Poznań 16 maja 2024

Politechnika Poznańska

Wydział Informatyki i Telekomunikacji

Algorytmy i Struktury Danych, Informatyka (semestr 2)
Sprawozdanie #5 — Programowanie dynamiczne

wykonał

Konrad Ceglarski

[156912]

1. Wprowadzenie

Programowanie dynamiczne to jedna z najważniejszych technik algorytmicznych, stosowana do rozwiązywania problemów optymalizacyjnych. Metoda ta polega na rozkładaniu problemu na mniejsze, łatwiejsze do rozwiązania podproblemy, a następnie na wykorzystaniu ich rozwiązań do konstrukcji rozwiązania problemu pierwotnego. W programowaniu dynamicznym kluczowe jest przechowywanie wyników podproblemów, aby uniknąć ich wielokrotnego przeliczania, co znacząco zwiększa efektywność algorytmu.

Jednym z klasycznych przykładów zastosowania programowania dynamicznego jest **0-1 problem plecakowy** (*ang. 0-1 Knapsack Problem*). Problem ten polega na tym, że mając dany zbiór przedmiotów, z których każdy ma określoną wagę i wartość, oraz plecak o ograniczonej pojemności, należy wybrać taki podzbiór przedmiotów, aby ich łączna wartość była maksymalna, a łączna waga nie przekraczała pojemności plecaka. Nazwa "0-1" wskazuje, że każdy przedmiot może być albo w całości włożony do plecaka (1), albo wcale (0).

Do problemu plecakowego możemy podejść na dwa sposoby:

- **optymalizacyjnie**
  czyli znaleźć taki podzbiór elementów, że mieszczą się one do plecaka o podanej pojemności, a ich suma wartości jest maksymalna.
  $\sum_{i}^{n} w_i \leq c$ oraz $max \sum_{i}^{n} p_i$
- **decyzyjnie**
  czyli sprawdzić, czy istnieje taki podzbiór elementów, że mieszczą się one do plecaka o podanej pojemności, a ich suma wartości jest równa co najmniej określonej wartości b.
  $\sum_{i}^{n} w_i \leq c$ oraz $\sum_{i}^{n} p_i \geq b$

Moim zadaniem w ramach zajęć laboratoryjnych było zaimplementowanie rozwiązań tego właśnie problemu w postaci algorytmów:

- **dynamicznego**
- **siłowego** (*ang. Brute Force*)

2. Klasy złożoności problemu plecakowego oraz złożoność obliczeniowa wybranych algorytmów rozwiązywania

Klasy złożoności problemu plecakowego

Problem optymalizacyjny	Problem decyzyjny
Klasa złożoności problemu: **NP-trudny**	*Klasa złożoności problemu:* **NP-zupełny**

Złożoność obliczeniowa zastosowanych algorytmów

Algorytm dynamiczny	
Złożoność czasowa: **$O(n*c)$**	Złożoność pamięciowa: **$O(n*c)$**

* gdzie **n** to ilość przedmiotów, a **c** to pojemność plecaka

podane złożoności wynikają z:
konieczności wypełnienia oraz przechowywania macierzy n*c

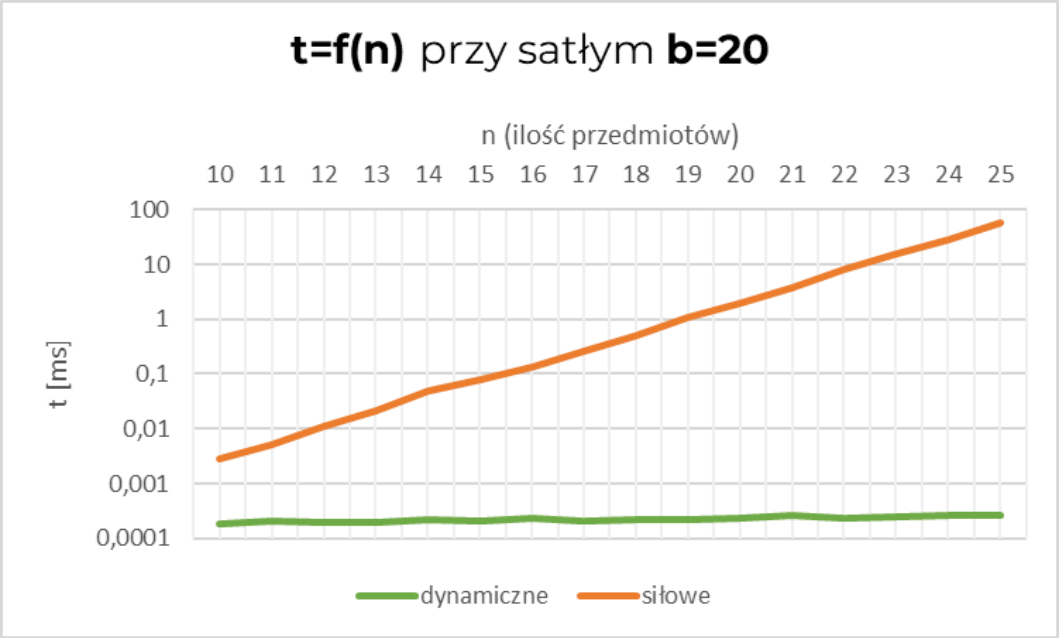
Algorytm siłowy (*ang. Brute Force*)	
Złożoność czasowa: **$O(2^n)$**	Złożoność pamięciowa: **$O(n)$**

* gdzie **n** to ilość przedmiotów

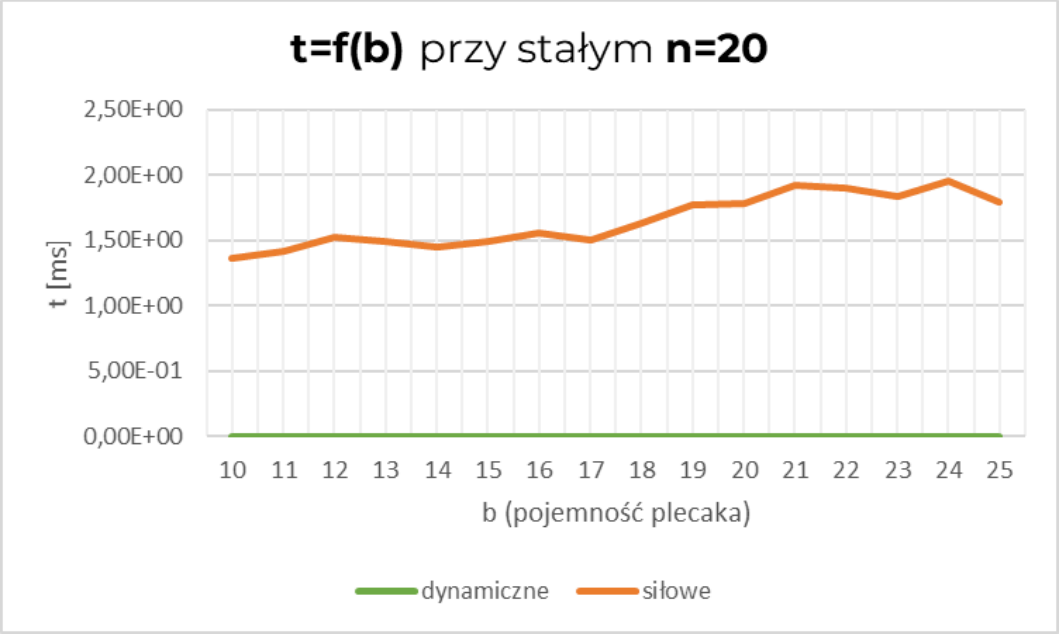
podana złożoność czasowa wynika z:
sprawdzenia wszystkich (2) możliwości dla każdego przedmiotu

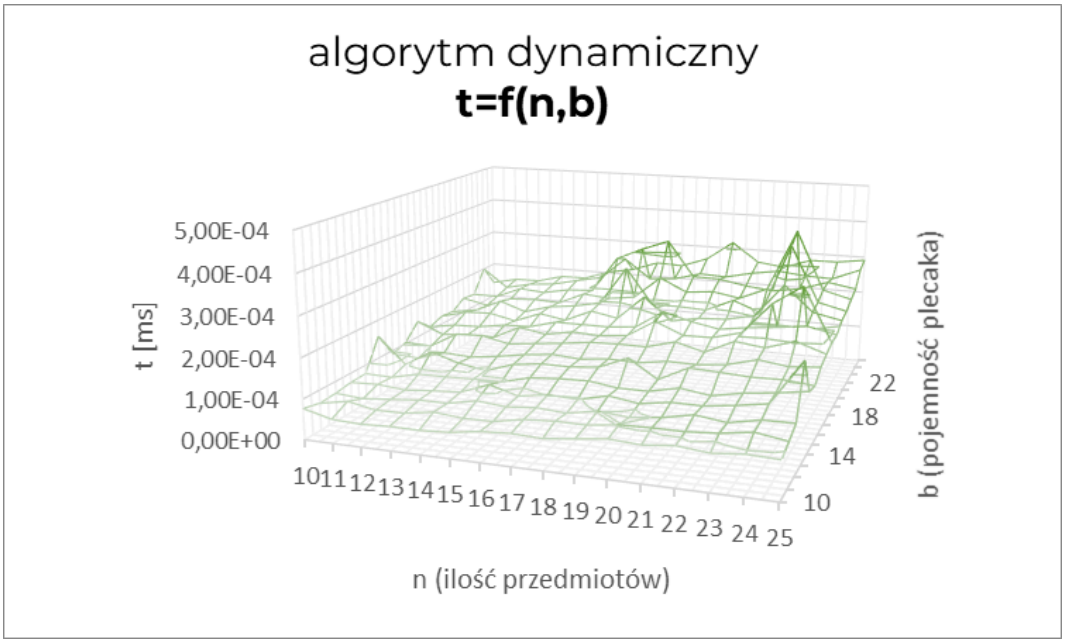
podana złożoność pamięciowa wynika z:
przechowywania bieżącej kombinacji przedmiotów

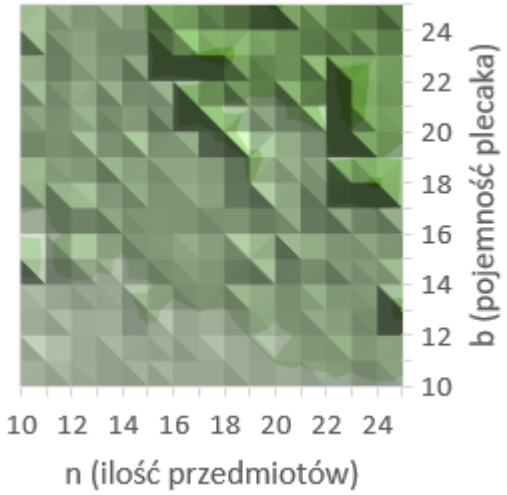
3. Wykresy obrazujące efektywność wybranych algorytmów



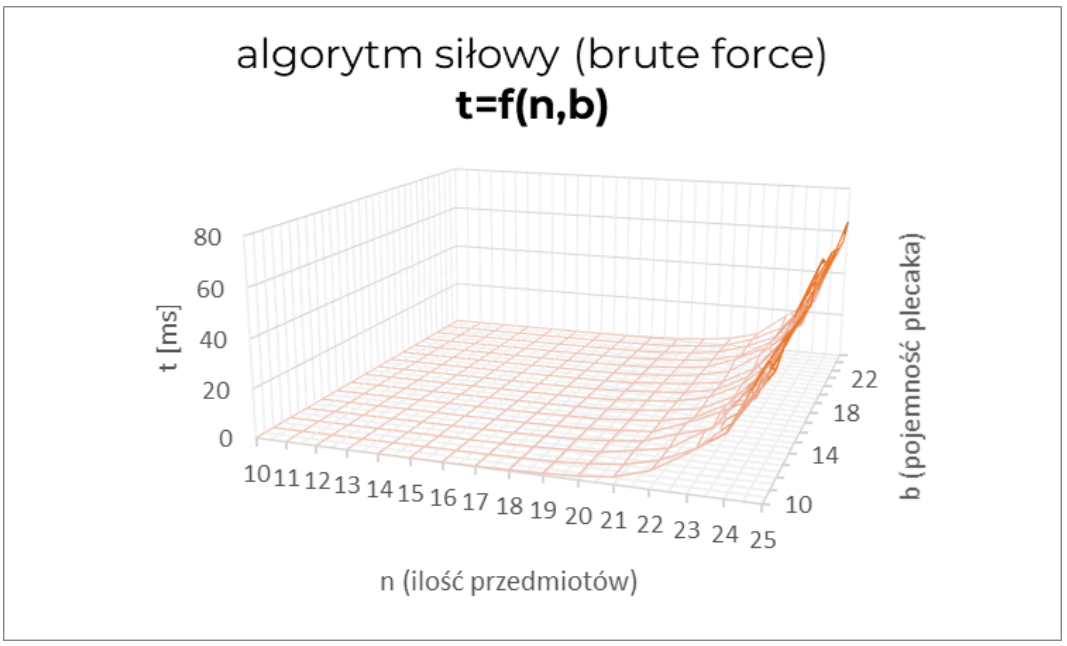
* zastosowano skalę logarytmiczną

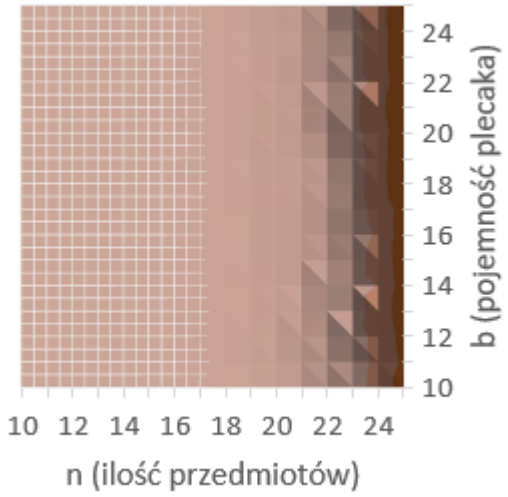






* im bardziej nasycony i ciemniejszy kolor
  tym dłużej algorytm szuka rozwiązania
* przedstawiona losowość na wykresie jest wyolbrzymiona
  przez małą skalę i niską amplitudę zmian wartości czasu





* im bardziej nasycony i ciemniejszy kolor
  tym dłużej algorytm szuka rozwiązania

4. Obserwacje związane z działaniem zaimplementowanych algorytmów dla grafów o różnym nasyceniu.

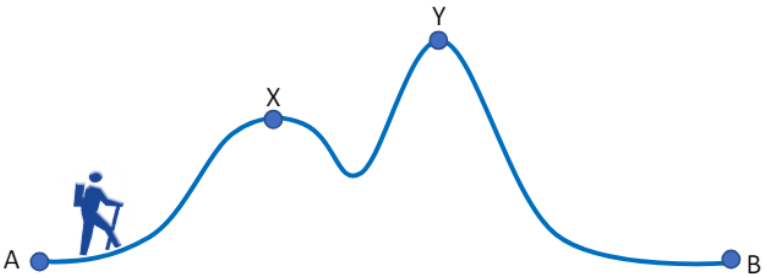
Wykresy powierzchniowe **`t=f(n,b)`** dobrze odzwierciedlają złożoność czasową obu algorytmów:

- Dla algorytmu **dynamicznego** wartości czasu rosną w stronę prawego-górnego rogu, czyli tam gdzie `n` (ilość przedmiotów) oraz `b` (pojemność plecaka) są największe, co pokrywa się z złożonością czasową **`O(n*b)`**.
- Dla algorytmu **siłowego** natomiast wartości niezależnie od pojemności plecaka rosną wykładniczo wraz ze wzrostem `n`, czyli ilości przedmiotów, które trzeba przetestować.

5. Czy można ustalić w jakich przypadkach algorytm zachłanny nie daje rozwiązania optymalnego?

Mimo, że do mojego zadania nie należała implementacja algorytmu zachłannego to z prezentacji wysuwam wnioski, że algorytm ten skupiając się na przedmiotach o wysokim stosunku wartości do wagi (lub wagi do wartości) pominie nieoczywiste kombinacje, które mogą okazać się bardziej optymalne.

Przykład
Cel: wejdź na najwyższy punkt masywu.



Rozwiązanie:

a) Zachłanny turysta, który startuje z pkt. A dotrze to pkt. X (rozwiązanie suboptymalne).
b) Zachłanny turysta, który startuje z pkt. B dotrze do pkt. Y (rozwiązanie optymalne).

Język implementacji:

Python 🐍 (Python 3.10.10)

Platforma realizacji testów:

Windows 10 x64 (Microsoft Windows [Version 10.0.19045.4170])
CPU: AMD Ryzen 7 3750H (4 cores/8 threads) 2.30 GHz
RAM: 16 GB

Repozytorium:

https://github.com/xKond3i/put_aisd

Źródła:

[1]:cs.put.poznan.pl/mszachniuk
[2]:www.ekursy.put.poznan.pl
[3]:www.eduinf.waw.pl/inf/alg