Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Primerjava iskanja z razpolavljanjem in interpolacijskega iskanja

Damjan Klemenčič

16. september 2014

Kazalo

1 Opis problema

Včasih želimo v urejeni tabeli z n elementi poiskati nek element. To lahko storimo tako, da se z zanko sprehodimo čez tabelo in poiščemo element. Vendar je to pri velikih tabelah zelo počasno (v času O(n)). Lahko pa uporabimo algoritem za iskanje z razpolavljanjem (Binary search), ki poišče željeni element veliko hitreje (v času O(log n)). Druga opcija je, da uporabimo algoritem za interpolacijsko iskanje (Interpolation search). V tej nalogi sem primerjal algoritem Binary search (funkcija `binarySearch`) in algoritem Interpolation search (funkcija `interpolationSearch`).

2 Rešitvi

2.1 binarySearch

Algoritem reši problem tako, da v vsaki iteraciji zanke `while` razpolovi interval iskanja in testira srednji element. Če je iskani element manjši od srednjega se premaknemo na levo polovico, če je večji se premaknemo na desno polovico in ponovimo postopek. V primeru, da je testirani element enak iskanemu vrnemo indeks elementa v tabeli. Ko se `while` zanka izteče vrnemo -1, ker iskanega elementa nismo našli.

2.2 interpolationSearch

Algoritem pa deluje na podoben način samo, da ne razpolavlja intervala iskanja ampak izračuna `mid` po formuli `low + ((myX - myArray[low]) / (myArray[high] - myArray[low]))` in na ta način ugotovi približno lokacijo iskanega elementa. Če je `mid` manjši od iskanega elementa prestavimo levo mejo `low` na pozicijo `mid + 1` se pravi desno od testiranega elementa, če je iskani element večji prestavimo desno mejo `high` na pozicijo `mid - 1` se pravi desno od testiranega elementa. Če noben od prejšnjih dveh pogojev ni izpolnjen pomeni da je iskani element enak testiranemu in vrnemo indeks elementa v tabeli. V primeru, da niso izpolnjeni pogoji zanke `while` pademo ven iz zanke in pogledamo če smo našli element, drugače vnemo -1.

3 Časovna zahtevnst algoritmov

3.1 binarySearch

Časonvan zahtevnost v najslabšem primeru je O(log n), v najboljšem pa celo O(1). Povprečna časovna zahtevnost je O(log n).

3.2 interpolationSearch

Časonvan zahtevnost v najslabšem primeru je O(n). Če so podatki enakomerno razporejeni lahko dosežemo časovno zahtevnost O(log log n).

4 Testni podatki

Testne podatke sem dobil od asistenta Andreja Bukoška in se mu za to tudi zahvaljujem. Poslal mi je štiri datoteke in sicer:

- data_R.txt - Datoteka vsebuje nabor podatkov, med katerimi iščemo posamezne ključe.
- data_Cp.txt - Datoteka vsebuje ključe katere iščemo med podatki iz zgornje datoteke.
- data_Re.txt - Datoteka vsebuje nabor podatkov med katerimi iščemo posamezne kjuče. Od datoteke data_R.txt se razlikuje v tem, da tu podatki naraščajo hitreje (eksponentno).
- data_Cep.txt - Datoteka vsebuje ključe katere iščemo med podatki iz zgornje datoteke.

5 Rezultati

Algoritma sem primerjal predvsem pri številu primerjav, ki jih opravi algoritem, da pride do željenega razultata. Rezultati testov so bili kar presenetljivi saj se pri podatkih iz datoteke `data_R.txt` bolje izkaše `interpolationSearch`, ki naredi 1884 primerjav medtem, ko `binarySearch` naredi 3891 primerjav. Do tega pride, ker `binarySearch` vedno zmanjpuje interval iskanja tako da razdeli tabelo približno na pol. Zato mora za elemente, ki se nahajajo na

začetku ali na koncu izbranega intervala narediti več primerjav. Algoritem `interpolationSearch` pa z izračunom `mid` elementa omeji interval tudi glede na iskano število in tako lahko hitreje poišče iskani element.
Pri podatkih iz datoteke `data_Re.txt` pa se veliko bolje izkaže `binarySearch`, ki naredi 3900 primerjav medtem, ko `interpolationSearch` naredi 49296 primerjav. V tej datoteki podatki naraščajo hitreje in zato algoritem `interpolationSearch` ni tako učinkovit. Rezultati se nahajajo v datoteki `result.txt`.