

SADJE

Domen Hribernik, Žan Kazar

Maribor, maj 2024

KAZALO

[1 OPIS PROBLEMA 3](#_Toc166517303)

[1.1 Navodila naloge 3](#_Toc166517304)

[1.2 Vizualizacija enostavne instance problema 3](#_Toc166517305)

[1.3 Primer iz realnega sveta 4](#_Toc166517306)

[2 OPIS REŠITVE Z METODO GROBE SILE 4](#_Toc166517307)

[2.1 Reševanje po korakih 4](#_Toc166517308)

[2.2 Psevdokoda 5](#_Toc166517309)

[2.3 Robni primeri 5](#_Toc166517310)

[3 OPIS REŠITVE S PRIMERNO STRATEGIJO: DINAMIČNO PROGRAMIRANJE 6](#_Toc166517311)

[3.1 Reševanje z dinamičnim programiranjem 6](#_Toc166517312)

[3.2 Psevdokoda 7](#_Toc166517313)

[3.3 Robni primeri 7](#_Toc166517314)

[3.4 Vizualizacija 8](#_Toc166517315)

[4 ZGLED TEŽJE INSTANCE PROBLEMA 9](#_Toc166517316)

[4.1 Vhodni podatki 9](#_Toc166517317)

[4.2 Vizualizacija 9](#_Toc166517318)

[5 PREDSTAVITEV IN PRIMERJAVA REZULTATOV 11](#_Toc166517319)

[5.1 Opis testnih primerov instanc problema 11](#_Toc166517320)

[5.2 Meritve časa in zasedenosti pomnilnika 11](#_Toc166517321)

[5.3 Ocenjena časovna zahtevnost 12](#_Toc166517322)

[6 ZAKLJUČEK 13](#_Toc166517323)

[7 LITERATURA 14](#_Toc166517324)

KAZALO SLIK

[Slika 1 - Psevdokoda metode grobe sile 5](#_Toc166510177)

[Slika 2 - Psevdokoda dinamično programiranje 7](#_Toc166510178)

[Slika 3 – Inicializacija tabel 8](#_Toc166510179)

[Slika 4 – Vizualizacija posodabljanja tabel 8](#_Toc166510180)

[Slika 5 – Vizualizacija prvih 50 znakov 10](#_Toc166510181)

[Slika 6 – Vizualizacija prvih 200 znakov 10](#_Toc166510181)

[Slika 7 - Primerjava rezultatov med grobo silo in dinamičnim programiranjem 12](#_Toc166510181)

# OPIS PROBLEMA

## Navodila naloge

Naloga, ki temelji na ljudski pesmi, se osredotoča na igro med dvema igralcema, Iva in Ivo, kjer se tekmujeta v manipulaciji zaporedja sadja, ki vsebuje samo jabolka in hruške. Cilj Ive je v zaporedje vstaviti določeno število jabolk tako, da bo Ivo, ki pobira vsak m-ti sadež iz zaporedja, pobral čim manj jabolk. Ivo želi pobrati čim več jabolk, medtem ko Iva manipulira zaporedje v svojo korist.

Označimo:

* 𝑛 kot začetno število sadežev.
* 𝑘 - kot največje število jabolk, ki jih lahko Iva vrine v zaporedje.
* 𝑚 - kot interval pobiranja sadežev s strani Iva.

Zaporedje sadja je predstavljeno kot niz znakov, kjer '0' predstavlja hruško, '1' pa jabolko. Iva manipulira zaporedje tako, da vstavi do

𝑘 jabolk na strateških mestih, da minimizira število jabolk, ki jih Ivo pobere na vsakem m-tem mestu.

## Vizualizacija enostavne instance problema

Recimo, da je začetno zaporedje sadežev '0101' (jabolko, hruška, jabolko, hruška), Iva lahko doda do dve jabolki, in Ivo pobira vsak drugi sadež:

* Začetno zaporedje: 0 1 0 1
* Možna manipulacija: Iva doda jabolko na prvo mesto (1 0 1 0 1)
* Ivo pobere: Drugi in četrti sadež, ki sta zdaj 0 in 0.

Za boljšo vizualizacijo, si lahko predstavljamo tabelo ali graf, ki prikazuje kako dodajanje sadežev vpliva na izbor Iva. V tem primeru, z dodajanjem jabolka na tretje mesto, Iva zagotovi, da Ivo pobere samo eno jabolko namesto dveh.

## Primer iz realnega sveta

Podobno kot pri tej igri manipulacije zaporedja, lahko ta problem primerjamo z optimizacijskimi izzivi v logistiki, na primer kako razporediti blago v skladišču, da se optimizira izbira artiklov glede na pogostost naročil. Tako kot Iva razporeja jabolka za minimalni izplen Iva, lahko logistični menedžerji razporejajo izdelke za maksimalno učinkovitost izbire.

# OPIS REŠITVE Z METODO GROBE SILE

## Reševanje po korakih

Metoda grobe sile vključuje preizkušanje vseh možnih načinov, kako lahko Iva vrine dodatna jabolka v zaporedje, in nato ocenjevanje vsake konfiguracije glede na to, koliko jabolk bo Ivo pobral. To se izvede z naslednjimi koraki:

1. Preverjanje vseh možnih mest za vrivanje jabolk: Za vsako pozicijo v zaporedju odločimo, ali bomo na to mesto vstavili jabolko ali ne. To se izvaja rekurzivno za vsako mesto v zaporedju.
2. Izračun učinka vsake konfiguracije: Po vsakem vrivanju jabolk izvedemo skeniranje zaporedja, da ugotovimo, koliko jabolk bo Ivo pobral, ko na vrsto pride njegovo pobiranje sadežev po vsakem 𝑚-tem sadežu.
3. Izbor najboljše konfiguracije: Primerjamo rezultate izračuna za vse možne konfiguracije in izberemo tisto, pri kateri Ivo pobere najmanj jabolk.
4. Ponovitev za vse pozicije, dokler ne porabimo vseh dodatnih jabolk ali dokler ne dosežemo konca zaporedja.

## Psevdokoda

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen

Slika 1 - psevdokoda metode grobe sile

## Robni primeri

1. Vse hruške, ni jabolk na začetku: Najboljša strategija bi bila, da Iva ne vrine nobenega jabolka, saj že obstoječe stanje zagotavlja, da Ivo ne more pobrati nobenega jabolka.
2. Vse so že jabolka: Ne glede na to, kje Iva vrine dodatna jabolka, bo Ivo vedno pobral največje možno število jabolk.
3. Ekstremne vrednosti 𝑘 in 𝑚: Kadar je 𝑘 zelo visoko ali 𝑚 zelo nizko, moramo posebej paziti na to, kako algoritem ravna z velikimi števili, saj lahko prihaja do znatnih časovnih zamikov pri izračunu.

Ta pristop z metodo grobe sile omogoča, da eksaktno analiziramo vsako možno konfiguracijo, vendar je lahko zelo časovno potraten, še posebej pri večjih vrednostih 𝑛 in 𝑘. Optimizirani algoritmi ali hevristike bi lahko zagotovili boljše rešitve v praksi, še posebej za večje primere.

# OPIS REŠITVE S PRIMERNO STRATEGIJO: DINAMIČNO PROGRAMIRANJE

## Reševanje z dinamičnim programiranjem

Rešitev problema z uporabo dinamičnega programiranja omogoča optimalno in učinkovito iskanje rešitev, zlasti pri velikih vrednostih n in k.

Algoritem deluje na naslednji način:

1. Inicializacija dinamične tabele: V dinamični tabeli dp shranjujemo minimalno število jabolk, ki jih Ivo pobere, pri določenem številu vrivanj na določenem koraku zaporedja. Path tabela se uporablja za rekonstrukcijo odločitev o vstavitvi. Struktura dp je velikosti (n+1) \* (k+1), saj nam takšna tabela generira vse možne pozicije. Vrednosti dp tabele so inicializirane na INT\_MAX. Struktura path je prav tako velikosti (n+1) \* (k+1). Njene vrednosti so inicializirane na -1.
2. Izpolnjevanje dinamične tabele:

Za vsako možno pozicijo in število izbranih jabolk preveri, ali je bolje ne vstaviti jabolko ali vstaviti jabolko.

* **Brez vstavitve**: Če ne vstavimo jabolka, preverimo, ali je trenutni sadež na mestu, ki ga bo Ivo pobral (vsak m-ti sadež), in posodobimo tabelo dp glede na to, ali je sadež jabolko.
* **Z vstavitvijo**: Če se odločimo vstaviti jabolko, ponovno ocenimo, ali bi Ivo pobral to jabolko, če bi bilo na mestu m-tega sadeža in posodobimo dp tabelo.

Pri vsakem koraku posodobi dp matriko z optimalnimi vrednostmi in path matriko z informacijami o poteh.

1. Iskanje minimalnega rezultata in rekonstrukcija rešitve: Po izpolnitvi tabele iščemo minimalno število jabolk, ki jih lahko Ivo pobere. S pomočjo tabele path rekonstruiramo, kje vse so bila jabolka vstavljena.

## Psevdokoda

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen

Slika 2 - psevdokoda dinamično programiranje

## Robni primeri

1. Če je n enak 0, je treba preprosto vstaviti k jabolk, saj ni nobenega sadja za razmislek.
2. Če je k enak 0, potem ni treba vstavljati nobenega jabolka, zato je optimalno število izbranih jabolk enako 0.
3. Če je m enak 0, potem ne bo izbrano nobeno sadje.
4. Če je m enak 1, potem bo vsako jabolko izbrano, saj se izbira vrši na vsakem koraku.
5. Če je m večji od n, ne bo izbrano nobeno sadje.

Uporaba dinamičnega programiranja v tem problemu omogoča sistematično in učinkovito analizo vsake možne konfiguracije z optimalnim izračunom. Algoritem je še posebej učinkovit v primerjavi z metodo grobe sile, saj ne zahteva preizkušanja vsake možnosti posebej, ampak sistematično gradi rešitev z zagotavljanjem najboljšega možnega izida na vsakem koraku.

## Vizualizacija

Inicializacija dp in path tabele:

A black and white diagram

Description automatically generated

Slika 3 – Inicializacija tabel

Posodabljanje tabel po vsaki iteraciji:

A couple of squares with a black line

Description automatically generated with medium confidence

Slika 4 - Vizualizacija posodbaljanje tabel

Z vgnezdeno zanko (i, j) se pomikamo po 2D tabeli, kjer stolpec predstavlja j in vrstico predstavlja i. Sprememba odtenka predstavlja novo iteracijo j, nova barva pa predstavlja novo iteracijo i.

# ZGLED TEŽJE INSTANCE PROBLEMA

## Vhodni podatki

* n: 10000
* k: 45
* m: 3
* Zaporedje: prvih 200 znakov (za prikaz): 100011100111101000111110010101100010111111000100101100001011110101011111111011110101101001010011001111011101110110111111011011010101110101101110100000101001011000010100111101000110111101111011001011111110000000110010101010110101111111100000011110111001110111011100010000000110001010010010110100...

## Vizualizacija

Zaradi kompleksnosti in velikosti podatkov, bomo uporabili skripto, ki vizualizira spremembe v prvih nekaj vrsticah dinamične tabele dp, ki predstavlja minimalno število jabolk, ki jih Ivo pobere za vsak (i, j).

Težavnost te instance izhaja iz kombinacije visoke dolžine zaporedja (n=10000), relativno visokega števila možnih vstavitev jabolk (k=45) in pogostega intervala pobiranja (m=3). Vsako dodatno jabolko vpliva na več mest v dinamični tabeli, kar poveča število možnosti za analizo. Interval pobiranja m=3 pomeni, da Ivo pobira veliko sadežev, kar oteži Ivino nalogo minimiziranja števila pobranih jabolk.

Slika, ki vsebuje besede besedilo, posnetek zaslona, barvitost, vrstica

Opis je samodejno ustvarjen

Slika 5 - Vizualizacija prvih 50 znakov

Slika, ki vsebuje besede besedilo, posnetek zaslona, barvitost, vrstica

Opis je samodejno ustvarjen

Slika 6 - Vizualizacija prvih 200 znakov

# PREDSTAVITEV IN PRIMERJAVA REZULTATOV

## Opis testnih primerov instanc problema

Za analizo algoritmov smo uporabili serijo testnih primerov z različnimi dolžinami nizov sadja (od kratkih do zelo dolgih). Zaradi konsistentnosti, smo pri vsakem testnem primeru k nastavili na maksimalno velikost 50 (razen pri primerih, kjer je n < k, pri teh primerih je k = 5), da bo velikost tabel dp in path večja, parameter m smo pa nastavili na 3. Velikosti primerov glede na n: 10, 20, 30, 50, 100, 250, 500, 1000, 5000, 10000.

## Meritve časa in zasedenosti pomnilnika

Meritve časa izvajanja kažejo, da metoda grobe pri večjih instancah problema (n > 1000) ali večjem številu dovoljenih vstavitev jabolk (k > 20) postane praktično neuporabna zaradi eksponentnega naraščanja časa izvajanja. V primerjavi z njo algoritem dinamičnega programiranja vztrajno ostaja v polinomskem času, kar omogoča obvladljive izračune celo pri velikih vrednostih n in k.

Zasedenost pomnilnika je bila prav tako analizirana, pri čemer "dp" metoda uporablja več pomnilnika zaradi potrebe po shranjevanju vmesnih rezultatov v dinamični tabeli. Kljub večji porabi pomnilnika pa je prednost v hitrosti, ki jo "dp" prinaša, odločilna.

Na spodnjem grafu vidimo, kako čas izvajanja narašča za obe metodi. Črta "bruteforce" eksponentno naraste z naraščanjem velikosti problema, medtem ko črta za "dp" metode kaže polinomsko rast. To jasno kaže na prednost uporabe "dp" v praksi.

A graph with numbers and lines

Description automatically generated

Slika 7 - Primerjava rezultatov med grobo silo in dinamičnim programiranjem

Rezultati kažejo, da je metoda dinamičnega programiranja bistveno boljša izbira za reševanje problema, ko gre za večje instance. Ne samo, da dinamično programiranje omogoča reševanje večjih problemov v razumskem času, temveč tudi zmanjšuje tveganje za prekoračitev časovnih in prostorskih omejitev, ki jih grobe sile metoda redno dosega.

## Ocenjena časovna zahtevnost

* Groba sila: Časovna zahtevnost metode je eksponentna, običajno O(2n+k), kar pomeni, da čas izvajanja eksponentno raste z naraščanjem velikosti niza in števila vstavitev.
* Dinamično programiranje (DP): DP metoda običajno deluje v časovni zahtevnosti O(n\*k), kar je veliko bolj učinkovito, saj je zahtevnost linearno odvisna od velikosti problema in števila dovoljenih vstavitev.

# ZAKLJUČEK

Najbolj problematičen del naloge je bilo izvedeti, katera strategija daje optimalno rešitev, saj je nalogo možno rešiti na več različnih načinov in z različnimi strategijami. Prav tako je bilo težko vizualitizirati delovanje algoritma. Naša implementacija algoritma, ki uporablja prinicip dimačnega programiranja je učinkovita, saj najde optimalne rešitve v zelo dobrem času.

# LITERATURA

Alg. In teh. za učinkovito reševanje problemov – Dinamično programiranje (Damjan Strnad).