

Kopica

Računalništvo 1

Avtor:

Gal Zakrajšek

Ljubljana, 2020

Kazalo

[Uvod 3](#_Toc29059646)

[Kopica in podatkovna struktura drevo 3](#_Toc29059647)

[Definicija 4](#_Toc29059648)

[Operacije 4](#_Toc29059649)

[Metoda vstavi(element) 4](#_Toc29059650)

[Zgled vstavljanja in plavanja navgor 5](#_Toc29059651)

[Metoda odstrani() 6](#_Toc29059652)

[Zgled odstranjevanja in plavanja navdol 6](#_Toc29059653)

[Metoda sestavi(elementi) 8](#_Toc29059654)

[Zgled sestavljanja iz manjših kopic 8](#_Toc29059655)

[Uporaba 10](#_Toc29059656)

[Heapsort 10](#_Toc29059657)

[Ostale uporabe 11](#_Toc29059658)

[Časovna zahtevnost 11](#_Toc29059659)

[Viri 12](#_Toc29059660)

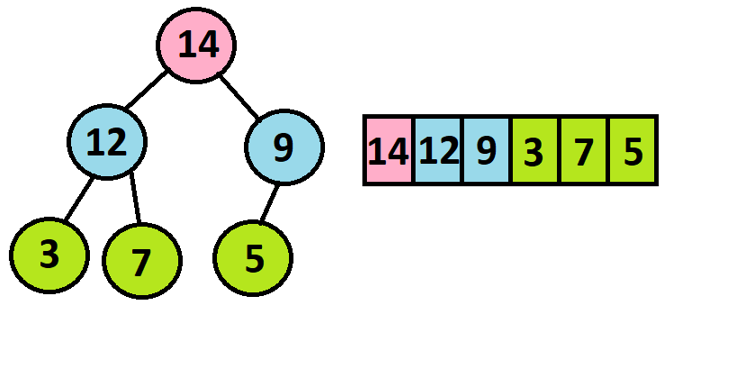
# Uvod

Podatkovna struktura kopica se uporablja v primerih, ko je potrebno v nekem naboru elementov poiskati element z največjo ali najmanjšo vrednostjo. Poznamo dve vrsti kopic in sicer minimalno, ki jo uporabimo takrat, ko bomo potrebovali najmanjši podatek med vsemi in pa maksimalno, s katero bo najlažje najti največji podatek. V predstavitvi bodo vse razlage in primeri metod narejeni za maksimalno kopico, saj se v primeru če imamo minimalno kopico spremeni zelo malo.

# Kopica in podatkovna struktura drevo

V predstavitvi bomo kopico predstavljali kot levo poravnano dvojisko drevo, saj si tako lažje predstavljamo vse njene lastnosti. V resnici pri kopici ne gre za drevo ampak za podatkovno strukturo, ki ima nekatere značilnosti dvojiskega drevesa, saj se tudi tukaj pojavijo očetje, sinovi ter listi. Zato si prvo poglejmo predstavitev drevesa z tabelo.

Ko predstavljamo neko drevo s tabelo, ga v njo zapisujemo po nivojih. Na prvo mesto zapišemo podatek v korenu, na drugo in tretje mesto pa levega in desnega sina od korena. Nato nadaljujemo po nivojih, zato bi na četrto mesto v tabeli prišel skrajno levi element v tretjem nivoju.



Slika 1: Prikaz predstavitve drevesa z tabelo.

Tak način hranjenja podatkov je uporaben zato, saj lahko v tabeli preko indeksov hitro dostopamo do sina nekega elementa ali pa do njegovega očeta. Če želimo dostopati do sinov in očetov uporabimo naslednje formule:

Pri tem prdstavlja indeks nekega elementa v tabeli.

# Definicija

V literaturah se pojavita dve različni definiciji kopice. Med seboj se razlikujeta o poravnanosti drevesa in sicer prva definicija pravi, da je kopica dvojiško drevo, ki ima podatek v korenu večji od podatkov v sinovih in da sta obe poddrevesi tudi kopici. Medted ko druga definicija pravi enako kot prva le, da imamo dodatni pogoj in sicer, da je dvojiško drevo levo poravnano. V primeru da ni levo poravnano pa rečemo, da ima samo kopično lastnost.

V predstavitvi bomo uporabljali drugo definicijo in sicer:

*Kopica je podatkovna struktura, ki temelji na drevesu, ki je levo poravnano in ima podatek v korenu večji od podatkov v sinovih. Oba sinova pa sta tudi kopici.*

# Operacije

Ko govorimo o implementaciji kopice, lahko pričakujemo, da na njej delujejo naslednje operacije:

* pripravi() – metoda ustvari novo podatkovno strukturo kopica
* prazna() – vrne True, če je kopica nad katero izvedemo metodo prazna, v nasprotnem primeru vrne False.
* največji() / najmanjši() – vrne element v kopici, ki je največji /najmanjši.
* vstavi(element) – metoda vstavi v kopico na pravo mesto podani element.
* odstrani() – iz kopice odstrani element, ki je največji ali najmanjši odvisno od tipa kopice.

V praksi pa se velikokrat uporablja tudi metoda, ki ni osnovna a je zelo uporabna in sicer, da iz podane tabele elementov sestavi veljavno kopico.

* Sestavi(tabela\_elementov) – v kopico doda vse elemente iz tabele\_elementov.

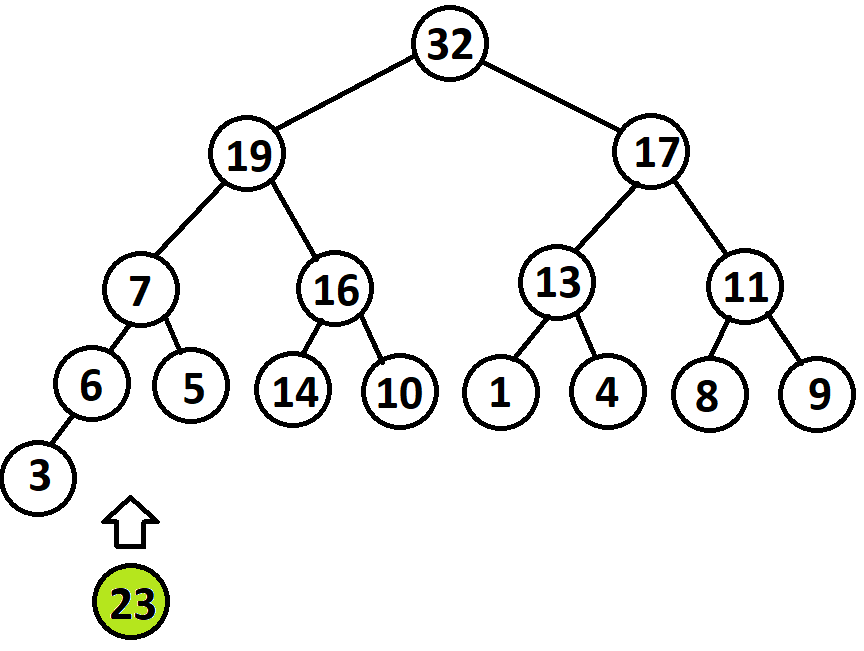
V nadaljevanju si bomo pogledali bolj natančno, kako delujejo osnovne metode. Metode pripravi, prazna in pa največji so čisto preproste, zato jim nebomo posvetili posebne pozornosti.

# Metoda vstavi(element)

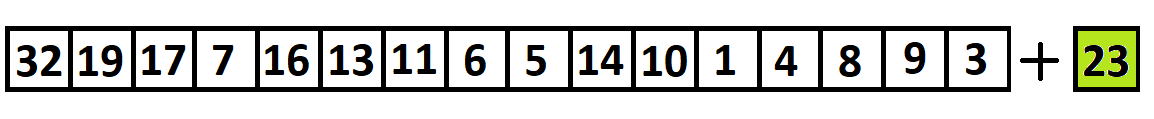
Metoda poskrbi, da se element vstavi v kopico na pravo mesto. To naredi tako, da na konec vstavi nov element, vendar se pri tem lahko zgodi, da izgubimo lastnost kopice, saj lahko vstavimo element, ki je večji od svojega očeta. Zato se po dodajanju elementa izvrši še tako imenovan algoritem plavanje, ki poskrbi, da element splava na pravilno mesto v kopici. Algoritem je dokaj preprost, nov element primerjamo z očetom in v primeru, da je oče manjši, ju zamenjamo. Ta postopek nato ponavljamo in vedno gledamo element, ki plava navzgor in naslednjega očeta. Ustavimo se takrat, ko naletimo na očeta, ki je večji od plavajočega elementa ali pa ko element pride v koren. Ker se element premika navzgor po nivojih, rečemo temu plavanje navzgor.

## Zgled vstavljanja in plavanja navgor

Za primer bomo vzeli naslednjo kopico in ji dodali element z podatkom 23.



Slika 2: Vstavljanje elementa 23 v kopico predstavljeno z drevesom.

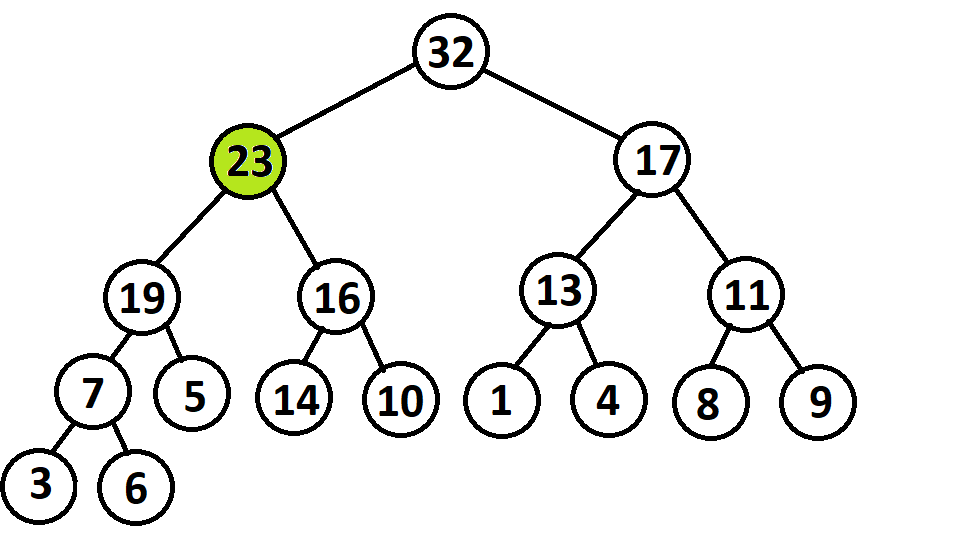


Slika 3: Vstavljanje elementa 23 v kopico predstavljeno z tabelo.

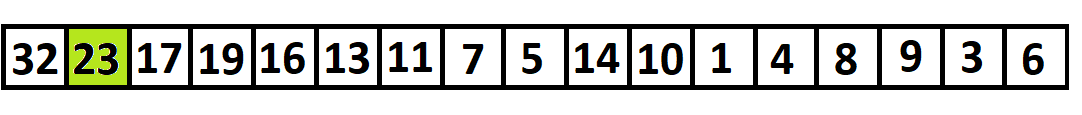
Sprva bomo vstavili podatek na zadnje mesto v kopici, kot je prikazano na zgornji sliki. Potem moramo poskrbeti še, da podatek splava navgor na pravilno mesto. Če pogledamo sliko ugotovimo, da mora podatek splavati na mesto, kjer je zdaj vrednost 19. Sedaj si poglejmo vse primerjave in zamenjave, ki se zgodijo pri plavanju.

1. Primerjamo 23 in 6, ker je 23 večja se podatka zamenjata.
2. Primerjamo 23 in 7, ker je 23 večja se podatka ponovno zamenjata.
3. Primerjamo 23 in 19, ker je 23 večja se podatka ponovno zamenjata.
4. Primerjamo 23 in 32, ker je 32 večje pomeni, da je zdaj naš plavajoči element na pravilnem mestu.

Po naštetih zamenjavah smo ponovno vzpostavili kopično lastnost. V primeru, da bi imeli minimalno kopico bi se podatki menjavali, v primeru da bi bil plavajoči element manjši od očeta. Končni rezultat prikazuje naslednja slika.



Slika 4: Prikaz rešitve vstavljanja in plavanja v kopici predstavljeno z drevesom.



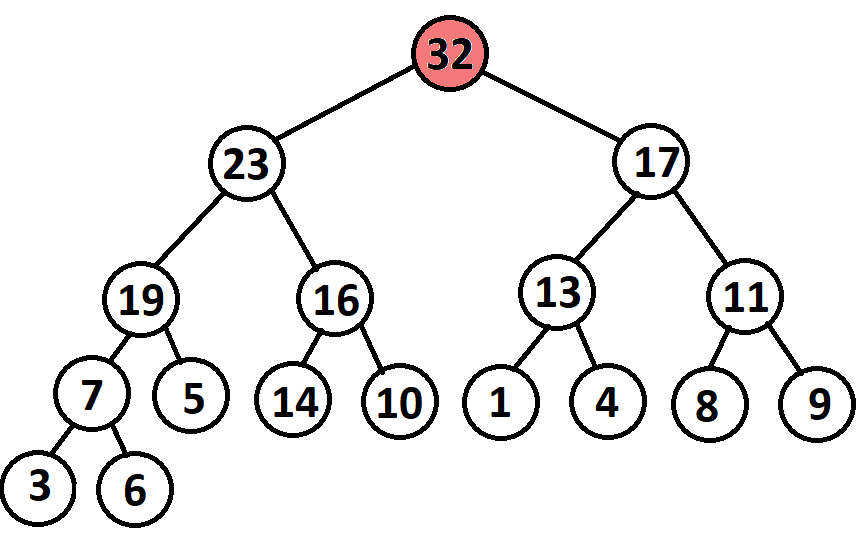
Slika 5: Prikaz rešitve vstavljanja in plavanja v kopici predstavljeno z tabelo.

# Metoda odstrani()

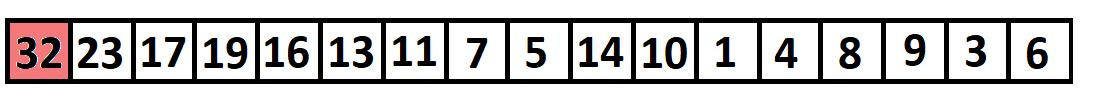
Pri metodi odstrani velja, da vedno odstranimo element v korenu oziroma največji element. Prvo vzamemo podatek iz zadnjega lista v drevesu in z njim prepišemo koren. Potem zadnji list izbrišemo. Zdaj opazimo, da smo porušili kopično lastnost, saj element v korenu ni več največji element. Tukaj ponovno uporabimo algoritem plavanja. V tem primeru bomo element spuščali po nivojih, zato temu rečemu plavanje navzdol. Algoritem deluje tako, da pogledamo plavajoči element, ki je trenutno v korenu in pa oba sinova. Ugotovimo kateri sin je večji in nato primerjamo koren in večjega sina. Če je podatek v korenu manjši, zamenjamo večjega sina in ta koren, potem pogledamo plavajoči element in ponovno oba sinova. V primeru, da je plavajoči element večji od večjega sina ali pa da je prišel v najnižji nivo in postal list postopek prekinemo, saj smo povrnili kopično lastnost.

## Zgled odstranjevanja in plavanja navdol

Kot primer bomo vzeli naslednjo kopico.



Slika 6: Brisanje elementa iz kopice predstavljene z drevesom.

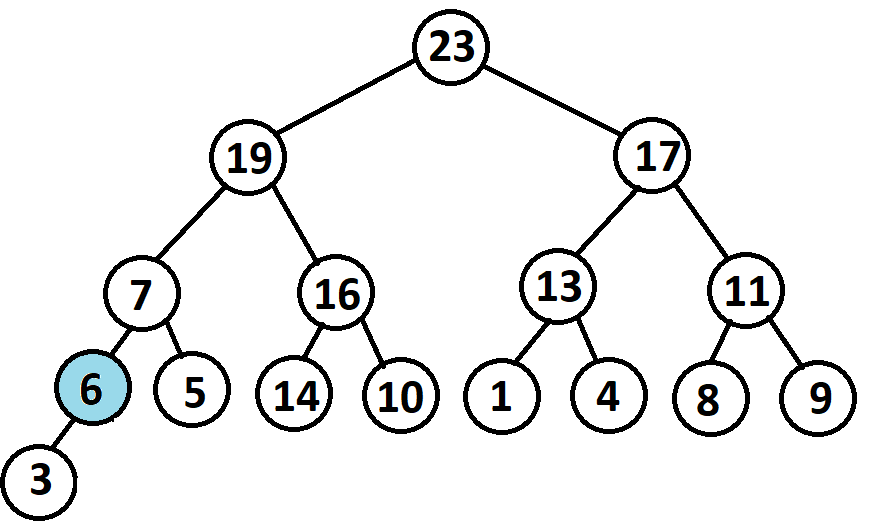


Slika 7: Brisanje elementa iz kopice predstavljene z tabelo,

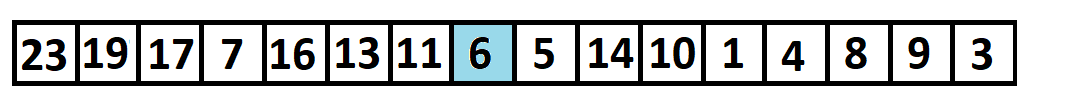
Najprej prepišemo koren z zadnjim elementom v kopici, ki je v našem primeru 6. Nato izbrišemo zadnji element. Zdaj je potrebno samo vrniti kopično lastnost, kar počnemo z plavanjem navdol. Primerjanja in zamenjave, ki se zgodijo so naslednja.

1. Primerjamo koren in večjega od sinov, v našem primeru sta to 6 in 23. Ker je 6 manjše zamenjamo podatka
2. Primerjamo plavajoči podatek in večjega od naslednjih sinov, to sta 6 in 19. Ker je 6 manjše ponovno zamenjamo podatka.
3. Primerjamo 6 in večjega od sinov 7. Ker je 6 manjše ponovno zamenjamo podatka.
4. Primerjamo 6 in pa edinega sina, ki je ostal. Ker je 6 večje od 3 vemo, da je podatek na pravem mestu.

Tako smo ponovno zagotovili kopično lastnost. V primeru minimalne kopice, bi bilo ravno obratno in sicer če bi bil pravajoči člen večji od manjšega sina, bi ju zamenjali. Naslednja slika prikazuje rezultat po odstranitvi. Modri element je element, s katerim smo prepisali podatek v korenu in ga nato z plavanjem spustili na pravo mesto.



Slika 8: Prikaz končnega rezultata odstranjevanja največjega elementa v kopici predstavljeni z drevesom.



Slika 9: Prikaz končnega rezultata odstranjevanja največjega elementa v kopici predstavljeni z tabelo.

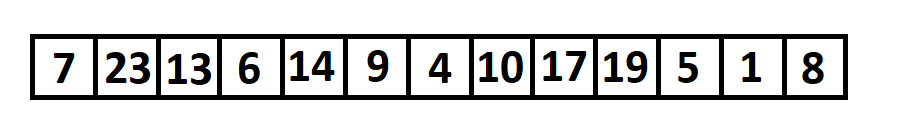
# Metoda sestavi(elementi)

Cilj metode sestavi je, da iz podane tabele elementov metoda sestavi in vrne kopico. To bi lahko naredili na preprost način in sicer, najprej bi ustvarili kopico in se nato z zanko zapeljali čez vse elemente ter pri tem uporabljali kopično osnovno metodo vstavi. Časovna zahtevnost bi bila v tem primeru .

Izkaže pa se, da v primeru ko imamo že na začetku podane vse podatke, lahko kopico ustvarimo na hitrejši način in sicer z sestavljanjem iz manjših kopic. Ideja je, da začnemo od zadaj in usvarimo več manjših kopic ter jih nato povezujemo še z novimi elementi in na koncu dobimo ponovno eno kopico z vsemi elementi.

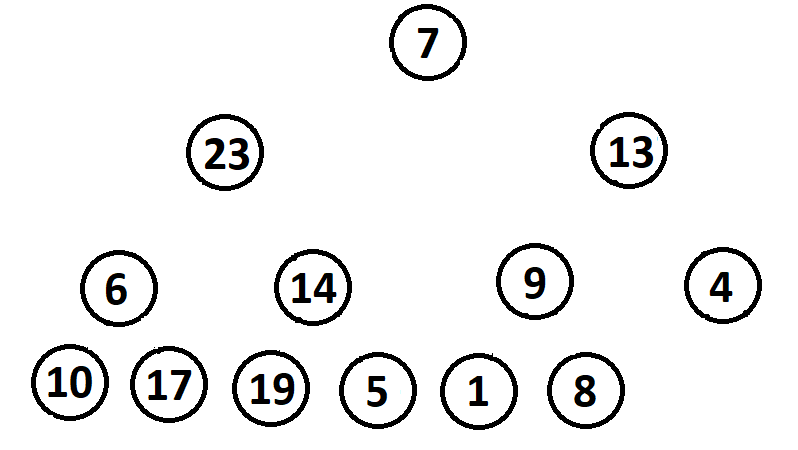
## Zgled sestavljanja iz manjših kopic

Za primer bomo vzeli naslednjo tabelo elementov:



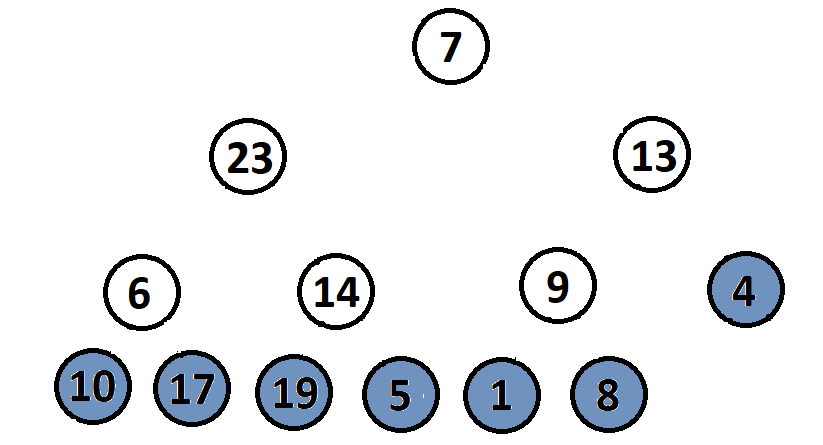
Slika 10: Tabela elementov za zgled sestavljanja iz manjših kopic.

Za lažjo predstavo bomo sedaj vozle razporedili v strukturo drevesa.



Slika 11: Elementi za zgled predstavljeni v obliki drevesa.

Postopek začnemo tako, da sprva ugotovimo, koliko listov bo imela končna kopica. Iz slike lahko razberemo, da bo imela končna kopica 8 listov in sicer z podatki: 8, 1, 5, 19, 17,10 in 4. V programu, kjer slike nimamo to naredimo tako, da kazalec postavimo na konec tabele in se premikamo nazaj toliko časa, dokler ne pridemo do elementa, ki ima vsaj enega sina. To preverjamo tako, da zmanjšujemo indeks toliko časa, dokler je večji od velikosti cele tabele. S tem smo našli prvi element, ki ima sina. Sedaj lahko vse liste proglasimo za kopice z enim elementom. Tako imamo trenutno 6 kopic, ki so na naslednji sliki obarvane z modro in pa ostale elemente zaenkrat še neorgarnizirane.



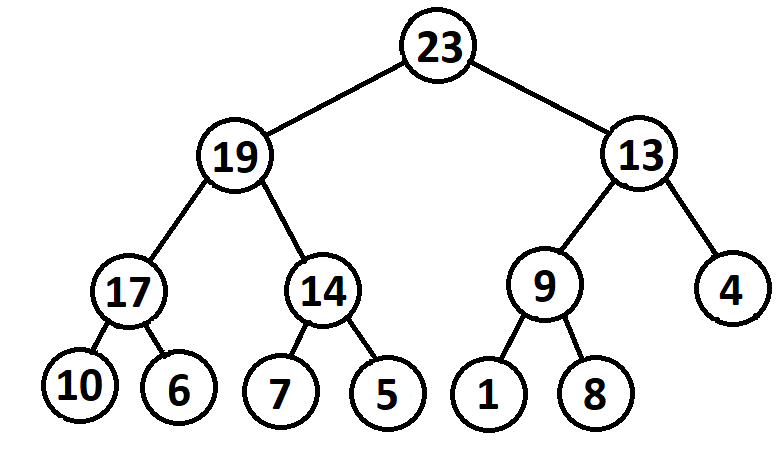
Slika 12: Označene kopice in ostali elementi.

Začnemo z naslednjim delou postopka in sicer premaknemo se na prvi element, ki ni več list in ga spremenimo v kopico ter za levega in desnega sina povežemo zadnje dve kopici. V našem primeru vzamemo podatek 9 in mu za levega sina podamo kopico z elementom 1 in za desnega sina kopico z elementom 8. Potem poskrbimo, da podatek v korenu splava na pravo mesto in da s tem povrnemo kopično lastnost. V našem primeru je kopična lastnost že izpolnjena.

Sedaj to ponovimo še za ostale elemente in sicer po naslednjem vrstnem redu:

1. Element 14 bo imelo levega sina kopico z elementov 19 in desnega kopico z elementom 5, pri tem se bosta zaradi plavanja navzdol zamenjali vrednost 14 in 19.
2. Element 6 bo imelo levega sina 10 in desnega 17, pri tem se zamenjata vrednosti 6 in 17.
3. Element 13 bo imelo levega sina kopico z korenom 9 in desnega kopico z korenom 4, kopica ima že takoj kopično lasnost kar pomeni, da plavanja ne potrebujemo.
4. Element 23 bo imel levega sina kopico, ki je nastala v 2. točki in desnega sina kopico, ki je nastala v 1. točki. Pri tem do plavana ne pride, saj ima 23 oba sinova manjša od sebe.
5. Na zadnje povežemo še element 7, ki bo imel za levega sina kopico, ki nastane v točki 4. in za densega sina kopico, ki nastane v točki 3. Po tem bo prišlo tudi do plavanja in sicer 7 se bo najprej zamenjala z 23, nato se bo zamenjala še z 19 in na koncu še z 14.

Končen rezultat po metodi je sledeč:



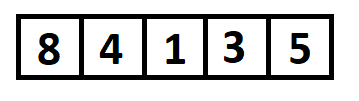
Slika 13: Rezultat metode sestavi iz manjših kopic.

# Uporaba

Sedaj, ko smo spoznali podatkovno strukturo kopica si lahko pogledamo kje se lahko uporablja. Ena izmed najbolj uporabnih je sortirni algoritm heapsort, ki uredi elementov z časovno zahtevnostjo Heapsort je uporaben zato, ker je hitrejši od nekaterih drugih sortirnih algoritmov, kot naprimer urejanje z mehurčki ali urejanje z vstavljanjem. Tam je časovna zahtevnost .

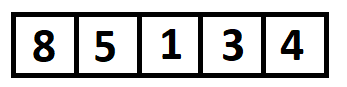
## Heapsort

Algoritem za parameter dobi neurjeno tabelo elementov. Iz teh podatkov sestavi maksimalno kopico. Potem začne z urejanjem tako, da prvo odstrani element z metodo, ki smo jo obravnavali prej. Ta element zapiše na konec tabele in sedaj ponovi postopek na preostalih elementih. Celotno delovanje si poglejmo na primeru:



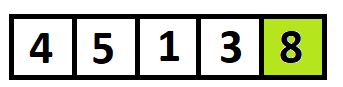
Slika 14: Primer tabele elementov nad katero bomo izvedli heapsort.

Algoritem najprej iz elementov naredi kopico z zgoraj omenjeno metodo sestavi. Kopica predstavljena s tabelo izgleda sledeče:



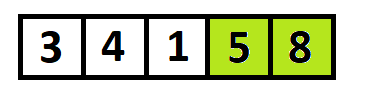
Slika 15: Kopica sestavljena iz elementov predstavljena s tabelo.

Tukaj izvede metodo odstrani, ker želimo hraniti zaporedje lahko metoda naredi tako, da zamenja prvi in zadnji element in nato začne algoritem ponavljati na tabeli elementov brez zadnjega. Prikazana slika z zeleno označuje elemente, ki so že razvrščeni pravilno in jih bo algoritem pustil pri mire. Na vseh ostalih pa se algoritem ponovi.



Slika 16: Stanje urejenosti tabele po enem koraku heapsorta.

Zdaj ponovimo postopek, sprva povrnemo kopično lastnost, v našem primeru zamenjamo samo elementa 4 in 5, potem odstranimo prvi element in ga zamenjamo z zadnjim, ki je še v naši kopici. V tem primeru z elementom 3. Po naslednjem koraku imamo naslednjo stanje.



Slika 17: Stanje urejenosti tabele po dveh korakih heapsorta.

Postopek se ponovi. Po naslednjem koraku imamo sledeč rezultat.



Slika 18: Stanje urejenosti tabele po treh korakih heapsorta.

Postopek ponavljamo dokler ne pridemo, do samo enega elementa v kopici. Takrat vemo, da smo uredili vse podatke.

## Ostale uporabe

Kopica pa je uporabna tudi pri drugje. Uporabimo jo lahko:

* Za realizacijo vrste s prednostjo, saj lahko v maksimalni kopici hranimo nujnosti nekaterih opravil in v primeru, da v vrsto pride element, ki ima prednost kopica z plavanjem poskrbi, da bo ta element na pravem mestu v vrsti.
* Kopica izbolša Dijkstrov algoritem, ki se uporablja za iskanje drevesa najkrajših poti. To služi na primer pri iskanju najboljše poti, če imamo mrežo avtobusov in želimo priti iz začetne točke do končne v najkrajšem času. Vsaka pot ima določen čas potovanja. Dijkstrov algoritem nam tako pomaga pri izbiri najboljših prestopanj.

# Časovna zahtevnost

Zdaj, ko smo spoznali osnove o kopici je potrebno razložiti še samo zakaj, bi se kopica uporabljala namesto tabel. Če pogledamo spodnjo tabelo, kjer primerjamo časovne zahtevnosti neurejene tabele, urejene tabele in pa kopice pri katerih n predstavlja število elemenentov. Iz teh podatkov je razvidno, da v nekaterih primerih imajo tabele boljše časovne zahtevnosti, a če pogledamo vse tri operacije skupaj vidimo, da kopica premaga obe navadni tabeli.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Neurejena tabela | Urejena tabela | Kopica |
| Vstavi |  |  |  |
| Največji / najmanjši |  |  |  |
| Odstrani največjega |  |  |  |

# Viri

Kete A. *Podatkovne structure – Kopica:* [http://www2.nauk.si/materials/316/out-839228/index.html?fbclid=IwAR3hlYBHN2VHrDjD9FBrIlP9J6Z3qhKMYXNrMZ589lRHPYZvwYkRUrRGdFY#state=14](http://www2.nauk.si/materials/316/out-839228/index.html?fbclid=IwAR3hlYBHN2VHrDjD9FBrIlP9J6Z3qhKMYXNrMZ589lRHPYZvwYkRUrRGdFY) [3. 1. 2020]

Adamchik V. S. *Binary heaps*: <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Binary%20Heaps/heaps.html> [3. 1. 2020]

<https://en.wikipedia.org/wiki/Heap_(data_structure)> [4. 1. 2020]

Prateek G. *Heaps/PriorityQueues*:<https://www.hackerearth.com/practice/data-structures/trees/heapspriority-queues/tutorial/> [4. 1. 2020]