SPRAWOZDANIE - LISTA 3

Zuzanna Pawlik, 282230

09.01.2025

FRAGMENTY KODÓW

CUT ROD

Fragment kodu CUTROD w wersji naiwnej:

```
int CUT_ROD(int p[], int n){
    if(n==0) return 0;

    int q = INT_MIN;
    for(int i = 1; i <= n; i++){
        q = max(q,p[i]+CUT_ROD(p,n-i));
    }
    return q;
}
```

Fragment kodu CUTROD w wersji ze spamiętywaniem:

```
int MEMORIZED_CUT_ROD(int p[], int r[], int s[], int n){
    if(r[n] >= 0) return r[n];
    int q;
    if(n==0){
        q = 0;
    }else{
        q = INT_MIN;
        for(int i = 1; i <= n; i++){
            int t = p[i] + MEMORIZED_CUT_ROD(p,r,s,n-i);
            if(q < t){
                q = t;
                s[n] = i;
            }
        }

    }
    r[n] = q;
    return q;
```

```
}
```

Fragment kodu CUTROD w wersji iteracyjnej:

```
void EXT_BOT_UP_CUT_ROD(int p[], int n, int r[], int s[]){
    r[0] = 0;

    for(int j = 1; j <= n; j++){
        int q = INT_MIN;
        for(int i = 1; i <= j; i++){
            if(q < p[i]+r[j-i]){
                q = p[i]+r[j-i];
                s[j] = i;
            }
        }
        r[j] = q;
    }
}
```

Zarówno wersja z zapamiętywaniem, jak i iteracyjna wykorzystują dodatkowo funkcje wypisujące rozwiązania, w których definiowane są również tablice pomocnicze r i s.

NAJDŁUŻSZY WSPÓLNY PODCIĄG

Fragment algorytmu LCS w wersji rekurencyjnej:

```
int LCS_REK(const string &X, const string &Y, int m, int n, int **b){
    if(m==0 || n==0) return 0;
    if(b[m][n] != -1) return b[m][n]; //juz wpisane
    if(X[m-1] == Y[n-1]) return b[m][n] = 1 + LCS_REK(X,Y,m-1,n-1,b);//sprawdz ostatni
    return b[m][n] = max(LCS_REK(X,Y,m,n-1,b),LCS_REK(X,Y,m-1,n,b));//wynik
}

void LCS_REK_SOLUTION(const string &X, const string &Y){
    int m = X.length();
    int n = Y.length();

    int **b = new int*[m+1];
    for(int i = 0; i<=m; i++){
        b[i] = new int[n+1];
        for(int j = 0; j <= n; j++){
            b[i][j] = -1;
        }
    }
```

```
    int k = LCS_REK(X,Y,m,n,b);

    string p = "";
    int i = m;
    int j = n;
    while(i > 0 && j > 0){
        if(X[i-1] == Y[j-1]){
            p = X[i-1]+p;
            i--;
            j--;
        }else if(b[i-1][j] > b[i][j-1]){
            i--;
        }else{
            j--;
        }
    }
    cout << "dlugosc podciagu: " << k << endl;
    cout << "podciag: " << p << endl;

    delete[] b;

}
```

Fragment algorytmu LCS w wersji iteracyjnej:

```
char** LCS_ITEROWANE(string& X, string& Y, int& d){
    int m = X.length();
    int n = Y.length();

    char** b = new char*[m+1];
    int** c = new int*[m+1];

    for(int i = 0; i <= m; i++){
        b[i] = new char[n+1];
        c[i] = new int[n+1];
    }

    for(int j = 0; j<=m; j++){
        c[j][0] = 0;
    }
    for(int j = 0; j<=n; j++){
        c[0][j] = 0;
    }

    for(int i = 1; i <= m; i++){
        for(int j = 1; j <= n; j++){
            if(X[i-1] == Y[j-1]){
```

```
                c[i][j] = c[i-1][j-1]+1;
                b[i][j] = 's';
            }else if(c[i-1][j] >= c[i][j-1]){
                c[i][j] = c[i-1][j];
                b[i][j] = 'g';
            }else{
                c[i][j] = c[i][j-1];
                b[i][j] = 'l';
            }
        }
    }
    d = c[m][n];
    return b;
}

void PRINT_SOLUTION(char** b, string& X, int i, int j){
    if(i>0 && j>0){
        if(b[i][j] == 's'){
            PRINT_SOLUTION(b,X,i-1,j-1);
            cout << X[i - 1];
        }else if(b[i][j] == 'g'){
            PRINT_SOLUTION(b,X,i-1,j);
        }else{
            PRINT_SOLUTION(b,X,i,j-1);
        }

    }
}
```

Obie wersje pozwalają wyznaczyć najdłuższy wspólny podciąg dwóch napisów, a także długość takiego ciągu. W LCS iterowanym zamiast strzałek użyta jest notacja: g - w górę, l - w lewo oraz s - na skos. Oba algorytmy wykorzystują dodatkowe funkcje pozwalające na wypisanie znalezionego ciągu.

WYBÓR ZAJĘĆ

0.0.1 REKURENCJA

Algorytm rekurencyjny:

```
void RECURSIVE_ACTIVITY_SELECTOR(int s[], int f[], int k, int n, int wyniki[], int& ilosc){
    int m = k+1;
    while(m<=n && s[m] < f[k]){
        m++;
    }

    if(m<=n){
```

```
        wyniki[ilosc++] = m;//dodajemy do zajęć
        RECURSIVE_ACTIVITY_SELECTOR(s,f,m,n,wyniki,ilosc);
    }
}

void PRINT_RAS(int s[], int f[], int n){
    int* wyniki = new int[n];
    int ilosc = 0;

    RECURSIVE_ACTIVITY_SELECTOR(s,f,0,n,wyniki,ilosc);
    cout << "rekurencja:" <<endl;
    for(int i = 0; i < ilosc; i++){
        cout << wyniki[i] << " ";

    }
    cout<<endl;


}
```

Modyfikacja:

```
void RAS_MOD(Zajecia lista_zajec[], int k, int n, int wyniki[], int& ilosc) {
    int m = k + 1;
    while (m <= n && lista_zajec[m].poczatek < lista_zajec[k].koniec){
        m++;
    }

    if (m <= n){
        wyniki[ilosc++] = lista_zajec[m].index;
        RAS_MOD(lista_zajec, m, n, wyniki, ilosc);
    }
}

void PRINT_RAS_MOD(Zajecia lista_zajec[], int n){
    int wyniki[n];
    int ilosc = 0;

    RAS_MOD(lista_zajec, 0, n, wyniki, ilosc);
    cout << "rekurencja modyfikacja:" <<endl;
    for(int i = 0; i < ilosc; i++){
        cout << wyniki[i] << " ";

    }
    cout<<endl;
}
```

ITERACJA

Algorytm iteracyjny:

```
void ACTIVITY_SELECTOR(int s[], int f[], int n, int wyniki[], int& ilosc){
    int k = 1;//poczatek
    wyniki[ilosc++] = 1;//pierwsze zajecia

    for(int m = 2; m <= n; m++){
        if(s[m] >= f[k]){
            wyniki[ilosc++] = m;
            k = m;
        }
    }

}

void PRINT_AS(int s[], int f[], int n){
    int wyniki[n];
    int ilosc = 0;

    ACTIVITY_SELECTOR(s,f,n,wyniki,ilosc);

    cout << "iteracyjna:"<<endl;
    for(int i = 0; i<ilosc; i++){
        cout << wyniki[i] << " ";

    }
    cout << endl;
}
```

Modyfikacja:

```
void AS_MOD(Zajecia lista_zajec[], int n, int wyniki[], int& ilosc){
    wyniki[ilosc++] = lista_zajec[1].index;
    int k = 1;

    for(int m = 2; m<=n; m++){
        if(lista_zajec[m].poczatek >= lista_zajec[k].koniec){
            wyniki[ilosc++] = lista_zajec[m].index;
            k = m;
        }
    }
}

void PRINT_AS_MOD(Zajecia lista_zajec[], int n){
    int wyniki[n];
```

```
    int ilosc = 0;

    AS_MOD(lista_zajec,n,wyniki,ilosc);

    cout << "iteracja modyfikacja:" <<endl;
    for(int i = 0; i < ilosc; i++){
        cout << wyniki[i] << " ";

    }
    cout<<endl;
}
```

Modyfikacje powyższych algorytmów wykorzystują strukturę *Zajecia*, dla której określamy *pocztek* - czas rozpoczęcia, *koniec* - czas zakończenia oraz *index* - początkową pozycję zajęć. Następnie takie zajęcia sortowane są po czasie rozpoczęcia i dopiero następuje wybór optymalnej opcji.

0.0.2 PROGRAMOWANIE DYNAMICZNE

Wersja algorytmu wykorzystująca programowanie dynamiczne:

```
void AS_DYNAM(int s[], int f[], int n){
    int max_zajec[n];
    int poprzedni[n];

    for (int i = 0; i < n; i++){
        max_zajec[i] = 1;
        poprzedni[i] = -1;
    }

    for (int i = 1; i < n; i++){
        for (int j = 0; j < i; j++){
            if (f[j] <= s[i]){
                if (max_zajec[j] + 1 > max_zajec[i]){//sprawdzenie czy wiecej zajec
                    max_zajec[i] = max_zajec[j] + 1;
                    poprzedni[i] = j;
                }
            }
        }
    }


    int index_max = 0;
    for (int i = 1; i < n; i++){
        if (max_zajec[i] > max_zajec[index_max]){//wybor maksymalnego zestawu
            index_max = i;
        }
```

```
    }


    int wynik[n];
    int ilosc = 0;

    while (index_max != -1){
        wynik[ilosc++] = index_max + 1;
        index_max = poprzedni[index_max];
    }


    cout << "prog. dynamiczne: ";
    for (int i = ilosc - 1; i >= 0; i--){
        cout << wynik[i] << " ";
    }
    cout << endl;
}
```

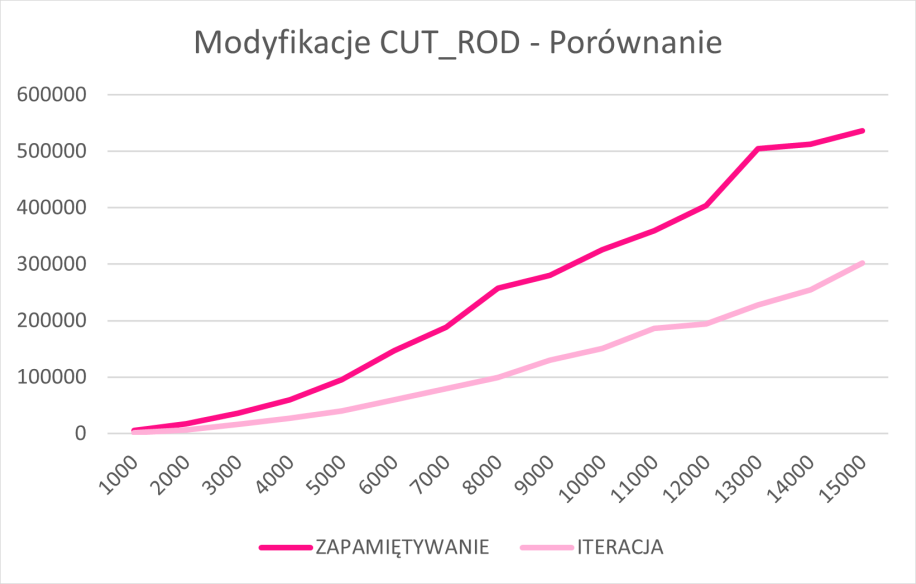
Kod ten przy pomocy tabel max_zajec oraz $poprzedni$ pozwala znaleźć optymalne rozwiązanie dla tego problemu. W podwójnej pętli for sprawdzane są zajęcia i względem wcześniejszych zajęć j. Ostatecznie wyznaczany jest $index_max$, czyli indeks, dla którego $max_zajec[index_max]$ jest największe, a zatem optymalne rozwiązanie.

TABELE I WYKRESY

PORÓWNANIE ALGORYTMÓW CUTROD

	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12000	13000	14000	15000
ZAPAMIĘTYWANIE	5030	17118	36120	59285	95481	146415	187983	257870	279777	325597	359696	403261	504312	512121	536420
ITERACJA	1560	6012	16064	26996	39499	59695	79947	99052	129553	150833	186455	194484	228129	254398	302190

	10	12	14	16	18	20	22	24	26	28	30
CUT_ROD	29	170	562	1234	3896	23638	55216	198697	756214	1993528	7199717

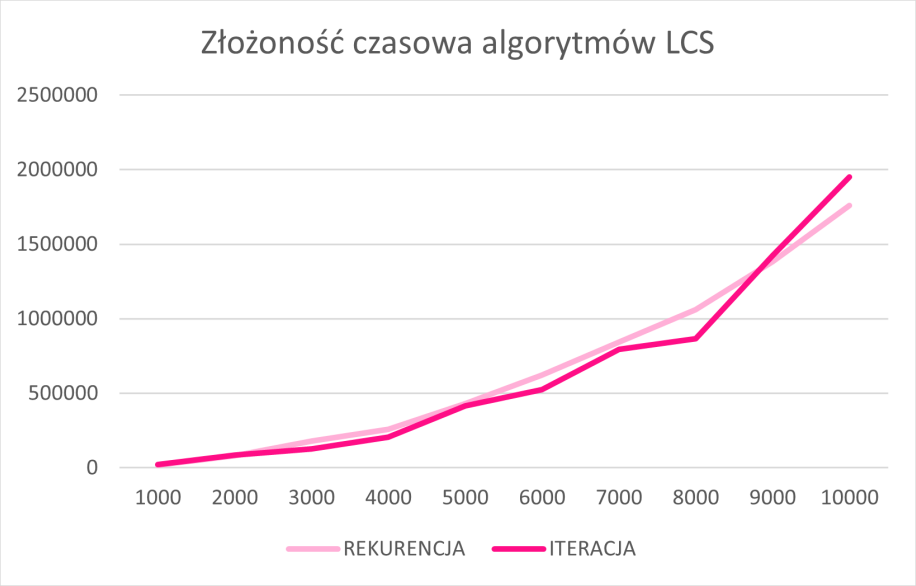


WNIOSKI

Powyższe porównanie pozwala stwierdzić, że najbardziej wydajną wersją tego algorytmu jest ta, wykorzystująca iterację. Jest ona zauważalnie szybsza od algorytmu rekurencyjnego, choć oba wykonują się raczej sprawnie w porównaniu z wersją naiwną. Niestety, dokładne zaobserwowanie tej różnicy okazało się niemożliwe, ponieważ zwykły CUTROD już na małych danych wykonywał się znacznie dłużej od pozostałych dwóch. Dla tak małych długości pomiar "lepszych" wersji algorytmu okazał się po prostu niemożliwy, natomiast warunki sprzętowe nie pozwoliły na próbę sprawdzenia algorytmu naiwnego na dużych danych.

PORÓWNANIE ALGORYTMÓW LCS

	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
REKURENCJA	18674	83482	178398	260061	430131	620945	844098	1062337	1385425	1758410
ITERACJA	24008	84979	128178	204654	414533	524642	795394	865568	1423717	1950303

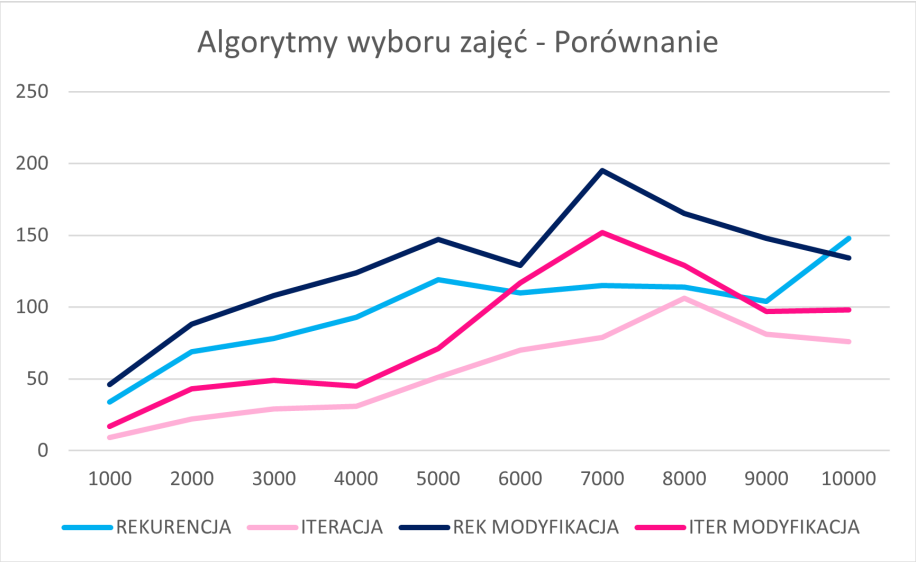


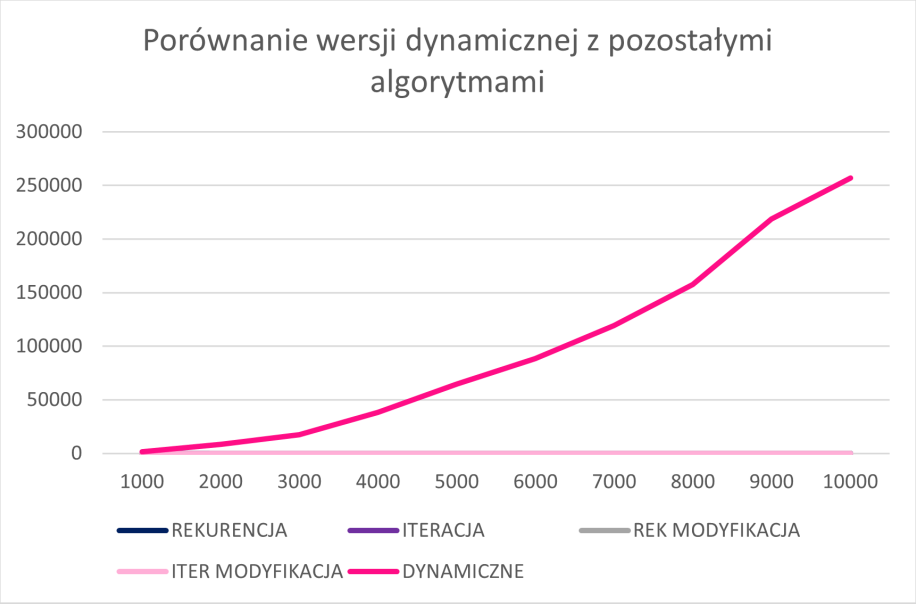
WNIOSKI

Oba algorytmy wykonywały się w bardzo podobnym czasie, bez konkretnego wskazania na przewagę jednego z nich. W większości przypadków wersja z iteracją była szybsza, ale różnice te nie były szczególnie zauważalne, a przewaga ta była niezbyt istotna.

PORÓWNANIE ALGORYTMÓW WYBORU ZAJĘĆ

	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
REKURENCJA	34	69	78	93	119	110	115	114	104	148
ITERACJA	9	22	29	31	51	70	79	106	81	76
REK MODYFIKACJA	46	88	108	124	147	129	195	165	148	134
ITER MODYFIKACJA	17	43	49	45	71	117	152	129	97	98
DYNAMICZNE	1496	8455	17226	38184	64847	88694	119487	157681	218805	256791





WNIOSKI

Oba podstawowe algorytmy, a także ich modyfikacje wykonywały się bardzo szybko, co spowodowało wahania w wynikach. Nie można zatem zaobserwować szczególnego wydłużenia się czasu wykonywania wraz ze wzrostem długości listy. Nieznacznie szybsza od pozostałych okazała się oryginalna wesja iteracyjna. Porównanie natomiast można wykonać z wersją dynamiczną, która okazała się o

wiele bardziej złożona, a czas jej wykonania wyraźnie wzrastał wraz ze wzrostem długości list.