Triedenie

Ako správne používať tento študijný text: Milý čitateľ, chystáš sa prečítať si kuchárku o triedení. Počas čítania narazíš na niekoľko problémov, ktoré ti odporúčame vyriešiť. Má to dopomôcť k tomu, aby si lepšie pochopil, o čom sa rozprávame. Samozrejme, nikto ti nezakazuje čítať ďalej, vystavuješ sa tým však riziku, že si prezradíš riešenie.

Obsah:

Ako napovedá názov textu, budeme sa rozprávať o triedení. Schopnosť zoradiť veci do poradia je totiž veľmi dôležitá a využívaná. Však stačí, že chcete zoradiť futbalové družstvá podľa počtu bodov, ktoré získali. Alebo hráte karty a chcete ich mať na ruke pekne po poradí od najmenšej po najväčšiu. Pri tomto všetkom potrebujeme usporiadať veci podľa určitej vlastnosti a od nás informatikov sa čaká, že to budeme vedieť robiť rýchlo a efektívne a to aj pre viac ako 14 kariet na ruke alebo 30 tímov vo futbalovom rebríčku.

Na oboznámenie sa s utriedenými postupnosťami skúste vyriešiť úlohu **poriadok**.

Pomalšie triedenia

Začnime s ľahšími (na nakódenie aj pochopenie), ale pomalšími algoritmami riešiacimi problém triedenia. Ten môžeme chápať nasledovne: V poli A máme n prvkov. Chceme preusporiadať pole A tak, aby v ňom prvky boli naukladané od najmenšieho po najväčší.

Dôležité je určiť si, čo znamená najmenší prvok a vôbec operátor `menší`. Keď porovnávame čísla, je to jasné. Používame normálne $<$, teda $2 < 8$, $47 < 212$ ... Nie vždy však pracujeme len s číslami. Môžeme dostať napríklad písmená. Tie môžeme porovnávať podľa poradia v abecede. Samozrejme, to nie je všetko, čo môžeme robiť. Našu operáciu `menší` si môžeme zvoliť podľa ľubovôle, stačí, aby nám dávala zmysel a po celý čas sme sa jej držali. V ďalšom texte však budeme predpokladať, že pracujeme s celými číslami a používame normálne $<$.

Aby som vás týmto úplne neunudil, vyriešte úlohu **vystava**.

Prečo práve takúto úlohu? Dostať najmenší prvok na začiatok poľa a nič iné nemeniť? Vedie nás to totiž k nášmu prvému triediacemu algoritmu – `Minsort`. Jeho myšlienka je veľmi jednoduchá. Zoberme si pole A celých čísel a chceme ho usporiadať od najmenšieho po najväčší prvok. Ktoré číslo sa bude nachádzať na začiatku takto utriedenej postupnosti? Samozrejme, že to najmenšie. Ak by to tak nebolo, potom by niektorá časť poľa predsa nebola utriedená. Vieme teda, čo je na prvej pozícii výsledného poľa. Zvyšných $n-1$ čísel, ktoré sú teraz na pozíciách 2 až n je však stále neusporiadaných.

Tu sa však dá použiť znova tá istá myšlienka. Najmenšie číslo, ktoré je teraz na pozícii 1 necháme tak, to už je umiestnené dobre. Skúsime nájsť číslo na pozíciu 2. A to bude samozrejme najmenšie číslo z čísel zvyšných, teda tých, čo sú na pozíciach 2 až n. A takto postupujeme ďalej a ďalej. Postupne priraďujeme čísla na prvú, druhú, tretiu až n-tú pozíciu a vždy ich nájdeme ako minimum z čísel zvyšných. Znamená to, že n krát urobíte to, čo ste robili v úlohe **vystava**, akurát na stále menšom poli. A to už predsa nie je problém. Výsledná časová zložitosť však bude $O(n^2)$. Nie zlé, ale stále nie najlepšie.

Predpokladám, že teraz už bez problémov viete naprogramovať vlastný `Minsort`, môžete si to teda hneď vyskúšať na úlohe **utried1**.

Asi vás veľmi neprekvapí, že na úlohu sa dá pozrieť aj opačne. Aké číslo je na poslednom mieste zoradeného poľa. Samozrejme, že je to maximum, čo dá vznik algoritmu `Maxsort`.

Zostaňme ešte chvíľu pri triedeniach so zložitosťou $O(n^2)$. Na začiatku sme spomínali ukladanie kariet. Tento proces je všetkým známy. Postupne, jednu po jednej, beriete karty z vašej kôpky a nachádzate im príslušné miesto na ruke. Ako však nájdete miesto, kam daná karta patrí? Začnete na konci. Porovnáte novú kartu s poslednou kartou v ruke. Ak je nová karta väčšia, rovno ju dáte na koniec, lebo viete, že všetky ostatné karty na ruke

sú od nej tiež menšie. Ak je však menšia ako posledná karta, znamená to, že ju treba zaradiť niekam ďalej. Pozriete sa teda na predposlednú kartu. Takto postupujete, až kým prvýkrát nie je nová karta väčšia ako karta, na ktorú sa pozeráte, alebo ste dosiahli úplný začiatok. Na toto miesto kartu vsuniete.

Tento istý prístup sa dá simulovať aj programom. Postupne beriete čísla a máte pole, kde si ukladáte výsledok. Postupne idete od konca poľa a sledujete, či vaše nové číslo nie je väčšie. Takto mu nájdete správne miesto. Problém však je, že neviete toto číslo strčiť do stredu poľa tak ako kartu. Toto môžete vyriešiť tak, že vždy, keď zistíte, že nové číslo je menšie ako číslo skúmané, posuniete toto číslo o jedna dozadu v poli. Takto si spravíte priestor pre nové číslo, ktoré tam potom viete pohodlne vložiť. Tento algoritmus má tiež zložitosť $O(n^2)$ a volá sa `Insertsort`. Kvôli problematickému vsúvaniu čísiel sa nedá viac vylepšiť. Jeho naprogramovanie si tiež môžete vyskúšať na úlohe **utried1**, dajte si však pozor, aby ste skontrolovali, či nie ste už na začiatku poľa.

Toto samozrejme nie je posledný zo série triediacich algoritmov so zložitosťou $O(n^2)$. Existuje ich ešte o dosť viac, od šialených až po tie rozumnejšie. Nie sú však až tak dôležité, vrhnime sa preto na algoritmy rýchlejšie.

Rýchle triedenia

V tejto časti sa budeme snažiť čas výpočtu znížiť na $O(n \log n)$. K tomu budeme používať rekurziu, ktorá bude rozdeľovať naše pole na menšie a menšie kusy, tie samostatne utriedi a nakoniec z nich poskladá riešenie.

Prvým z algoritmov bude `Mergesort`, teda triedenie spájaním. Tento algoritmus sa pýta, či ak rozdelím pole na dve polovice, každú z nich samostatne utriedim, či ich viem potom spojiť do jedného výsledného utriedeného poľa.

Pred pokračovaním čítania, kde bude postup riešenia vysvetlený, môžete sa nad tým zamyslieť sami pri úlohe **telesna**.

Predpokladajme, že máme dve utriedené polia čísel A a B a chceme ich teraz spojiť do jedného usporiadaného poľa. Ktorý prvok môže byť na začiatku výsledného poľa? Chceme tam mať najmenší prvok. A vieme, že na začiatku A je najmenší prvok aj na začiatku B je najmenší prvok, lebo sú utriedené. To ale znamená, že menšie z týchto dvoch čísel je najmenšie aj celkovo. Vieme teda prvý prvok výsledku. Nech je to prvé číslo z A. Vyhodíme ho teda zo začiatku A a dostávam tú istú situáciu. Znova, ktorýkoľvek z prvkov na začiatku polí A a B môže byť aktuálne najmenší a teda ísť na druhú pozíciu. Opakovaním tohoto postupu dostanem výsledné pole.

Čo sa týka konkrétnej implementácie, samozrejme nechceme prvky z poľa vyhadzovať, lebo to zaberá príliš veľa času. Radšej si zoberieme dve premenné a a b, ktoré budú ukazovať na pomyselný začiatok oboch polí. Ak teda „vyhodíme“ prvok zo začiatku A, znamená to, že len o 1 zväčšíme premennú a. Tiež si dajte pozor, aby ste kontrolovali, či ste niektoré pole nevyčerpali celé, aby ste nevypadli mimo pamäť.

S touto jednoduchou úvahou je už výsledný algoritmus jednoduchý. Ak mám pole veľkosti 1 alebo 2, toto pole viem jednoducho utriediť najviac jednou výmenou. Ak je naše pole väčšie, rozdelím ho na dve polovice a na tie rekurzívne zavolám ten istý algoritmus. To znamená, že ich dostanem obe utriedené a potrebujem ich spojiť. A na to použijem hore vysvetlené spájanie (merge). Na konci dostanem utriedené celé pole.

Treba si všimnúť, že algoritmus potrebuje naviac jedno pomocné pole, kam bude mergovať výsledok. Dá sa to síce aj bez neho, ale nie je to príjemné a pole naviac nám nijak neuškodí.

Otázkou zostáva, či je to naozaj také rýchle, ako som tvrdil. Aké rýchle je naše spájanie? No jednoducho lineárne od počtu prvkov, ktoré spájam. Predstavme si teraz, čo robí náš algoritmus. Najskôr rozdelí pole na dve polovice (to je prvá úroveň). Tie rozdelí na dokopy štyri štvrtiny (druhá úroveň) atď. až kým nedosiahne polia veľkosti 1. Keďže veľkosti častí na úrovniach sa stále delia na polovicu, znamená to, že mám približne $\log n$ úrovní. Otázkou zostáva, koľko práce vykonám na každej úrovni. Na to si stačí všimnúť, že každý prvok pôvodného poľa je na každej úrovni zastúpený práve raz, teda na každej úrovni je n prvkov. No a na každej úrovni použijem lineárny merge. To znamená, že spravím $O(n)$ operácií. Na $\log n$ úrovniach spravím $O(n)$ operácii, teda výsledný čas je $O(n \log n)$.

Svoju novonadobudnutú znalosť si hneď môžete vyskúšať na úlohe **utried2**.

To však nie je všetko. `Mergesort` totiž nie je jediný a dokonca ani najpoužívanejší triediaci algoritmus v čase $O(n \log n)$. To, čo sa používa oveľa viac, je takzvaný `Quicksort`. Tento algoritmus sa dá totiž oveľa ľahšie naprogramovať bez pamäte naviac a taktiež má veľmi malý počet cache-missov. Poďme sa teda pozrieť na to, čo robí. Predtým však môžete skúsiť vyriešiť úlohu **jablcka**.

Ako správne predpokládate, ani táto úloha sem nie je zaradená úplne náhodne a má s `Quicksortom` priamy súvis. Úloha znie tak, že máme pole celých čísel A a číslo p. Toto číslo nazývame pivot. Našou úlohou je rozdeliť pole na tri menšie polia, čísla menšie ako p, čísla rovnaké ako p a čísla väčšie ako p. Naviac nás vôbec nezaujíma vzájomné poradie prvkov v týchto poliach.

Riešenie je samozrejme veľmi jednoduché. Stačí postupne prechádzať naším poľom A a rozdeľovať prvky po poradí do príslušného výsledného poľa. Ako nám však toto pomôže pri triedení celého poľa A. No vieme, že výsledok bude vyzerať nasledovne: najskôr pôjdu zoradené prvky menšie ako p, potom prvky rovné p a na záver zoradené prvky väčšie ako p. To znamená, že sme si našu úlohu rozdelili opäť na dve menšie časti, na ktoré sa potrebujeme rekurzívne zavolať. Vidíme, že rozdeliť pole podľa pivota p – zpivotizovať ho nám trvá čas $O(n)$. A ak by sme si vybrali prvok p vždy tak, aby menšie prvky tvorili polovicu všetkých prvkov, mali by sme $\log n$ úrovní, na každej s prácou $O(n)$, takže výsledný čas by bol $O(n \log n)$.

To je síce pekné, ale príliš optimistické. Poďme sa pozrieť aj na to, ako najdlhšie môže bežať náš algoritmus. Môžeme mať smolu a prvok p si vždy vybrať ako najväčší prvok nášho neutriedeného poľa. To znamená, že namiesto $\log n$ úrovní, budeme mať úrovní n. A na každej bude stále o jedna menej prvkov. To znamená, že zložitosť bude $O(n^2)$. To ako vieme, je pre veľké polia nepoužiteľné. Čo s tým?

Ak by sme chceli, aby náš program rozdelil pole vždy na polovice, ako prvok p by sme si museli zvoliť stredný prvok postupnosti – medián. Pozitívne je, že existuje algoritmus, ktorý nájde medián v postupnosti v čase $O(n)$. Bohužiaľ, tento algoritmus je dosť komplikovaný (môžete si o ňom prečítať v samostatnej sekcii) a má privysokú konštantu, aby sa dal používať prakticky. To, čo teda spravíme je, že to necháme na náhodu.

Pozrime sa na to tak intuitívne. Ak vyberiem daný prvok z ešte neutriedených prvkov, náhodne, je predsa oveľa pravdepodobnejšie, že to bude prvok blízko stredu ako to, že to bude prvok maximálny. Existuje dôkaz (bohužiaľ trochu komplikovanejší), ktorý ukáže, že v priemernom prípade je časová zložitosť `Quicksortu` naozaj $O(n \log n)$ a je veľmi nepravdepodobné, že to bude viac. A to naozaj úplne stačí. Dostaneme jednoduchý a rýchly algoritmus.

Môžete si aj vyskúšať naprogramovať na úlohe **utried2**. Dávam vám však do pozornosti aj náš vzorový program. Ukazuje, ako sa dá `Quicksort` implementovať bez pomocného poľa, aj keď je to trochu trikové, lebo sa dá zapliesť v ± 1. Určite si ho teda naštudujte.

Triky na rýchlejšie riešenia

Čo mali spoločné všetky predchádzajúce algoritmy? Rozhodovali sa na základe porovnávania prvkov medzi sebou. Patria teda medzi `triedenia porovnávaním`. Ako si neskôr ukážeme, časová zložitosť $O(n \log n)$, ktorú sme dosiahli pomocou triedenia `Mergesortom` alebo `Quicksortom` je najlepšie, čo vieme dosiahnuť. Napriek tomu v niektorých špecifických prípadoch vieme spraviť iný, rýchlejší algoritmus.

Výhodou triedení porovnávaním je to, že nezáleží na tom, aké veľké sú prvky, ktoré triedime. Či sú to čísla po 10^9 alebo 10^{18}. Nič sa nemení, algoritmus funguje tak isto. Poďme sa však pozrieť, čo by sme vedeli robiť, ak by sme si obmedzili čísla, ktoré triedime, na čísla, ktoré sú menšie ako 10^6 – milión. Jedna z hlavných vecí, ktoré sa zmenia oproti napríklad 10^9 je to, že si vieme v časovom limite vytvoriť pole veľkosti milión a pracovať s ním.

Prichádza teda jedna veľmi jednoduchá myšlienka. Ak si vieme vytvoriť pole veľkosti milión, tak pre každý prvok v našej postupnosti, ktorú chceme utriediť, si vieme porátať, koľkokrát sa v danej postupnosti nachádza. Na to nám stačí vytvoriť pole $P[1000000]$ na začiatku naplnené nulami. Následne prejdem našou postupnosťou a ak vidím prvok x, zväčším o jedna $P[x]$. Po jednom prejdení budem teda pre každé číslo vedieť, koľkokrát sa nachádza v našej postupnosti. Na koniec, už len prejdem poľom od 0 po 1000000 a každé číslo vypíšem $P[x]$ krát.

Aká je zložitosť nášho riešenia? Potrebujeme prejsť postupnosťou – n operácií a potrebujem prejsť celým poľom, takže 1000000 operácií. Samozrejme, o takom veľkom čísle si už nepovieme, že je to konštanta, označme si ho ako r. Presnejšie, nech r je horná hranica na čísla, ktoré sa môžu vyskytovať v našej postupnosti. Výsledná časová zložitosť je teda $O(n + r)$. Táto zložitosť je v podstate lineárna od n.

Vyššie spomenutý algoritmus má názov `Countsort`. Všimnite si, že naozaj nepatrí medzi triedenia porovnávaním, lebo nikdy medzi sebou neporovná dva prvky postupnosti. A tak ako predchádzajúce, môžete si ho vyskúšať na úlohe **utried3**.

Tieto algoritmy vyzerajú lákavo, keďže sú také rýchle, v praxi sa však veľmi nepoužívajú. Väčšinou sa s nimi môžete stretnúť iba v teoretických príkladoch, kde záleží na skutočnej časovej zložitosti. V praktických súťažiach väčšinou stačí použiť `Quicksort`, ktorý je naviac implementovaný v C++.

Dolná hranica triedení porovnávaním

Čo robia triedenia porovnávaním? Porovnávajú prvky medzi sebou a podľa toho ich vymieňajú. Môžete si všimnúť, že `Mergesort` aj `Quicksort` mali $O(n \log n)$ porovnaní. Otázkou však je, či je toto najlepšie možné riešenie, či sa to nedá zlepšiť. Hľadáme teda dolnú hranicu na počet porovnaní, ktoré potrebuje ľubovoľný algoritmus.

Predstavme si, že algoritmy trochu upravíme. Najskôr spravia všetky porovnania a až potom prvky povymieňajú. Môžete si rozmyslieť, že takto sa na nich nič nezmení. Taktiež množina výmen je jednoznačne určená porovnaniami, ktoré algoritmus urobí. Taktiež sa zamýšľajme len nad postupnosťami, kde prvky tvoria permutáciu čísel 1 až n. Uvedomte si, že pre algoritmus je to absolútne nerozlíšiteľné a nám to pomôže v argumentácii.

Počet výmen, ktoré algoritmus reálne potrebuje na to, aby pole utriedil, je relatívne malý, vždy mu stačí $O(n)$ výmen. Problém však je, že vtedy si už algoritmus musí byť istý, ktorú permutáciu má pred sebou. Nemôžu mu zostať dve, lebo by sa mohol pomýliť. Nakreslime si teda takzvaný rozhodovací strom. Vrchol tohto stromu bude tvoriť porovnanie, ktoré algoritmus urobí. Z každého vrchola pôjdu dve hrany, podľa odpovede, ktorú dostane a listy stromu budú možné permutácie, ktoré mohol dostať. Takýto strom má teda $n!$ listov.

Každému algoritmu zodpovedá jeden takýto rozhodovací strom. Našou úlohou bude dokázať, že vždy obsahuje cestu dĺžky aspoň $O(n \log n)$. Tým pádom existuje vstup, na ktorom bude musieť spraviť aspoň toľko porovnaní a dokážeme, že potrebujeme zložitosť $\Omega(n \log n)$.

Poďme sa pozrieť na to, koľko úrovní (akú hĺbku) musí mať náš rozhodovací strom. Na prvej úrovni je jeden vrchol. Na druhej úrovni sú najviac dva, na tretej najviac štyri atď. To znamená, že $n!$ vrcholov a teda aj listov sa môže prvýkrát objaviť v hĺbke $\log n!$. Tým pádom dolná hranica na počet porovnaní je $\log n!$. Poďme si toto číslo odhadnúť.

Zhora vieme spraviť jednoduchý odhad $n^n \geq n!$. Zdola môžeme spraviť nasledovný odhad. Prvých $n/2$ členov faktoriálu, je väčších ako $n/2$, teda $n! \geq (n/2)^{n/2}$. Dostaneme teda $n^n \geq n! \geq (n/2)^{n/2}$. Po zlogaritmovaní: $n \log n \geq \log n! \geq \frac{n}{2} \log \frac{n}{2}$. A v O notácii dostanem $O(n \log n) \geq O(\log n!) \geq O(n \log n)$, teda $O(\log n!) = O(n \log n)$.

Smutný a zároveň veselý výsledok. Smutný je, lebo je nám jasné, že nevieme triediť rýchlejšie, čo je však oveľa lepšie, máme optimálne riešenie tohoto problému. A to je obrovský úspech, na ktorý môžeme ako informatici byť právom hrdí. A teraz už aj vy viete všetko o triedení.

Teraz sa môžete vrátiť a doriešiť si príklady, ktoré ste počas čítania preskočili alebo môžete vyskúšať zriešiť niektorý z nasledovných príkladov: **zavazia**, **lupez**, **zurka program**, **darcek**, **stavebnica** a **nakup**. Ich spoločnou témou je síce triedenie, je však podané netradičnejšie a skryte.