PSA2: DN3

Js sm sam reku, da pokra ne,
ker poker je najslabš!

Turčin

Turčin se je končno vzel v roke in začel delati. Ker je dober poznavalec iger na srečo, verjame v zakon velikih števil, zato je kupil na tisoče preprostih igralnih avtomatov, kjer po vstavljenem žetonu zasveti bodisi zelena bodisi rdeča luč. Zelena luč igralcu prinese eno točko, rdeča pa nobene. Verjetnost zelene luči je odvisna od avtomata in je pri i-tem enaka p_i, $p_i \in (0, 1)$.

Tržno nišo je Turčin našel v maratonskih igrah na srečo, kjer mora igralec igrati na vsakem od avtomatov natanko enkrat, na koncu pa bo njegov dobitek odvisen od števila točk T, ki jih bo pri tem zbral. Turčin mora zato izračunati verjetnosti $P_k = P(T = k)$ za vse k med 0 in A, kjer je A število igralnih avtomatov. Le tako bo lahko ustrezno prilagodil dobitke.

Naloga A (10 točk)

Tu pridete na vrsto vi! Iz seznama verjetnosti p_i (dolžine A) izračunajte vseh $A+1$ verjetnosti P_k. Turčin je pragmatičen človek in bo zadovoljen že s približki za P_k, ki se od dejanskih vrednosti razlikujejo za največ $\varepsilon = 10^{-10}$.

Velja podobno kot pri prvih dveh domačih nalogah: pustimo na miru `main.cpp`, spremenimo pa vsebino datotek `resitev.h` oz. `resitev.cpp`, kjer vas že čaka metoda `izracunajVerjetnosti`, ki sprejme vektor dolžine A, katerega i-ta komponenta je enaka p_i. Metoda vrne vektor dolžine $A+1$, katerega k-ta komponenta je enaka P_k. Končno verzijo kode oddajte na učilnici (samo izvorno kodo), ker jo bomo izvajalci predmeta še enkrat preverili.

Vhodni podatki

Tokrat sta podnalogi le dve: testna A0 in dejanska A1. Vhodni datoteki imata končnico `.in`, izhodni pa `.out`. V vsaki vrstici vhodne datoteke je en testni primer (verjetnosti p_i), v pripadajoči vrstici v izhodni datoteki pa rešitev za ta primer (verjetnosti P_k). Branje je narejeno tako, da se vrstica že pretvori v `vector<double>`, zato se lahko ukvarjate zgolj z računanjem.

Testnih primerov v nalogi A1 je 20. Pripadajoča števila avtomatov (vrednosti A) so 2^0, 2^1, 2^2, …, 2^{17}, 2^{17}, 2^{17}, 2^{17}.

Željena časovna zahtevnost

Tokrat je dovoljeni čas odvisen od A in je enak 1 sekundo ($A \leq 2^{11}$), (približno) 20 sekund ($2^{12} \leq A \leq 2^{16}$) ali 48 sekund ($A = 2^{17}$) **na testni primer**[1]. Naj bo $[A] = \{1, 2, \ldots, A\}$. Naivni algoritem, ki izračuna verjetnosti P_k neposredno po formuli

$$P_k = \sum_{\substack{S \subseteq [A] \\ |S|=k}} \left(\prod_{i \in S} p_i \right) \left(\prod_{i \notin S} 1 - p_i \right),$$

ki obravnava vse načine, na katere lahko dobi igralec k točk, bo veliko prepočasen. Njegova časovna zahtevnost je $\mathcal{O}(A \cdot 2^A)$.

Z (naivno) metodo deli in vladaj lahko dosežemo časovno zahtevnost $\mathcal{O}(A^2)$. Ta bi morala biti dovolj hitra za večino manjših testnih primerov. Če jo še nekoliko dodelamo (**in se spomnimo snovi, ki smo jih jemali**), pa lahko pridemo do subkvadratične, ki bo dovolj hitra za vse primere.

Če se potrudite z implementacijo, boste lahko (za opazen faktor) hitrejši od uradne rešitve.

Primer

Vsebina `A0.in` in `A0.out` je prikazana spodaj:

```
A0.in:
0.5 0.5 0.5 0.5
0.5 0.33...3 0.66...6 0.75

A0.out:
0.0625 0.25 0.375 0.25 0.0625
0.0277...7 0.18055...5 0.388...8 0.31944...4 0.0833...3
```

V dejanskih datotekah seveda ni treh pik. V prvem primeru so vse štiri verjetnosti p_i enake 1/2, zato so verjetnosti P_k po vrsti 1/16, 4/16, 6/16, 4/16 in 1/16. V drugem primeru so verjetnosti p_i po vrsti 1/2, 1/3, 2/3 in 3/4, verjetnosti P_k pa so, kakršne so.

[1] Uradna rešitev potrebuje 0,2 oz. 10 oz. 22 sekund.