Kako hitro bodo zgnile vse pomaranče?

Avtor: Klavdija Koren

Datum: 1. 12. 2021

Uvod

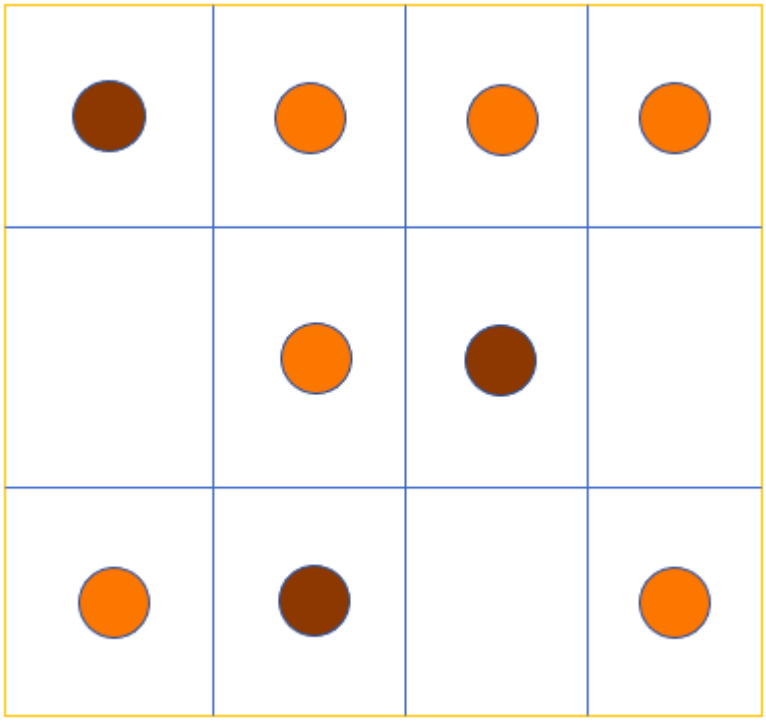
Podano dobimo matriko poljubne velikosti z vrednostmi 0, 1 in 2, kateri smo v nadaljevanju rekli kar "košara pomaranč". 0 v matriki predstavlja prazno celico matrike, 1 predstavlja svežo pomarančo, 2 pa gnilo pomarančo. Pomaranče zgnijejo, če imajo za sosedo gnilo pomarančo. Za sosede vzamemo le celice levo, desno, gor in dol, ne pa diagonalne. Zanima nas minimalno število korakov, da zgnijejo vse pomaranče. Lahko se zgodi, da ne morejo zgniti vse pomaranče, če je ena od pomaranč v košari izolirana. To pomeni, da v košari obstaja sveža pomaranča, ki ima za svoje sosede le prazne celice. V tem primeru ne bodo mogle zgniti vse pomaranče. Pomagali si bomo z vnaprej pripravljenim razredom Vrsta, ki smo ga definirali na predavanjih in vajah ter ga lahko najdete na tej povezavi: http://naslokar.fmf.uni-lj.si/FMF/Racunalnistvo1/vrsta.py.

Primer:

Vhodni podatki:

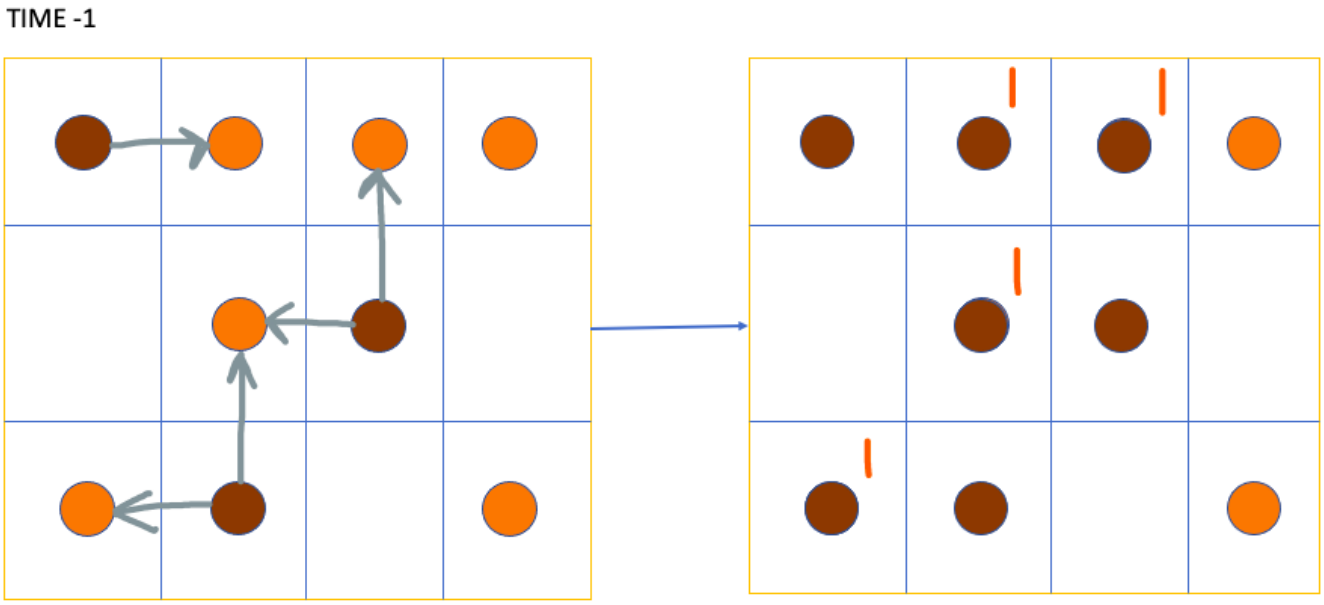
```
kosara = [[2, 1, 1, 1],
          [0, 1, 2, 0],
          [1, 2, 0, 1]]
```

Za lažjo predstavo korakov, bomo namesto številk v skicah uporabili rjave kroge, ki predstavljajo gnile pomaranče in oranžne kroge, ki predstavljajo sveže pomaranče.



1. korak:

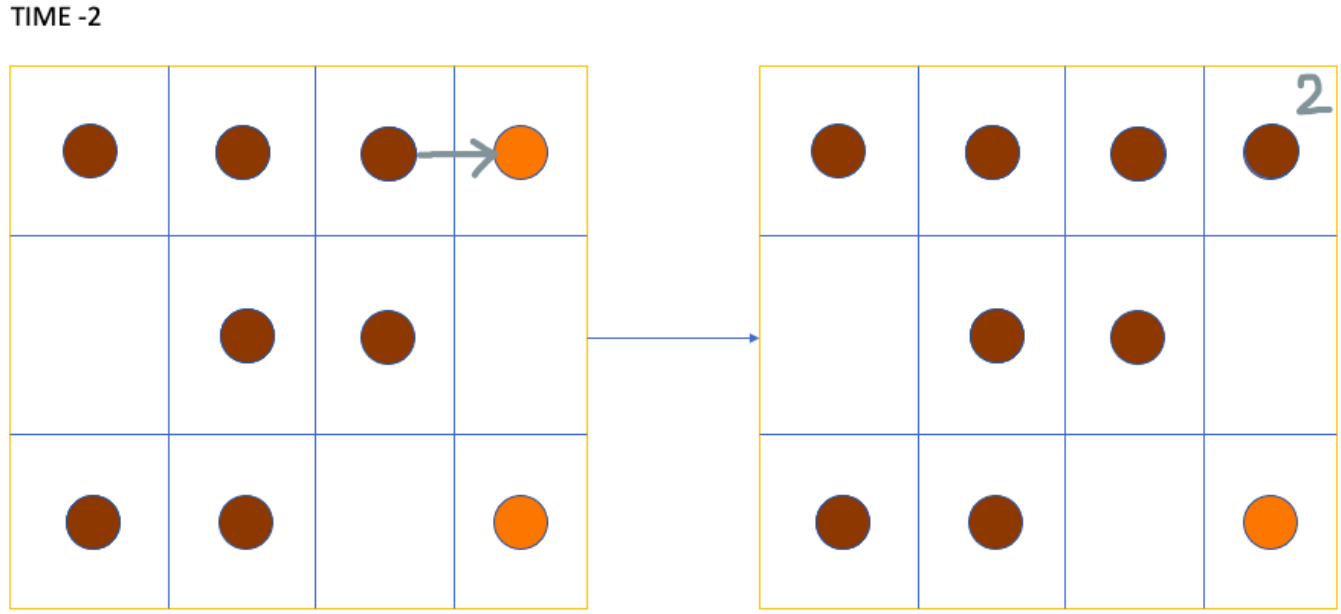
Na prvi sliki je prikazan prvi korak. Sive puščice predstavljajo katere pomaranče bodo zaradi gnilih sosed dodatno zgnile, številke 1 pa kažejo, katere pomaranče so torej zgnile v 1. koraku.



2. korak:

Na drugi sliki vidimo, da je narisana nova siva puščica, ter s številko 2 označena pomaranča, ki je na novo zgnila v drugem koraku. Opazimo, da je pomaranča v desnem spodnjem kotu ostala sveža, ker je izolirana. Ne glede na to koliko korakov naredimo, bo ta pomaranča ostala sveža. Zato v tem primeru ne morejo zgniti vse pomaranče.

Če izolirane pomaranče ne bi bilo in bi bila namesto nje gnila pomaranča ali prazna celica, bi bilo minimalno število korakov, da zgnijejo vse pomaranče, enako 2.



Algoritem 1

Najprej si poglejmo, kako bi lahko dobili rešitev brez uporabe vrste.

Ideja:

Osnovna ideja je, da simuliramo dogajanje v košari. Za to bi potrebovali dve matriki. Prva bi opisovala začetno stanje, druga pa novo. A temu se lahko izognimo tako, da v tabeli hranimo informacijo, v katerem koraku je pomaranča zgnila, zato uporabimo dodatno spremenjivko `gnilost`, s katero si bomo pomagali pri spreminjanju matrike ter štetju korakov.

Postopek:

Vse elemente matrike pregledamo s pomočjo dvojne zanke. Na začetku nastavimo spremenljivko gnilost na 2. To spremenljivko bomo potrebovali, da bomo pogledali, ali je pomaranča gnila, ter za štetje korakov. Če pridemo do gnile pomaranče, spremenimo vse možne sveže sosede tako, da jih označimo z vrednostjo spremenljivke gnilost. In sicer tako da so prvotne gnile pomaranče označene z 2, pomaranče, ki so zgnile v 1. koraku s 3, pomaranče, ki so zgnile v 2. koraku s 4 in tako naprej. Postopek ponavljamo dokler v enem koraku ne zgnije nobena nova pomaranča. Na koncu se še zadnjič sprehodimo čez matriko in preverimo, da ni ostala še kakšna sveža pomaranča. Če najdemo kakšno svežo pomarančo, pomeni, da ne morejo zgniti vse pomaranče in algoritem kot rezultat vrne -1. Če pa so zgnile vse pomaranče, pa vrnemo število korakov kar nam pove izraz `gnilost - 2` (odštejemo začetno vrednost spremenljivke gnilost).

Primer:

Vhodni podatki:

```
v = [[2, 1, 0, 2, 1],
     [1, 0, 1, 2, 1],
```

```
    [1, 0, 0, 2, 1]]
```

Kako se elementi matrike spreminjajo med koraki:

```
# najprej pregledujemo prvo gnilo pomarančo, torej element v
# 1. vrstici ter 1. stolpcu
# spremenimo pomarančo pod njo
[[2, 1, 0, 2, 1],
 [3, 0, 1, 2, 1],
 [1, 0, 0, 2, 1]]

# spremenimo pomarančo desno od nje
[[2, 3, 0, 2, 1],
 [3, 0, 1, 2, 1],
 [1, 0, 0, 2, 1]]

# pregledujemo pomarančo v 1. vrstici ter 4. stolpcu
# spremenimo pomarančo desno od nje
[[2, 3, 0, 2, 3],
 [3, 0, 1, 2, 1],
 [1, 0, 0, 2, 1]]

# pregledujemo pomarančo v 2. vrstici ter 4. stolpcu
# spremenimo pomarančo desno od nje
[[2, 3, 0, 2, 3],
 [3, 0, 1, 2, 3],
 [1, 0, 0, 2, 1]]

# spremenimo pomarančo levo od nje
[[2, 3, 0, 2, 3],
 [3, 0, 3, 2, 3],
 [1, 0, 0, 2, 1]]

# pregledujemo pomarančo v 3. vrstici ter 4. stolpcu
# spremenimo pomarančo desno od nje
[[2, 3, 0, 2, 3],
 [3, 0, 3, 2, 3],
 [1, 0, 0, 2, 3]]

# konec 1. koraka

# pregledujemo pomarančo v 2. vrstici ter 1. stolpcu
# spremenimo pomarančo pod njo
[[2, 3, 0, 2, 3],
 [3, 0, 3, 2, 3],
 [4, 0, 0, 2, 3]]

# konec 2. koraka
```

Implementacija v pythonu:

```
def gnile_pomarance(v):
    '''
    Funkcija pregleda v koliko korakih zgnijejo vse pomaranče.
    '''
    spremenjene = False
    gnilost = 2
    while (True):
        for i in range(V):
            for j in range(S):
                # preverimo, če je pomaranča zgnila na prejšnjem koraku
                if (v[i][j] == gnilost):
                    if (preveri(i + 1, j) and v[i + 1][j] == 1):
                        v[i + 1][j] = v[i][j] + 1
                        spremenjene = True
                        print(v)
                    if (preveri(i, j + 1) and v[i][j + 1] == 1):
                        v[i][j + 1] = v[i][j] + 1
                        spremenjene = True
                        print(v)
                    if (preveri(i - 1, j) and v[i - 1][j] == 1):
                        v[i - 1][j] = v[i][j] + 1
                        spremenjene = True
                        print(v)
                    if (preveri(i, j - 1) and v[i][j - 1] == 1):
                        v[i][j - 1] = v[i][j] + 1
                        spremenjene = True
                        print(v)
        # preverimo, če ni zgnila nobena pomaranča ter zanko zaključimo
        if (not spremenjene):
            break
        spremenjene = False
        gnilost += 1
    # preverimo, ali je ostala kakšna sveža pomaranča
    for i in range(V):
        for j in range(S):
            if (v[i][j] == 1):
                # našli smo pomarančo, ki ni zgnila
                return -1
    return gnilost - 2
```

Časovna in prostorska zahtevnost:

- Časovna zahtevnost algoritma: V najboljšem primeru je časovna zahtevnost enaka `O(V*S)`, v najslabšem pa `O((V*S)*(V*S))`, kjer V predstavlja število vrstic matrike, S pa število stolpcev. Matriko je treba pregledovati vsakič znova, dokler ni spremembe. To se lahko zgodi največ `(V*S)/2 - krat`, zato je časovna zahtevnost enaka `O((V*S)*(V*S))`.
- Prostorska zahtevnost: `O(1)`. Potrebujemo samo prostor za eno matriko.

Algoritem 2

Pri drugem algoritmu si bomo pomagali tako, da bomo uporabili podatkovno strukturo vrsta.

Vrsta je podatkovna struktura za skladiščenje podatkov. Podatke v vrsto dajemo na enem koncu vrste, jemljemo jih pa na drugem koncu. Torej iz vrste vedno vzamemo podatek, ki je najdlje v vrsti. Operacije za

delovanje vrste so: `pripravi`, `vstavi`, `začetek` (vrne začetni element), `odstrani` (odstrani začetni element) ter `prazna`.

Ideja:

V vrsti bomo hranili indekse gnilih pomaranč. Iz vrste bomo jemali gnile pmaranče eno po eno ter na konec vrste uvrščali tiste, ki zgnijejo zaradi te, ki smo jo iz vrste vzeli. Za pomoč pri štetju korakov si bomo pomagali z mejnim elementom, ki bo ločeval elemente v vrsti.

Postopek:

V prvem koraku pregledamo matriko in v prazno vrsto zapišemo vse gnile pomaranče ter preštejemo vse sveže pomaranče. Nato iz vrste jemljemo po eno gnilo pomarančo. Pogledamo sosede te gnile pomaranče. Te ki so zgnile v tem koraku vstavimo na konec vrste in prvo gnilo pomarančo odstranimo iz vrste. Ko nove pomaranče dodajamo v vrsto, sproti zmanjšujemo tudi števec svežih pomaranč. Da lahko štejemo korake, potrebujemo mejni element, ki v vrsti loči gnile pomaranče iz prvega koraka in gnile pomaranče iz drugega koraka. Ko pregledamo vse gnile pomaranče v nekem koraku in jih odstranimo iz vrste, pridemo do mejnega elementa. Odstranimo ga iz vrste in ga vstavimo na konec vrste, da loči naslednje korake, števec korakov pa povečamo za ena vsakič. Preden mejni korak prestavimo na konec vrste še preverimo, ali je vrsta v tistem trenutku, ko ga odstranimo prazna, če je, smo pregledali že vse gnile pomaranče in tudi tiste na novo zgnite, zato zanko prekinemo. Če je na koncu, ko se zanka ustavi, tudi število svežih pomaranč enako 0, pomeni, da so zgnile vse pomaranče in izpiše se nam minimalno število korakov. Če pa število svežih pomaranč na koncu ni enako 0, pa se izpiše, da vse pomaranče ne morejo zgniti.

Uporabljene funkcije:

- `preveri(celica)`: Funkcija, ki preveri veljavnost celice. Uporabili smo jo pri iskanju sosed gnile pomaranče. Preveriti smo morali, ali celica obstaja, ker robna gnila pomaranča nima vseh štirih sosed.
- `mejni_element(celica)`: Funkcija, ki preveri, ali je celica mejni element. Uporabili smo jo na začetku zanke, ko smo pogledali prvi element vrste. Če je bila to pomaranča, smo pogledali sosede, če pa je bil to mejni element, smo ga prestavili na konec vrste.
- `gnile_pomarance(kosara)`: Funkcija, ki izračuna in izpiše, koliko korakov potrebujemo, da zgnijejo vse pomaranče, če je le to možno.

Implementacija v pythonu:

```
from vrsta import Vrsta

kosara = [[2, 1, 0, 2, 1],
          [1, 0, 1, 2, 1],
          [1, 0, 0, 2, 1]]

V = len(kosara)
S = len(kosara[1])

def preveri(celica):
    '''
    Funkcija, ki preveri veljavnost celice v košari.
    '''
    return (celica[0] >= 0 and celica[1] >= 0 and celica[0] < V and celica[1] < S)

def mejni_element(celica):
    '''
    Funkcija, ki preveri ali je celica mejni element.
    '''
    return (celica[0] == -1 and celica[1] == -1)


def gnile_pomarance(kosara):
    '''
    Funkcija, ki izračuna in izpiše koliko korakov potrebujemo, da zgnijejo
    vse pomaranče, če je le to možno.
    '''
    vrsta = Vrsta()
    koraki = 0
    sveze = 0

    # sprehodimo se čez matriko in gnile pomaranče dodamo v vrsto, ter preštejemo
    # sveže pomaranče
    for i in range(V):
        for j in range(S):
            if kosara[i][j] == 2:
                vrsta.vstavi((i, j))
            if kosara[i][j] == 1:
                sveze += 1
    if vrsta.prazna():
        if sveze == 0:
            return print('Košara je prazna.')
        return print('Vse pomaranče ne morejo zgniti.')

    else:
        vrsta.vstavi((-1, -1))

    while True:
        gnila = vrsta.zacetek()
        desna = (gnila[0], gnila[1] + 1)
        leva = (gnila[0], gnila[1] - 1)
        gor = (gnila[0] - 1, gnila[1])
        dol = (gnila[0] + 1, gnila[1])
```

```
        # preverimo ali je začetni element vrste mejni element, če je, ga
        # odstranimo in vstavimo na konec vrste, ter prištejemo korak
        if mejni_element(gnila) is True:
            vrsta.odstrani()
            # preverimo, če je po odstranitvi mejnega elementa vrsta prazna
            if vrsta.prazna():
                # če je, pomeni da so zgnile že vse možne pomaranče in zanko prekinemo
                break
            vrsta.vstavi((-1, -1))
            koraki += 1

        else:
            # preverimo desno celico
            if preveri(desna) is True and kosara[desna[0]][desna[1]] == 1:
                kosara[desna[0]][desna[1]] = 2
                vrsta.vstavi(desna)
                sveze -= 1

            # preverimo levo celico
            if preveri(leva) is True and kosara[leva[0]][leva[1]] == 1:
                kosara[leva[0]][leva[1]] = 2
                vrsta.vstavi(leva)
                sveze -= 1

            # preverimo zgornjo celico
            if preveri(gor) is True and kosara[gor[0]][gor[1]] == 1:
                kosara[gor[0]][gor[1]] = 2
                vrsta.vstavi(gor)
                sveze -= 1

            # preverimo spodnjo celico
            if preveri(dol) is True and kosara[dol[0]][dol[1]] == 1:
                kosara[dol[0]][dol[1]] = 2
                vrsta.vstavi(dol)
                sveze -= 1
            vrsta.odstrani()

    if sveze != 0:
         return print('Vse pomaranče ne morejo zgniti.')
    else:
        return print('Minimalno število korakov, da zgnijejo vse pomaranče = ', koraki)
```

Časovna in prostorska zahtevnost:

- Časovna zahtevnost algoritma: `O(V*S)`, kjer V predstavlja število vrstic matrike, S pa število stolpcev. Vsak element matrike je lahko vstavljen v vrsto samo enkrat, zato je časovna zahtevnost enaka `O(V*S)`.
- Prostorska zahtevnost: `O(V*S)`. Za shranjevanje elementov v vrsto potrebujemo največ toliko prostora, da lahko vstavimo v vrsto vsak element enkrat, zato je prostorska zahtevnost enaka `O(V*S)`.

Odgovori na pogosta vprašanja:

- **Kaj se zgodi, če je košara pomaranč prazna?** Če je košara prazna, nam program izpiše `Košara je prazna.` To pogledamo že v prvem koraku, ko pregledamo matriko. Če je po pregledu matrike vrsta prazna in števec svežih pomaranč enak 0, vemo, da je košara prazna.
- **Kaj se zgodi, če so v košari samo gnile pomaranče ali pa samo sveže?** Če so v košari samo sveže pomaranče, nam program izpiše `Vse pomaranče ne morejo zgniti.` Program se ustavi pri istem koraku, kot če je košara prazna, saj pogledamo, ali je ob pregledu matrike vrsta prazna. Če je vrsta prazna, nimamo gnile pomaranče in ne glede na to koliko imamo svežih, ne bo zgnila nobena. Če pa so

v košari samo gnile pomaranče, pa nam program izpiše `Minimalno število korakov, da zgnijejo vse pomaranče = 0`. Tukaj se program ustavi malo kasneje. V vrsto vstavi vse pomaranče iz košare ter začne eno po eno pregledovati. Ker pa nobena od teh nima sveže sosede, vse samo odstranimo iz vrste. Na koncu odstranimo tudi mejni element in preden ga prestavimo na konec, pogledamo če je vrsta prazna. Ker ni zgnila nobena nova pomaranča je vrsta v tem trenutku prazna in zanka se ustavi. Ker mejnega elementa nikoli ni bilo treba prestaviti na konec vrste, bo število korakov enako 0.

- **Kako se v vrsti shranjujejo podatki?** V vrsti se shranjujejo indeksi elementov iz matrike kot par oblike `(i, j)`, kjer i predstavlja i-to vrstico matrike, j pa j-ti stolpec.

Viri

1. [https://www.geeksforgeeks.org/minimum-time-required-so-that-all-oranges-become-rotten/], 16. 11. 2021

2. [https://medium.com/trick-the-interviwer/rotting-oranges-e09ca22f6e24], 16. 11. 2021.

3. Matija Lokar, datoteka vrsta.py, pridobljeno s [http://naslokar.fmf.uni-lj.si/FMF/Racunalnistvo1/vrsta.py], 1. 12. 2021.