k uspořádání n záznamů jako $T(n)$, platí $T(0) = T(1) = 0$, $T(n) \leqq cn + d + T(n_1) + T(n_2)$, kde n_1, n_2 jsou délky úseků, které vzniknou prvním dělením posloupnosti podle čísla k a které je třeba uspořádat.

Pokud by uvedené dvě části byly zhruba stejně veliké, takže by bylo možné předpokládat, že $n_1, n_2 \leqq n/2$, pak bychom dostali rekurentní vzorec $T(n) \leqq cn + d + 2T(n/2)$, odkud by plynulo $T(n) \leqq cn \log n + 2nd$. Jak ukážeme v článku 10.2, nastává takové dělení velmi často, takže průměrná doba výpočtu je $O(n \log n)$.

Pomineme-li možnost, že by k bylo voleno tak nevhodně, že by klíče všech záznamů zpracovávaného úseku byly větší nebo naopak menší než k, je druhým extrémem případ, kdy je $n_1 = n - 1$ nebo $n_2 = n - 1$. Pak je $T(n) \sim cn + d + T(n - 1)$, což dává $T(n) \sim cn(n + 1)/2 - c + d(n - 1) = \Omega(n^2)$.

Z úvahy je vidět, že rychlost algoritmů 5.2.1 a 5.2.2 silně závisí na způsobu volby čísla k v bodě 2. Obvykle se volí k jako hodnota klíče některého ze záznamů. Volba $k = K_L$ nebo $k = K_P$ není, jak uvidíme ve cvičeních, příliš vhodná, neboť výsledkem je dlouhá doba zpracování takřka uspořádaných posloupností, které se v praxi často vyskytují. Dobré výsledky je možné získat volbou klíče záznamu ležícího zhruba uprostřed děleného úseku, tedy $k = K_i$, kde $i \doteq (P + L)/2$. I když i v tomto případě je doba výpočtu v nejhorším případě úměrná n^2, průměrná doba výpočtu je jak z teoretického hlediska, tak na základě praktických zkušeností velmi krátká. Existují i složitější způsoby, např. volba k jako aritmetického průměru klíčů záznamů, které leží v první, druhé a třetí čtvrtině děleného úseku atd.

5.3. Dolní odhady

V tomto článku dokážeme, že pro jistou třídu algoritmů pro třídění, která zahrnuje všechny postupy, popsané v článcích 5.1, 5.2, nelze odhad $O(n \log n)$ pro čas potřebný k uspořádání nepříznivé kombinace n záznamů zlepšit. Tato třída zahrnuje, zhruba řečeno, algoritmy, které řídí výpočet na základě porovnávání klíčů záznamů, popř. klíče záznamu a daného čísla.

Abychom mohli podat přesnou definici této třídy, budeme předpokládat, že každý uvažovaný algoritmus je zapsán ve formě programu pro počítač s libovolným přístupem a jeho úkolem je uspořádání posloupnosti čísel zapsané na vstupní pásce.

Zavedeme si následující pomocnou definici:

Definice 5.3.1. *Popis výpočtu počítače s libovolným přístupem* je posloupnost dvojic čísel $(p_1, i_1), (p_2, i_2), \ldots$, kde p_j, resp. i_j je obsah programového, resp. indexového registru po provedení j-tého kroku uvažovaného výpočtu. ●

Popis výpočtu jednoznačně určuje, který příkaz se v každém z výpočetních kroků provede a se kterými paměťovými buňkami operuje.

Nechť nyní jsou $(p_1, i_1), \ldots$ a $(p'_1, i'_1), \ldots$ dva popisy výpočtu téhož algoritmu a nechť j je nejmenší číslo takové, že platí $(p_j, i_j) \neq (p'_j, i'_j)$. Pak nastává jedna z následujících dvou možností. Buď byl v j-tém kroce proveden podmíněný příkaz skoku, přičemž v jednom výpočtu ke skoku došlo a v druhém ne (v tomto případě je $i_j = i'_j$), nebo bylo příkazem *STORE* změněno číslo uložené v indexovém registru, a to v každém z výpočtů jinak (pak je jistě $p_j = p'_j$). Cílem druhé z těchto možností je obvykle dosáhnout toho, aby následující příkaz přesunu paměti nebo aritmetické operace využívající indexového adresování mohl v každém z výpočtů operovat s jinou paměťovou buňkou.

Řízení výpočtu podmíněnými skoky je typické pro algoritmy z článků 5.1, 5.2, zatímco algoritmy následujícího článku 5.4 budou řídit výpočet pomocí adres uložených v indexovém registru.

Definice 5.3.2. Algoritmus pro třídění (popsaný jako program pro počítač s libovolným přístupem) nazveme *porovnávacím*, jestliže pro libovolnou posloupnost čísel $a_1, \ldots, a_n$ a libovolné permutace $r_1, \ldots, r_n$ a $s_1, \ldots, s_n$ čísel $1, \ldots, n$ platí: je-li $(p_1, i_1), \ldots$, resp. $(p'_1, i'_1), \ldots$ popis výpočtu algoritmu při uspořádávání posloupnosti $a_{r_1}, \ldots, a_{r_n}$, resp. $a_{s_1}, \ldots, a_{s_n}$ a k nejmenší číslo takové, že

$$(p_k, i_k) \neq (p'_k, i'_k), \text{ pak } p_k \neq p'_k. \quad \bullet$$

Naším cílem je dokázat následující větu:

Věta 5.3.3. *Pro každý porovnávací algoritmus a každé přirozené n existuje permutace čísel $1, \ldots, n$, pro jejíž uspořádání potřebuje uvažovaný algoritmus čas alespoň* $\log n!$.

Důkaz: Nechť je každá permutace čísel $1, \ldots, n$ zpracována v čase nejvýše t. Každé permutaci Π přiřadíme posloupnost $P(\Pi)$ skládající se z t nul a jedniček, přičemž na jejím i-tém místě je jednička právě tehdy, jestliže v i-tém kroku zpracovávání permutace Π je prováděn podmíněný příkaz skoku, při kterém skutečně dojde ke skoku.

Popisy výpočtů, při kterých jsou zpracovávány dvě různé permutace Π_1 a Π_2, se musí lišit. Jelikož zkoumáme porovnávací algoritmus, musí se lišit i posloupnosti $P(\Pi_1)$ a $P(\Pi_2)$. Těchto posloupností je nejvýše 2^t, přitom jejich počet je roven $n!$. Proto musí platit $2^t \geqq n!$. ●

Pro vhodnou konstantu c platí $\log n! \geqq cn \log n$, takže pro porovnávací algoritmy je odhad $O(n \log n)$ pro dobu zpracovávání n záznamů nejlepší možný. Je možné dokázat, že algoritmus 5.2.1 je porovnávací; je až na multiplikativní faktor z hlediska nejhoršího možného chování optimální porovnávací algoritmus pro třídění.

5.4. Třídění rozdělováním

Algoritmy pro třídění, popsané v článcích 5.1, 5.2, jsou sice použitelné univerzálně, avšak pro jejich rychlost platí omezení $\Omega(n \log n)$ odvozené v článku 5.3. Nyní popíšeme třídu algoritmů, které jsou sice použitelné jen v některých případech, ale jsou-li aplikovatelné, uspořádají n záznamů v čase $O(n)$.

Základní varianta tohoto postupu je použitelná v případě, kdy klíče záznamů jsou celá čísla ležící mezi dvěma nepříliš vzdálenými čísly D a H. Obvykle se požaduje, aby rozdíl $H - D$ byl řádově nejvýše tisíce, a čím je menší, tím je použití třídění rozdělováním výhodnější.

Podle této metody postupujeme takto: vytvoříme „přihrádky", označené čísly $D, D+1, \ldots, H-1, H$, které jsou na začátku výpočtu prázdné, a potom postupně probíráme záznamy, které uspořádáváme, a záznam s klíčem K uložíme do K-té přihrádky. Nakonec seřadíme obsahy přihrádek za sebe.

Tento postup byl běžně používán děrnoštítkovými stroji, u nichž „přihrádky" byly skutečnými přihrádkami na štítky. Na elektronickém počítači je vhodné reprezentovat „přihrádky" seznamy. Při použití spojových seznamů použijeme pole $ZAHL(i)$, $i = D, \ldots, H$, obsahující záhlaví jednotlivých seznamů a záznam s klíčem K se uloží do seznamu se záhlavím $ZAHL(K)$. Použití kompaktních seznamů naráží na jednu obtíž: není předem jasné, kolik záznamů bude uloženo do jednotlivých přihrádek, a dimenzovat je na všech n záznamů by bylo neekonomické. Algoritmus 5.4.1 řeší tuto obtíž tím, že si nejprve spočítá pro každé $K = D, \ldots, H$, kolik záznamů má klíč K, a teprve potom provede rozdělení přiřazené oblasti paměti do přihrádek.

Algoritmus 5.4.1. Jednoprůchodové třídění rozdělováním

Vstupní data: Celá čísla D, H a záznamy $Z_1, \ldots, Z_n$ s klíči $K_1, \ldots, K_n$, představovanými celými čísly v rozmezí od D do H včetně.
Úkol: Uspořádat záznamy podle hodnot klíčů.
Pomocné proměnné: Výstupní posloupnost $W_1, \ldots, W_n$, do níž budou uloženy vstupní záznamy, celá čísla C_i, $i = D, \ldots, H$.

1. [*Inicializace.*] Pro $j := D, \ldots, H$ položme $C_j := 0$.
2. [*Určení velikosti přihrádek.*] Pro $i := 1, \ldots, n$ položme $K := K_i$; $C_K := C_K + 1$ (po ukončení je C_K rovno počtu záznamů s klíčem K).
3. [*Určení přihrádek.*] Pro $j := D+1, \ldots, H$ (v tomto pořadí) proveďme $C_j := C_{j-1} + C_j$. (Po ukončení je C_j počet záznamů s klíčem nejvýše j.)
4. [*Rozdělování.*] Pro $i := n, n-1, \ldots, 2, 1$ (v tomto pořadí) proveďme následující příkazy:
$K := K_i$; $C := C_K$; $W_C := Z_i$; $C_K := C_K - 1$. ●

V bodě 4 jsme záznamy probírali v opačném pořadí, aby měl algoritmus vlastnost, kterou nyní popíšeme:

Definice 5.4.2. Algoritmus pro třídění se nazývá *stabilní*, jestliže každé dva záznamy se stejnými klíči jsou ve výstupní posloupnosti v témž pořadí jako ve vstupní posloupnosti. ●

Mají-li například záznamy Z_1 až Z_6 po řadě klíče 1, 3, 2, 5, 3, 6, pak algoritmus, který je uspořádá na $Z_1, Z_3, Z_5, Z_2, Z_4, Z_6$ stabilní není, neboť jediné možné stabilní uspořádání je

$$Z_1, Z_3, Z_2, Z_5, Z_4, Z_6$$