1 Sortowanie

Sortowanie tablic danych (wygenerowanych w eksperymencie lub numerycznie) jest czesto przydatne. Poprzez sortowanie rozumiemy tu ustawienie elementow tablicy w kolejnosci rosnacej (rosnacych wartosci) dla jednej zmiennej. Na przyklad mozemy miec zbior danych eksperymentalnych dla dwoch zmiennych - czasu (t) i amplitudy (A(t)). Wowczas dwukolumnowa tablice punktow (t,A) warto ustawic w kolejnosci zgodnej z rosnacym czasem.

Najlepsze algorytmy sortujace wymagaja kilka razy $N \log_2 N$ operacji dla N elementowego zbioru. Tworcy algorytmow daza do tego by te 'kilka razy' bylo jak najmniejsze. Do najlepszych algorytmow naleza Quicksort autorstwa C.A.R.Hoare oraz Heapsort napisany przez J.W.J.Williams.

Zaczniemy jednak od najprostszego mozliwego algorytmu 'wstawianie wprost' ('straight insertion'). Metoda ta jest wystarczajaco szybka jedynie dla $N < 20$. Wymaga ona N^2 operacji i dlatego powinna byc uzywana bardzo ostroznie.

Dla $N < 50$ dobra metoda jest niewiele bardziej skomplikowana 'metoda Shell'a', wymagajaca w najgorszym razie $N^{3/2}$ operacji (na ogol jest nawet szybsza - zalezy to od poczatkowego ulozenia elemantow w tablicy).

1.1 Metoda 'wstawiania wprost'

Metoda przypomina te, ktora stosuje sie przy ukladaniu kart do gry. Wybiera sie druga karte i umieszcza w odpowiedniej kolejnosci w stosunku do pierwszej. Nastepnie trzecia karte uklada sie w kolejnosci wzgledem pierwszych dwoch, itd. az ulozy sie wszystkie karty. Algorytm wyglada nastepujaco:

$$
\begin{array}{l}
for \quad j=2 \quad to \quad n\\
begin\\
\quad a = tab[j]\\
\quad i = j-1\\
\quad while \quad (i>0 \quad and \quad tab[i]>a)\\
\quad begin\\
\quad\quad tab[i+1] = tab[i]\\
\quad\quad i = i-1\\
\quad end\\
\quad tab[i+1] = a\\
end
\end{array}
$$

1.2 Metoda Shell'a

Metoda ta jest bardzo wydajnym wariantem 'wstawiania wprost'. Pomysl pokazemy na przykladzie sortowania 16 elementow $n_1 \cdots n_{16}$. W pierwszym kroku sortujemy przy pomocy metody 'wprost' kazda z 8 grup po dwa elementy, $(n_1, n_9), (n_2, n_{10}), \cdots, (n_8, n_{16})$. W drugim kroku sorujemy kazda z 4 grup po 4 elementy $(n_1, n_5, n_9, n_{13}), \cdots, (n_4, n_8, n_{12}, n_{16})$. W czwartym kroku sortujemy ostatecznie cala tabele 16 elementowa.

Oczywiscie, ostatni krok jest wystarczajacy by uporzadkowac tablice. Jaki jest wiec cel wczesniejszych krokow? Odpowiedz jest tak, ze wczesniejsze sortowania pozwalaja elementom tablicy ustawic sie blisko ich wlasciwej pozycji. Dzieki temu koncowe sortowanie 'wprost' rzadko kiedy musi przejsc przez wiecej niz kilka pozycji zanim znajdzie wlasciwe miejsce dla danego elementu.

Odstepy miedzy elementami sortowanymi w kazdym kolejnym kroku (8,4,2,1 w powyzszym przykladzie) nazywa sie 'wzrostem' ('increment'). Przeprowadzono wiele badan nad tym jak najlepiej wybrac zestaw wzrostow, ale optymalny wybor jest nieznany. Zestaw $\cdots, 8, 4, 2, 1$ nie jest zreszta dobrym wyborem, zwlaszcza dla N bedacych potega 2. Duzo lepiej wziac kolejno zbior

$$(3^k - 1)/2, \cdots, 40, 13, 4, 1 \tag{1}$$

,ktory mozna stworzyc poprzez rekurencje

$$i_1 = 1, \quad i_{k+1} = 3i_k + 1, \quad k = 1, 2, \cdots \tag{2}$$

Mozna pokazac, ze dla takiego zbioru liczba potrzebnych operacji wynosi $N^{3/2}$ w przypadku najgorszego poczatkowego ustawienia elementow w tablicy. Dla 'losowo' ulozenia elemantow sortowanie wymaga $N^{1.25}$ operacji, przynajmniej dla $N < 6000$.

Algorytm dla tablicy $tab[n]$ wyglada nastepujaco

```
wzr = 1
do
begin
  wzr = 3 * wzr + 1
end
while   (wzr ≤ n)

do
begin
  wzr = wzr/3
  for   i = wzr + 1   to   n
  begin
    a = tab[j]
    j = i
    while(tab[j − wzr] > a)
    begin
      tab[j] = tab[j − wzr]
      j = wzr − 1
      if   (j ≤ wzr)   break
    end
    tab[j] = v
  end
end
while   (wzr > 1)
```

1.3 Metoda Heapsort

Metoda ta wymaga rzedu $N \log_2 N$ operacji. Niestety przedstawiamy tutaj jedynie jej podstawy bez szczegolow.

Mowimy, ze zestaw N elementow a_i, $i = 1, \cdots, N$ tworzy 'stos' ('heap') jezeli elemnty spelniaja nastepujace relacje

$$a_{j/2} \geq a_j \quad for \quad 1 \leq j/2 < j \leq N \tag{3}$$

Dzielenie $j/2$ oznacza tu 'dzielenie calkowite' czyli takie, ktorego wynikiem jest liczba calkowita lub wynik zaokraglany jest w dol do najblizszej liczby calkowitej. Powyzsza definicja staje sie bardziej zrozumiala jezeli pomysli sie, ze uporzadkowane elementy a_i tworza binarne drzewo, na ktorego wiercholku znajduje sie 'szef', wezel a_1, dwa wezly a_2 i a_3 to 'podwladni', dla ktorych z kolei podwladnymi sa kolejne cztery wezly a_4 do a_7, etc. W tej formie stos ma kazdego 'zwierzchnika' wiekszego lub rownego swoim dwom 'podwladnym', od gory do spodu hierarchi.

Jezeli juz powiedzie sie ustawienie elementow zbioru w postaci stosu, jego sortowanie jest bardzo proste. Zdejmujemy szczyt stosu, bedacy najwiekszym elementem jeszcze nie sortowanym. Nastepnie 'promujemy' na szczyt jego najwiekszego podwladnego. Pozniej promujemy jego najwiekszego podwladnego, itd. Proces przypomina to co sie dzieje (lub powinno dziac) w korporacji w chwili przejscia na emeryture naczelnego dyrektora. Proces zabiera $N \log_2 N$ operacji gdyz odejscie kazdego kolejnego dyrektora prowadzi do promocji $\log_2 N$ podwladnych.

Stworzenie stosu z danej tablicy ponownie przypomina proces selektywnej promocji w gore w korporacji. Wyobrazmy sobie, ze korporacja zaczyna z $N/2$ pracownikami na linii produkcyjnej, ale bez ich zwierchnikow. Nastepnie zatrudnia sie zwierzchnika majacego zajmowac sie dwojka pracownikow. Jezeli jest on mniej zdolny niz ktorys z jego podwladnych musi zamienic sie z nim miejscami w hierarchii. Po zatrudnieniu pierwszych zwierzchnikow zatrudnia sie kolejno ich zwierchnikow itd.

Kod dla tego modelu znalezc mozna na stronie 337-ej 'Numerical Recipies In C'.