Županijsko natjecanje iz informatike

Srednja škola

Druga podskupina (3. i 4. razred)

15. veljače 2019.

OPISI ALGORITAMA

Autori zadataka: Domagoj Bradač (*Dionice*), Adrian Satja Kurdija (*Trokut, Kvadrat*)

Zadatke, rješenja i test podatke pripremili: Domagoj Bradač, Mislav Bradač i Adrian Satja Kurdija

Ministarstvo znanosti i obrazovanja

Agencija za odgoj i obrazovanje

HRVATSKI SAVEZ INFORMATIČARA

Ovaj zadatak najjednostavnije je riješiti grubom silom: za svaki broj C unutar trokuta provjeravamo je li ABC jednakostraničan trokut. Kako to provjeriti? Trokut je jednakostraničan ako su zadovoljena sljedeća tri uvjeta:

1. dva od vrhova A, B, C leže u istom redu,
2. dva od vrhova A, B, C leže na istoj dijagonali jednog tipa,
3. dva od vrhova A, B, C leže na istoj dijagonali drugog tipa.

Da bismo provjerili ove uvjete, dovoljno je unaprijed proći po trokutu i za svaki broj zapisati u kojem se redu i na kojim dijagonalama nalazi. To možemo implementirati tako da trokut spremimo kao matricu čiji redak po redak popunjavamo redovima trokuta:

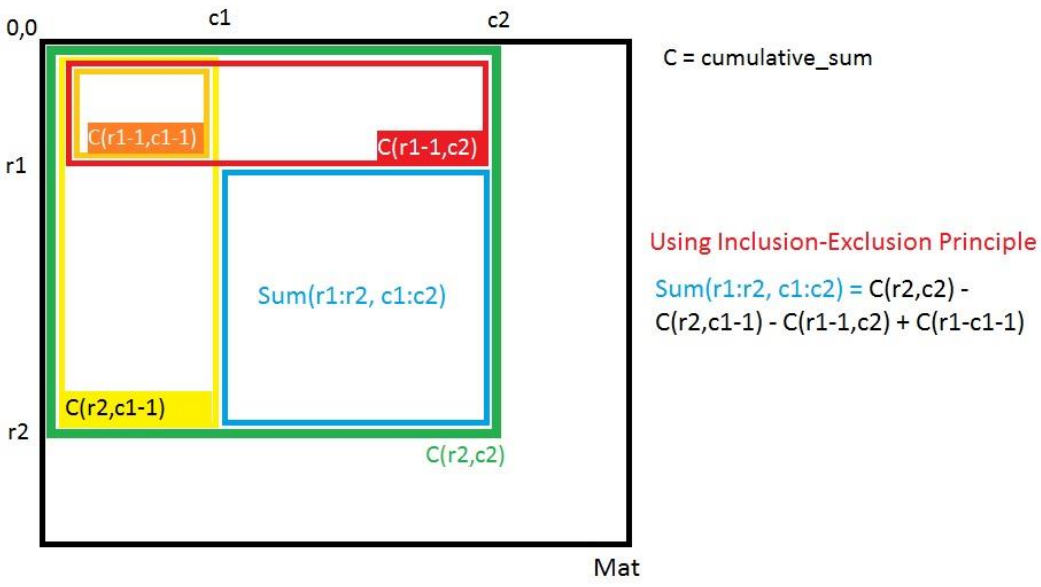
1
2 3
4 5 6
7 8 9 …

Tada dijagonala u jednom smjeru odgovara stupcu ove matrice, a dijagonala u drugom smjeru odgovara razlici retka i stupca.

Lako je provjeriti sve moguće kvadrate u matrici, ali teže je to učini dovoljno brzo - složenost naivnog pristupa bit će $O(N^5)$ jer biramo $O(N^3)$ kvadrata i za svaki od njih računamo plodnost u $O(N^2)$.

Traženo rješenje u složenosti $O(N^3)$ odabire sredinu mogućeg kvadrata i širi ga iz te sredine dok ne naiđe na rub. Svaki put kada povećamo promatrani kvadrat, možemo u konstantnom vremenu ažurirati njegovu plodnost. Kako? Promotrimo kvadrat K x K, te novi kvadrat koji dobivamo dodavanjem jednog "ruba" tj. "okvira" oko tog kvadrata. Da bismo dobili plodnost novog kvadrata, moramo dodati plodnosti novih rubnih polja, ali i povećati za jedan faktore koji množe plodnosti polja u unutarnjem kvadratu. To je kao da smo plodnosti tih unutarnjih polja još jednom pribrojili. Zaključujemo da je razlika plodnosti novog i starog kvadrata jednaka zbroju svih polja u novom kvadratu.

Kako brzo izračunati zbroj vrijednosti u nekom kvadratu unutar matrice? Riječ je o relativno poznatom problemu, za koji nije presudno radi li se o kvadratu ili pravokutniku. Taj zbroj (plavi kvadrat na donjoj slici) moguće je brzo izračunati koristeći formulu uključivanja-isključivanja kad bismo imali unaprijed izračunate zbrojeve svih pravokutnika koji počinju u gornjem lijevom kutu zemljišta (zeleni, žuti, crveni i narančasti pravokutnik):[1]



Algoritam, dakle, najprije u jednom prolazu po matrici računa zbrojeve plodnosti svih pravokutnika koji počinju u gornjem lijevom kutu zemljišta (s pomoću iste formule, računajući pojedinu vrijednost koristeći prethodno izračunate vrijednosti) i zapisuje ih u pomoćnu tablicu C. Potom vršimo gore opisanu provjeru svih kvadrata širenjem iz središta, koju je potrebno izvrtiti za dva slučaja: kad je sredina kvadrata neko polje matrice (tj. kad je stranica kvadrata neparne duljine) i kad se sredina kvadrata ne podudara ni s jednim poljem matrice (tj. kad je stranica kvadrata parne duljine).

[1] Slika preuzeta s *https://stackoverflow.com/questions/39937212/maximum-subarray-of-size-hxw-within-a-2d-matrix*

Za $N \leq 20$ bilo je dovoljno provjeriti sve moguće strategije kojih ima ukupno 2^N jer svaku ponudu možemo prihvatiti ili odbiti. Ukupna složenost ovog rješenja je $O(N \cdot 2^N)$ i nosilo je 20% bodova.

Za $N \leq 5000$, zadatak možemo riješiti dinamičkim programiranjem. Primijetimo da nam je u određenom trenutku bitno jedino koliko dionica trenutno posjedujemo. Neka *dp(i, k)* označava maksimalnu razliku ukupne prodaje i kupnje da bismo nakon prvih *i* zahtjeva posjedovali točno *k* dionica. Kako na svaki zahtjev možemo djelovati na dva načina, prijelaze odrađujemo u konstantnoj složenosti. Ukupna složenost ovog rješenja je $O(N^2)$ i nosilo je 40%bodova.

Slučaj kada su zahtjevi za kupnjom padajući nosio je 40% bodova i korak je prema punom rješenju. Možemo ga riješiti pohlepnim algoritmom. Zahtjeve obrađujemo s lijeva na desno. O zahtjevima za prodajom ne odlučujemo odmah. Kada naiđe zahtjev za kupnjom, tj. prilika da prodamo dionicu, odabrat ćemo najmanji neiskorišteni zahtjev za prodajom te ga "upariti" s ovim zahtjevom za kupnjom ako nam to donosi zaradu, tj. ako je cijena po kojoj trenutno možemo prodati viša od najniže cijene po kojoj možemo kupiti. Održavamo li zahtjeve za prodajom u prikladnoj strukturi (*set*, *priority queue* ili slično) ovaj algoritam možemo implementirati u složenosti O(N log N). Potičemo čitatelja da zastane i pokuša dokazati ispravnost ovog algoritma jer će to uvelike pomoći u razumijevanju punog rješenja.

Za početak, razmislimo zašto prethodni algoritam neće raditi kada zahtjevi za kupnjom nisu padajući. Odgovor je jednostavan: kasnije u nizu će nam se možda pružiti bolja cijena po kojoj možemo prodati dionicu od one koju smo odabrali. Prethodno opisani pohlepni algoritam modificirat ćemo na sljedeći način: kada obrađujemo zahtjev za kupnjom, odabrat ćemo najbolju od sljedeće tri opcije:

- Ne učinimo ništa.
- Uparimo ovaj zahtjev za kupnjom s najmanjim neiskorištenim zahtjevom za prodajom.
- Uklonimo najmanju prodaju koju smo dosad učinili i zamijenimo ju ovom.

Primijetite da se prijašnji algoritam sastojao od samo prve dvije od navedenih opcija. Koristeći još jednu strukturu istog tipa, i ovaj algoritam možemo implementirati u složenosti O(N log N). Ispravnost ovog algoritma može se dokazati induktivno. Pretpostavimo da je algoritam optimalan za neki niz zahtjeva duljine *k* te promatramo što se događa kada dodamo još jedan zahtjev na kraj niza. Glavna ideja dokaza je sljedeća: optimalno rješenje za *k + 1* zahtjev ne može imati manje prodanih dionica niti više od jedne prodane dionice više nego optimalno rješenje za prvih *k* zahtjeva. Potrebno je analizirati nekoliko slučajeva što ostavljamo čitatelju za vježbu.