Analiza porównawcza algorytmów optymalizacyjnych: Programowanie Dynamiczne vs Zachłanne

Sprawozdanie z laboratorium

13 stycznia 2026

Streszczenie

Niniejsze sprawozdanie przedstawia analizę wydajności i poprawności implementacji wybranych algorytmów: cięcia pręta (Cut Rod), najdłuższego wspólnego podciągu (LCS), problemu wyboru zajęć (Activity Selection) oraz kodowania Huffmana (wariant binarny i ternarny). Badania przeprowadzono dla danych losowych o rozmiarach N dochodzących do 30 000 elementów. Kluczowym elementem analizy jest porównanie podejścia dynamicznego z zachłannym, weryfikacja narzutu czasowego rekurencji oraz analiza strukturalna drzew Huffmana.

Spis treści

1 Wstęp i cel ćwiczenia

Celem laboratorium było zaimplementowanie i przetestowanie klasycznych problemów algorytmicznych. Szczególny nacisk położono na:

1. Porównanie złożoności teoretycznej z rzeczywistym czasem wykonania.
2. Analizę "kosztu"memoizacji w podejściu rekurencyjnym.
3. Weryfikację skuteczności algorytmów zachłannych w problemie Activity Selection.
4. Porównanie efektywności i struktury kodowania Huffmana w wariancie binarnym i trójkowym (ternarnym).

2 Analiza Implementacji

2.1 Activity Selector - Modyfikacja (Start Time)

Standardowy algorytm zachłanny sortuje zajęcia po czasie zakończenia (f_i). W ramach ćwiczenia zaimplementowano wariant sortujący po czasie rozpoczęcia (s_i). Algorytm ten działa analogicznie, lecz iteruje tablicę "od tyłu", wybierając zadania, których koniec jest wcześniejszy niż start poprzednio wybranego.

```
vector<Activity> activity_start_sorted(vector<Activity> acts) {
    // 1. Sortowanie po czasie STARTU
    sort(acts.begin(), acts.end(), [](const Activity& a, const Activity&
    b){
        return a.s < b.s;
    });

    vector<Activity> res;
    int n = acts.size();

    // 2. Wybor ostatniego elementu (najpozniejszy start)
    Activity last_selected = acts[n-1];
    res.push_back(last_selected);

    // 3. Iteracja wsteczna
    for (int i = n - 2; i >= 0; i--) {
        if (acts[i].f <= last_selected.s) { // Warunek nienachodzenia
            res.push_back(acts[i]);
            last_selected = acts[i];
        }
    }
    // 4. Przywrocenie kolejnosci chronologicznej
    reverse(res.begin(), res.end());
    return res;
}
```

Listing 1: Modyfikacja Activity Selector (Sortowanie po starcie)

2.2 Cut Rod - Iteracja z rekonstrukcją

Zamiast rekurencji, zastosowano podejście iteracyjne (Bottom-Up), które eliminuje ryzyko przepełnienia stosu. Dodatkowa tablica $s[j]$ przechowuje miejsce cięcia, co pozwala na odtworzenie pełnego rozwiązania.

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

Listing 2: Cut Rod Iterative

2.3 Kodowanie Huffmana: Binarne vs Ternarne

W ramach laboratorium zaimplementowano dwa warianty algorytmu Huffmana: klasyczny (binarny) oraz jego uogólnienie dla drzewa trójkowego (ternarny). Oba algorytmy wykorzystują kolejkę priorytetową (Min-Heap).

2.3.1 Różnice w redukcji kolejki i padding

- **Wariant Binarny ($k = 2$):** W każdym kroku pobierane są 2 najmniejsze węzły, a wstawiany jest 1 nowy. Liczba elementów maleje o 1.
- **Wariant Ternarny ($k = 3$):** Pobierane są 3 węzły, wstawiany 1. Liczba elementów maleje o 2.

Aby w wariancie ternarnym możliwe było poprawne zakończenie algorytmu (redukcja do 1 korzenia), początkowa liczba węzłów N musi być nieparzysta ($N \equiv 1 \pmod 2$). Jeśli N jest parzyste, dodajemy sztuczny węzeł o wadze 0:

```
// Jesli Q.size() jest parzyste, na koncu zostalyby 2 elementy.
// Dodajemy pusty wezel, aby umozliwic grupowanie po 3.
if (Q.size() % 2 == 0) {
    Q.push(new HuffNode(0, '\0'));
}

while (Q.size() > 1) {
    HuffNode *x = Q.top(); Q.pop();
    HuffNode *y = Q.top(); Q.pop();
    HuffNode *z = Q.top(); Q.pop();
    // ... laczenie x, y, z w wezel rodzica ...
}
```

Listing 3: Huffman Ternary - Obsługa parzystości

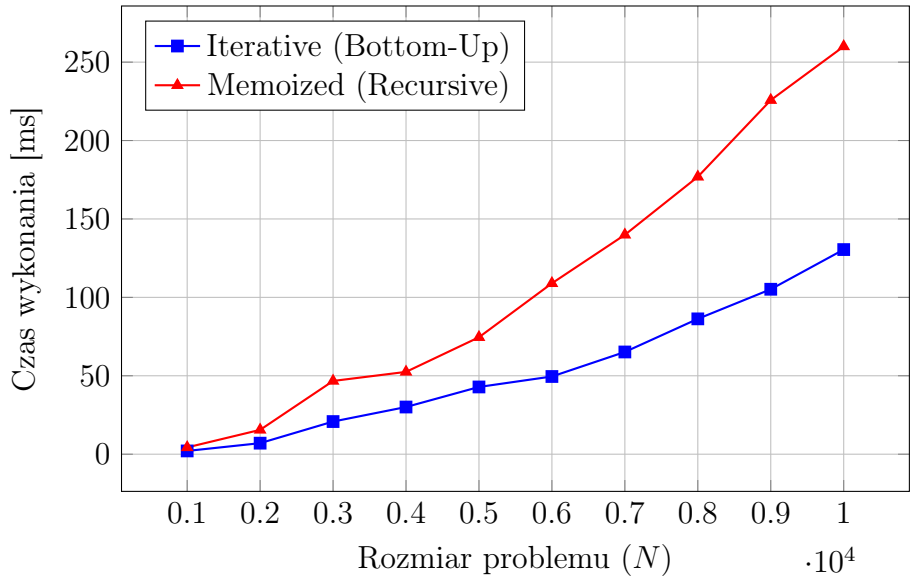
3 Wyniki eksperymentów

Poniższe wykresy wygenerowano na podstawie danych z pliku `benchmark_results.csv`.

3.1 Cut Rod: Koszt rekurencji

Porównanie wersji iteracyjnej i zmemoizowanej rekurencji dla $N \in [1000, 10000]$.

Wykres 1: Cut Rod - Iteracja vs Memoizacja

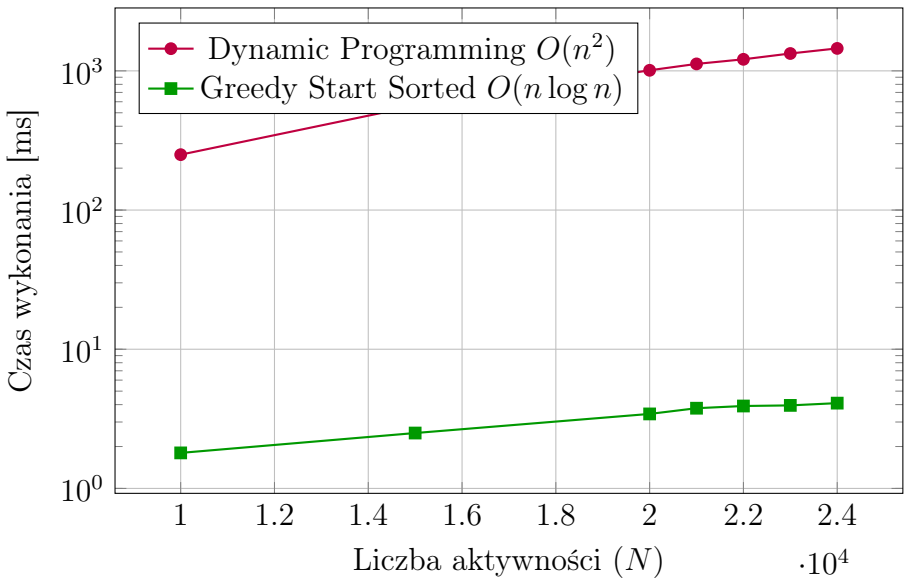


Rysunek 1: Wersja iteracyjna jest ok. **2x szybsza**. Wynika to z braku narzutu wywołań funkcji i lepszego wykorzystania pamięci podręcznej.

3.2 Activity Selector: Przepaść wydajnościowa

Zestawienie algorytmu programowania dynamicznego ($O(n^2)$) z podejściem zachłannym ($O(n \log n)$).

Wykres 2: Activity Selector - DP vs Greedy

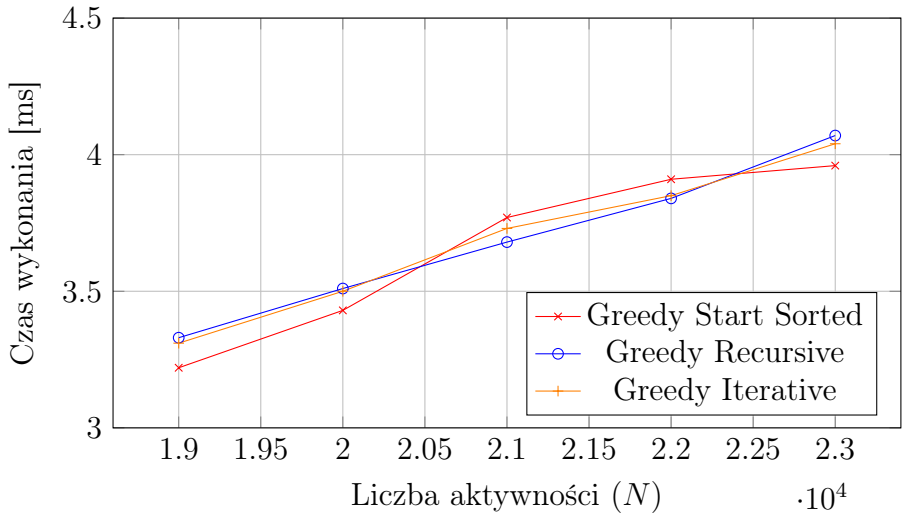


Rysunek 2: Dla $N = 24000$, DP potrzebuje ok. 1.4 sekundy, podczas gdy algorytm zachłanny kończy pracę w 4 milisekundy.

3.3 Porównanie wariantów zachłannych

Szczegółowa analiza algorytmów zachłannych dla dużych N (zoom).

Wykres 3: Porównanie wariantów Greedy (Zoom)



Rysunek 3: Wszystkie warianty zachłanne osiągają zbliżoną wydajność. Sortowanie po czasie startu jest równie efektywne co standardowe podejście.

3.4 Analiza strukturalna drzew Huffmana

Porównano właściwości drzew wygenerowanych dla tych samych danych wejściowych.

Tabela 1: Porównanie właściwości wariantów Huffmana

Cecha	**Huffman Binarny**	**Huffman Ternarny**
Stopień drzewa (k)	2	3
Alfabet kodowy	$\{0, 1\}$	$\{0, 1, 2\}$
Redukcja w kroku	1 węzeł	2 węzły
Wysokość drzewa	Wyższa ($\approx \log_2 N$)	Niższa ($\approx \log_3 N$)
Średnia długość kodu	Dłuższa	Krótsza
Złożoność obliczeniowa	$O(N \log N)$	$O(N \log N)$

Drzewo ternarne jest "płytsze", co daje krótsze napisy kodowe (w sensie liczby znaków), ale wymaga użycia alfabetu trójkowego.

4 Wnioski końcowe

Na podstawie przeprowadzonych badań sformułowano następujące wnioski:

1. **Dominacja algorytmów zachłannych:** W problemie wyboru zajęć podejście zachłanne jest bezkonkurencyjne (rzędy wielkości szybsze od DP). Potwierdza to zasadność stosowania strategii zachłannej, gdy spełniona jest *własność wyboru zachłannego*.

2. **Koszt abstrakcji (Memoizacja vs Iteracja):** W problemie Cut Rod obie metody mają tę samą złożoność asymptotyczną $O(n^2)$, lecz wersja iteracyjna jest stale około dwukrotnie szybsza i bezpieczniejsza pamięciowo (brak stosu rekurencji).

3. **Uniwersalność sortowania:** Wykazano, że sortowanie zajęć po czasie *rozpoczęcia* i przetwarzanie ich od końca jest równie wydajną metodą co klasyczne sortowanie po czasie zakończenia.

4. **Ograniczenia rekurencji:** Dla problemu LCS przy dużych danych ($N > 5000$) zaobserwowano ryzyko przepełnienia stosu w wersji rekurencyjnej.

5. **Generalizacja Huffmana:** Implementacja wariantu ternarnego potwierdziła, że algorytm Huffmana poddaje się uogólnieniu. Kluczowym wyzwaniem jest zapewnienie parzystości liczby węzłów początkowych (poprzez padding), tak aby redukcja trójkowa zawsze zakończyła się na jednym korzeniu.