Opis in načrt dela

tim.mulej

April 2022

1 Uvod

Za projektno nalogo sem si izbral Wordle s slovenskimi besedami. Moj plan je, da najprej igro repruduciram, kot spletno aplikacijo v ASP.NET okolju, kjer bom za programski jezik uporabil C#. Besede bom shranil v bazi, do katere bom dostopal s programskim jezikom SQL. Nato bom za vse besede ocenil, koliko informacije ti dajo o iskani besedi, in potem izbral tisto, ki ti da največ. Oceno bom konstruiral kot pričakovano vrednost informacije, ki ti jo ta poda.

2 Opis dela

Cilj besedne igre Wordle je, da s čim manj poizkusi uganeš izbrano besedo s petimi črkami. Po vsakem poizkusu ti računalnik pove, katere črke si zadel, in katerih črk iskana beseda nima. Torej, če se ti katera od črk obarva zeleno, to pomeni, da je tudi v izbrani besedi ta črka na tem mestu. Če se ti črka obarva rumeno, pomeni da si črko zadel, vendar je njeno mesto napačno, če pa se črka obarva sivo, pa pomeni, da te črke ni v iskani besedi. Če se ti vseh pet črk obarva zeleno, si zmagal igro.

Ta algoritem za igranje Wordla temelji na tem, da poizkuša z vsakim poizkusom čim bolj zmanjšati število možnih pravilnih odgovorov. Število možnih odgovorov najbolj elegantno opišemo z informacijo, ki je definirana na način

$$I = \log_2 \left(\frac{n}{m} \right),$$

kjer je n število vseh možnih besed in m število še možnih pravilnih besed. Na tak način je zelo priročnen, ker če dobimo na primer 8 informacije iz ene besede, to pomeni da smo število možnih odgovorov zmanjšali za faktor 2^8. Poleg tega pa je informacija aditivna, torej če nam da prva beseda 6 informacije in druga 4, smo zmanjšali število možnih odgovorov za 2^{10}-krat.

Ta algoritem za vse besede izračuna pričakovano informacijo, na način

$$E(I) = \sum_{i=1}^{n} p_i \log_2 \left(\frac{1}{p_i} \right),$$

kjer je p_i verjetnost, da se nam bo pojavil i-ti vzorec, na primer (rumena, siva, siva, siva, zelena), potem pa izbere besedo z največjo pričakovano vrednostjo. To ponavlja dokler ne ugane prave besede.

3 Načrt dela

Najprej sem si moral naložiti programe, v katerih sem delal, kot so MS SQL strežnik in MS SQL Manegement studio za obdelavo podatkovnih baz, potem pa še Visual Studio z paketom ASPNET and web development.

Potem sem iz spletne strani http://bos.zrc-sazu.si/sbsj.html pobral vse besede z petimi črkami, ki sem jih shranil v bazo z imenom dolzine_pet tako,da je bila vsaka črka v svojem stolpcu. To mi je potem pomagalo pri iskanju besed. Čeprav sem imel v bazi samo besede s petimi črkami, sem moral nekatere izbrisati, ker so vsebovale številke, tuje črke, vezaje in pike. Vseh besed, ki so ustrezale je bilo malo več kot 13000, kar je $13,65$ informacije. Potem sem moral narediti še dve kopije te baze besed. Prvo kopijo sem poimenoval dolzine_pet_skrajsan in jo uporabljam za hranjenje vseh še možnih besed na primer, če z ugibanjem ugotovim, da je prva črka iskane besede k bodo v bazi dolzine_pet_skrajsan vstale samo še besede, ki se začnejo na k. Tretjo bazo pa sem poimenoval newDolzine_pet_skrajsan, ki mi pomaga pri resetiranju indeksov v bazi dolzine_pet_skrajsan.

Moja aplikacija je sestavljena iz štirih glavnih map. Prva mapa se imenuje wwwroot in v njej so shranjene vse datoteke, ki vsebujejo grafične elemente, na primer škatle okoli črk. Druga mapa se imenuje Models, notri imam shranjen razred Beseda, katerega uporabljam kot osnovno podatkovno strukturo. Ta ima 12 atributov. V prvih 10 imam shranjene črke in njihove stopnje, zadnje dva atributa pa vsebujeta povprečno informacijo, ki naj bi jo ta beseda prinesla v danih okoliščinah in dejansko informacijo, ki smo jo dobili iz te besede. Tretja mapa se imenuje Pages, v njej so shranjeni cshtml teksti strani in njihovi controllerji. Četrta mapa se imenuje services v njej imam shranjene vse funkcije, ki jih moja aplikacija počne. Poleg teh glavnih map pa aplikacija vsebuje še Connected Services, Dependencies, Properties, appsettings.json in Program.cs, ki skrbijo, da se aplikacija pravilno zažene.

Največ dela sem vložil v mapo services, v kateri sem ustvaril tri datoteke. V datoteki GetServices sem napisal funkcije, s katerimi sem pobral, prefiltriral in prenesel podatke. Naslednja datoteka GameServices vsebuje vsa pravila igre Wordle. Tukaj sem imel največ problemov s tem, kako pobarvati črke. Paziti sem moral na vrstni red in število ponavljajočih se črk. Saj če uporabnik poizkusi besedo, ki ima dve isti črki in je druga pravilna, se mora prva pobarvati sivo in ne rumeno. Tretjo datoteko pa sem poimenoval BotServices. V njej so funkcije, ki mi pomagajo izračunati optimalno besedo. Tu mi je največ problemov povzročalo sestavljanje wherestringa, s katerim sem hotel zajeti samo besede, ki zadoščajo določenemu vzorcu. Drugi način, ki ne bi zahteval wherestringa, je, da bi za vsako besedo poizkusil, kako se obarvajo vse ostale besede, in potem bi preštel, kolikokrat se vsak vzorec obarvanja pojavi, vendar

bi za ta postopek moral za vsako besedo poslati toliko poizvedb v bazo, kolikor je preostalih besed, kar pa bi porabilo preveč časa, ta moja metoda pa pošlje samo $3^5 = 243$ poizvedb. Pri sestavljanju wherestringa sem moral paziti pri besedah, ki vsebujejo več enakih črk. Na primer, če beseda vsebuje dva a-ja, jaz pa bi hotel najti vse besede, ki ustrezajo vzorcu, da je prvi a rumen in drugi siv, bi moral poiskati samo besede z enim a-jem. Če bi bile barve a-jev zamenjani, pa temu vzorcu ne bi ustrezala prav nobena beseda, ker wordle vedno pobarva samo prvo možno črko. Čeprav sem naredil izboljšavo z wherestringom, moja aplikacija še vedno potrebuje veliko časa, da izračuna optimalno besedo, ker je vseh vzorcev obarvanja besede $3^5 = 243$, moram tolikokrat iti skozi seznam besed, za vsako besedo. Moj računalnik zmore predelati eno besedo v eni do dveh sekundah, torej bi mu za vse besede vzelo med 3 in 6 ur. Na srečo bo za prvi poizkus vedno ista beseda, ki je karen z $5,9$ pričakovane informacije, zato lahko za prvo iskanje spustim celoten proces, naslednja iskanja optimalne besede pa ne vzamejo toliko časa, ker moj program išče samo po tistih besedah, ki so še mogoča. V povprečju mi vstane okoli $(13000/2^{5,9}) = 217$ besed, kar pa traja okoli 4 minute.

Poleg tega, da ti program predlaga optimalno besedo, sem še naredil, da program oceni vsak poizkus uporabnika, s tem da mu pove, koliko je bila pričakovana informacija njegove besede in nato mu še pove, koliko dejanske informacije je pridobil s tem poizkusom.

Med igranjem wordla sem opazil, da si je lažje spomniti besedo, če vidiš katere črke si že uporabil in katere nisi, zato sem dodal še eno tabelo, v kateri kaže katere črke si že uporabil in kako so se ti obarvale.

4 WhereString

Ker je kreiranje WhereStringa najbolj zahteven del te naloge, bi še kaj napisal o tem.

WhereSringe sem potreboval, ko sem za vsako besedo ocenil, koliko besed opisujejo določeni vzorci pobarvanja te besede. Glavna funkcija pri kreiranju WhereStringa je CommandString, ki sprejme urejen list barv črk in besedo. Najprej ta funkcija shrani pozicije črk in same črke, ki so zelene, rumene in sive v svoje liste, potem pa to naredi še za vse črke, ki niso zelene. Potem pa gre s for zanko skozi vse pozicije črk in če je črka na i-ti poziciji zelena, poišče še vse ostale iste črke v besedi, ki so zelene in doda WhereStringu ustrezno komando za toliko zelenih črk, kot jih je. Poleg tega pa si zapomni, da je to črko že porabila v zeleni barvi. Če pa je črka na i-ti poziciji rumena, pa preveri, če je še kakšna ista črka v besedi rumena ali siva; če najde še kakšno sivo uporabi funkcijo GrayandYellow, ki doda ustrezno komando k whereStringu, če pa najde samo rumene, pa uporabi funkcijo Yellow za dodajanje komande k whereStringu, potem pa si še zapomni to črko, da jo je že porabila pri sivih in rumenih. Če pa je črka siva, pa doda komando, da vsak stolpec, ki ni zelen ne more biti enak te črki in si to črko zapomne, da jo je že porabila kot sivo.

Funkcija GrayandYellow sprejme vse še možne pozicije, pozicije na katerih so

sive, pozicije na katerih so rumene in črko, vrne pa del whereStringa. Funkcija najprej preveri koliko je pozicij z rumeno barvo in če jih je več kot 2, že takoj vrne nemogoče. Potem preveri, če se da razvrstiti toliko črk, kolikor je pozicij z rumeno barvo med pozicije ki niso ne sive in ne rumene, in če se ne da, potem vrne nemogoče. Če pa se da, pa na podoben način kot logični veznik XOR napiše del whereStringa.

Funkcija Yellow pa sprejme vse še možne pozicije, pozicije na katerih so rumene in črko, vrne pa del whereStringa, ki opisuje kje še lahko stoji ta črka.Ta funkcija WhereString kreira na način, da najprej prešteje rumene pozicije in če je rumenih pozicij več ali enako 3 vrne, da je nemogoče, če jih je pa manj, pa preveri koliko je še prostih pozicij in če se da to črko razporediti še tja. Na primer če sta dve rumeni poziciji in štiri še prosta, bo funkcija yellow vrnila, da morata biti te črki na prostih mestih, ki nista rumeni.

5 Možne izboljšave

Ta program deluje tako, da ko dobi informacijo o iskani besedi, izloči vse besede, ki ne ustrezajo tej informaciji. To počne toliko časa, dokler mu ne ostane samo še ena beseda in nato izbere to besedo, ki mu ostane. Možno je, da obstaja beseda, ki ne ustreza vsem kriterijem in bi podala več informacije o iskani besedi, vendar jo moj program ne izbere. Lahko bi napisal, da bi program vsakič šel skozi vse besede in poiskal tisto, z največ informacije, vendar bi potem program delal prepočasi.

Druga izboljšava bi bila, da bi program naredil dvostopenjsko iskanje. To bi potekalo tako, da bi za vsako besedo izračunal, koliko je maksimalna pričakovana informacija vseh ostalih besed v naslednjem poizkušanju, če program poizkusi to besedo. Tak način iskanja optimalne besede bi porabilo veliko preveč časa.

6 Rezultati

Ta program v povprečju najde pravilno besedo v 4 poizkusih. Najboljša otvoritvena beseda je karen z 5.9 pričakovane informacije, kar pomeni da v povprečju zmanjša število možnih besed za $2^{5.9} \approx 60$-krat. Poleg karen sta dobri otvoritveni besedi tudi teran in tarok s 5.82 in 5.72 pričakovane informacije.