Analiza Algorytmów

Ksawery Józefowski
Politechnika Wrocławska

January 15, 2025

Contents

1 Wstep

W ramach analizy zaimplementowano i przetestowano trzy algorytmy:

- Algorytm znajdowania najdłuższego wspólnego podciagu (LCS),
- Algorytm wyboru aktywności,
- Algorytm ciecia preta.

Każdy algorytm został zaimplementowany w różnych wersjach: rekurencyjnej, iteracyjnej i dynamicznej. Celem analizy było porównanie ich efektywności i złożoności czasowej.

2 Fragmenty kodów

2.1 LCS

Algorytm LCS (Longest Common Subsequence) pozwala znaleźć najdłuższy wspólny podciag dwóch sekwencji znaków. Jest używany w takich dziedzinach jak bioinformatyka czy porównywanie tekstów. Poniżej przedstawiono implementacje rekurencyjna tego algorytmu:

```
int LCS_Rec(char X[], char Y[], int m, int n, int c[MAX_M +
    1][MAX_N + 1]) {
    if (m == 0 || n == 0) return 0;
    if (X[m - 1] == Y[n - 1]) {
        comparisons++;
        assignments++;
        c[m][n] = 1 + LCS_Rec(X, Y, m - 1, n - 1, c,
            assignments, comparisons);
    } else {
        comparisons++;
        assignments++;
        c[m][n] = max(LCS_Rec(X, Y, m - 1, n, c, assignments,
            comparisons),
                      LCS_Rec(X, Y, m, n - 1, c,
                          assignments, comparisons));
    }
    return c[m][n];
}
```

2.2 Algorytm wyboru aktywności

Algorytm wyboru aktywności znajduje maksymalny zbiór niepokrywajacych sie czasowo aktywności. Jest to problem klasyczny w optymalizacji, szczególnie

w harmonogramowaniu zadań. Poniżej przedstawiono implementacje dynamiczna tego algorytmu:

```
void dynamicActivitySelector(Activity activities[], int n,
    int selected[], int& selectedCount) {
    int dp[n];
    std::sort(activities, activities + n, compare);
    dp[0] = 1;
    for (int i = 1; i < n; ++i) {
        dp[i] = 1;
        for (int j = i - 1; j >= 0; --j) {
            if (activities[i].start >= activities[j].finish)
                {
                dp[i] = std::max(dp[i], dp[j] + 1);
            }
        }
    }
}
```

2.3 Algorytm ciecia preta

Algorytm ciecia preta (Rod Cutting) rozwiazuje problem maksymalizacji zysku przy cieciu preta na kawałki o określonych długościach. Jest to klasyczny problem dynamiczny, stosowany w optymalizacji zasobów. Poniżej przedstawiono wersje Extended Bottom Up:

```
1 void Bottom_Up_Cut_Rod(const int p[], int* r, int* s, int n)
    {
2     r[0] = 0;
3     for (int j = 1; j <= n; j++) {
4         int q = -1;
5         for (int i = 1; i <= j; i++) {
6             if (q < p[i] + r[j - i]) {
7                 q = p[i] + r[j - i];
8                 s[j] = i;
9             }
10        }
11        r[j] = q;
12    }
13 }
```

3 Analiza i wyniki

Porównano algorytmy pod wzgledem liczby operacji i czasu wykonania dla różnych rozmiarów danych wejściowych.

3.1 Tabele wyników

3.1.1 LCS

Rozmiar danych	Algorytm	Porównania	Przypisania	Czas (ms)
1	LCS (rekurencyjny)	2	2	920 ns
1	LCS (iteracyjny)	7	10	1210 ns
10	LCS (rekurencyjny)	564	564	4700 ns
10	LCS (iteracyjny)	140	100	1500 ns
100	LCS (rekurencyjny)	7919	7919	254640 ns
100	LCS (iteracyjny)	1291	847	5080 ns

Table 1

3.1.2 Cut Rod

Rozmiar danych	Algorytm	Porównania	Przypisania	Czas (ms)
1	Cut Rod	1	1	50 ns
1	MemorizedCutRod	1	4	570 ns
1	Bottom$_U pCutRod$	1	4	180 ns
10	Cut Rod	1023	1023	6850 ns
10	MemorizedCutRod	100	76	1080 ns
10	Bottom$_U pCutRod$	55	31	360 ns
20	Cut Rod	1048575	1048575	7116060 ns
20	MemorizedCutRod	400	251	2190 ns
20	Bottom$_U pCutRod$	210	61	820 ns

Table 2

3.1.3 Activity Selector

Rozmiar danych	Algorytm	Porównania	Przypisania	Czas (ms)
1	ActivitySelector (rekurencyjny)	1	4	1200 ns
1	ActivitySelector (iteracyjny)	2	5	33 ns
1	ActivitySelector (dynamiczny)	3	11	300 ns
10	ActivitySelector (rekurencyjny)	11	20	933 ns
10	ActivitySelector (iteracyjny)	15	14	133 ns
10	ActivitySelector (dynamiczny)	78	64	900 ns
100	ActivitySelector (rekurencyjny)	100	125	1366 ns
100	ActivitySelector (iteracyjny)	112	38	400 ns
100	ActivitySelector (dynamiczny)	5151	3371	34333 ns

Table 3

3.2 Wykresy Wyników

3.2.1 Activity Selector

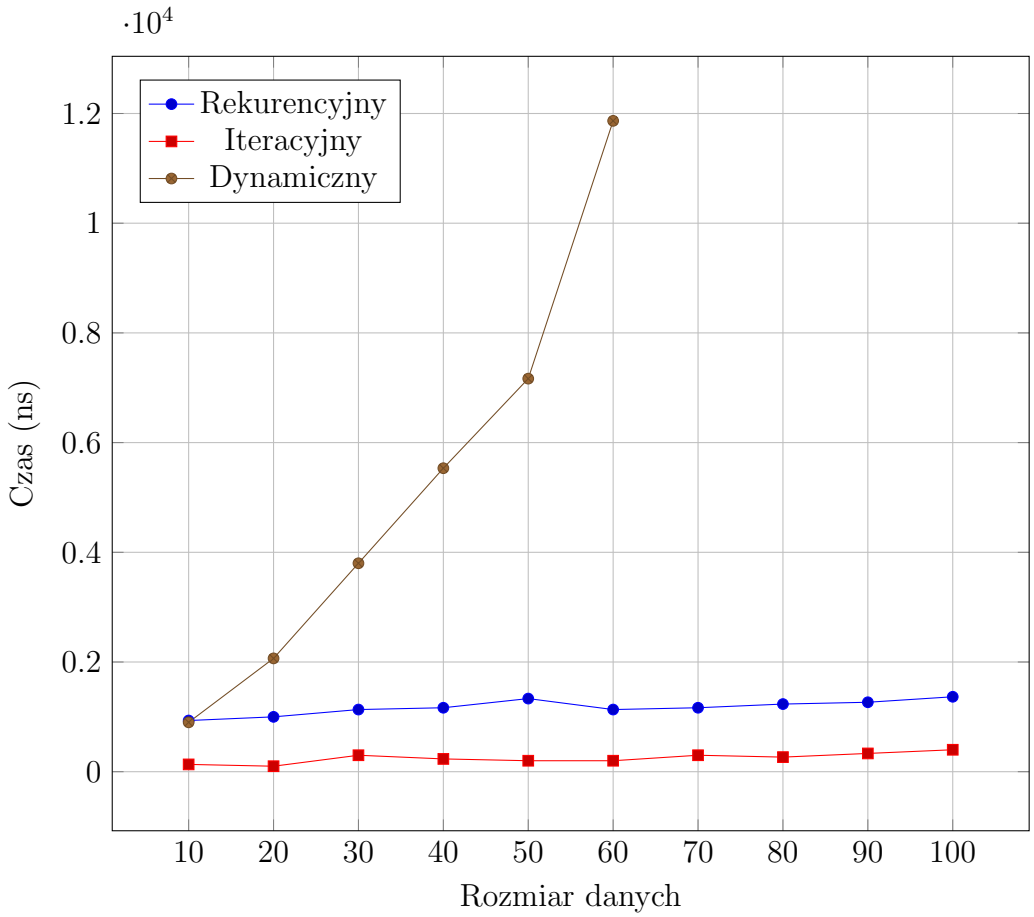

Figure 1: Porównanie czasu wykonania algorytmów

3.2.2 Cut Rod

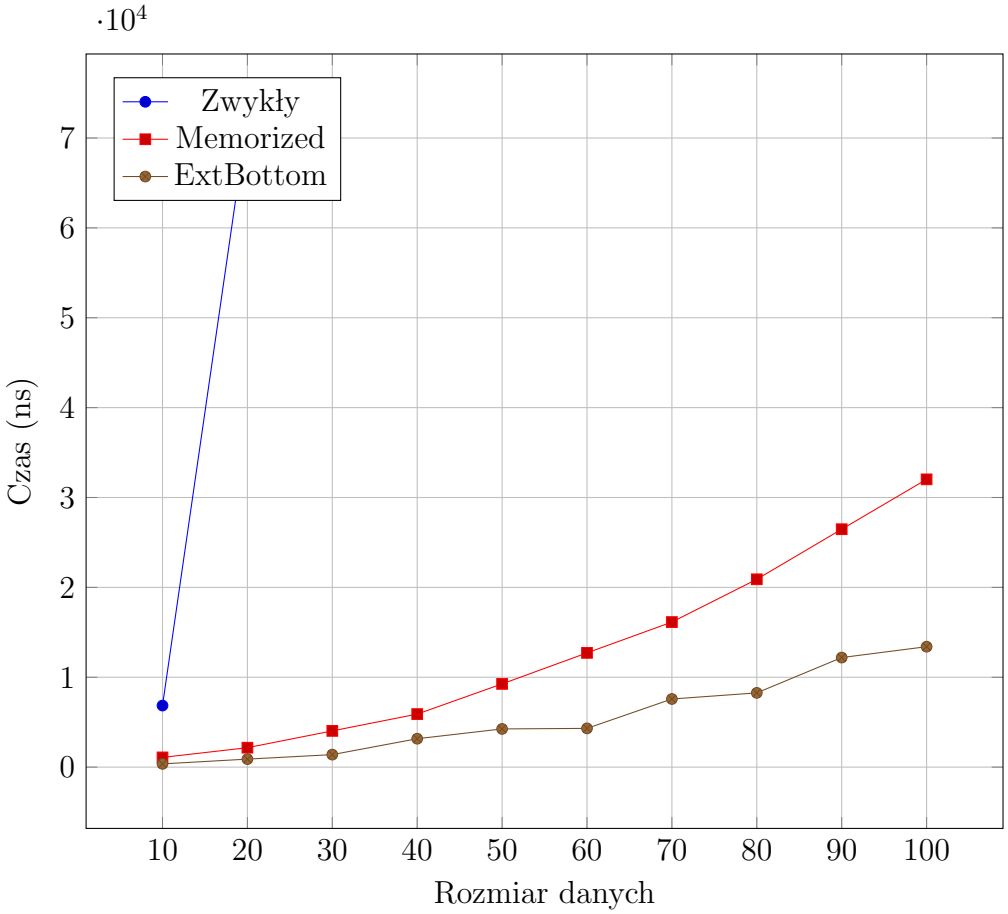

Figure 2: Porównanie czasu wykonania algorytmów

3.2.3 LCS

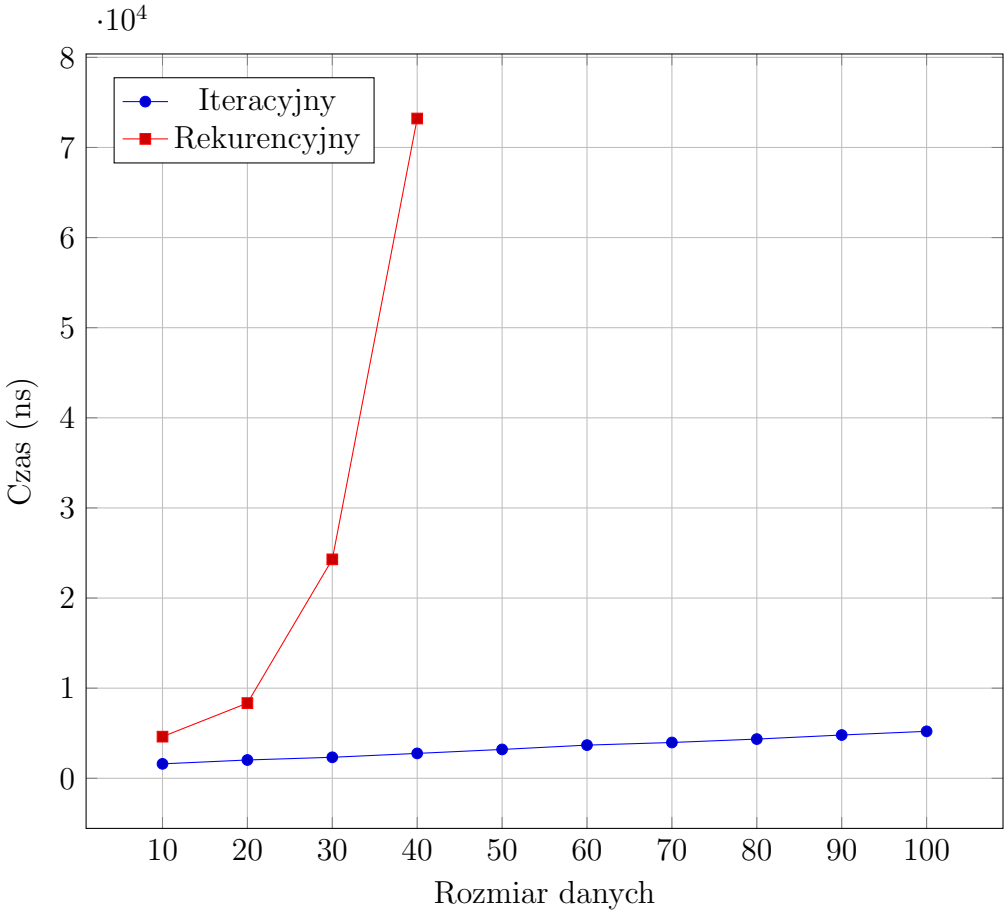

Figure 3: Porównanie czasu wykonania algorytmów

4 Wnioski

Na podstawie przedstawionych wyników i wykresów dotyczacych trzech algorytmów: LCS, Cut Rod oraz Activity Selector, można wyciagnać kilka istotnych wniosków:

- **Algorytmy rekurencyjne vs. iteracyjne:**
  - W przypadku algorytmu **LCS**, algorytm rekurencyjny wykazuje znaczaco dłuższy czas wykonania niż algorytm iteracyjny. Wynika to z braku optymalizacji w wersji rekurencyjnej, która może prowadzić do nadmiernych obliczeń dla wiekszych danych.
  - W **Cut Rod**, wersja z pamiecia (MemorizedCutRod) jest wyraźnie szybsza od klasycznego algorytmu (Cut Rod), co wskazuje na korzyści płynace z zapamietywania wyników podproblemów i unikania ich wielokrotnego obliczania.
- **Algorytmy dynamiczne:**
  - W przypadku algorytmu **Activity Selector**, wersja dynamiczna ma tendencje do wydłużania czasu obliczeń w porównaniu do wersji iteracyjnej, szczególnie dla wiekszych rozmiarów danych. Mimo to, algorytmy dynamiczne sa skuteczne w rozwiazywaniu problemów o wiekszej złożoności.
  - W **Activity Selector**, wersja iteracyjna jest zdecydowanie najszybsza, co może wynikać z prostszej implementacji i mniejszej liczby operacji wymaganych do rozwiazania problemu.
- **Zależność czasu wykonania od rozmiaru danych:**
  - W przypadku **LCS** i **Cut Rod**, czas wykonania algorytmów wzrasta w miare zwiekszania rozmiaru danych, co jest zgodne z oczekiwaniami dotyczacymi algorytmów o złożoności czasowej zależnej od rozmiaru wejścia.
  - Dla algorytmu **Activity Selector**, wersja dynamiczna charakteryzuje sie znacznie wiekszym czasem wykonania w porównaniu do wersji rekurencyjnej i iteracyjnej, szczególnie przy wiekszych rozmiarach danych, co sugeruje wieksza złożoność obliczeniowa tego podejścia.