RYSUNEK 7.13. Metoda tablic równoległych na przykładzie

Tablica określająca *kolejność danych*

3	5	4	2	1	-1	?	?

Tablica zawierająca *pola informacyjne*

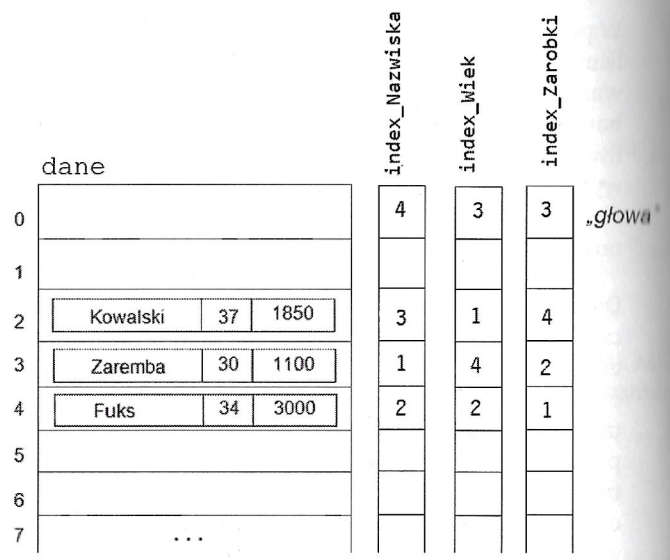
znak[*MaxTab*]		E	O	K	T	K	?	?

będzie zawierać indeks pierwszego rzeczywistego elementu listy. W naszym przykładzie jest to 3, zatem w dolnej tablicy pod pozycją 3 znajduje się pierwszy element listy — jest nim znak K. Aby dowiedzieć się, co następuje po K, musimy odczytać `nastepny [3]`. Jest to 2 i tam też jest umieszczona kolejna litera wyrazu „KOTEK". Koniec listy jest zaznaczany umownie poprzez wartość -1.

Rozwiązanie to można uznać za eleganckie i elastyczne. Dopisanie funkcji, które obsługują taką strukturę danych, nie jest trudne. Występuje tu pełna analogia do już wcześniej przedstawionych funkcji (np. obsługujących listy jednokierunkowe), dlatego też zadanie ewentualnego opracowania ich pozostawiam czytelnikowi.

Dlaczego jednak nie pójść o krok dalej i nie używać *kilku* tablic pozwalających na określenie kolejności sekwencji danych?! Zbliżylibyśmy się wówczas do wersji zaprezentowanej na rysunkach 7.9 i 7.10, otrzymując jednak o wiele prostsze w realizacji zadanie. Realizacja takiego pomysłu jest jak najbardziej możliwa, ilustrowano na rysunku 7.14, gdzie przedstawiona została minibaza danych (dane pierwotne) obok kilku tablic ukazujących ich możliwe sortowanie (indeksy).

RYSUNEK 7.14. Metoda tablic równoległych



Obok tablicy danych możemy zauważyć trzy osobne tablice, które umożliwiają dostęp do danych widzianych jako listy posortowane wedle przeróżnych kryteriów. Tablica `dane` zawiera rekordy z danymi (w naszym przykładzie mogą to być obiekty, tak jak poprzednio), przy czym efektywne informacje zaczynają się od komórki 2 w górę. Dlaczego tak dziwnie? Otóż zabieg ten zapewnia nam odpowiedniość 1 do 1 tablicy danych i tablic realizujących sortowanie (`index_Nazwiska`, `index_Wiek` i `index_Zarobki` są w rzeczywistości zwykłymi tablicami liczb całkowitych).

W tych tablicach bowiem komórki 0 i 1 są zarezerwowane, odpowiednio, na adres początku listy i znacznik jej końca. Należy to rozumieć w ten sposób, że `index_Nazwiska[0]` zawierający liczbę 4 informuje nas, że `dane[4]` są pierwszym rekordem na liście. A jaki jest następny rekord? Oczywiście `index_Nazwiska[4]=2`, co oznacza, że drugim rekordem na liście danych jest 2. Postępując tak dalej, odtwarzamy całą listę: 4, 2, 3 — łatwo zauważyć, że jest to lista posortowana alfabetycznie według nazwisk. Skąd jednak wiemy, że rekord 3 jest ostatnim rekordem na liście? Otóż `index_Nazwiska[3]` zawiera 1, co stanowi zgodnie z naszą umową znacznik końca listy.

Analogicznie postępując, możemy zbudować listy `index_Wiek` i `index_Zarobki` uporządkowane, odpowiednio, według wieku i zarobków.

Reprezentacja list, w której nastąpiło oddzielenie danych od informacji określających ich wzajemną kolejność, pozwala na zapamiętanie w tym samym obszarze pamięci kilku list jednocześnie — o ile oczywiście ich elementy składowe w jakiś sposób się pokrywają. W aplikacjach, w których występuje taka sytuacja, jest to cenna właściwość przyczyniająca się do zmniejszenia zużycia pamięci. Takie postępowanie będzie miało sens, gdy objętościowo listy indeksów będą znacznie mniejsze od list danych i programista będzie musiał sam ocenić, co oznacza słowo „znacznie".

Model klasowy realizacji pomysłu z rysunku 7.14 jest koncepcyjnie prosty, choć wcale nie tak łatwy w pełnej realizacji. Popatrz na szkic modelu danych:

```
class RekordDanych:
  def __init__(self, ...):
    self.wiek = ...
    self.zarobki = ...
    ...
class BazaTablicowa:          # Klasa obejmuje tablicę danych oraz dodatkowe indeksy i pozwala
                              # na ich sortowanie
  def __init__(self, MaxTab):
    self.Licznik = ...        # Bieżąca liczba rekordów w tablicy
    self.MaxElt = ...         # Maksymalna liczba elementów listy danych w tablicy
    self.Licznik = 0          # Liczba wpisów na początku to 0
    self.Rozmiar = MaxTab     # Zapamiętajmy maksymalny rozmiar
    # Tworzy statyczną tablicę o rozmiarze MaxTab:
    self.tab = [RekordDanych() for x in range(MaxTab)]
    # Indeksy – pierwsze komórki są zarezerwowane, dane zaczynają się od indeksu 2.!
    self.index_Nazwiska = [-1 for x in range(MaxTab+2)] # Lista indeksowa 1.
    self.index_Wiek     = [-1 for x in range(MaxTab+2)] # Lista indeksowa 2.
    self.index_Zarobki  = [-1 for x in range(MaxTab+2)] # Lista indeksowa 3.
```

Dodajmy na koniec, że opisane wyżej przykłady nie zapewniają oszczędnego gospodarowania pamięcią, ale to może być koszt do poniesienia za cenę wygody użytkowania!