Wrocław,27.01.2014r.

Kacper Przezak 200616

Termin zajęć: środa 9:15 TN

Projektowanie efektywnych algorytmów
Sprawozdanie z ćwiczenia nr 2:
Problem plecakowy

Prowadzący: dr inż. Andrzej Rusiecki

1.Opis implementacji:

Program składa się z klasy Plecak, oraz struktury Przedmiot. Użytkownik ma dwie możliwości użytkowania programu może wczytać dane testowe bezpośrednio z pliku, lub też podać ilość elementów, oraz pojemność plecaka, wówczas dane zostaną zapisane w pliku ”generowany.txt”. Waga przedmiotu wynosi od 1 do połowy pojemności plecaka, wartość od 1 do pojemności plecaka. Do generowania danych wykorzystałem funkcję ”rand()”, oraz operator ”%” zwracający resztę z dzielenia. Odpowiednie dobranie powyższych wielkości ma istotny wpływ na czas wykonywania się algorytmu, oraz ilość używanej przez program pamięci operacyjnej. W celu ułatwienia wykonywania pomiarów czasu wykonywania się algorytmu obiekty klasy Plecak można tworzyć przy użyciu konstruktora parametrowego. Dzięki temu podczas pojedynczego uruchomienia programu dokonujemy kilku pomiarów.

```
44          if (menu == 1)
45          {
46              cout << "\npodaj nazwe pliku z instancja testowa\n";
47              cin >> nazwa_pliku;
48          }
49          if (menu == 2)
50          {
51              ofstream plik_generuj;
52              int pom_pojemnosc;
53              int pom_ilosc_elementow;
54              plik_generuj.open("generowany.txt");
55              cout << "Podaj pojemnosc plecaka\n";
56              cin >> pom_pojemnosc;
57              plik_generuj << "\n" << pom_pojemnosc;
58              cout << "Podaj ilosc elementow\n";
59              cin >> pom_ilosc_elementow;
60              plik_generuj << "\n" << pom_ilosc_elementow;
61
62              for (int i = 0; i < pom_ilosc_elementow; i++)
63              {
64
65                      plik_generuj << "\n" << rand() %(int) (pom_pojemnosc*0.5) + 1 << "\t" << rand() % pom_pojemnosc + 1;
66              }
67              nazwa_pliku = "generowany.txt";
68              plik_generuj.close();
69
```

Listing 1 wczytanie lub generowanie danych wejściowych

Kolejnym etapem działania programu jest wczytanie danych z pliku, przy użyciu funkcji wczytaj_plik_testowy() . Funkcja wykorzystuje klasę fstream służącą do operacji na plikach. Pierwszą wykonaną czynnością jest pobranie każdej linii z pliku do zmiennej o nazwie ”linia” typu string, oraz dodanie na koniec wektora pobranej wartości. Następnie wykonywana jest konwersja danych zawartych w wektorze odczytu na typ int i zapisanie ich do wektora_liczb. Do zamiany typów została użyta funkcja atoi(), która jako parametr przyjmuje cstring a zwraca int. Jeśli konwersja nie powiedzie się funkcja atoi() zwraca 0.

```
161     bool wczytaj_plik_testowy(string nazwa_pliku)
162     {
163         vector<string> wektor_odczytu;
164         fstream plik;
165         plik.open(nazwa_pliku, ios::in);
166         if (plik.good())
167         {
168             //wczytanie całego pliku, każda linia jest oddzielnym elementem wektora_odczytu
169             string linia;
170             while (!plik.eof())
171             {
172                 getline(plik, linia);
173                 wektor_odczytu.push_back(linia);
174             }
175             for each (string linia in wektor_odczytu)
176             {
177                 stringstream strumien(linia);
178                 do
179                 {
180                     string sub;
181                     strumien >> sub;
182                     if (atoi(sub.c_str()))
183                     {
184                         wektor_liczb.push_back(atoi(sub.c_str()));
185                     }
186 
187                 } while (strumien);
188             }
189             return true;
190         }
191         else
192         {
193             cout << "Niepoprawny odczyt pliku\n";
194             return false;
195         }
196 
197     }
```

Listing 2 funkcja wczytująca dane z pliku

Następną wykonywaną przez program czynnością jest pobranie z wektor_liczb dwóch pierwszych wartości które oznaczają pojemność plecaka, oraz ilość jego elementów, po pobraniu każdej z wartość zostaje ona usunięta z wektora. Pobrane wartości są przypisywane zmiennym klasy Plecak. Następną czynnością jest sprawdzenie poprawności ilości pobranych danych i wypełnienie wektora zawierającego obiekty o strukturze Przedmiot.

```
72          if (menu == 1 || menu == 2)
73          {
74          if (wczytaj_plik_testowy(nazwa_pliku))
75          {
76              //pobranie pojemnosci plecaka, oraz usuniecie tej danej z wektora
77              pojemnosc = wektor_liczb.front();
78              //usuniecie pierwszego elementu wektora
79              wektor_liczb.erase(wektor_liczb.begin());
80              //pobranie ilosci elementow, oraz usuniecie tej danej z wektora
81              ilosc_elementow = wektor_liczb.front();
82              //usuniecie pierwszego elementu wektora
83              wektor_liczb.erase(wektor_liczb.begin());
84              cout << "Problem plecakowy\nzlodziej ma do dyspozycji\t" << ilosc_elementow << " elementow\n" << "pojemnosc plecaka wynosi\t" << pojemnosc << "\n";
85              if (wektor_liczb.size() == 2 * ilosc_elementow)
86              {
87                  //stworzenie wektora zawierajacego elementy plecaka
88                  for (int i = 0; i < wektor_liczb.size(); i = i + 2)
89                  {
90                      wektor_przedmiotow.push_back(Przedmiot(wektor_liczb[i], wektor_liczb[i + 1]));
91                  }
92
93                  tablica_wartosci = new  int*[ilosc_elementow+1];
94                  for (int i = 0; i < ilosc_elementow+1; i++)
95                  {
96                      tablica_wartosci[i] = new int[pojemnosc+1];
97
98                  }
99                  programowanie_dynamiczne();
100             }
101         }
102         else
103         {
104             cout << "Niepoprawna ilosc wczytanych wartosci\nalgorytm nie wykona sie :(\n";
105         }
```

Listing 3 stworzenie wektora przedmiotów

Następnym etapem programu jest wywołanie funkcji programowanie dynamiczne która jest najistotniejszą częścią zadania. Pierwszym zadaniem tej funkcji jest dynamiczne stworzenie tablicy o ilości wierszy równej liczbie elementów i ilości kolumn równej pojemności plecaka. Kolejną czynnością jest wypełnienie tablicy wartości. Jeśli w dla rozpatrywanego elementu i danej pojemności plecaka rozmiar przekracza pojemność wówczas tablica wartości jest wypełniana elementem znajdującym się wiersz wyżej. W przypadku gdy rozpatrywany element zmieści się w plecaku do tablicy zostaje dodana większa z wartości tablica wartości o wiersz o 1 mniejszy lub dany przedmiot i tablica wartości o poprzednim wierszu i nr kolumny (nr kolumny oznacza wykorzystaną pojemność plecaka) równym różnicy rozpatrywanej kolumny i wagi danego przedmiotu. Skutkiem wykonania powyższych instrukcji jest znalezienie optymalnego rozwiązania problemu, które zapisane jest w tablicy wartości o wierszu równym ilości elementów i kolumnie równej pojemności plecaka.

```
void  programowanie_dynamiczne()
{
    //srand(time(NULL));
    //clock_t start, koniec;
    //start = clock();
    auto begin = std::chrono::high_resolution_clock::now();
    for (int i = 0; i <= ilosc_elementow;i++)
    {
        tablica_wartosci[i][0] = 0;
    }
    for (int i = 0; i <= pojemnosc; i++)
    {
        tablica_wartosci[0][i] = 0;
    }

    for (int i = 1; i <= ilosc_elementow;i++)
    {
        for (int j = 0; j <= pojemnosc; j++)
        {
            if (wektor_przedmiotow[i-1].rozmiar > j)
            {
                tablica_wartosci[i][j] = tablica_wartosci[i - 1][j];
            }
            else
            {
                tablica_wartosci[i][j] = max(tablica_wartosci[i - 1][j], wektor_przedmiotow[i-1].wartosc + tablica_wartosci[i - 1][j - wektor_przedmiotow[i-1].rozmiar]);
            }
        }
    }
```

Listing 4 programowanie dynamiczne

W celu sprawdzenia poprawności wykonania algorytmu konieczne jest wyświetlenie wybranych elementów. Sposób w jaki jest to realizowane przedstawiony jest na poniższym listingu.

```
    bool log = true;
    int wiersz = ilosc_elementow;
    int kolumna = pojemnosc;
    do
    {
        if (tablica_wartosci[wiersz][kolumna] == tablica_wartosci[wiersz-1][kolumna])
        {
            wiersz--;
        }
        else
        {
            cout << "\n wartosc elementu\t" << wektor_przedmiotow[wiersz-1].wartosc << "\t rozmiar \t"<<wektor_przedmiotow[wiersz-1].rozmiar;

            kolumna = kolumna - wektor_przedmiotow[wiersz-1].rozmiar;
            wiersz--;
        }
        if (kolumna < 0 || wiersz == 0)
        {
            log = false;
        }

    } while (log);

    cout << "\n\nnajlepsze rozwiazanie\t" << tablica_wartosci[ilosc_elementow][pojemnosc];


}
```

Listing 5 wyświetlenie wybranych elementów

2.Pomiary czasu wykonywania algorytmu

charakterystyka sprzętowa

- System operacyjny Windows 8.1 64-bit
- Procesor Intel Core i7-2670QM CPU 2.2GHz
- Pamięć operacyjna 8GB 1333MHz

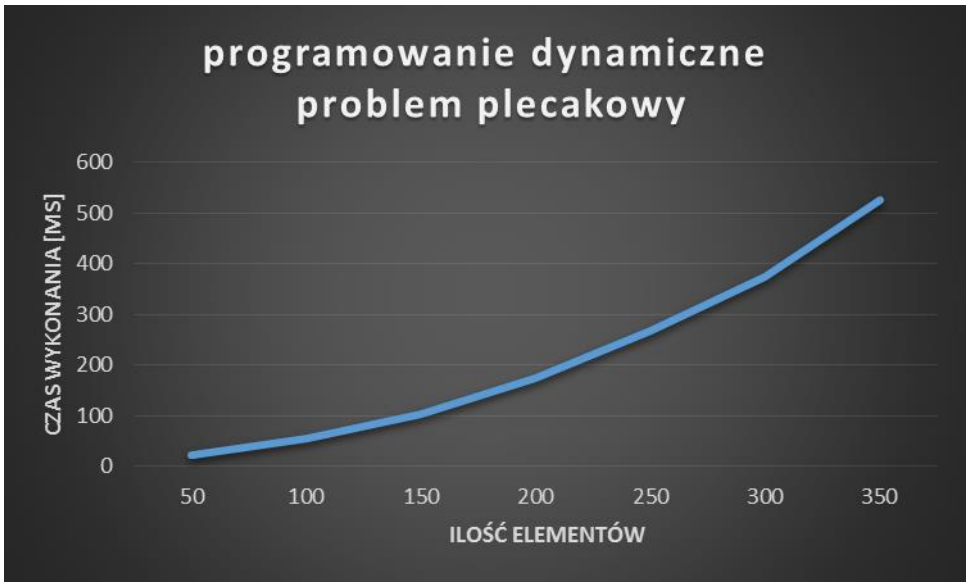
Podczas wykonywania pomiarów komputer nie był w znaczący sposób obciążony wykonywanie innych zadań. Dane służące do testowania algorytmu zostały wczytane z plików znajdujących się w folderze Debug. Dla każdych danych testowych wykonano 10 pomiarów które następnie zostały uśrednione

50 elementów	100 elementów	150 elementów	200 elementów	250 elementów	300 elementów	350 elementów
15	31	95	171	265	359	520
15	62	94	185	267	379	528
15	46	93	178	275	360	526
15	46	109	171	266	380	531
15	60	101	156	262	380	532
31	37	109	203	265	371	517
31	62	125	171	264	380	517
31	66	111	162	266	375	528
31	78	93	158	266	377	528
15	46	109	171	267	376	527

Tabela 1 pomiary czasu wykonywania się algorytmu wyrażone w ms

50 elementów	100 elementów	150 elementów	200 elementów	250 elementów	300 elementów	350 elementów
21,4	53,4	103,9	172,6	266,3	373,7	525,4

Tabela 2 uśrednione pomiary czasu wykonywania się algorytmu wyrażone w ms



wykres 1wykres zależności czasu wykonywania się algorytmu od ilości elementów

3.Wnioski

Krzywa widoczna na wykresie 1 pokazuje zależność czasu rozwiązywania problemu od ilości elementów. Nietrudno zauważyć iż zależność ta rośnie w sposób wykładniczy co jest zgodne z założeniami algorytmu. Oprócz ilości elementów czas wykonywania zależy w znacznym stopniu od pojemności plecaka, wraz ze wzrostem rozmiaru plecaka ilość rozpatrywanych elementów rośnie. Ze względu na ograniczoną ilość pamięci program nie zadziała poprawnie dla instancji o znacznej wartości iloczynu ilości elementów i pojemności plecaka.
.