli, pa bi lahko postopek gnali še naprej, dokler ne obiščemo vseh točk v grafu. S tem bi dobili *drevo* najkrajših poti iz točke A do vseh drugih točk v grafu (zakaj drevo, bo jasno iz konkretnega primera spodaj). O drevesu smo maloprej povedali nekaj lepega: v drevesu je do vsake točke mogoče priti le na en način. Koren drevesa najkrajših poti je začetna točka, A, torej nam bo drevo povedalo, kako iz A najhitreje priti do vseh drugih točk.

Le še, kako se postopek začne, smo pozabili povedati. A četudi se ne bi spomnili, je menda očitno: začnemo tako, da imamo le eno obiskano točko, namreč začetno točko, A. Pot do nje je dolga 0.

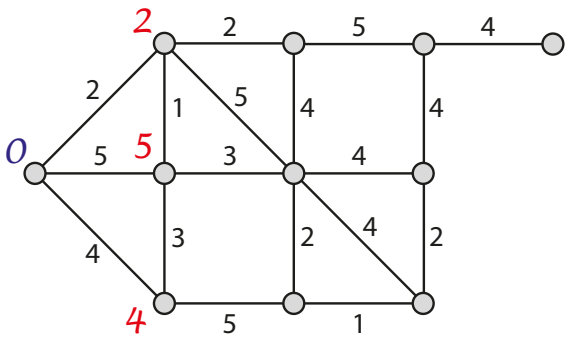
Kaj pa, če ciljno vozlišče iz začetnega sploh ni dosegljivo? Algoritem bo to pravzaprav opazil: zgodilo se mu bo, da bo mejnih točk zmanjkalo, ciljna pa še vedno ne bo obiskana. V tem primeru pač iščemo najkrajšo pot, ki je ni in neuspeh je neizbežen.

In zdaj konkretni primer: poiskati želimo najkrajšo pot od A do B na grafu, ki ga kaže slika.

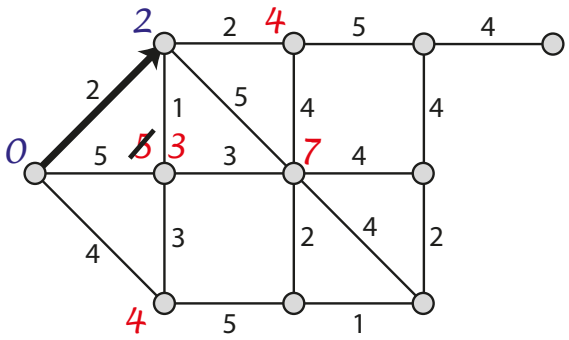


Potek algoritma je ilustriran spodaj.

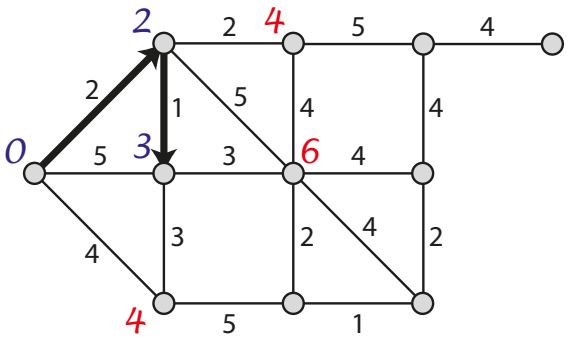
1.



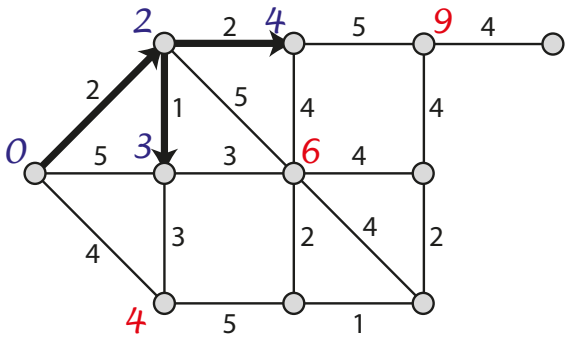
2.



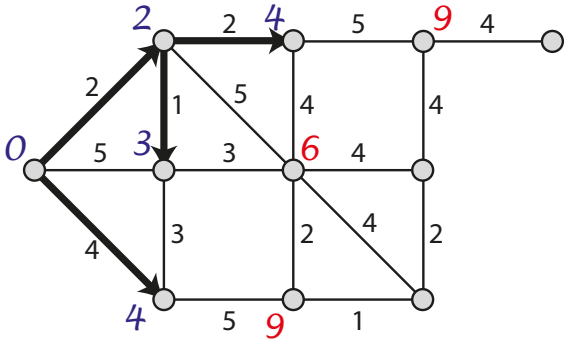
3.



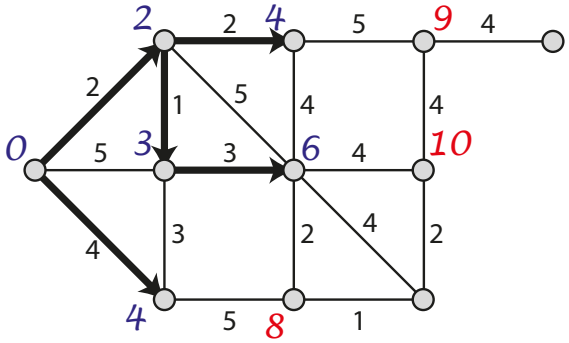
4.



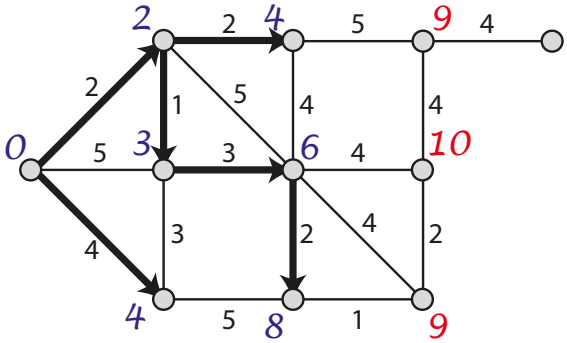
5.



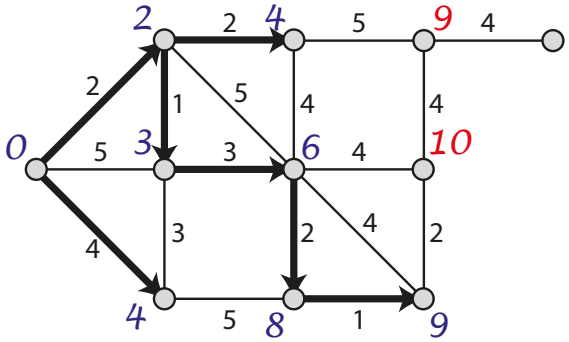
6.



7.



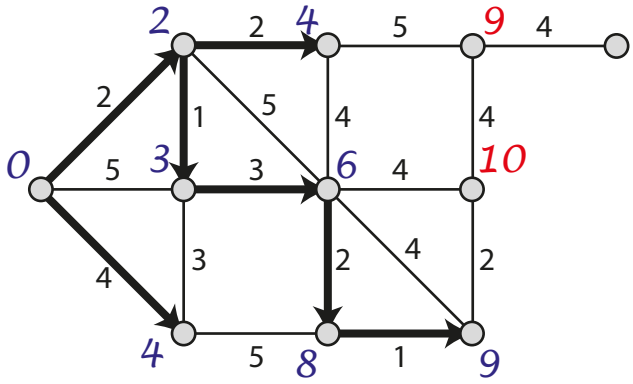
8.



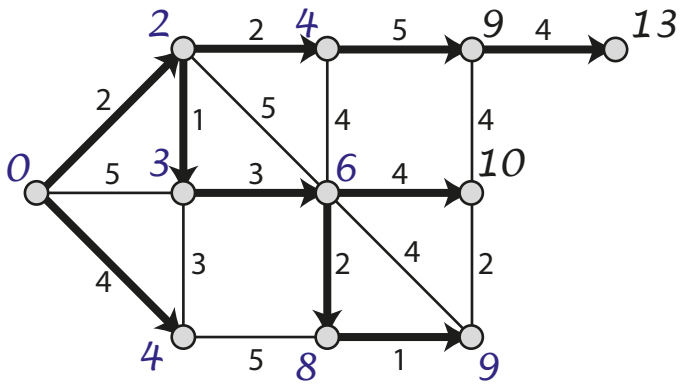
1. V začetku je obiskana točka A, razdalja do nje je 0. Razdalje do treh mejnih točk so 2, 5 in 4.
2. V naslednjem koraku razglasimo najbližjo mejno točko za obiskano. S tem dobimo dve novi mejni točki; razdalji do njiju sta 4 in 7. Do mejne točke, ki je bila prej oddaljena 5, zdaj poznamo krajšo pot dolžine 3.
3. Med obiskane prestavimo mejno točko, oddaljeno 3. To ne prinese novih mejnih točk, le pri eni od obstoječih popravimo razdaljo s 7 na 6.

4. Zdaj imamo dve mejni točki na razdalji 4. Odločimo se za katerokoli od njiju; izbira lahko vpliva na to, kakšna bo najkrajša pot, ki jo bomo našli, ne pa tudi na to, kako dolga bo. Če, recimo, izberemo zgornjo točko, to prinese novo mejno točko na razdalji 9.
5. Med tremi mejnimi točkamim (razdalje do njih so 4, 6 in 9) izberemo najbližjo in jo prestavimo med obiskane. Dobil smo novo mejno točko, razdalja do nje je 9.
6. Zdaj prestavimo med obiskane mejno točko, ki je na razdalji 6. To nam da eno novo mejno točko (na razdalji 10) in zmanjša razdaljo do ene od obstoječih mejnih točk z 9 na 8.
7. Zdaj obiščemo mejno točko, ki je na razdalji 8. To prestavi ciljno točko med mejne točke ... nismo pa še prepričani, da že poznamo najkrajšo razdaljo do nje.
8. Pravzaprav jo: mejne točke so oddaljene 9, 10 in 9. Ciljna točka je najbližja mejna točka (no, ena od njih), torej jo obiščemo. Delo je končano, najkrajša pot in njena dolžina sta znani.

Mimogrede smo izračunali še najkrajše poti do vseh drugih obiskanih vozlišč. Najkrajših poti do ostalih mejnih in do morebitnih vozlišč nismo izračunali in nas tudi ne zanimajo. Lahko pa bi nadaljevali, dokler ne obiščemo vseh vozlišč; tako bi izvedeli najkrajše poti od A do vseh drugih vozlišč.



Če izpustimo povezave, ki nas ne zanimajo, saj ne nastopajo v najkrajših poteh, dobimo drevo.



Da mora biti rezultat drevo, je očitno: povezave vodijo do vseh vozlišč (vsaj do vseh vozlišč v tistem delu grafa, ki je dosegljiv iz začetne točke) in do vsakega vozlišča vodi le ena povezava, samo smo vsako vozlišče obiskali (to je, premaknili iz množice mejnih v množico obiskanih točk) le enkrat.

Ne spreglejmo, da drevo govori le o najkrajših poteh iz A v B, ne pa tudi o najkrajših potem med drugimi pari vozlišč. No, med nekaterimi že: najkrajša pot iz vozlišča označenega z 2 do vozlišča z oznako 13 gre gotovo točno tam, kjer jo kaže drevo; če bi obstajala kaka krajša, bi jo uporabili tudi tu. Tudi pot od 2 do 10 vodi tako, kot kaže tole drevo. Pot od 8 do 13 pa vodi bogvekje. Če bi jo hoteli poiskati, bi morali pognati Dijkstrin algoritem iz točke 8.

Dijsktrin algoritem je hiter: čas izvajanja je sorazmeren $M \log N$, kjer je M število povezav in N število točk v grafu. Če predpostavimo, da imajo vse točke bolj ali manj podobno stopnjo, na primer k, bo čas izvajanja sorazmeren $k\,N \log N$: za dvakrat večji graf bo algoritem potreboval nekaj več kot dvakrat daljši čas. O tem se sicer ni težko prepričati, vendar zahteva, da vemo, recimo, kaj je kopica, tega pa ne vedo ne otroci ne mnogi izmed bralcev tega besedila (razen v agronomskem pomenu besede, seveda).

Mimogrede se spomnimo čebele in dinamičnega programiranja. Dijkstrin algoritem je lep primer algoritma, sestavljenega po načelu dinamičnega programiranja. Tako kot smo pri čebeli počasi širili fronto od začetnega cveta proti spodnji vrstici in za vsak cvet opazovali, odkod se nam najbolj splača priti vanj, tudi tu počasi širimo fronto od začetnega vozlišča. Razlika je pravzaprav le v tem, da pri čebeli iščemo maksimum, tu pa minimum, in da imamo pri čebeli dobičke na vozliščih grafa, tu pa ceno na njegovih povezavah. Sicer pa gre za eno in isto reč.

Tule je še lep primer naloge, v kateri je potrebno razmišljati malenkost drugače.