O algorytmice

Mariusz Blank

1

Intro:

- Development environment:

  Visual Studio Code (https://code.visualstudio.com/)

  Extensions:
  - Remote - SSH from Microsoft
  - C/C++ from Microsoft

  platform: ideone.com – for home/private exercises

- Contact:

  mariusz.blank@nokia.com tel. 609 824 909

2

Algorytmika

- Algorytmy – przykłady (szachy, 4 kolory, wyjście z labiryntu, układanka)
- W czym tkwi trudność
- Rekurencja
- Algorytmy zachłanne
- Programowanie dynamiczne
- Sortowanie
- Struktury danych – wektor, lista, stos, kolejka, drzewo binarne, sterta, tablica typu hash
- Miara oceniania algorytmów

Algorytmy i sposoby ich przedstawienia

- Algorytmy na przestrzeni wieków – rys historyczny
- Reprezentacje algorytmów
  - Słowny opis
  - Opis w postaci listy kroków
  - Schemat blokowy
  - Program zapisany w języku programowania np. C/C++
  - Programy wymagają precyzyjnej składni
  - Języki programowania – dlaczego nie algorytmiczne Esperanto?

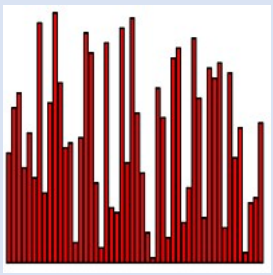
Zadania – przykłady algorytmów.

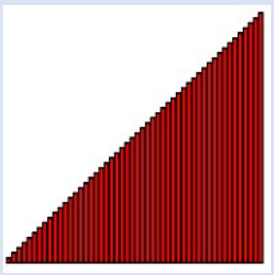
- Gra: Mamy liczbę 0, dwóch graczy wskazuje liczby naprzemiennie, następna liczba musi być większa od poprzedniej, ale o nie więcej niż 10. Kto pierwszy powie 100 – wygrywa.
- Znajdowanie maksymalnego elementu zbioru/tablicy.
- Gra nim
- Zadania: https://app.codility.com/public-link/Nokia-task1/ , {0,1..7}
- *Ćwiczenia – zadania z listy.*
- Określenie poprawności wyrażenia nawiasowego N. |N| < 1000000, e.g.: N="())())"
- Czy słowo S jest palindromem? |S| < 1000000, litery a-z.
- Czy słowa S i T są anagramami? |S| < 1000000 i |T| < 1000000.
- Porządkowanie ciągu elementów
  - Algorytm bąbelkowy
  - Algorytm kubełkowy/zliczanie
  - Quicksort
- Zapisywanie liczb w systemie binarnym – bajty, bity.

5

Sortowanie

- **Zadanie**
- Dany ciąg *n* liczb całkowitych posortować rosnąco ($a_i \leq a_{i+1}$), 0 <= i < n.
- Dane czytamy ze standartowego wejścia. W pierwszym wierszu podano liczbę $n \leq 1\,000\,000$. W kolejnych *n* wierszach zapisano po jednej liczbie całkowitej ($\leq 10^6$).
- **Przykład**
- Dla pliku:
- 4
- 3
- 1
- 3
- 4
- Odpowiedź: 1 3 3 4





6

Sortowanie bąbelkowe

```
//////////////////////////////
// Sortowanie miliona liczb z
// zakresu 0..1 000 000
//////////////////////////////
#include <cstdio>
#include <iostream>
using namespace std;
const int maks = 1000000;
long int n;
long int tab[maks];
int  main()
{
        long int i, j, k;
        cin >> n;
        for (i = 0; i < n; i++) {
                cin >> tab[i];
        }
        // babelki do gory
        bool swap = false;
        for (i = 0; i < n; i++)  {
                swap = false;
                for( j = 0; j < n - i - 1; j++)
                {
                        if ( tab[j] > tab[j + 1] )
                        {
                                swap = true;
                                long int temp = tab[j];
                                tab[j] = tab[j + 1];
                                tab[j + 1] = temp;
                        }
                }
                if ( !swap)    break;
        }
        for ( i = 0; i < n; i++)   cout << tab[i] << " ";
        return 0;
}
```

Czytamy dane z pliku (za standardowego wejścia).

7

Qsort

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
   vector <int> v;    //
   int n;   // ilosc elementow
   cin >> n;
   for (int i = 0; i < n ; ++i )
   {
        int l;
        cin >> l;
        v.push_back(l);
   }
   sort(v.begin(), v.end());
   cout << "Sorted \n";
   for (int i = 0; i < v.size(); ++i )
      cout << v[i] << " ";
   return 0;
}
```

```
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
        int t[1000000];
        int n;
        cin >> n; // ilosc elementow tablicy, n < 1000
        for (int i = 0; i < n; i++)
        {
                cin >> t[i];
        }
        sort(t, t+n);
        cout << "Sorted \n";
        for (int i = 0; i < n; i++)
        {
                cout << t[i] << " ";
        }
        return 0;
}
```

8

Sortowanie przez zliczanie

Sortowanie przez zliczanie

```
///
// Sortowanie liczb z zakresu 0..1 000 000
//

#include <iostream>
using namespace std;
const int maks = 1000000;
long int n;
long int tab[maks + 1];

int main()
{
        long int i, j, k;
        cin >> n;
        for (i = 0; i < n; i++)
        {
                cin >> k;
                tab[k]++;
        }

        for ( i = 0; i < maks; i++)
        {
             for ( j = 0; j < tab[i]; j++)
              {
                   cout << i << " ";
              }
         }
        return 0;
}
```

Zadanie: Jaka litera we wczytanym słowie S powtarza się maksymalna ilośc razy. |S| < 1000000. S sklada się z liter a-z.

9

Funkcje rekurencyjne

- Rekurencyjne
  - Silnia
  - Liczby Fibbonaciego
  - Wieża Hanoi
- Iteracyjne
  - Liczby Fibbonaciego
  - Silnia
  - Zgadywanie liczb

10

```
// Wyznaczanie wartosci funkcji f(n) = n!
// Def.: n! = 1 * 2 * 3 * ... * n, 0! = 1.

#include <iostream>
using namespace std;
long long silnia(long long i)
{
        cout << "Entered: i = ” << i << endl;
        if ( i < 1 )
                return 1;
        else
                return i * silnia(i - 1);
}
int main()
{
        int n;
        cout << "Podaj n: ";
        cin >> n;
        cout << silnia(n) << endl;
        return 0;
}
```

11

```
// Wyznaczanie k-tej liczby Fibbonacciego
// def.: f(k) = f(k - 1) + f(k - 2)
#include <iostream>
using namespace std;
long long fibb(long long k)
{
        if ( k < 3)
                return 1;
        else
                return  fibb(k - 2) + fibb(k - 1);
}
int main()
{
        int n;
        cout << "Podaj n: ";
        cin >> n;
        cout << n << "-ta liczba Fibbonacciego wynosi: " << fibb(n) << endl;
        return 0;
}
```

12

```
//Wieza Hanoi
#include <iostream>
#include <cstdlib>
using namespace std;
int hanoi(int ile)
{
      if ( ile == 1)
      {
            return 1;
      }
      return 2 * hanoi(ile - 1) + 1;
}
int main(int argc, char **argv)
{
      int n;
      n = 3;
      if ( argc == 2)
            n = atoi(argv[1]);
      long long l = hanoi(n);
      cout << "Ilosc ruchow = " << l << endl;
      return 0;
}
```

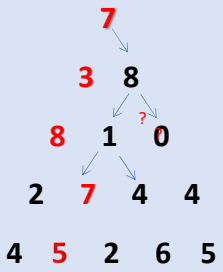
Wczytywanie parametrów z linii komend.

```
// Wypisujemy wykonywane ruchy
#include <iostream>
#include <cstdlib>
using namespace std;

int hanoi(int ile, int skad, int dokad)
{
    if ( ile == 1)
    {
       cout  << skad << " -> " << dokad << endl;
       return 0;
    }
     hanoi(ile - 1, skad, 3 - skad - dokad);
     cout << skad << " -> " << dokad << endl;
     hanoi(ile - 1, 3 - skad - dokad, dokad);
     return 0;
}

int main(int argc, char **argv)
{
      int n;
      n = 3;
      if ( argc == 2)
      n = atoi(argv[1]);
      hanoi(n, 0, 2);
      return 0;
}
```

13

Trójkąt

maksymalna jazda z góry na dół



Do 100 pięter

Ile zgarniemy?

Czy to jest łatwe?

14

Zadanie: Trójkąt

- **Napisz program obliczający maksymalną sumę liczb, przez jakie można przejść po drodze, która zaczyna się w wierzchołku i kończy w dowolnym punkcie podstawy.**
- Każdy krok można wykonać albo skośnie w lewo w dół, albo skośnie w prawo w dół.
- Liczba wierszy nie przekracza 100
- Wszystkie liczby zapisane w trójkącie są całkowite i mieszczą się pomiędzy 0 i 99.
-
- **7**
- **3 8**
- **8 1 0**
- **2 7 4 4**
- **4 5 2 6 5**
- **Dane wejściowe**
- W pierwszym wierszu pliku wejściowego podano *n* – liczbę wierszy w trójkącie. W kolejnych *n* wierszach zapisano wiersze trójkąta. Dla przykładu powyżej plik wejściowy jest następujący:

5

7

3 8

8 1 0

2 7 4 4

4 5 2 6 5

-
- **Dane wynikowe**
- Maksymalną sumę jako liczbę całkowitą zapisujemy na standartowe wyjście.


```
#include <iostream>
using namespace std;

/* global vars */
const int N = 100;
int n;
long int maks;
int tab[N][N] = {0};

int search(int wiersz, int kol, int suma)
{
	suma += tab[wiersz][kol];
	if (wiersz < n)
	{
		search(wiersz + 1, kol, suma);
		search(wiersz + 1, kol + 1,suma);
	}
	else
	{
		if (maks < suma)
			maks = suma;
	}
}
```

```
int main()
{
	cin >> n;
	int i;
	for (i = 0; i < n; i++)
	for (int j = 0; j <= i; j++)
	{
		cin >> tab[i][j];
	}
	search(0, 0, 0);
	cout << maks << endl;
	return 0;
}


/*
  Is it OK?
  How long does it take to complete the task for n = 100?
*/
```

A L G O R Y T M Y Z A C H L A N N E

- *Na każdym kroku algorytmu zachłannego dokonujemy wyboru "najlepszego w danej chwili" elementu. Proces ten kontynuujemy do znalezienie rozwiązania.*

17

Algorytmy zachłanne/żarłoczne

- **Problem 1.**

Napisz program wydający żądaną sumę pieniędzy przy pomocy minimalnej ilości banknotów i monet. [200, 100, 50, 20, 10, 5 , 2, 1, 0.50, 0.2, 0.1, 0.05, 0.02, 0.01]

- **Problem 2.** Złodziej w banku.

Złodziej udźwignie plecak z łupem o pewnej masie.
Po włamaniu do banku znalazł kilogramowe sztaby złota, srebra, miedzi w różnych ilościach.
W jaki sposób powinien wybierać sztaby, aby wynieść łup o maksymalnej wartości?
Dane:

p

z s m

Gdzie p - pojemność plecaka, z – ilość sztab złota, s - ilość sztab srebra, m – ilość sztab miedzi.

Przykład:
30

8 15 50 8Au, 15Ag, 7Cu

18

STACJA **zastanowić się nad rowiązaniem 11.09.22**

Stacja powinna obsłużyć możliwie największą liczbę spośród *n* klientów, gdzie *n* < 3500.
Każdy klient wymaga tylko jednorazowej obsługi, która musi się zacząć w ustalonej chwili początkowej oraz trwać do ustalonej chwili końcowej, ściśle – obsługa *i*-tego klienta musi być wykonana nieprzerwanie w przedziale domknięto-otwartym $[s_i, k_i)$, gdzie $0 <= s_i <= k_i < T = 32000$.
Stacja nie może obsługiwać jednocześnie dwóch klientów.
Oznacza to, że jeśli przedziały obsługi dwóch różnych klientów przecinają się, to stacja będzie mogła obsłużyć tylko jednego z nich.

Zadanie
Napisz program, który:
•Wczytuje zna wejściu liczbę klientów *n* oraz dla każdego klienta początek I koniec przedziału czasu w którym powinien być obsłużony,
•Znajduje maksymalny podzbiór klientów, których można bezkolizyjnie obsłużyć, tzn. takich, że ich przedziały obsługi nie przecinają się.

Wejście
W pierwszym wierszu pliku wejściowego zapisano liczbę całkowitą dodatnią n < 3500. Jest to liczba wszystkich klientów.
W każdym z kolejnych *n* wierszy są zapisane dwie liczby całkowite nieujemne, nie większe niż T = 32000.
Liczby w i-tym z tych wierszy: s_i oraz k_i to odpowiednio czas, w którym należy zacząć i zakończyć obsługę klienta numer *i*.

Wyjście
Dane wyprowadzamy na standartowe wyjście.
W pierwszym wierszu należy zapisać jedną liczbę całkowitą nieujemną M, tj. maksymalną liczbę klientów, jaką możne bezkolizyjnie obsłużyć na stacji.
W kolejnych M wierszach należy zapisać plan obsługi M klientów – w każdym wierszu numer jednego klienta według kolejności w jakiej będą obsługiwani przez stacje.

Przykład: Dla
3
7 9
3 8
1 5

19

PROGRAMOWANIE DYNAMICZNE

Szukamy rozwiązania problemu elementarnego.

Na jego podstawie obliczamy problem wyższego rzędu i kontynuujemy obliczenia, aż do otrzymania rozwiązania problemu wyjściowego.

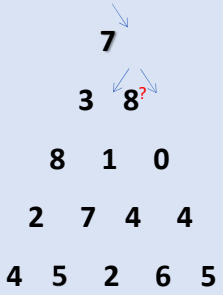
20

```
// Programowanie dynamiczne
// Wyznaczanie wartosci funkcji n!
// Def.: n! = 1 * 2 * 3 * ... * n
#include <iostream>
using namespace std;
long long silnia[100];//silnia[54]=54!
int main()
{
        int n;
        cin >> n;

        silnia[0] = silnia[1] = 1;
        for ( int i = 2; i <= n; i++)
        {
                silnia[i] = i * silnia[i – 1];
        }
        cout << silnia[n] << endl;
        return 0;
}

// Wyznaczanie k-tej liczby Fibbonacciego
```

```
// def.: f(k) = f(k – 1) + f(k – 2)
#include <iostream>
using namespace std;
long long fibb[100000];
int main()
{
    int n;
    cout << "Podaj n: ";
    cin >> n;
    fibb[0] = fibb[1] = 1;
    for ( int i = 2; i < n; i++) {
        fibb[i] = fibb[ i - 1] + fibb[i - 2];
    }
    cout << n << "-ta liczba Fibbonacciego wynosi: "
        << fibb[n - 1] << endl;
    return 0;
}
```


Powróćmy do zadania:
Trójkąt

maksymalna jazda z góry na dół



Do 100 pięter

Ile zgarniemy?

Czy to jest łatwe?

```
// Programowanie dynamiczne
#include <iostream>
using namespace std;
int n;
int tab[100][100] = { 0 };
int search(int tab[][100])
{
  for ( int i = n – 2; i >= 0; i--) // level, pietro
  for ( int j = 0; j <= i; j++) {
   tab[i][j] += (tab[i+1][j] > tab[i+1][j+1] ?
        tab[i+1][j]: tab[i+1][j+1]);
  }
  return tab[0][0];
}
```

```
int main()
{
      cin >> n;
      int I;
      for (I = 0; I < n; I++)
      for (int j = 0; j <= I; j++)
            cin >> tab[i][j];
      cout << search(tab) << endl;
      return 0;
 }
```

23

Analiza algorytmów – złożoność, notacja O()

Analizę stosujemy aby:
Porównać różne algorytmy wykonujące te same zadania
Przewidywać wydajność algorytmu w nowym środowisku
Ustalać wartości *parametrów* algorytmów

Czasy działania algorytmów są proporcjonalne do jednej z poniższych funkcji:

1	czas działania jest stały	
log *n*	logarytmiczny	(zgadywanie liczby)
n	liniowy	(suma min + max)
*n*log*n*	liniowo-logarytmiczny	(qsort)
n^2	kwadratowy, jeśli n podwaja się, to czas wzrasta 4-krotnie (szukanie w tablicy)	
n^k	wielomianowy	
2^n	wykładniczy, jeśli n podwaja się, to czas wzrasta wykładniczo	

24

- **Duże „O”**

Jest to narzędzie matematyczne ułatwiające analizę algorytmu.

Def.

Funkcja $g(n)$ jest rzędu $O(f(n))$, jeśli istnieje taka stała rzeczywista c i liczba naturalna m, że dla każdego $n > m$ zachodzi $g(n) < cf(n)$.

Uwaga:

Stałe c i m często maskują szczegóły implementacyjne często bardzo ważne. Mogą one kryć bardzo duże liczby co obniża wartość algorytmu.

Ex.

Określ złożoność algorytmu znajdującego pierwszy elementu w ciągu. O(1)
Określ złożoność algorytmu znajdującego sumę pierwszego i ostatniego elementu w ciągu. O(n)

Określ złożoność algorytmu znajdującego maksymalny element w ciągu. O(n)

Określ złożoność algorytmu znajdującego maksymalny element w tablicy $n \times n$. O(n2)

Określ złożoność algorytmu znajdującego maksymalny element w tablicy symetrycznej $n \times n$.

Określ złożoność algorytmu sprawdzającego czy w posortowanym ciągu znajduje się dany element.

25

Struktury danych z biblioteki STL

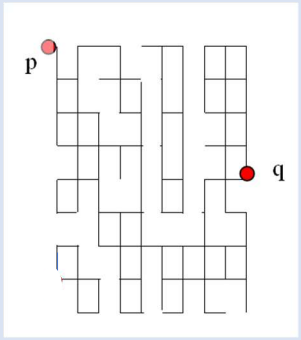
Aby wykorzystać w programie, wybieramy własciwy nagłówek np.:
#include <vector>
using namespace std;

Iteratory – specjalne wskaźniki, które mogą przechodzić po elementach kontenerów.
Deklaracja: kontener::iterator it; np. vector<int>::iterator it;

- Lista
- Tablica/wektor
- Tablica hash – set, map
- Stos
- FIFO
- Drzewa
- Grafy
  - Reprezentacje grafów
  - Przeszukiwanie grafów

26

Problem połączeń



Pytanie: czy w celu prowadzenia komunikacji między komputerami *p* i *q* potrzebne jest nawiązanie nowego połączenia bezpośredniego między nimi, czy też wystarczy skorzystać z połączeń istniejących.

27

Scalanie

```
// Algorytm scalanie
// format pliku: w pierwszej linii: ilosc wierzcholkow,
// w drugiej linii: start i end - punkty miedzy ktorymi
// szukamy polaczenia.
#include <iostream>
using namespace std;
int id[100000];
int main()
{
    int i;
    // Wlasnosc id[i] == id[j] kiedy i oraz j sa polaczone w sieci
    int p,q ;
    int t;
    int n;
    int start, end;
    cin >> n;
    // kolorowanie
    for ( i = 0; i < n; i++)   id[i] = i;
    cin >> start >> end;
    while ( cin >> p >> q )
    {
        t = id[p];
        if ( id[p] == id[q] ) continue;

        for ( i = 0; i < n ; i++)
        if ( id[i] == t )
            id[i] = id[q];
    }
    if ( id[start] == id[end] )
        cout << 1<< endl;
    else
        cout << 0;
    return 0;
}
```

28

Fastscal

```
// Algorytm szybkie scalanie: z uzyciem drzewa
// format pliku: w pierwszej linii: ilosc wierzcholkow,
// w drugiej linii start i end punkty miedzy ktorymi
// szukamy polaczenia.
#include <iostream>
using namespace std;
int id[100000];
// id[i] wskazuje na ojca w budowanym drzewie polaczen.
// Wierzcholki z istniejacym polaczeniem maja wspolny korzen.
int  main()
{
      int i, j;
      int p,q ;
      int t;
      int n;
      int start, end;
      cin >> n;
      for ( i = 0; i < n; i++) id[i] = i;
      cin >> start >> end;
       while ( cin >> p >> q )
       {
            for ( i = p; i != id[i]; i = id[i]);
            for ( j = q; j != id[j]; j = id[j]);
            if ( i == j ) continue;
            id[i] = j;
       }
       for ( i = start; i != id[i]; i = id[i]);
       for ( j = end; j != id[j]; j = id[j]);
       if ( i == j )
            cout << 1 << endl;
       else
            cout << 0 << endl;
        return 0;
}
```

29

Balance tree

```
// Algorytm szybkie scalanie: uzycie drzewa zbalansowanego
// format pliku: w pierwszej linii - ilosc wierzcholkow,
// w drugiej linii start i end punkty miedzy ktorymi
// szukamy polaczenia.
// dodatkowa tablica sz[] dla kazdego korzenia (id[i] == i)
// przechowuje liczbj jego wezlow, po to aby operacja scalania
// polegala na dolaczeniu mniejszego drzewa do wiekszego.
#include <iostream>
using namespace std;
char sz[100000];
int id[100000];
int  main()
{
      int i, j;
      int p,q ;
      int t, n;
      int start, end;
      cin >> n;
      for ( i = 0; i < n; i++) {
            id[i] = i;
            sz[i] = 1;
      }
      cin >> start >> end;
      while ( cin >> p >> q )
       {
            for ( i = p; i != id[i]; i = id[i]);
            for ( j = q; j != id[j]; j = id[j]);
            if ( i == j ) continue;
            // rownowazenie drzewa
            if ( sz[i] < sz[j] )    {
                  id[i] = j;
                  sz[j] += sz[i];
            }
            else     {
                  id[j] = i;
                  sz[i] += sz[j];
            }
       }
      for ( i = start; i != id[i]; i = id[i]);
      for ( j = end; j != id[j]; j = id[j]);
      cout << ( i == j ? 1 : 0) << endl;;
return 0;
}
```

30

Compress

```
// Algorytm szybkie scalanie: uzycie drzewa zbalansowanego
// z kompresja sciezek.
// format pliku: w pierwszej linii - ilosc wierzcholkow,
// w drugiej linii: start i end punkty miedzy ktorymi
// szukamy polaczenia.
//
// Modyfikacja poprzedniego algorytmu.
// Dodatkowo skracamy sciezki o polowe przez co splaszczamy drzewo polaczen.

#include <iostream>
using namespace std;
char sz[100000];
int id[100000];
int  main()
{
    int i, j;
    int p,q ;
    int t, n;
    int start, end;
    cin >> n;
    for ( i = 0; i < n; i++)
    {
        id[i] = i;
        sz[i] = 1;
    }
    cin >> start >> end;
    while ( cin >> p >> q)
    {
        // kompresja sciezek
        for ( i = p; i != id[i]; i = id[i])
            id[i] = id[id[i]];
        for ( j = q; j != id[j]; j = id[j])
            id[j] = id[id[j]];
        if ( i == j ) continue;
        // rownowazenie drzewa
        if ( sz[i] < sz[j] )
        {
            id[i] = j;
            sz[j] += sz[i];
        }
        else
        {
            id[j] = i;
            sz[i] += sz[j];
        }
    }
    for ( i = start; i != id[i]; i = id[i]);
    for ( j = end; j != id[j]; j = id[j]);
    if ( i == j )
        cout << 1 << endl;
    else
        cout << 0 << endl;
    return 0;
}
```