Analiza i porównanie algorytmów CUT_ROD, LCS i ACTIVITY_SELECTOR

Tomasz Warzecha

February 1, 2025

1 Wstęp

W sprawozdaniu moim celem jest analiza i porównanie wydajności trzech algorytmów: CUT_ROD, LCS oraz ACTIVITY_SELECTOR. Algorytmy te zostały zaimplementowane w różnych wersjach: rekurencyjnej, ze spamiętywaniem oraz iteracyjnej. Dodatkowo w ACTIVITY_SELECTOR został zmodyfikowany w taki sposób aby działał na danych wejściowych posortowanych względem czasu rozpoczęcia, nie zakończenia. Niektóre z algorytmów również posiadają funkcje odzyskiwania rozwiązań. Badania skupiają się na porównaniu teoretycznych złożoności czasowych oraz praktycznych czasów wykonania dla różnych rozmiarów danych.

2 Implementacja algorytmów

Algorytmy zostały zaimplementowane zgodnie z klasycznymi wersjami, z uwzględnieniem modyfikacji wskazanych w poleceniach.

Najciekawsze fragmenty kodu

Poniżej przedstawię najciekawsze fragmenty kodu, wraz z objaśnieniami, tak aby łatwiej zrozumieć działanie algorytmów.

2.1 *CUT_ROD*

W algorytmie CUT_ROD wersja ze spamiętywaniem wykorzystuje tablicę do przechowywania wyników pośrednich, co znacznie redukuje złożoność czasową. Poniżej przedstawiono kluczowy fragment kodu, który wraz z fragmentem zapisującym wynik dla danego n w naszej tablicy tworzy główne usprawnienie naiwnego algorytmu:

```
if (R[n-1]>=0){
    return R[n-1];
}
```

Ten fragment sprawdza, czy wynik dla danego ‘n‘ został już obliczony i zapisany w tablicy ‘R‘. Jeśli tak, zwraca zapisaną wartość, unikając ponownego obliczania.

2.2 *LCS*

W algorytmie LCS wersja iteracyjna wykorzystuje dwie tablice do przechowywania długości najdłuższego wspólnego podciągu oraz kierunku, w którym należy się poruszać, aby odtworzyć rozwiązanie. Poniżej znajduje się kluczowy fragment kodu:

```
if (X[i-1] == Y[j-1]) {
    C[i][j] = C[i-1][j-1] + 1;
    B[i][j] = '-';
}
```

Ten fragment sprawdza, czy znaki w obu ciągach są takie same. Jeśli tak, zwiększa długość najdłuższego wspólnego podciągu i zapisuje kierunek w tablicy ‘B‘.

2.3 *ACTIVITY_SELECTOR*

W algorytmie ACTIVITY_SELECTOR wersja dynamiczna wykorzystuje dwie tablice do przechowywania liczby zajęć oraz indeksów zajęć, które należy wybrać. Poniżej przedstawiono kluczowy fragment kodu:

```
while (f[i] < f[k]) {
    if (f[i] <= s[k] && f[k] <= s[j] && c[i][k]+c[k][j]+1> c[i][j])
        {
        c[i][j] = c[i][k]+c[k][j]+1;
        act[i][j] = k;
    }
    k--;
}
```

Ten fragment sprawdza, czy zajęcia się nie nakładają i czy wybór danego zajęcia zwiększa liczbę możliwych do wybrania zajęć. Jeśli tak, aktualizuje tablice ‘c‘ i ‘act‘, tak aby zawierała aktualnie najwiekszą ilość zadań między dwoma indeksami.

3 Porównanie algorytmów

3.1 Porównanie teoretyczne

Poniżej zestawiono złożoności czasowe oraz pamięciowe dla różnych wersji algorytmów:

Algorytm	**Rekurencyjna**	**Spamiętywanie**	**Iteracyjna**
CUT_ROD	$O(2^n)$	$O(n^2)$	$O(n^2)$
LCS	$O(2^{m+n})$	$O(mn)$	$O(mn)$
ACTIVITY_SELECTOR	$O(n)$	-	$O(n)$

Table 1: Porównanie teoretyczne algorytmów

3.2 Porównanie czasów działania

Zaprezentowano wyniki pomiarów na wykresach:

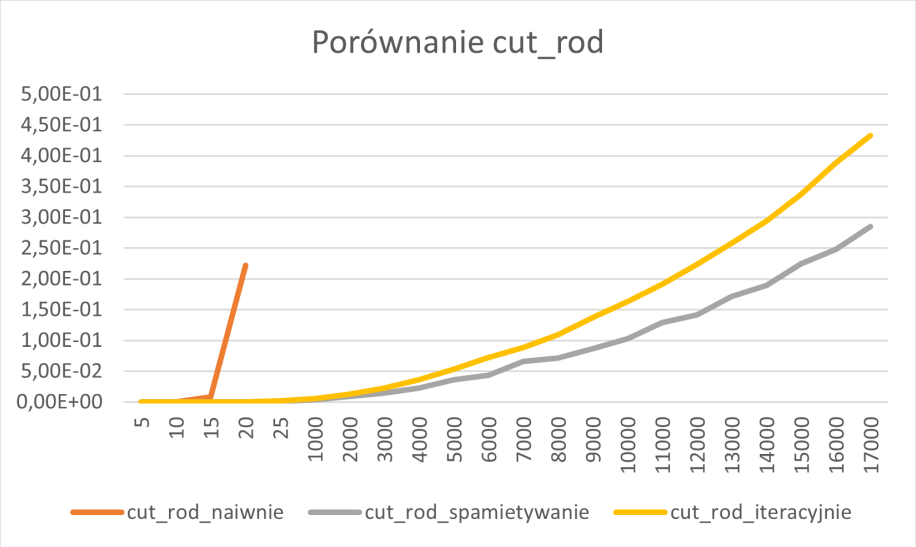

Figure 1: Porównanie czasów działania *CUT_ROD* w różnych wersjach.

Łatwo możemy zauważyć na wykresie, że naiwna wersja CUT_ROD jest bardzo czasochłonna. Jest to spowodowane wykładniczą złożonością czasową, podczas gdy wersje ze spamiętywaniem i iteracyjna mają złożoność kwadratową. Zatem dwie ostatnie wersje są znacząco szybsze i podobne czasowo do siebie.

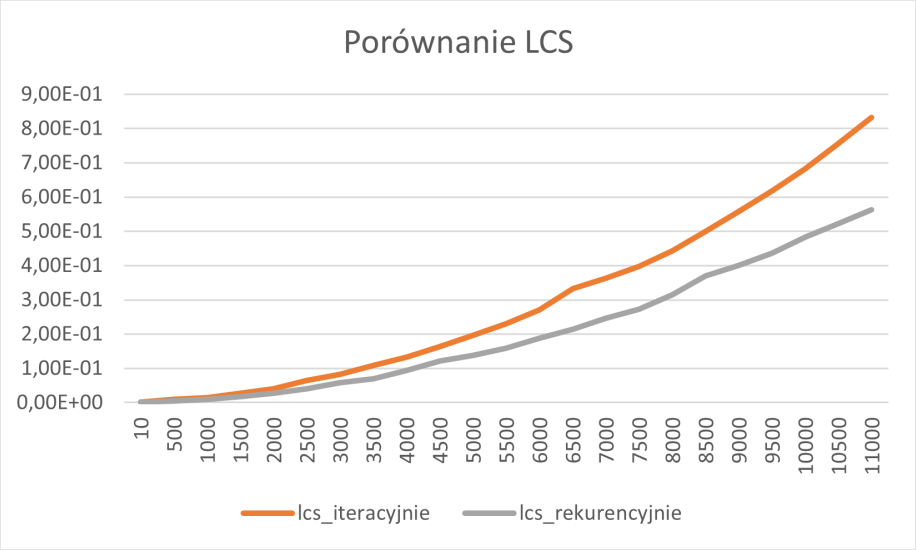

Figure 2: Porównanie czasów działania *LCS* w różnych wersjach.

Możemy zauważyć że czas wykonania lcs w wersji iteracyjnej i rekurencyjnej jest podobny, różnice są niewielkie. Jest to spowodowane taką samą złożonością algorytmów - n^2.

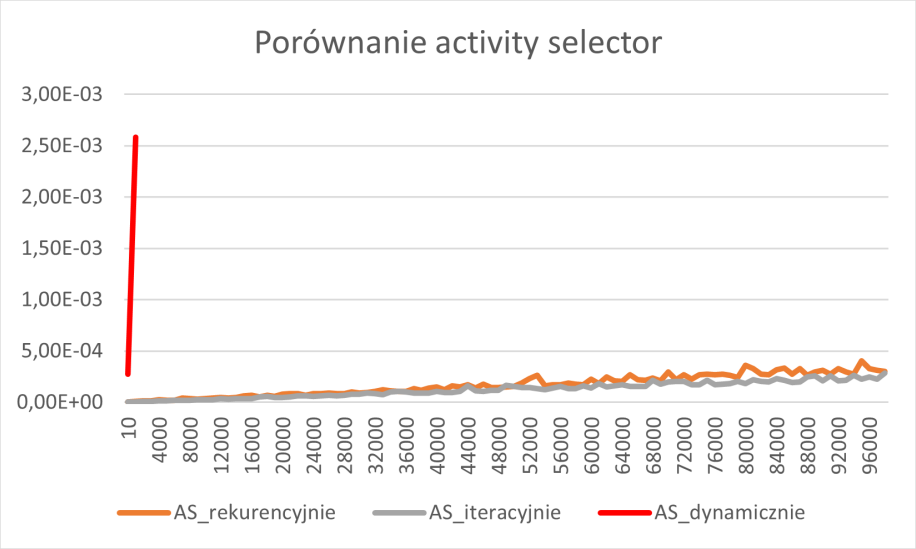

Figure 3: Porównanie czasów działania *ACTIVITY_SELECTOR* w różnych wersjach.

Jak możemy zauważyć, dynamiczna wersja AS ma dużo większą złożoność czasową, ponieważ aż n^3, co od razu widać na rysunku. Wersje rekurencyjna i iteracyjna mają złożoność liniową i nieznacznie się od siebie różnią.

4 Porównanie modyfikacji

4.1 *ACTIVITY_SELECTOR*

Zmodyfikowana wersja algorytmu ACTIVITY_SELECTOR działa na danych posortowanych względem czasu rozpoczęcia. Poniżej przedstawiono wyniki porównania czasów działania:

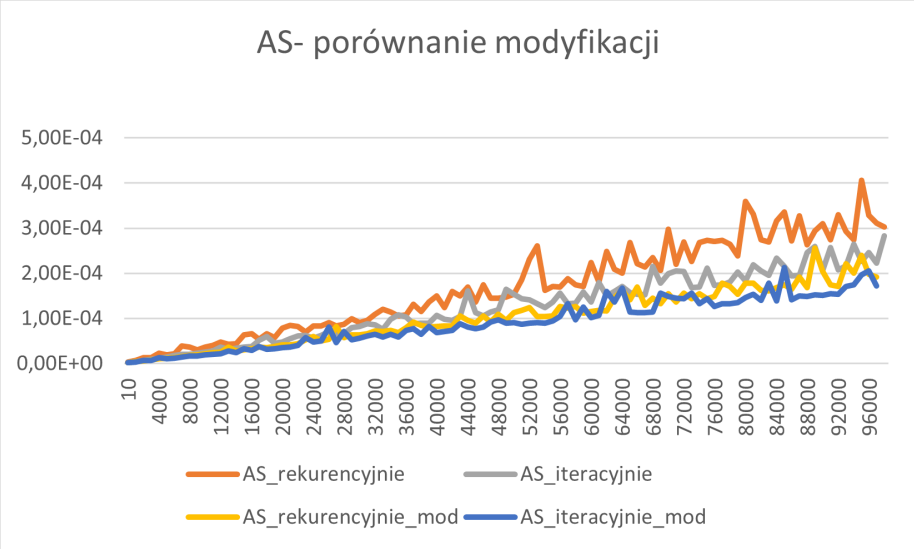

Figure 4: Porównanie czasów działania *ACTIVITY_SELECTOR* i jego modyfikacji.

Modyfikacja algorytmu polegała na otrzymywaniu danych wejściowych posortowanych względem czasu rozpoczęcia zamiast zakończenia. Na wykresie możemy zobaczyć, że niezależnie od tego czy mamy posortowane czasy rozpoczęcia, czy zakończenia czas działania pozostaje na podobnym poziomie, delikatne różnice mogą wynikać z różnego obciążenia komputera podczas testów.

5 Wnioski

Przeprowadzone testy pokazują, że wersje algorytmów ze spamiętywaniem oraz iteracyjne są w większości przypadków znacznie bardziej wydajne niż ich rekurencyjne odpowiedniki. Jedynie w przypadku ACTIVITY_SELECTOR wersja rekurencyjna jest na tym samym poziomie, jednak wynika to z istoty problemu. Modyfikacje algorytmów nie wpływają znacząco na poprawę wydajności oraz powodują większe skomplikowanie kodu, zatem nie warto z nich korzystać.