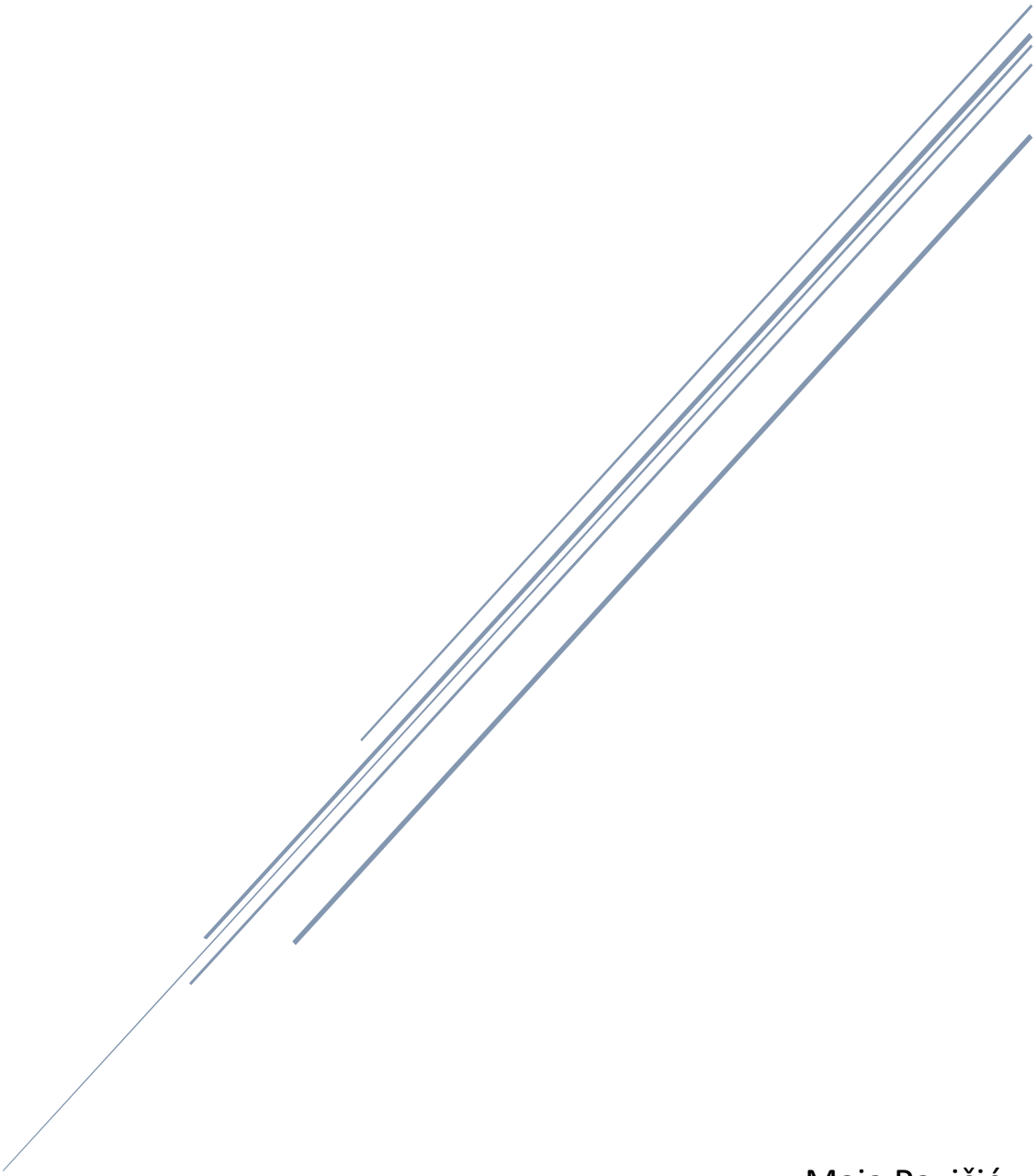
PROBLEM REZANJA ŠIPKE

empirijska i teorijska analiza algoritma, rekurzivno i dinamičko programiranje



Maja Pavičić
Oblikovanje i analiza algoritama, doc. dr. sc. Matej Mihelčić
Prirodoslovno - matematički fakultet, Sveučilište u Zagrebu
studeni 2023.

Opis problema

Dana je šipka poznate duljine N, koju je moguće rezati na manje dijelove. Svakoj mogućoj duljini šipke pridružena je vrijednost. Rezanje šipke je besplatno. Potrebno je odrediti optimalan način rezanja šipke, koji će maksimizirati dobit.
Zadani problem potrebno je riješiti rekurzivnim i dinamičkim programiranjem te provesti teorijsku i empirijsku analizu složenosti.

Iako nisu eksplicitno navedeni u opisu problema, uzeto je da vrijede još neki uvjeti kako bi problem bio preciznije definiran ili jednostavniji za implementaciju:

- početna duljina šipke i duljine svih odrezanih dijelova su prirodni brojevi (da bi se problem mogao riješiti rekurzivnim i dinamičkim metodama potrebno je da duljine budu diskretne vrijednosti, a onda je najjednostavnije raditi sa prirodnim brojevima)
- vrijednosti šipki su prirodni brojevi (to je najjednostavnije za implementaciju, a irelevantno za račun složenosti)
- vrijednosti su iz intervala [1, duljina šipke * 1.5] i uniformno su distribuirane (da bismo mogli izgenerirati cijene potrebno je zadati neki interval)
- funkcija cijene u ovisnosti o duljini šipke je rastuća (nije eksplicitno navedeno kao pretpostavka, ali je razumno za pretpostaviti)

Algoritamska rješenja

Problem rezanja šipki spada u probleme dinamičkog programiranja i zadovoljava sljedeće principe svojstvene toj skupini problema:

- princip optimalnosti: Da bi šipka određene duljine bila optimalno narezana, „podšipka“ koju iz nje dobijemo micanjem najljevijeg dijela također mora biti optimalno narezana.
- princip neovisnosti/invarijantnosti: Optimalan način rezanja preostalog desnog dijela šipke ne ovisi o duljini najljevijeg dijela.
- princip ulaganja/parametrizacije: Problem optimalnog rezanja „poddijelova“ početne šipke je iste vrste kao i početni problem rezanja cijele šipke.

Navedene principe koristimo u konstrukciji tri različita algoritma za rješenje problema: rekurziju, memoiziranu rekurziju i dinamičko rješenje. Zatim ćemo usporediti operacijsku i vremensku složenost konstruiranih algoritama.

Napomenimo još da u našoj implementaciji nismo pamtili koji način rezanja je optimalan, nego samo koliku maksimalnu vrijednost možemo postići. Za rekonstrukciju rješenja bilo bi potrebno koristiti još jedno polje u koje bismo u svakom koraku spremali izabrane duljine najljevijih dijelova, što bi usporilo program, ali ne bi u konačnici utjecalo na složenost.

Također, u našim rješenjima razlikujemo na primjer particiju šipke 3|5|2 od 2|3|5, što nema smisla, jer za ovaj problem sve permutacije narezanih dijelova u konačnici daju istu vrijednost. Moguće je implementirati i algoritme koji bi to uzimali u obzir, primjerice tako da duljine najljevijih dijelova moraju činiti rastući niz. To bi ubrzalo algoritam, ali bi zakompliciralo račun složenosti, pa to u ovom slučaju nismo napravili.

Primijetimo na kraju da rješenje ovog problema do na poredak permutacija nije jedinstveno te da će naši algoritmi pronaći samo jedno od mogućih rješenja.

Rekurzivno rješenje

Ideja rekurzivnog rješenja je sljedeća: u svakom koraku rekurzije za šipku dane duljine odredimo kolika će biti duljina najljevijeg odrezanog dijela i zatim na ostatak šipke rekurzivno primijenimo isti algoritam. Pritom isprobamo sve moguće duljine najljevijeg dijela i na kraju se odlučimo za onu za koju ćemo ukupno dobiti najveću vrijednost.

```
CUT_ROD(p,n)
    if n == 0
        return 0

    q = -inf
    for i = 1 to n
        q = max{q,p[i] + CUT_ROD(p, n-i)}
    return q
```

Slika 1. – Pseudokod za običan rekurzivni algoritam

Ovo je najočitije rješenje problema, implementacija rekurzivne relacije (1) koju problem zadovoljava, no nedostatak mu je što iste potprobleme rješava mnogo puta.

Memoizirano rekurzivno rješenje

Bolji način je koristiti memoizaciju rekurzije, odnosno jednom izračunatu vrijednost funkcije za neku duljinu zapamtiti u memoriji i sljedeći put kad ju zatrebamo traženu vrijednost samo pročitati.

Za to je potrebno na početku rješenja alocirati i inicijalizirati memoriju za pamćenje podrezultata. Kasnije ćemo vidjeti da ovim postupkom znatno popravljamo vremensku složenost, no to plaćamo potrošnjom memorije (primjer za time-memory trade-off).

```
MEMOIZED_CUT_ROD(p, n)
    let r[0:n] be a new array
    for i = 0 to n
        r[i] = -inf
    return MEMOIZED_CUT_ROD_AUX(p, n, r)


MEMOIZED_CUT_ROD_AUX(p, n, r)
    if r[n] >= 0
        return r[n]
    if n == 0
        q = 0
    else
        q = -inf
        for i = 1 to n
            q = max{q, p[i], MEMOIZED_CUT_ROD_AUX(p, n-i, r)}
        r[n] = q
        return q
```

Slika 2. – Pseudokod za memoizirani rekurzivni algoritam

Rješenje dinamičkim programiranjem

Rješenje dinamičkim programiranjem … umjesto da krenemo od cijele duljine šipke N, a manje potprobleme rješavamo kad nam zatrebaju njihova rješenja, možemo odmah primijetiti da za određivanje rješenja većeg problema moramo znati rješenja svih manjih problema N-1, N-2, …, 1. Zato možemo krenuti od rješavanja manjih problema prema većima. Tako ćemo u svakom koraku sva potrebna manja rješenja već imati izračunata. Dakle umjesto da problem rješavamo rekurzivno odozgo prema dolje (top-down), rješavamo ga dinamikom odozdo prema gore (bottom-up).

Kao i za memoiziranu rekurziju, za to nam je potrebna memorija za pamćenje međurezultata, ali je napredak što ne moramo raditi puno funkcijskih poziva, koji su vremenski zahtjevni i zauzimaju memoriju na stogu.

```
BOTTOM_UP_CUT_ROD(p, n)
    let r[0:n] be a new array
    r[0] = 0

    for j = 1 to n
        q = -inf
        for i = 1 to j
            q = max{q, p[i] + r[j-i]}
        r[j] = q

    return r[n]
```

Slika 3. – Pseudokod za dinamički algoritam

Teorijska analiza

Teorijsku analizu složenosti radimo promatrajući pseudokodove iz prethodnog odlomka.

Za račun složenosti rekurzivnih algoritama najznačajnije je koliko će se puta pozvati funkcija.
Zato definiramo:
$T(n) := \ broj\ poziva\ funkcije\ CUT\_ROD\ za\ zadanu\ početnu\ duljinu\ štapa\ n$
Rekurzivna relacija koja se koristi u rješenju:
$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$ (1)
Uz početni uvjet $T(0) = 1$.
U formuli (1) broj 1 označava inicijalni poziv funkcije za duljinu n, a članovi sume odgovaraju funkcijskim pozivima unutar for petlje.
To možemo napisati i kao $T(n) = T(n-1) + [T(n-2) + T(n-3) + \cdots + T(2) + T(1)]$.
Sada je lakše primijetiti da vrijedi $T(n) = 2 \cdot T(n-1)$.
Ovo je vrlo jednostavna rekurzivna relacija za kakve smo formulu za rješavanje dali na predavanju:
$T(n) = 2^n \cdot T(0) = 2^n$
$\Longrightarrow\ T(n) \in \theta(2^n)$
Zaključujemo da rekurzivni algoritma ima eksponencijalnu složenost.

Izračunajmo sada odmah složenost za dinamičko rješenje. U tom rješenju imamo samo jedan funkcijski poziv, ali možemo kao relevantnu operaciju uzeti određivanje maksimuma. Naime, većina operacija će se izvršiti jednom ako su izvan petlje ili n puta ako su unutar vanjske for petlje. No operacije iz unutarnje for

petlje će se izvršiti $1 + 2 + \cdots + n = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$ puta, u svakom slučaju. Zaključujemo da je složenost dinamičkog rješenja polinomijalna, odnosno $T(n) \in \theta(n^2)$.

Složenost memoiziranog rekurzivnog rješenja je malo teža za vidjeti. Opet promatramo broj poziva funkcije MEMOIZED_CUT_ROD_AUX. Funkcija se inicijalno poziva za duljinu n te se zatim rekurzija spušta u dubinu sve do duljine 1. Zatim se rekurzija krene penjati nazad te u proizvoljnom koraku k u for petlji poziva funkciju MEMOIZED_CUT_ROD_AUX za duljine $1, \ldots, k-1$. No sva su ta manja rješenja već ranije izračunata pa sada rekurzije idu samo do dubine 1 (odnosno služe za dohvaćanje zapamćenog rezultata iz memorije). To se najjednostavnije može shvatiti na stablu rekurzije. Dakle, ukupni broj poziva funkcije je opet $1 + 2 + \cdots + n = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$, a složenost memoiziranog rekurzivnog rješenja polinomijalna $T(n) \in \theta(n^2)$.

Kao što smo već spomenuli, očekujemo da će usprkos jednakoj složenosti memoizacija imati veće konstante, odnosno dulje vrijeme izvršavanja, zbog učestalih funkcijskih poziva.

Empirijska analiza

Mjerenja su izvršavana na računalu sljedećih karakteristika: procesor i7-1165 G7 2.80GHz, 64GB RAM. Inicijalna veličina stoga je 1MB, ali je ta vrijednost za potrebe ovog projekta povećana na 4MB. Program smo implementirali u jeziku C++ i kompajlirali Visual C++ 2019 kompajlerom.

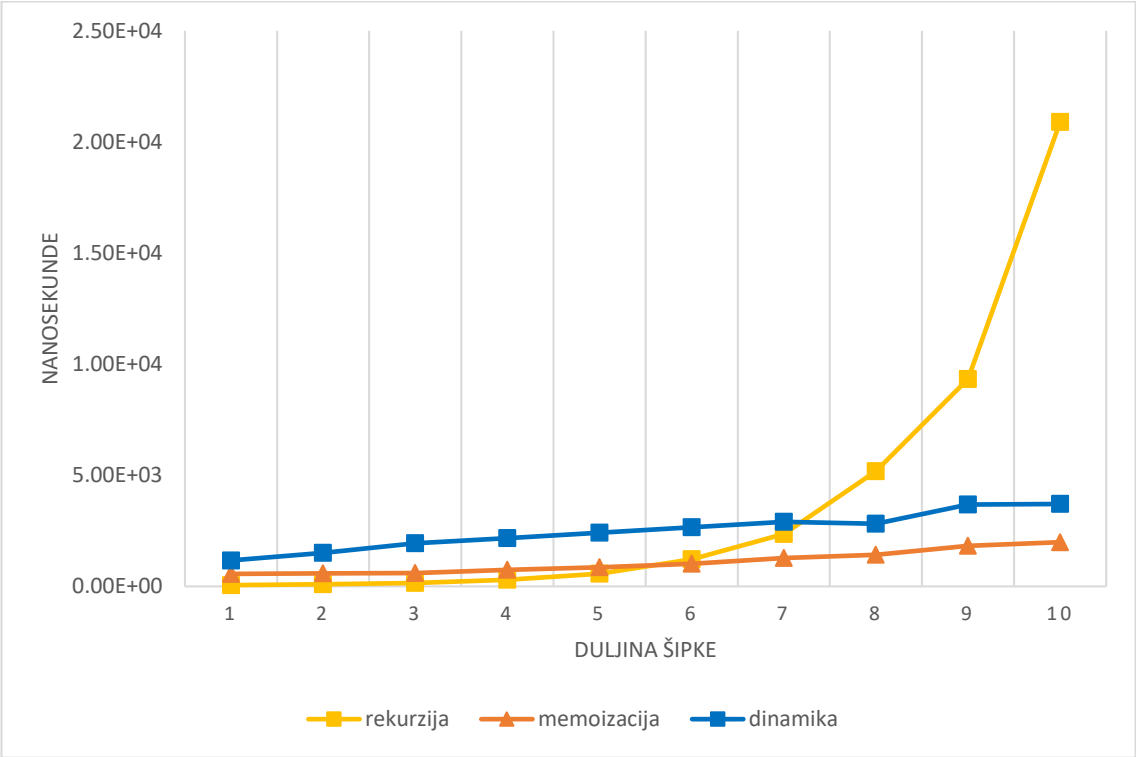
Cijelu implementaciju smo smjestili u jedan program, ali smo kod organizirali tako da se lako može odrediti koji će algoritmi biti pozvani u kojem izvršavanju. Uz to smo u programu koristili sljedeće važne parametre: najmanja i najveća duljina šipke za koju izvršavamo algoritme, korak za koji povećavamo duljinu i broj ponavljanja algoritama za pojedinu duljinu šipke (kako bismo izračunali prosjek vremena izvršavanja i dobili manje oscilacije u mjerenjima).

U izvršenim mjerenjima broj ponavljanja iznosio je 100.

Za manje duljine šipke smo izvršavali sva tri algoritma.

duljina šipke	rekurzija	memoizacija	dinamika
1	5.70E+01	5.63E+02	1.17E+03
2	9.80E+01	5.85E+02	1.50E+03
3	1.56E+02	6.03E+02	1.94E+03
4	2.93E+02	7.46E+02	2.17E+03
5	5.78E+02	8.58E+02	2.41E+03
6	1.22E+03	1.02E+03	2.67E+03
7	2.36E+03	1.28E+03	2.91E+03
8	5.18E+03	1.43E+03	2.81E+03
9	9.33E+03	1.83E+03	3.69E+03
10	2.09E+04	1.99E+03	3.71E+03

Tablica 1. – šipke duljine od 1 do 10, povećanje duljine za 1, 100 ponavljanja, mjereno u nanosekundama



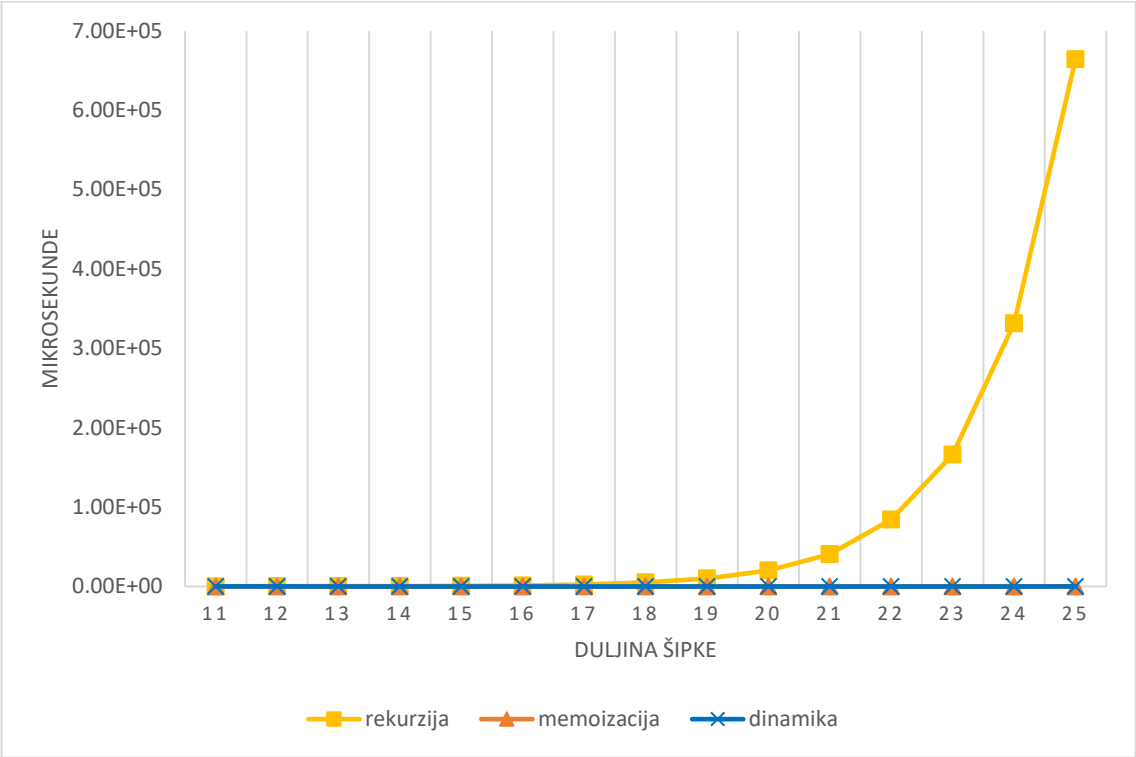
Graf 1. – Ovisnost trajanja izvršavanja algoritama o duljini šipke, prikaz mjerenja iz Tablice 1.

Već se za najmanjih 10 dužina može vidjeti da se memoizacija i rekurzija ponašaju relativno slično, dok rekurzija odudara. Također primjećujemo da za najmanje dužine rekurzija traje najkraće, a dinamika najdulje. Jedan od razloga je što nema potrebe za alokacijom niti inicijalizacijom dodatne memorije. Međutim, vrijeme izvršavanja rekurzije raste znatno brže pa već za duljinu 8 postaje najsporija.

I u narednih 15 mjerenja možemo vidjeti da se sličan trend nastavlja.

duljina šipke	rekurzija	memoizacija	dinamika
11	4.13E+01	2.01E+00	3.13E+00
12	7.80E+01	2.03E+00	3.19E+00
13	1.60E+02	2.08E+00	4.15E+00
14	3.20E+02	3.04E+00	4.76E+00
15	6.08E+02	3.77E+00	5.69E+00
16	1.25E+03	4.51E+00	5.32E+00
17	2.47E+03	4.08E+00	4.34E+00
18	5.01E+03	4.19E+00	5.18E+00
19	1.00E+04	4.95E+00	5.19E+00
20	2.02E+04	5.16E+00	6.01E+00
21	4.05E+04	6.24E+00	6.27E+00
22	8.42E+04	7.10E+00	6.22E+00
23	1.66E+05	7.06E+00	6.18E+00
24	3.31E+05	8.12E+00	7.16E+00
25	6.64E+05	8.30E+00	7.31E+00

Tablica 2. – šipke duljine od 11 do 25, povećanje duljine za 1, 100 ponavljanja, mjereno u mikrosekundama



Graf 2. – Ovisnost trajanja izvršavanja algoritama o duljini šipke, prikaz mjerenja iz Tablice 2.

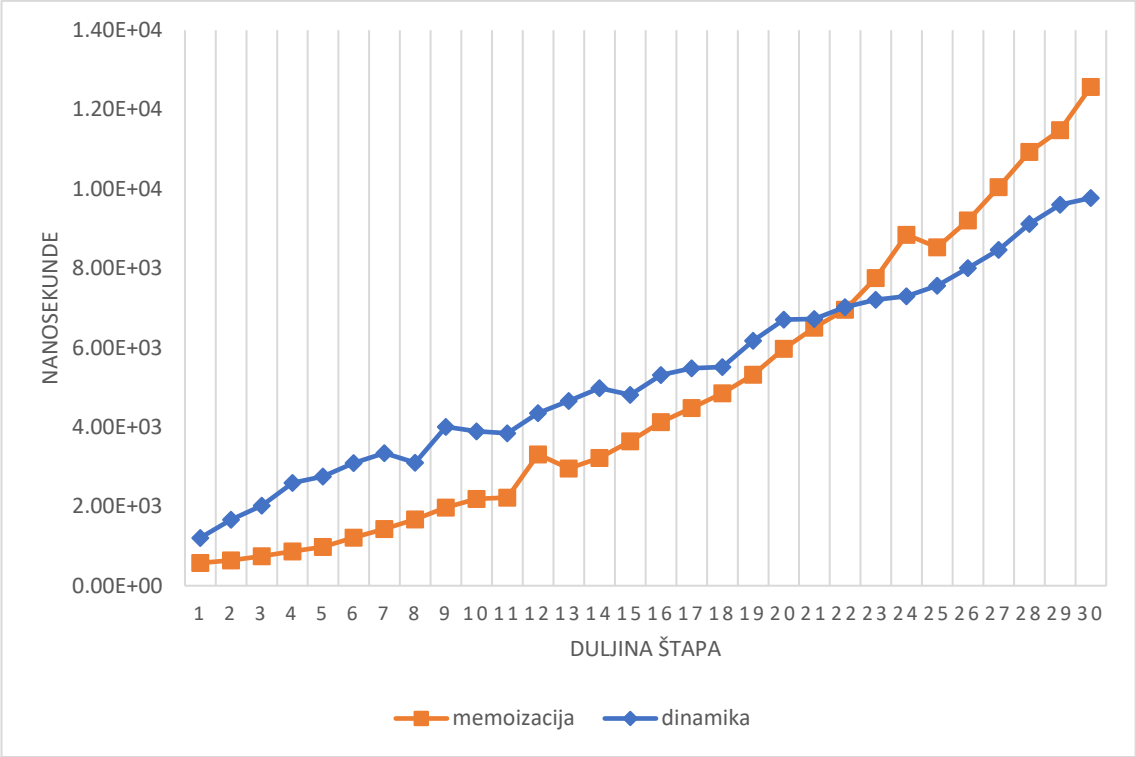
U preostalim mjerenjima nećemo više promatrati običnu rekurziju, jer bi nam njeno izvršavanje za veće duljine previše usporilo mjerenje, a na grafu ne bismo mogli vidjeti kako se ponašaju memoizacija i dinamika.

Najprije ponovimo mjerenja za duljine 1 – 30. Sada jasno vidimo da rekurzivno rješenje na početku ima manje vrijeme izvršavanja, ali je porast trajanja veći i oko duljine 22 dinamičko rješenje postaje brže.

duljina šipke	**memoizacija**	**dinamika**
1	5.74E+02	1.21E+03
2	6.37E+02	1.66E+03
3	7.43E+02	2.02E+03
4	8.63E+02	2.59E+03
5	9.81E+02	2.75E+03
6	1.22E+03	3.09E+03
7	1.43E+03	3.34E+03
8	1.67E+03	3.10E+03
9	1.97E+03	4.00E+03
10	2.19E+03	3.89E+03
11	2.22E+03	3.84E+03
12	3.31E+03	4.35E+03
13	2.96E+03	4.66E+03
14	3.22E+03	4.98E+03
15	3.64E+03	4.81E+03
16	4.12E+03	5.31E+03
17	4.48E+03	5.48E+03
18	4.85E+03	5.51E+03
19	5.32E+03	6.18E+03

20	5.97E+03	6.71E+03
21	6.51E+03	6.72E+03
22	6.96E+03	7.02E+03
23	7.76E+03	7.21E+03
24	8.85E+03	7.29E+03
25	8.53E+03	7.56E+03
26	9.21E+03	8.01E+03
27	1.00E+04	8.47E+03
28	1.09E+04	9.12E+03
29	1.15E+04	9.61E+03
30	1.26E+04	9.77E+03

Tablica 3. – šipke duljine od 1 do 30, povećanje duljine za 1, 100 ponavljanja, mjereno u nanosekundama



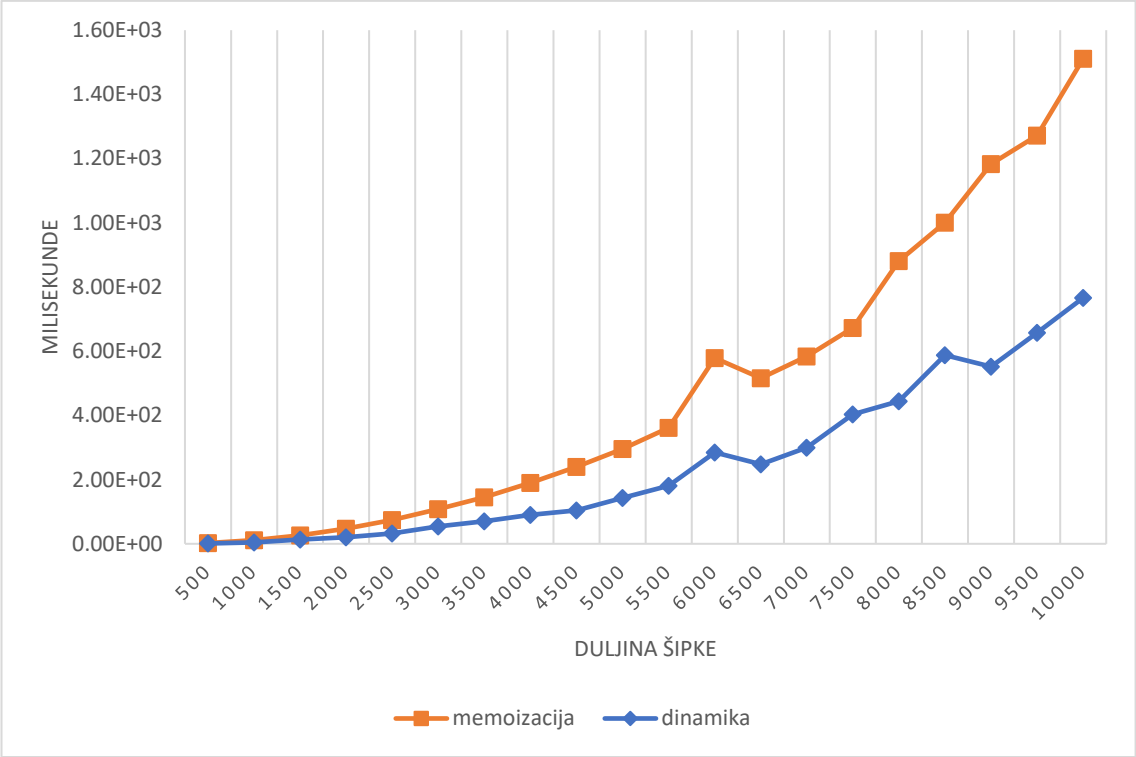
Graf 3. – Ovisnost trajanja izvršavanja algoritama o duljini šipke, prikaz mjerenja iz Tablice 3.

Obzirom na vrijeme izvršavanja izmjereno u nanosekundama, zaključujemo da ove algoritme možemo izvesti i za znatno dulje štapove U sljedećem mjerenju krenuli smo s početnom duljinom 500 i uz korak 500 došli do duljine od 10 000. Trajanje smo ovaj put mjerili u milisekundama. Pritom je bilo potrebno povećati inicijalnu veličinu stoga od 1MB na 4MB, jer je zbog dubine rekurzije došlo do overflowa.

duljina šipke	memoizacija	dinamika
500	2.26E+00	1.00E+00
1000	1.13E+01	4.25E+00
1500	2.65E+01	1.31E+01
2000	4.70E+01	2.04E+01
2500	7.41E+01	3.18E+01
3000	1.08E+02	5.40E+01
3500	1.45E+02	6.98E+01
4000	1.90E+02	9.01E+01
4500	2.40E+02	1.04E+02

5000	2.96E+02	1.43E+02
5500	3.61E+02	1.81E+02
6000	5.79E+02	2.85E+02
6500	5.16E+02	2.48E+02
7000	5.83E+02	2.99E+02
7500	6.73E+02	4.03E+02
8000	8.81E+02	4.44E+02
8500	1.00E+03	5.88E+02
9000	1.18E+03	5.52E+02
9500	1.27E+03	6.57E+02
10000	1.51E+03	7.66E+02

Tablica 4. – šipke duljine od 500 do 10 000, povećanje duljine za 500, 100 ponavljanja, mjereno u milisekundama



Graf 4. – Ovisnost trajanja izvršavanja algoritama o duljini šipke, prikaz mjerenja iz Tablice 4.

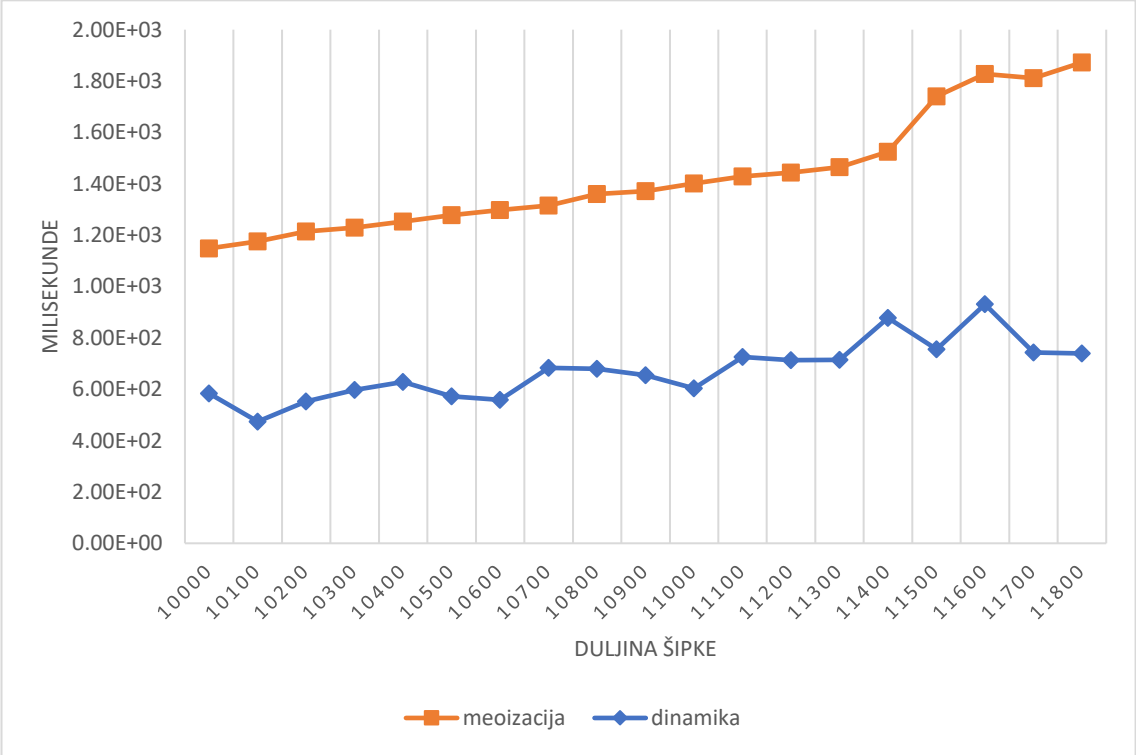
Na ovom je grafu već jasno vidljivo da porast vremena izvršavanja nije linearan ni za jedno od rješenja te da trajanje memoizirane rekurzije raste brže od dinamike.

Na kraju smo još željeli vidjeti do koje duljine možemo doći sa alociranih 4MB RAM-a. Eksperimentalno smo dobili da se za duljinu 11 800 algoritmi izvršavaju, dok za duljinu 11 900 dobivamo grešku sa kodom -1073741571, što označava „stack overflow“. Navodimo rezultate i tih mjerenja. Kako nam ova mjerenja nisu bila posebno zanimljiva za promatranje složenosti, za svaku smo duljinu mjerenje ponovili 3 puta.

duljina štapa	memoizacija	dinamika
10000	1.15E+03	5.84E+02
10100	1.18E+03	4.74E+02
10200	1.21E+03	5.52E+02
10300	1.23E+03	5.98E+02
10400	1.25E+03	6.29E+02
10500	1.28E+03	5.73E+02
10600	1.30E+03	5.58E+02

10700	1.32E+03	6.84E+02
10800	1.36E+03	6.79E+02
10900	1.37E+03	6.54E+02
11000	1.40E+03	6.03E+02
11100	1.43E+03	7.25E+02
11200	1.44E+03	7.13E+02
11300	1.47E+03	7.15E+02
11400	1.52E+03	8.78E+02
11500	1.74E+03	7.55E+02
11600	1.83E+03	9.31E+02
11700	1.81E+03	7.43E+02
11800	1.87E+03	7.40E+02

Tablica 5. – šipke duljine od 10 000 do 11 800, povećanje duljine za 100, 100 ponavljanja, mjereno u milisekundama



Graf 5. – Ovisnost trajanja izvršavanja algoritama o duljini šipke, prikaz mjerenja iz Tablice 5.

Izvori

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms (Fourth Edition). MIT Press. https://mitpress.mit.edu/algorithms/
2. Mihelčić, M. 30.10.2023. "Dinamičko programiranje - nastavak, pohlepni algoritmi - uvod". kolegij Oblikovanje i analiza algoritama. Prirodoslovno - matematički fakultet, Sveučilište u Zagrebu https://web.math.pmf.unizg.hr/nastava/oaa/materijali/predavanjaMM/Predavanje6.pdf