Univerza v Ljubljani
Fakulteta za matematiko in fiziko

SCHEDULING TRIANGLES

Finančni praktikum

Avtorja:
Blaž Arh, Matic Matušek

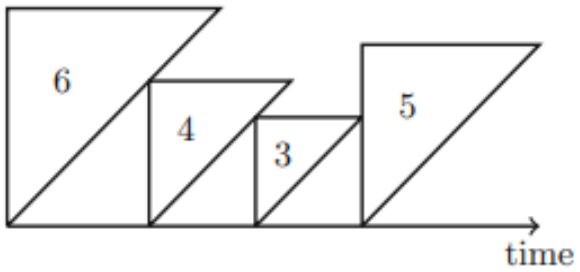
Ljubljana, 2022

Kazalo

1 Uvod

1.1 Navodilo naloge

Imamo n pravokotnih enakokrakih trikotnikov, ki so določeni z dolžino kraka d_i za $i = 1, 2, ..., n$ in pravim kotom med krakoma. Trikotnike postavimo, z nogo na x-os, na sledeči način:



Trikotniki morajo biti postavljeni tako, da je hipotenuza trikotnika naraščajoča glede na x-os, natančneje naklon hipotenuze je enak 1. Trikotnike želimo postaviti tako, da je dolžina teh trikotnikov najkrajša. Trikotniki se med seboj ne smejo prekrivati, prav tako pa mora biti krak noge pravokoten na x-os.

1.2 Uporabnost

Pomembno je, da znamo ta matematični problem interpretirati v praksi. Trikotnike lahko interpretiramo kot opravila na našem urniku. Radi bi minimizirali trajanje našega urnika, medtem ko bi radi prioritizirali bolj kritična oziroma pomembna opravila. Tako je torej:

- pravokotni enakokraki trikotniki = opravila,
- dolžina kraka d_i = pomembnost opravila.

Želimo si sestaviti urnik iz n opravil. Naj bo d_i pomembnost i-tega opravila. Naš cilj je sestaviti čim krajši urnik. Želimo, da se pomembnejša opravila izvedejo, v primeru zakasnitve le teh, pa lahko manj pomembna opravila izpustimo. Večji kot je trikotnik, bolj pomembno je opravilo. Kot je razvidno iz zgornje slike, lahko opravila z manjšo pomembnostjo vstavimo pod opravila z večjo pomembnostjo. To pomeni, da manjše trikotnike vstavljamo pod večje. Torej, če pride do zakasnitve pomembnejšega opravila (večjega trikotnika), manj pomembno opravilo izpustimo (manjši trikotnik).

2 Reševanje problema s pomočjo različnih algoritmov

Dürr et al. [1] za reševanje problema opišejo Greedy algoritem. Pred Greedy algoritmom sva se odločila sprogramirati brute force algoritem, ki je nekoliko manj težaven, je pa precej časovno zahteven in sicer $O(n!)$. Nato sva napisala Greedy algoritem, ki ima nizko časovno zahtevnost, vendar ne vrne optimalnega rezultata. Zato sva za konec napisala tudi optimalen algoritem, kjer sva uporabila celoštevilsko linearno programiranje, ki je malo manj časovno zahtevno kot brute force. Torej najbolj časovno zahteven algoritem je brute force, takoj za njem pa celoštevilsko linearno programiranje. Najmanj časovno zahteven algoritem je Greedy algoritem, ki rešitev poišče v polinomskem času $O(n^2)$, vendar ne vrne optimalne rešitve.

2.1 Brute force algoritem

2.1.1 Opis algoritma

Imamo slovar z n trikotniki in seznam, sestavljen z $n!$ podseznami dolžine n (vse možne permutacije vrstnega reda trikotnikov). Potem za vsako permutacijo izračuna dolžino urnika in jo zabeleži v nov seznam. Ko pregleda vse permutacije, poišče prvo možno permutacijo, ki vrne optimalno rešitev (minimalno dolžino v seznamu vseh dolžin) in jo vzame za rešitev.

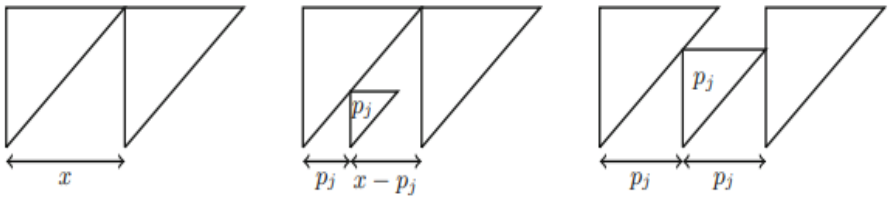
2.1.2 Psevdokoda

```
1.     Vhod: trikotniki
2.     n = length(trikotniki)
3.     dolzina_urnika = []
4.     perutacije = permutations( [0, 1, ..., n-1] )
5.     for i in permutacije
6.           for j in 0:(n-1)
7.             trikotnik["vrstni_red"][j] = permutacija[j]
8.       dolzina_urnika.append( dolzina(trikotniki) )
9.     index = dolzina_urnika.index( min(dolzina_urnika) )
10.    for i in 0:(n-1)
11.      trikotniki[i]["vrstni red"] = permutacije[index][i]
12.    Izhod: trikotniki
```

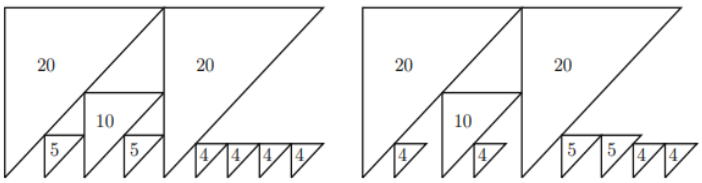
2.2 Greedy algoritem

2.2.1 Opis algoritma

Greedy algoritem je kateri koli algoritem, ki rešuje problem tako, da na vsakem koraku naredi lokalno optimalno izbiro. Prvi trikotnik razvrsti na $x = 0$, kar ustvari prostor pod trikotnikom 1. Nato vsak trikotnik $j = 2, ..., n$ postavi na mesto, kjer je dolžina že postavljenih trikotnikov najkrajša. Če je več enakih prostorov, izbere prvega povrsti. Če ima izbrani prostor pod trikotnikom dolžino l in se začne v času x_i, potem je trikotnik j postavljen na mesto $x_j = x_i + d_j$. Če je $2d_j \geq l$, potem se vsi trikotniki k, za katere je $x_k \geq x_j$ zamaknejo za $2d_j - l$, da se ne prekrivajo. Opisano dogajanje prikazuje spodnji primer, kjer je $x = l$ in $p_i = d_i$.



Pri številnih problemih Greedy algoritem ne poišče optimalne rešitve in nič drugače ni pri našem problemu. Na naslednji sliki prikažemo primer, ko Greedy ni optimalen. Greedy algoritem (desna slika) vrne dolžino 42 in ne vrne optimalne dolžine, ki je 40 (leva slika).



Hitrost Greedy algoritma je polinomska. Da se pokazati, da je aproksimacijsko razmerje zgornje meje točnosti Greedy algoritma 1.5 optimalnega časa. Torej, če je optimalni čas določenega primerna 40 časovnih enot, potem Greedy pokaže največ 60 časovnih enot.

2.2.2 Psevdokoda

```
Vhod: trikotniki
n = length(trikotniki)
trikotniki_lokalno = dict()
for i in 0:(n-1)
    trikotniki_lokalno[i] = trikotniki[i]
    m = length(trikotniki_lokalno)
    lokalna_dolzina_urnika = []
    for j in 0:(m-1):
        trikotniki_lokalno[i]["vrstni_red"] = j
        *popravi_mesta_ostalim_trikotnikom
        lokalna_dolzina_urnika.append( dolzina_urnika(trikotniki_lokalno) )
    index = lokalna_dolzina_urnika.index( min(lokalna_dolzina_urnika) )
    trikotniki_lokalno[i]["vrstni_red"] = index
    *popravi_mesta_ostalim_trikotnikom
Izhod: trikotniki_lokalno
```

Funkcija `popravi_mesta_ostalim_trikotnikom` nastavi mesta tistih trikotnikov, ki so v slovarju `trikotniki_lokalno`, tako da so še vedno urejeni po vrsti in vsak trikotnik ima enolično določeno mesto. V prvi `for` zanki se z vsakim korakom doda nov trikotnik v slovar `trikotniki_lokalno`, potem naslednja `for` zanka pregleda vsa mesta, kamor lahko postavi na novo dodani trikotnik in si dolžine urnika beleži v seznam `lokalna_dolzina_urnika`. Po zaključeni drugi `for` zanki, se izbere optimalen oziroma najkrajši čas urnika in se za ta določen čas na trikotnike aplicira njihov vrstni red. Ko se zaključijo vse zanke, imamo v slovarju `trikotniki_lokalno` dodane vse vhodne trikotnike, ki so bili postavljeni na lokalno optimalno mesto.

2.3 Celoštevilsko linearno programiranje

V spodnjem primeru privzamemo, da položaj noge oziroma x-koordinato označuje s_i, t pa označuje dolžino urnika. Spremenljivka $x_{i,j}$ je definirana na naslednji način

$$x_{i,j} = \begin{cases} 1, & \text{če } i\text{-ti trikotnik za } j\text{-tim trikotnikom} \\ 0, & \text{sicer} \end{cases}$$

in lahko zavzame le vrednosti 0 in 1, medtem ko s_i in t zavzemata pozitivne realne vrednosti. Pogoj $|s_i - s_j| \geq \min\{d_i, d_j\}$ je glavni pogoj v najinem linearnem programu, ostali pogoji pa več ali manj služijo omejevanju spremenljivk.

S pogojem $x_{i,j} + x_{j,i} = 1$ povemo, da mora veljati, da je j-ti trikotnik pred i-tim ali pa, da je i-ti trikotnik pred j-tim. T označuje $\sum_{i=1}^{n} d_i$ in se uporabi zato, da v primeru, ko je $s_j < s_i$ pogoji 2 v spodnjem zgledu vseeno veljajo. Linearni program bi v matematičnem zapisu izgledal tako:

$$\min t$$
$$\text{p.p.}$$
$$1.\ \forall i : 0 \leq s_i \leq t - d_i$$
$$2.\ \forall i, j \text{ kjer } i \neq j : s_j - s_i \geq \max\{d_i, d_j\} - T \cdot x_{i,j}$$
$$3.\ \forall i, j \text{ kjer } i \neq j : x_{i,j} + x_{j,i} = 1$$

2.3.1 Psevdokoda

```
1.      Vhod: seznam_dolzin_trikotnikov
2.      T = sum(seznam_dolzin_trikotnikov)
3.      t = new_variable()
4.      s = new_variable()
5.      x = new_variable(binary = True)
6.      set_objective(t, maximization = False)
7.      for i, d in enumerate(seznam_dolzin_trikotnikov)
8.          add_constraint(s[i] >= 0)
9.          add_constraint(s[i] + d <= t)
10.         for j, e in enumerate(seznam_dolzin_trikotnikov[i+1:], i+1):
11.             m = min(d, e)
12.             add_constraint(x[i, j]+x[j, i] = 1)
13.             add_constraint(s[i] - s[j] >= m - T*x[j, i])
14.             add_constraint(s[j] - s[i] >= m - T*x[i, j])
15.     Izhod: seznam s
```

V prvih 6-ih vrsticah definiramo spremenljivke in minimizacijsko funkcijo. V `for` zankah pa podamo vse pogoje. Izhodni podatek je seznam `s`, ki vsebuje položaje nog trikotnikov.

2.4 Generiranje podatkov

Pri generiranju podatkov sva potrebovala zgenerirati n dolžin krakov d_i. Če sva hotela, da je dolžina d_i celoštevilska, sva si pomagala s funkcijo `random.randint`. Če pa sva želela, da je dolžina kraka d_i realno število, pa sva si pomagala s funkcijo `random.uniform`. Funkcijama je potrebno podati zgornjo in spodnjo mejo, torej interval iz katerega izbirata naključne vrednosti.

V zgledu na koncu poročila sva za spodnjo mejo izbrala število 1, za zgornjo mejo pa število 50. Funkcija, ki generira podatke ima parametra a in b, ki omogočata izbiro katerega koli intervala, tako da velja $d_i \in [a, b]$; $a, b \in \mathbf{R}^+$ in $a \leq b$. Funkcija, ki generira podatke, dopušča tudi izbiro celoštevilskih podatkov.

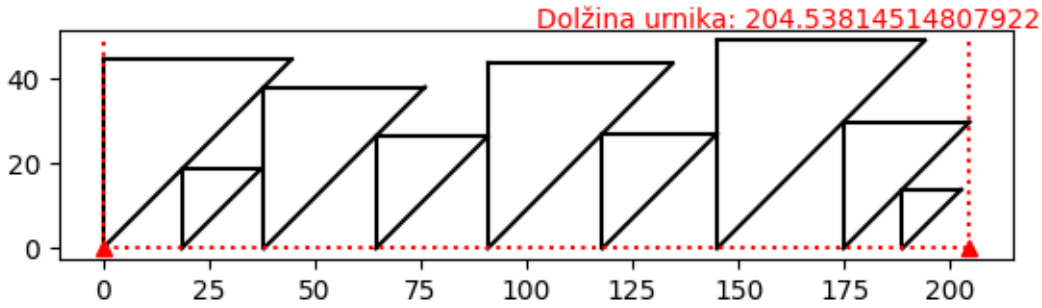
2.5 Predstavitev rezultatov

Pri prikazu delovanja vseh treh algoritmov si bomo pomagali z zgledom. Za testno množico sva izbrala 9 enakokrakih trikotnikov z naslednjimi dolžinami.

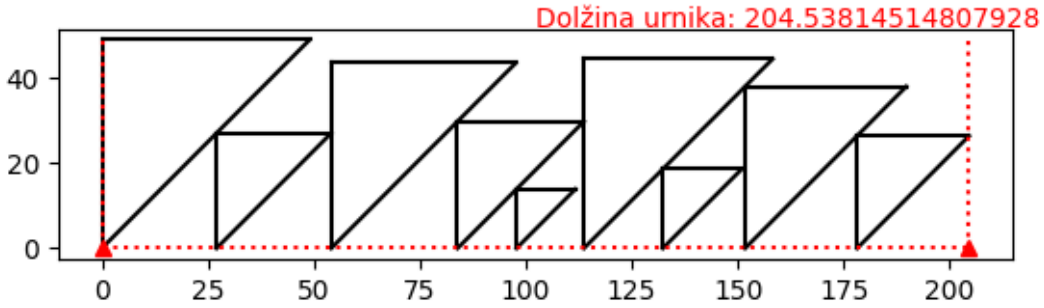
```
test = naredi_trikotnike(
            9,
            [
            44.513033703890834,
            18.613265453372076,
            27.04233495394404,
            29.803779538038164,
            38.01467639495788,
            43.53220098011497,
            13.90897209250403,
            26.415619884578483,
            49.01780933958118
            ]
        )
```

Rezultati, ki jih vrnejo algoritmi so sestavljeni iz dolžine celotnega urnika in razporeditve trikotnikov. Rezultati zgleda so sledeči:

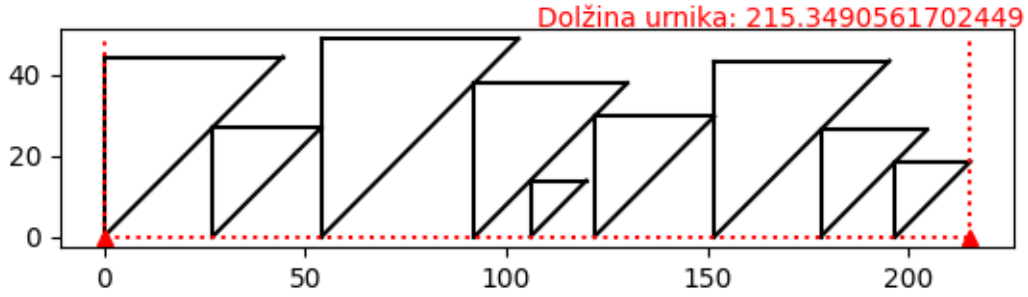
- Brute force algoritem



- Celoštevilsko linearno programiranje



- Greedy algoritem



Sklep zgleda: Opazimo, da tako brute force algoritem in CLP vrneta enako dolžino, kar je tudi pričakovano, saj oba algoritma poiščeta optimalno rešitev. Razlikujeta pa se po razporeditvi trikotnikov. Pri zgornjem zgledu je 72 različnih razporeditev trikotnikov, ki podajo najmanjšo razdaljo oz. optimalno rešitev. Z uporabo greedy algoritma do optimalne rešitve nismo prišli. Velikost napake je cca $5,29$ %.

Literatura

[1] Dürr, C., Hanzálek, Z., Konrad, C. et al. The triangle scheduling problem. J Sched 21, 305-312 (2018). https://doi.org/10.1007/s10951-017-0533-1