**Žaba**

(Žan PLAHUTA, Matjaž CENTRIH)

Šentrupert, januar 2022

VSEBINA

[UVOD 3](#_Toc92219928)

[Primer iz realnega sveta 3](#_Toc92219929)

[Vhodni podatki 3](#_Toc92219930)

[Omejitve vhodnih podatkov 3](#_Toc92219931)

[Primer 3](#_Toc92219932)

[OPIS REŠITVE PRIMERA Z GROBO SILO 4](#_Toc92219933)

[Groba sila 1 4](#_Toc92219934)

[Groba sila 2 5](#_Toc92219935)

[OPIS REŠITVE PRIMERA S PRIMERNO STRATEGIJO 6](#_Toc92219936)

[VIZUALIZACIJA REŠEVANJA S POMOČJO GRAFOV 7](#_Toc92219937)

[PRIMERJAVA REZULTATOV 7](#_Toc92219938)

[ČASOVNA ZAHTEVNOST 8](#_Toc92219939)

[ZAKLJUČEK 8](#_Toc92219940)

[LITERATURA 9](#_Toc92219941)

KAZALO SLIK

[Slika 1: Prvi skok 4](#_Toc92220077)

[Slika 2: Razred lokvanj 4  
Slika 3: ZgradiGlavnegaOčeta 4](#_Toc92220078)

[Slika 4: BF rešitev 1 4](#_Toc92220079)

[Slika 5: Najdaljša pot 5](#_Toc92220080)

[Slika 6: BF rešitev 2 5](#_Toc92220081)

[Slika 7: Iskanje najdaljše poti 5](#_Toc92220082)

[Slika 8: Optimalna rešitev 6](#_Toc92220083)

# UVOD

V velikem ribniku za vasjo plava v ravni vrsti n lokvanjev, ki so enakomerno razporejeni v ravno vrsto. Na bregu stoji žaba in pogleduje proti lokvanjem. Na vsakem lokvanju se pase nekaj muh; njih število je različno od lokvanja do lokvanja. Žaba se odžene z brega in pristane na k-tem lokvanju. Skakanje nadaljuje, pri čemer je dolžina dveh zaporednih skokov različna največ za 1 (hitrosti ne more spremeniti na brzino). Prav tako se ne sme ustaviti. Ko zmanjka lokvanjev, pristane v vodi in si z njo poplahne grlo. Največ koliko muh lahko poje? Opis problema.

## Primer iz realnega sveta

Enostaven primer iz realnega življenja bi lahko bila organizacija in sledenje zaužitim kalorijam, med različnimi obroki.

Po boju s hudo boleznijo smo izgubili kar nekaj kilogramov, zato želimo čim hitreje pojesti čim več kalorij, glede na naše že sestavljene jedilnike. Že pripravljene jedi pregledujemo po vrsti od prve do zadnje, vsaka od njih pa ima tudi že v naprej izračunano število kalorij. Jedi izbiramo glede na naš določen korak in se za vsako sproti odločamo katero je najbolje izbrati naslednjo in sicer tako, da bo vsebovala čim več kalorij. Izbira dveh zaporednih jedi ne sme biti različna za več kot K –1 ali K + 1, saj so jedi že v naprej pripravljene. Ko zmanjka jedi v vrsti smo pregledali vse jedi in ostane nam optimalna vrsta, s katero lahko zaužijemo največ kalorij.

N: število vseh obrokov, ki jih imamo na voljo,

K: prestop s katerim začnemo jemati jedi,

mi: kalorije posameznega obroka.

## Vhodni podatki

V prvi vrstici se nahajata celi števili n in k, ki sta ločeni s presledkom. Druga vrstica vsebuje n s presledki ločenih celih števil mi , kjer je mi število muh na i-tem lokvanju. (Prvi lokvanj je najbližji bregu.)

**Časovna omejitev:** *5 s*

**Pomnilniška omejitev:** *256 MiB*

## Omejitve vhodnih podatkov

• 1 ≤ n ≤ 10 000

• 1 ≤ k ≤ n

• 0 ≤ mi ≤ 1000

## Primer

**Vhod:**    
12 4    
1 0 1 2 7 12 1 10 2 1 2 1

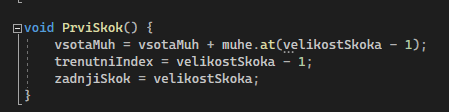
**Izhod:**    
14

**Komentar:**    
V optimalnem primeru žaba skoči s 4. na 8. lokvanj, nato na 11. lokvanj, za tem pa pristane v vodi. Tako poje 2 + 10 + 2 = 14 muh.

# OPIS REŠITVE PRIMERA Z GROBO SILO

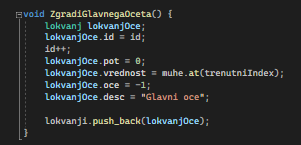
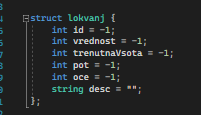
Najin problem sva rešila na 3 načine. Dvakrat z grobo silo in enkrat s primerno strategijo. Oba groba načina sta si podobna, vendar delujeta na drugačen način. Edina skupna stvar jima je prvi skok.

## Groba sila 1

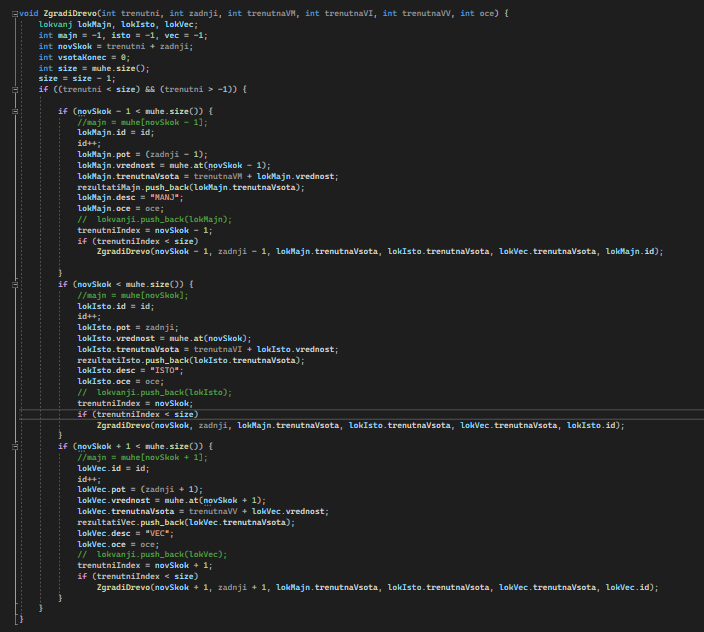


Slika : Prvi skok

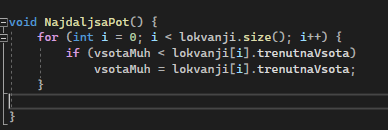
Pri prvem pristopu sva ustvarila “Struct” oziroma razred z imenom “lokvanj” v katerega sva zapisovala podatke o očetu, številu muh na lokvanju, ceno poti, trenutno število zaužitih muh itd. Za tem sva vse te lokvanje zapisovala v vektor in na koncu našla tistega z največjim številom zaužitih muh. Čisto na začetku po opravljenem prvem skoku pa zgradiva tudi očeta, ki je v bistvu zapis prvega skoka.



Slika : Razred lokvanj Slika : ZgradiGlavnegaOčeta



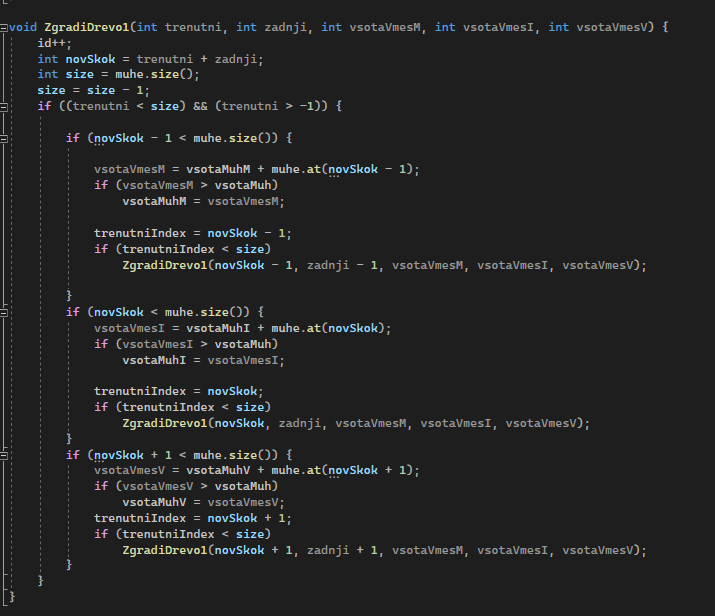
Slika : BF rešitev 1



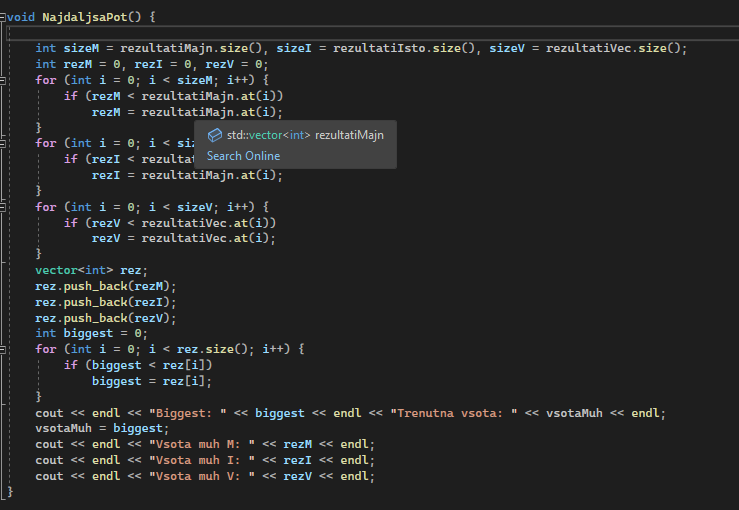
Slika : Najdaljša pot

## Groba sila 2

Drugi pristop deluje prav tako rekurzivno samo, da ne uporablja svojega razreda in vektorja, ampak samo sproti zapisujeva tri vrste vsot (k-1, k, k+1), katere na koncu med sabo preverjava in izpiševa največjo vrednost.



Slika : BF rešitev 2

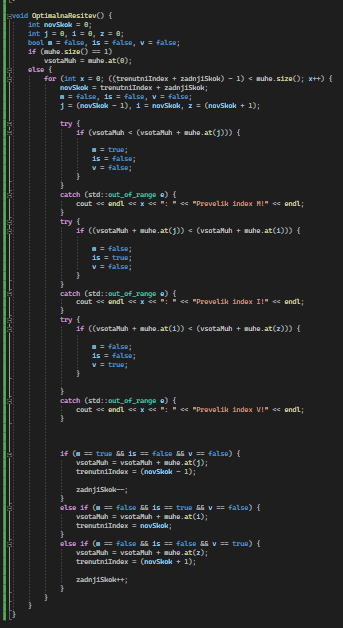


Slika : Iskanje najdaljše poti

Težava pri obeh dveh primerih pa je, da nastane preveč rekurzivnih klicev kar pomeni, da sistem vrne “Segmentation fault c++ vector”. Tako, da to nebi bila primerna pristopa za reševanja tega problema.

# OPIS REŠITVE PRIMERA S PRIMERNO STRATEGIJO

Primerna strategija za reševanje najinega problema bi bila dinamično programiranje. Podobno kot pri primeru z vrvmi. Najin program prejeme podatke, jih prebere in začne s iskanjem največje vsote. Metoda vsebuje 3 vrste if-ov, za k-1, k in k+1. Na podlagi česa veva, ali je treba skoke povečevati, manjšati ali ohraniti in ažurno posodabljava vsoto. Ta strategija je veliko hitrejša in manj kompleksna.



Slika : Optimalna rešitev

# VIZUALIZACIJA REŠEVANJA S POMOČJO GRAFOV

Kot je že razvidno iz grafa sva ugotovila, da je groba sila porabila več časa pri vseh rešenih primerih. Zato je bolj primerna optimalna rešitev za reševanje tega problema.

Pri reševanju problemov nobena od najinih rešitev ni presegla določeno mejo za izvajanje programa.

# PRIMERJAVA REZULTATOV

Rezultati so pri vseh opravljenih primerih bili enaki. Različen je bil čas opravljanja.

Vseh primerov pa nama ni uspelo preveriti, ker najin program ni deloval pravilno pri optimalni rešitvi.

# ČASOVNA ZAHTEVNOST

Časovna zahtevnost implementacije metode z grobo silo je O(log(n)), medtem ko je časovna zahtevnost implementacije s primerno strategijo enaka O(n).

Glede na postavljene zahteve je čas pri uporabi dinamičnega programiranja odvisen le od števila lokvanjev in velikosti skoka. Ker je največje število lokvanjev lahko 10 000 in če gledamo na n kot število števk, v tem primeru nmax = 10 000, lahko dobimo dodaten čas izvedbe glede na iskanje po tabeli za vsak skok.

# ZAKLJUČEK

Problema sva se prvo lotila idejo binarnih dreves, katero sva gradila z razredom lokvanj v katerem sva imela podatke (id, id\_oceta, st\_muh, trenutna\_vsota). Ker pa je bil ta pristop zamuden sva uporabila rekurzivno klicanje metode in sva vsoto gradila kar sproti oziroma 3 vsota, jih na koncu primerjala in nato izpisala največjo. Problem je tukaj nastal pri preveč rekurzivnih klicih, zato je program pri »težjih/ večjih« problemih vrnil napako in zaključil.

Ko sva začela uporabljati primerno strategijo se je uporaba spomina in časovna zahtevnost zmanjšala, vendar sva naletela na težavo, katere nisva znala odpraviti in tako nama tudi ni uspelo pripraviti rešitev, ki bi rešila vse podane probleme. Vsekakor pa sva svojo časovno zahtevnost spremenila iz linearne v O(n).

Sama rešitev pa je dokaj enostavna:

1. Branje datoteke
2. Pregled vseh možnih poti
   1. Izbira skoka na k-1, k, k+1
   2. Opravljanje skoka
   3. Ponovitev do konca ribnika
3. Izbira tiste najboljše (čim več muh)
4. Izpis

# LITERATURA

<https://estudij.um.si/pluginfile.php/514921/mod_resource/content/5/04c%20-%20ATURP%20-%20Dinami%C4%8Dno%20programiranje.pdf> Dostop: 04.01.2022

<https://estudij.um.si/pluginfile.php/673583/mod_resource/content/2/seminarske_naloge_ATURP_VS_2021_2022.pdf> Dostop: 04.01.2022

<https://feri.um.si/> Dostop: 04.01.2022