INFORMATYKA – MÓJ SPOSÓB NA POZNANIE I OPISANIE ŚWIATA

PROGRAM NAUCZANIA INFORMATYKI Z ELEMENTAMI PRZEDMIOTÓW MATEMATYCZNO-PRZYRODNICZYCH

Informatyka – poziom rozszerzony

Algorytmy porządkowania danych

Iwona i Ireneusz Bujnowscy

Człowiek - najlepsza inwestycja

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

Tytuł: ***Algorytmy porządkowania danych***

Autor: ***Iwona i Ireneusz Bujnowscy***

Redaktor merytoryczny: ***prof. dr hab. Maciej M. Sysło***

Materiał dydaktyczny opracowany w ramach projektu edukacyjnego
Informatyka – mój sposób na poznanie i opisanie świata. Program nauczania informatyki z elementami przedmiotów matematyczno-przyrodniczych

www.info-plus.wwsi.edu.pl

infoplus@wwsi.edu.pl

Wydawca: Warszawska Wyższa Szkoła Informatyki
ul. Lewartowskiego 17, 00-169 Warszawa
www.wwsi.edu.pl
rektorat@wwsi.edu.pl

Projekt graficzny: *Marzena Kamasa*

Warszawa 2013

Człowiek - najlepsza inwestycja

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego

SCENARIUSZ TEMATYCZNY

ALGORYTMY PORZĄDKOWANIA DANYCH

INFORMATYKA – POZIOM ROZSZERZONY

OPRACOWANY W RAMACH PROJEKTU:

INFORMATYKA – MÓJ SPOSÓB NA POZNANIE I OPISANIE ŚWIATA.

PROGRAM NAUCZANIA INFORMATYKI

Z ELEMENTAMI PRZEDMIOTÓW MATEMATYCZNO-PRZYRODNICZYCH

Streszczenie

Nowa podstawa programowa kształcenia ogólnego zakłada, że uczeń po zakończeniu nauki będzie potrafił rozwiązywać problemy, podejmować decyzje z wykorzystaniem komputera i zastosowaniem podejścia algorytmicznego. W treściach nauczania wymieniono algorytmy, których znajomość i umiejętność analizy działania jest kluczowa w procesie edukacji: wskazano tam m.in. algorytmy dotyczące porządkowania danych – sortowanie bąbelkowe, przez wybór, przez wstawianie liniowe i binarne, przez scalanie, sortowanie szybkie i kubełkowe.

W przedstawionych scenariuszach skupimy się na najprostszych algorytmach sortowania danych, takich, od których rozpoczyna się naukę analizy algorytmów w zakresie złożoności: na sortowaniu bąbelkowym, sortowaniu przez wybieranie oraz sortowaniu przez zliczanie. W kolejnych scenariuszach, które będą tworzone w trakcie trwania projektu zostaną opracowane pozostałe algorytmy wymienione w podstawie programowej.

Czas realizacji

4 x 45 minut

Tematy lekcji

1. Algorytmy sortowania danych. Algorytm sortowania bąbelkowego (2 x 45 minut)
2. Sortowanie przez wybieranie – analiza działania i implementacja (1 x 45 minut)
3. Sortowanie przez zliczanie (1 x 45 minut)

Podstawa programowa

Etap edukacyjny IV, przedmiot: informatyka (poziom rozszerzony)

Cele kształcenia wymagania ogólne

III. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego.

Treści nauczania – wymagania szczegółowe

poziom rozszerzony

5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń:
   1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin;
   2) stosuje podejście algorytmiczne do rozwiązywania problemu;
   3) formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera;
   4) dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji;
   5) posługuje się podstawowymi technikami algorytmicznymi;
   6) ocenia własności rozwiązania algorytmicznego (komputerowego), np. zgodność ze specyfikacją, efektywność działania;
   7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania;
   11) opisuje podstawowe algorytmy i stosuje
      b) algorytmy wyszukiwania i porządkowania informacji – algorytmy sortowania ciągu liczb: bąbelkowy, przez wybór, przez wstawianie.

Cel

Zapoznanie uczniów z wybranymi algorytmami porządkowania danych.

Słowa kluczowe

sortowanie, sortowanie bąbelkowe, sortowanie przez zliczanie, sortowanie przez wybór

LEKCJA NR 1

TEMAT: Sortowanie bąbelkowe

Co przygotować

- Prezentacja 1 „Sortowanie bąbelkowe”

MATERIAŁ TEORETYCZNY

Algorytmy dotyczące porządkowania danych kojarzą się przede wszystkim z sortowaniem. Wiele problemów algorytmicznych da się rozwiązać porządkując elementy rosnąco lub malejąco. Tematyka związana z sortowaniem danych jest na tyle ciekawa, że doczekała się wielu opracowań. Wymyślono i opublikowano sporą liczbę różnych algorytmów różniących się od siebie ideą, złożonością czy skomplikowaniem kodu.

Sortowanie bąbelkowe

Nasze rozważania zacznijmy od opisu działania i kodu jednego z najbardziej łatwego do zrozumienia i zakodowania algorytmu sortowania bąbelkowego.

Algorytm sortowania bąbelkowego opiera się na porównywaniu kolejnych sąsiednich par elementów i sprawdzaniu, czy ich wzajemne położenie w tablicy jest odpowiednie z punktu widzenia wymaganego końcowego uporządkowania.

Poniższy fragment kodu ilustruje główną część tego algorytmu:

```
1. for (int i=0; i<n-1; i++)
2.   if (t[i] > t[i+1]) swap(t[i], t[i+1]);
```

Instrukcja pętli zawarta w wierszu 1 wymusza zmianę zmiennej *i* począwszy od wartości *1* do wartości *n-2*. Zawarta w linii 2 instrukcja warunkowa porównuje w każdej iteracji pętli element o indeksie *i* z elementem następnym, czyli takim, który ma indeks *i+1*. Jeśli wartość elementu następnego jest mniejsza od tego, który go poprzedza, to następuje ich zamiana.

Warto zauważyć, że po zrealizowaniu przedstawionej pętli po raz pierwszy największy element przesunie się z dowolnej zajmowanej dotychczas pozycji na miejsce ostatniego elementu w tablicy. Gdyby pętla wraz z instrukcją warunkową wykonała się powtórnie, to kolejny, co do wielkości element trafi na właściwą pozycję w tablicy. Po kolejnej realizacji tego kodu na właściwych pozycjach będą już więc co najmniej dwa elementy itd. Stąd już niedaleko do zapisu pełnego kodu sortowania bąbelkowego:

```
1. for (int p=1; p<n; p++)
2.   for (int i=0; i<n-1; i++)
3.     if (t[i] > t[i+1]) swap(t[i], t[i+1]);
```

Pętla ze zmienną sterującą *p* odlicza kolejną realizację omówionych wcześniej instrukcji, a jednocześnie zmienna *p* otrzymuje wartość wskazującą, co najmniej ile elementów będzie na właściwych pozycjach w tablicy po wykonaniu instrukcji z wierszy 2 i 3. Gdy *p* ma wartość *1*, to po wykonaniu instrukcji z wierszy 2 i 3. jeden z elementów z całą pewnością zajmie miejsce właściwe. Gdy *p* będzie miało wartość *2*, to po wykonaniu instrukcji z wierszy 2 i 3 już co najmniej dwa elementy będą na właściwych pozycjach itd. Oczywiście po *n-1* krotnym wykonaniu instrukcji z wierszy linii 2 i 3. wszystkie elementy będą na właściwych miejscach w tablicy.

Opisane instrukcje działają skutecznie, ale nieoptymalnie i można poprawić ich działanie. Po przyjrzeniu się ostatniemu kodowi, łatwo zauważyć, że instrukcja warunkowa wykona się około $n2$ razy. Takie algorytmy, gdzie instrukcja dominująca jest wykonywana ok. $n2$ razy, oznacza się symbolem O($n2$).

W jaki sposób można poprawić działanie zaproponowanego algorytmu? Można to zrobić na kilka sposobów. Najczęściej stosowane są dwa:

a)można skrócić działanie drugiej pętli (ze zmienną sterującą *i*) w ten sposób, aby nie były już sprawdzane te elementy, które trafiły już na właściwe pozycje,

b)można również sprawdzać, czy w trakcie realizacji instrukcji w wierszach 2 i 3 nastąpiła choć jednokrotna zamiana elementów. Gdyby takiej zamiany nie było, to z całą pewnością można stwierdzić, że tablica jest posortowana.

Tab.1. Działanie algorytmu na tablicy 10-elementowej.

	Zawartość tablicy									
-	7	2	9	5	3	1	8	0	4	6
p=1	2	7	5	3	1	8	0	4	6	9
p=2	2	5	3	1	7	0	4	6	8	9
p=3	2	3	1	5	0	4	6	7	8	9
p=4	2	1	3	0	4	5	6	7	8	9
p=5	1	2	0	3	4	5	6	7	8	9
p=6	1	0	2	3	4	5	6	7	8	9
p=7	0	1	2	3	4	5	6	7	8	9
p=8	0	1	2	3	4	5	6	7	8	9
p=9	0	1	2	3	4	5	6	7	8	9

W kolejnych wierszach tablicy 1 znaleźć można zawartość tablicy *t* po kolejnych pełnych realizacjach instrukcji z pętli z wierszu 2. Widać tu, że pełne posortowanie tablicy nastąpiło już przy $p = 7$.

Spróbujmy teraz zapisać kod powyższego algorytmu, który po każdym przebiegu pętli ze zmienną sterującą *p* będzie zmniejszał liczbę porównywanych par sąsiednich elementów tak, by elementy, które są na właściwej pozycji nie były wielokrotnie sprawdzane:

```
1. for (int p=1; p<n; p++)
2.   for (int i=0; i<n-p; i++)
3.     if (t[i] > t[i+1])
4.       swap(t[i], t[i+1]);
```

Mała modyfikacja w wierszu drugim spowodowała, że teraz ostatnimi elementami porównywanymi w tablicy są elementy o indeksach $n - p$ oraz $n - p + 1$. Dzięki temu pętla wewnętrzna dla różnych (coraz większych) wartości zmiennej *p* będzie wykonywała się coraz mniejszą liczbę razy.

Teraz przykładowy wariant sortowania bąbelkowego, który przerywa działanie algorytmu, gdy podczas wykonywania pętli wewnętrznej nie nastąpiła żadna zamiana elementów:

```
1. for (int p=1; p<n; p++)
2.  {
3.    lzamian=0;
4.    for (int i=0; i<n-p; i++)
5.      if (t[i] > t[i+1])
6.       {
7.         lzamian++;
8.         swap(t[i], t[i+1]);
9.       }
10.    if (lzamian == 0) break;
11. }
```

Program działa w ten sposób, że w każdym obiegu pętli wewnętrznej zliczana jest liczba zamian elementów sąsiednich. Jeśli po którejś pętli zmienna *lzamian* ma wartość 0, to oznacza, że wszystkie elementy są już uporządkowane i algorytm zostaje przerwany instrukcją break.

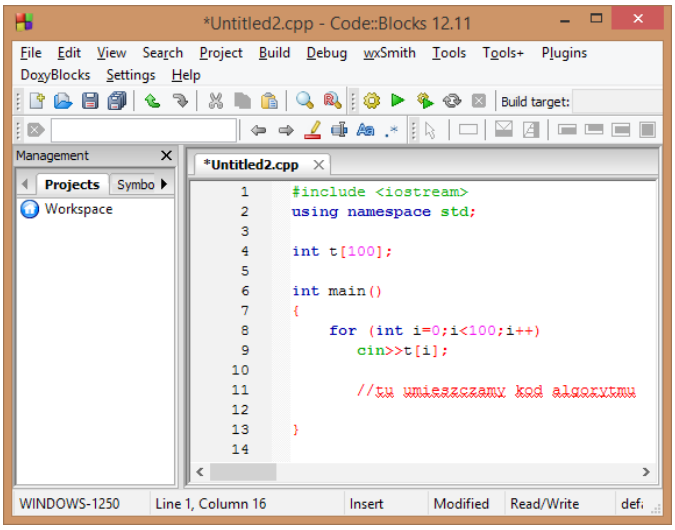
Zaproponowane ulepszenia algorytmu, choć dla pewnych danych poprawiają działanie, nie zmieniają ogólnej złożoności. Pesymistyczna złożoność, dla danych „złośliwych", pozostaje niezmieniona i wynosi O(*n*2).

Wczytywanie danych do programu

Bardzo przydatną umiejętnością jest wczytywanie danych do programu, zwłaszcza wtedy, gdy jest ich bardzo dużo. Ręczne wpisywanie z klawiatury np. 100 liczb jest uciążliwe, więc przy testowaniu programu warto uzupełnić powyższe instrukcje o czytanie danych:

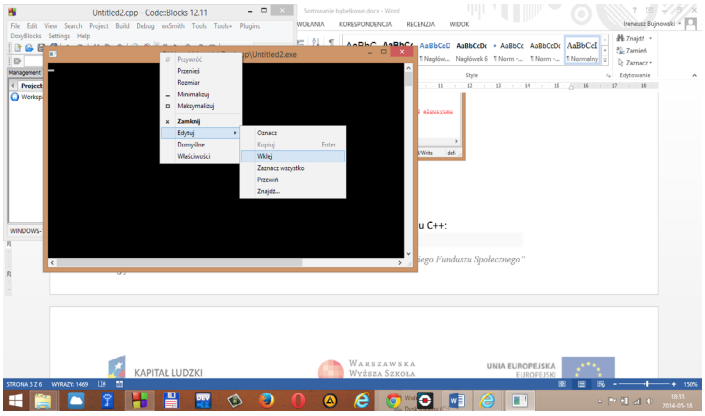
a)wczytywanie danych ze schowka – jeśli dane testowe są w jakimś pliku, lub zapisane w postaci komentarza na końcu pliku z programem, to do ich wczytania możemy wykorzystać schowek systemowy. Umieszczamy dane w schowku.

Po uruchomieniu programu wymagającego wczytania danych



pojawi się czarne okienko (okienko służące do komunikacji z programem).

Dane można wczytać klikając prawym przyciskiem myszy na pasku tytułowym tego okna, po wybraniu opcji Edytuj, a potem opcji Wklej z menu kontekstowego.



b) wczytanie danych z pliku wykorzystując strumienie:

```
1. #include <iostream>
2. #include <fstream>
3. using namespace std;
4.
5. int t[100];
6. int main()
7.  {
8.   // wczytywanie danych z pliku
9.   ifstream plik1 („dane.in”);
10.    for (int i=0;i<100;i++)
11.      plik1 >> t[i];
12.    plik1.close();
13.
14.    //zapis wyniku w pliku
15.    ofstream plik2 („wyniki.out”);
16.    for (int i=0;i<100;i++)
17.      plik2 << t[i];
18.    plik2.close();
19.
20.}
```

W wierszu 2 zawarta jest instrukcja dołączająca bibliotekę odpowiedzialną za strumienie plikowe. W wierszach 9-12 zapisane są instrukcje, które wykorzystując mechanizm strumieni wczytują dane z pliku dane. in. Analogiczne instrukcje, powodujące zapis wyniku do pliku, zawarte są w wierszach 15-18.

Wymienione dwa sposoby wczytywania danych do programu, przez schowek i z pliku, warto stosować przy rozwiązywaniu zadań, bowiem umożliwiają wielokrotne testowanie i poprawianie (debugowanie) działania zastosowanego algorytmu i jego komputerowej realizacji (implementacji).

Przebieg zajęć

Czynności organizacyjne (5 minut)

Sprawdzenie obecności, podanie tematu, określenie celów lekcji.

Wprowadzenie (10 minut)

Omówienie materiału teoretycznego przygotowanego do lekcji – prezentacja:

- sortowanie bąbelkowe,
- wczytywanie danych do programu z wykorzystaniem schowka,
- wczytywanie danych do programu z wykorzystaniem plików.

Praca zespołowa (20 minut)

Uczniowie pracując w grupach rozwiązują proste zadania (programy) wymagające sortowania danych.

➠ Ćwiczenie 1.1

Napisz program, który posortuje dane zawarte w pliku posortuj.dat i zapisze je do pliku posortowane.dat.

Dyskusja podsumowująca (5 minut)

Pisząc programy w C++ możemy wczytywać i wypisywać dane wykorzystując strumienie plikowe i schowek. Dzięki temu można wielokrotnie testować program na tych samych, nieraz rozbudowanych zestawach danych.

Omówienie z uczniami występujących trudności i sposobów unikania w przyszłości popełnianych błędów.

Praca domowa (5 minut)

➠ Ćwiczenie 1.2

Napisz program, który wczyta dane do tablicy zawartość pliku posortuj.dat oraz zliczy liczbę porównań elementów sąsiednich, wykonywanych podczas sortowania bąbelkowego w wersji bez optymalizacji.

➠ Ćwiczenie 1.3

Napisz program, który wczyta dane zawarte w pliku posortuj.dat i obliczy liczbę zamian elementów, wykonywanych przez zoptymalizowaną wersję algorytmu sortowania bąbelkowego.

Sprawdzanie wiedzy

Sprawdzenie wiedzy na podstawie testu.

Ocenianie

Ocena pracy na podstawie rozwiązanych ćwiczeń.

Dostępne pliki

1. Prezentacja
2. Zadania
3. Test
4. Filmy

LEKCJA NR 2

TEMAT: Sortowanie przez wybieranie

Co przygotować

- Prezentacja 2 „Sortowanie przez wybieranie”

MATERIAŁ TEORETYCZNY

Algorytmy porządkowania danych są idealnym problemem, na podstawie którego można pokazać różnorodność sposobów podejścia do rozwiązania zadania. W sposób przystępny da się przy okazji przedstawić, czym jest złożoność algorytmów i jak ją mierzyć. Po sortowaniu bąbelkowym, kolejnym algorytmem sortowania, z którym warto zapoznać uczniów jest algorytm sortowania przez wybieranie, czyli SelectionSort.

Znajdowanie minimum w tablicy

Nim przejdziemy do algorytmu sortowania przez wybieranie warto pokazać (czasem przypomnieć) algorytm wyszukiwania elementu najmniejszego (minimalnego) w tablicy lub fragmencie tablicy.

Popatrzmy na poniższy kod:

```
1. emin = t[0];
2.   for (int i=1; i<n-1; i++)
3.     if (t[i]< emin) emin = t[i];
```

Ostatni wiersz mógłby wyglądać np. tak:

```
3.     emin = min(emin, t[i]);
```

co może się bardziej podobać, ale nie zmienia to ogólnego działania algorytmu.

Efektem działania przedstawionego kodu jest zapisanie w zmiennej emin najmniejszej wartości występującej w tablicy pomiędzy indeksami *0*, a *n – 1*.

Ten algorytm można nieco zmodyfikować: zamiast szukać najmniejszej wartości można po prostu pamiętać, gdzie w tablicy taka wartość występuje:

```
1. pmin = 0;
2.   for (int i=1; i<n-1; i++)
3.     if (t[i]< t[pmin]) pmin = i;
```

Zmienna pmin na początku działania algorytmu wskazuje na element spod indeksu 0, który jest z założenia kandydatem na element najmniejszy. W wierszach 2-3 znajduje się pętla, w której sprawdzane są kolejne elementy i porównywane z elementem z pozycji pmin. Jeśli testowany element jest mniejszy, to od tego momentu zmienna pmin wskazuje na jego indeks.

Sortowanie przez wybieranie

Chcąc nauczyć się programować, czy budować działające algorytmy warto spróbować opisywać słownie ich działanie. Dla algorytmu sortowania przez wybieranie prawdziwe są przynajmniej dwa sformułowania:

1. W wyniku działania algorytmu na kolejne pozycje i w tablicy wybierane są najmniejsze elementy z pozycji od *i* do *n* – 1.
2. W sortowanej tablicy wyróżniamy dwie jej części – posortowaną lewą i nieposortowaną prawą. Początkowo lewa część jest pusta, lecz w kolejnym kroku jest powiększana o jeden element, najmniejszy wybrany spośród wszystkich elementów z nieposortowanej części tablicy.

Wizualizacja sortowania przez wybieranie:

Przykład:

tablica do posortowania

1	7	8	9	2	4	3

Znaleziony najmniejszy element to 1 – nic nie zamieniamy, bo jest na swoim miejscu.

1	7	8	9	2	4	3

Od drugiego elementu poszukujemy najmniejszego → to 2.

1	7	8	9	2	4	3

Zamieniamy miejscami drugi element, czyli 7, ze znalezionym najmniejszym.

Efekt to:

1	2	8	9	7	4	3

Dwa elementy są już na swoich miejscach, poszukujemy najmniejszego elementu począwszy od trzeciego to 3:

1	2	8	9	7	4	3

Zamieniamy miejscami.

Po zamianie:

1	2	3	9	7	4	8

Trzy elementy są już na swoich miejscach; najmniejszy element licząc od czwartego to 4:

1	2	3	9	7	4	8

Po zamianie (cztery elementy na miejscu).

1	2	3	4	7	9	8

Najmniejszy element na białym tle to 7 i jest na swoim miejscu – brak zamiany.

1	2	3	4	7	9	8

1	2	3	4	7	9	8

Kolejny najmniejszy element to 8.

Po zamianie, ostatni element jest już na swoim miejscu:

1	2	3	4	7	8	9

tablica posortowana

1	2	3	4	7	8	9

Złożoność algorytmu sortowania przez wybieranie wynosi również O(*n*2).

Teraz już czas na przedstawienie kodu omówionego algorytmu:

```
5. for (int p=0; p<n-1; p++)
6.  {
7.      pmin = p;
8.      for (int i=p+1; i<n; i++)
9.        if (t[i]< t[pmin]) pmin = i;
10.     swap(t[i], t[pmin])
11.  }
```

Przebieg zajęć

Czynności organizacyjne (5 minut)

Sprawdzenie obecności, podanie tematu, określenie celów lekcji.

Wprowadzenie (10 minut)

Omówienie materiału teoretycznego przygotowanego do lekcji – prezentacja:

- sortowanie przez wybieranie,
- przypomnienie wczytywania danych do programu z wykorzystaniem schowka,
- przypomnienie jak wczytać dane z plików.

Praca zespołowa (20 minut)

Uczniowie pracując w grupach rozwiązują proste zadania (programy) wymagające sortowania danych.

Ćwiczenie 2.1

Napisz program, który posortuje dane zawarte w pliku posortuj.dat i zapisze je do pliku posortowane.dat. Do rozwiązania zadania wykorzystaj sortowanie przez wybieranie.

Dyskusja podsumowująca (5 minut)

Porównanie prędkości działania algorytmu sortowania bąbelkowego i sortowania przez wybieranie.

Warto razem z młodzieżą przeanalizować działanie algorytmu sortowania, gdy dane są uporządkowane rosnąco oraz gdy są uporządkowane malejąco. Podczas analizy wariantu sortowania uzależniającego kolejne przebiegi algorytmu od liczby dokonanych wcześniej zamian okazuje się, że przy próbie posortowania ciągu rosnącego algorytm wykona jedynie ok. n operacji, co daje złożoność liniową O(n), a to znaczy, że działa jak najszybszy algorytm sortujący. W przypadku analizy działania sortowania przez wybierania dla identycznych danych złożoność jest kwadratowa O(n2).

Omówienie z młodzieżą występujących trudności i sposobów unikania w przyszłości popełnianych błędów.

Praca domowa (5 minut)

Ćwiczenie 2.2

Napisz program, który wczyta dane do tablicy zawartość pliku posortuj.dat oraz zliczy liczbę porównań elementów sąsiednich wykonywanych przez algorytm sortowania przez wybieranie.

Ćwiczenie 2.3

Napisz program, który umożliwi porównanie czasów działania algorytmu dla danych zawartych w plikach posortuj*.dat. Możesz to zrobić wczytując dane strumieniami z pliku, tak by czas wczytania nie miał wielkiego wpływu na ogólny czas działania algorytmu.

Uwaga: środowisko CodeBlocks po uruchomieniu i zakończeniu programu zawsze wyświetla czas działania:

Sprawdzanie wiedzy

Sprawdzenie wiedzy na podstawie testu.

Ocenianie

Ocena pracy na podstawie rozwiązanych ćwiczeń.

Dostępne pliki

1. Prezentacja
2. Zadania
3. Test
4. Filmy

LEKCJA NR 3

TEMAT: Sortowanie przez zliczanie

Co przygotować

- Prezentacja 3 „Sortowanie przez zliczanie”

MATERIAŁ TEORETYCZNY

Przedstawione dotychczas algorytmy sortowania działały stosunkowo wolno. Zarówno sortowanie bąbelkowe, jak i sortowanie przez wybieranie były klasy O(*n*2), to znaczy, że aby wykonać sortowanie *n* elementów, należy w najgorszym razie wykonać około *n*2 operacji dominujących (porównań). Do analizy działania algorytmów możemy założyć, że współczesne komputery potrafią wykonać ok. 108-109 operacji arytmetycznych na sekundę. Gdyby tak było w rzeczywistości, to wymienione algorytmy sortowania do uporządkowania 106 elementów potrzebowałyby wykonania 1012 operacji. To w połączeniu z prędkością komputera oznaczałoby czas sortowania na około 1000s.

Aby móc posortować tak dużo danych, warto skorzystać z algorytmów działających szybciej. Okazuje się, że przy spełnieniu pewnych ograniczeń można wykonać sortowanie liniowo tzn. o złożoności, która jest proporcjonalna do liczby elementów – O(*n*). Posortowanie 106 elementów zajmowałoby ok. 1/1000-1/100s.

Sortowanie przez zliczanie

Jedną z takich metod sortowania, działającą w czasie proporcjonalnym do liczby elementów, jest sortowanie przez zliczanie. Metodę tę można stosować wtedy, gdy naszym zadaniem jest posortowanie liczb całkowitych o wartościach z pewnego małego zakresu: aby móc policzyć wystąpienia elementów, należy stworzyć dodatkową, pomocniczą tablicę, w której zlicza się wystąpienia elementów w tablicy, którą chcemy posortować.

Jak działa sortowanie przez zliczanie?

Najłatwiej prześledzić to analizując konkretny przykład:

Należy posortować tablicę *T[n]* daną poniżej, *n=8*.

T[0]	*T[1]*	*T[2]*	*T[3]*	*T[4]*	*T[5]*	*T[6]*	*T[7]*
3	**4**	**2**	**4**	**9**	**2**	**2**	**8**

Tablica jest wypełniona elementami całkowitymi dodatnimi z zakresu od 2 do 9.

Dla ułatwienia zakres liczb, który wpływa na rozmiar tablicy pomocniczej *z* można przyjąć jako <0,9>.

1. Mając już wczytaną tablicę *T* tworzymy pomocniczą tablicę *Pom*.
2. Wypełniamy tablicę *Pom* zerami.

Poniższy fragment kodu ilustruje początkową część tego algorytmu:

```
1. for (int i=0; i<z; i++)
2.   Pom[i]=0;
```

Instrukcja pętli zawarta w wierszach 1-2 umożliwia przypisanie wszystkim elementom tablicy Pom wartości 0*:*

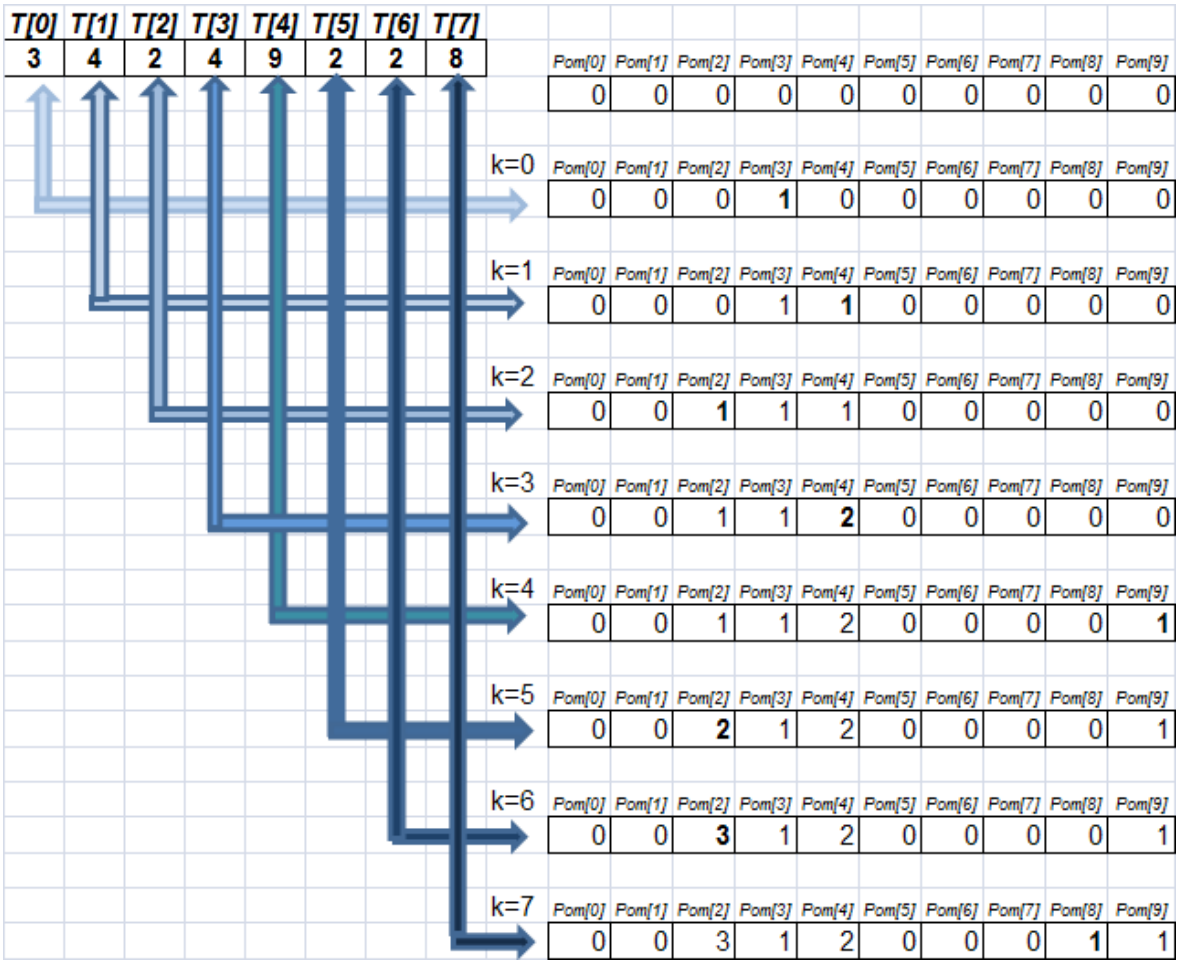
Pom[0]	*Pom[1]*	*Pom[2]*	*Pom[3]*	*Pom[4]*	*Pom[5]*	*Pom[6]*	*Pom[7]*	*Pom[8]*	*Pom[9]*
0	0	0	0	0	0	0	0	0	0

Następnie poszczególne wartości tablicy *T* traktujemy jako indeksy komórek tablicy *Pom*, które należy powiększyć o 1 w razie ich każdorazowego wystąpienia w tablicy T.

```
3. for (int k=0; k<n; k++)
4.   Pom[T[k]]++;
```

Poniżej przedstawiona jest wizualizacja ilustrująca jak zmienia się tablica pomocnicza *Pom* w trakcie działania powyższej pętli:

dla k=0 T[0]=3 Pom[T[0]] zwiększa się o 1, a więc Pom[3]=1
dla k=1 T[1]=4 Pom[T[1]] zwiększa się o 1, a więc Pom[4]=1
dla k=2 T[2]=2 Pom[T[2]] zwiększa się o 1, a więc Pom[2]=1
dla k=3 T[3]=4 Pom[T[3]] zwiększa się o 1, a więc Pom[4]=2 itd.



Tablice *Pom* po zakończeniu działania powyższej pętli wygląda następująco:

Pom[0]	*Pom[1]*	*Pom[2]*	*Pom[3]*	*Pom[4]*	*Pom[5]*	*Pom[6]*	*Pom[7]*	*Pom[8]*	*Pom[9]*
0	0	3	1	2	0	0	0	1	1

Po zliczeniu liczby wystąpień elementów w tablicy T i wypełnieniu tablicy Pom należy przeprowadzić operację polegającą na wypełnieniu odpowiednimi liczbami komórek tablicy T na podstawie zawartości tablicy Pom.

```
6. for (int j=0; j<z; j++)
7.   for (int l=0; l<Pom[j]; l++)
8.     cout<<j<<" ";
```

Pom[0]	*Pom[1]*	*Pom[2]*	*Pom[3]*	*Pom[4]*	*Pom[5]*	*Pom[6]*	*Pom[7]*	*Pom[8]*	*Pom[9]*
0	0	3	1	2	0	0	0	1	1

Czyli dla j=0 Pom[0]=0 instrukcja 7 nie zostanie wykonana:

dla j=1 pom[1]=0 instrukcja 7 nie zostanie wykonana;
dla j=2 pom[2]=3 instrukcja 7 zostanie wykonana 3 razy;→2 2 2
dla j=3 pom[3]=1 instrukcja 7 zostanie wykonana 1 raz;→3.
dla j=4 pom[4]=2 instrukcja 7 zostanie wykonana 2 razy;→4 4
dla j=5 pom[5]=0 instrukcja 7 nie zostanie wykonana;
dla j=6 pom[6]=0 instrukcja 7 nie zostanie wykonana;
dla j=7 pom[7]=0 instrukcja 7 nie zostanie wykonana;
dla j=8 pom[8]=1 instrukcja 7 zostanie wykonana 1 raz;→8
dla j=9 pom[9]=1 instrukcja 7 zostanie wykonana 1 raz;→9

końcowy efekt to:
2 2 2 3 4 4 8 9

Należy zwrócić uwagę, że powyższy kod wypisał wprawdzie posortowane elementy rosnąco, ale nie zapisał tego w tablicy.
Jak należałoby zmienić kod, aby posortowane elementy znalazły się w tablicy?
Jak należałoby zmienić kod, by elementy były wypisywane malejąco?

Wady i zalety sortowania przez zliczanie:

- Zaleta: działa w czasie liniowym (jest szybki); złożoność O(n) – *n* to liczba elementów
- Wada: może sortować wyłącznie liczby całkowite, a nie nadaje się zupełnie do sortowania takich typów jak liczby rzeczywiste czy zmienne tekstowe
- Wada: wymaga dodatkowej pamięci na tablicę pomocniczą

Powyższy kod porządkował liczby całkowite nieujemne.

Jak należałoby zmienić kod, aby sortował liczby całkowite dodatnie oraz niedodatnie.

Przebieg zajęć

Czynności organizacyjne (5 minut)

Sprawdzenie obecności, podanie tematu, określenie celów lekcji.

Wprowadzenie (10 minut)

Omówienie materiału teoretycznego przygotowanego do lekcji – prezentacja:

– sortowanie przez zliczanie.

Praca zespołowa (20 minut)

Uczniowie pracując w grupach rozwiązują proste zadania (programy) wymagające sortowania danych metodą przez zliczanie.

➠ Ćwiczenie 3.1

Napisz program, który posortuje dane zawarte w pliku posortuj.dat i zapisze je do pliku posortowane.dat.

Dyskusja podsumowująca (5 minut)

Omówienie z uczniami występujących trudności i sposobów unikania w przyszłości popełnianych błędów.

Praca domowa (5 minut)

➠ Ćwiczenie 3.2

Napisz program, który wczyta dane do tablicy zawartość pliku posortuj.dat oraz zapisze posortowane rosnąco przez zliczanie elementy do tablicy.

➠ Ćwiczenie 3.3

Zmodyfikuj program z Ćwiczenia 3.2, tak by elementy w tablicy docelowej były posortowane malejąco.

Sprawdzanie wiedzy

Sprawdzenie wiedzy na podstawie testu.

Ocenianie

Ocena pracy na podstawie rozwiązanych ćwiczeń.

Dostępne pliki

1. Prezentacja
2. Zadania
3. Test
4. Filmy

Człowiek - najlepsza inwestycja

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska
Wyższa Szkoła
Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego