Grayevi kodovi

Uzelac, Ivona

Grayevi kodovi

Uzelac, Ivona

Sveučilište u Zagrebu
Prirodoslovno-matematički fakultet
Matematički odsjek

Ivona Uzelac

Grayevi kodovi

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Vedran Krčadinac

Zagreb, studeni 2020.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____ , predsjednik
2. _____ , član
3. _____ , član

Povjerenstvo je rad ocijenilo ocjenom _____ .

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

1 Uvod

Grayev kod za binarne nizove ciklički je uređaj na skupu svih binarnih nizova duljine n kod kojeg se susjedni nizovi razlikuju točno u jednom bitu. Frank Gray iz Bell labaratorija patentirao je 1953. godine jedan takav kod s ciljem minimalizacije pogrešaka pri pretvaranju analognih signala u digitalne. Općenitije, Grayevi kodovi su metode uzastopnog generiranja kombinatornih objekata kod kojih se susjedni objekti razlikuju minimalno ili na unaprijed zadan način. U ovom diplomskom radu cilj je analizirati i implementirati u programskom jeziku Python algoritme za generiranje Grayevih kodova za podskupove, kombinacije, permutacije i particije.

U drugom poglavlju definiramo binarni niz i implementiramo dva algoritma za ispis binarnih n-torki u standardnom poretku. Prvi primjer Grayevih kodova za binarne nizove je *zrcaljeni Grayev kod* kojeg generiramo rekurzivnim i iterativnim algoritmom. Idući je *monoton Grayev kod* koji ima dodatan zahtjev na broj jedinica uzastopnih parova binarnih nizova.

U trećem poglavlju primjenjujemo ideju Grayevog koda na k-kombinacije n-članog skupa. Minimalna razlika sada je simetrična razlika susjednih kombinacija s kardinalitetom 2. Prvi Grayev kod za kombinacije koji proučavamo generiramo na dva načina, algoritmom koji koristi zrcaljeni Grayev kod i *revolving door* algoritmom. Drugi Grayev kod za kombinacije generiramo algoritmom *minimalnih promjena*.

U četvrtom poglavlju obrađujemo algoritme koji generiraju Grayeve kodove za permutacije. Kako se dvije permutacije razlikuju barem na dva mjesta, minimalna razlika susjednih objekata kod Grayevih kodova za permutacije je jedna transpozicija. U ovom poglavlju obrađujemo jedan algoritam za generiranje takvog koda, takozvani *Johnson-Trotterov* algoritam. Također, u ovom poglavlju obrađujemo algoritam koji permutacije generira na način da se svake dvije susjedne razlikuju maksimalno, odnosno na svim mjestima. Takav kod zovemo kod za permutacije s deranžmanima.

U posljednjem poglavlju obrađujemo Grayeve kodove za particije skupova gdje se susjedne particije razlikuju za premještanje jednog elementa između dva susjedna podskupa. Algoritam koji generira takav kod nazivamo *Knuthovim algoritmom*.

2 Grayevi kodovi za binarne nizove

2.1 Definicija Grayeva koda

Binarni niz duljine n je uređena n-torka nula i jedinica $(b_1, b_2, \ldots, b_n), b_i \in \{0,1\}$. Skup svih binarnih nizova je Kartezijev produkt $\{0,1\}^n = \{0,1\} \times \{0,1\} \times \ldots \times \{0,1\}$ koji po principu produkta ima 2^n elemenata. Prije nego definiramo Grayeve kodove za binarne nizove, prisjetimo se nekih kombinatornih teorema i dokaza.

Teorem 2.1. *Skup od n elemenata ima 2^n podskupova.*

Dokaz 1. Neka je $S = \{a_1, a_2, \ldots, a_n\}$ proizvoljan n-člani skup i neka je $\mathcal{P}(S)$ njegov partitivni skup. Konstruirajmo bijekciju između $\mathcal{P}(S)$ i skupa svih binarnih nizova duljine n.

Za svaki $X \subseteq S$ definiramo funkciju $f_X : \{1, 2, \ldots, n\} \to \{0,1\}$,

$$f_X(i) = \begin{cases} 1, & a_i \in X \\ 0, & a_i \notin X \end{cases}, \text{za } i = 1, 2, \ldots, n.$$

Drugim riječima, podskupovima od S pridružujemo binarne n-torke, $F : \mathcal{P}(S) \to \{0,1\}^n$, $F(X) = f_X$. Tvrdimo da je funkcija F bijekcija. To vrijedi jer ima inverznu funkciju $G : \{0,1\}^n \to \mathcal{P}(S)$,

$$G(b_1, b_2, \ldots, b_n) = \{a_i \mid b_i = 1, 1 \leq i \leq n\},$$

gdje je $(b_1, b_2, \ldots, b_n)$ binarna n-torka iz skupa $\{0,1\}^n$. Očito vrijedi

$$G \circ F = id_{\mathcal{P}(S)} \quad \text{i} \quad F \circ G = id_{\{0,1\}^n},$$

odnosno kompozicije funkcija G i F su identitete. Stoga možemo zaključiti da su kardinaliteti skupova $\mathcal{P}(S)$ i $\{0,1\}^n$ jednaki. Skup binarnih n-torki $\{0,1\}^n$ po principu produkta ima 2^n elemenata pa je $|\mathcal{P}(S)| = 2^n$. □

Ako su elementi n-članog skupa S numerirani, kao u prethodnom dokazu, podskupove od S možemo izjednačiti s binarnim nizovima duljine n.

Dokaz 2. Dokažimo sada teorem 2.1 pomoću indukcije. Baza indukcije očito je zadovoljena za $n = 1$. Pretpostavimo da tvrdnja teorema vrijedi za skupove s $n-1$ elemenata, odnosno da skup od $n-1$ elemenata ima 2^{n-1} podskupova.

Neka je S skup od n elemenata. Fiksirajmo neki element $a \in S$ te podijelimo podskupove od S u dvije klase: oni koji sadrže a i oni koji ne sadrže a. Oni podskupovi koji ne sadrže a su podskupovi skupa $S' = S \setminus \{a\}$ koji ima

$n-1$ elemenata, stoga prema pretpostavci indukcije takvih podskupova ima 2^{n-1}. Oni podskupovi koji sadrže a mogu se napisati kao unija od $\{a\}$ i nekog podskupa iz S', stoga i takvih ima 2^{n-1}. Dakle, prema principu sume, broj podskupova od S je $2^{n-1}+2^{n-1} = 2^n$, čime smo dokazali korak indukcije. □

Napišimo rekurzivni algoritam koji generira binarne nizove na način kao u dokazu 2. Algoritam definira početne binarne nizove 0 i 1 te dodaje 0 i 1 u prefikse svih binarnih nizova za $n = 2, 3, \ldots$.

Algoritam 2.2. Binarne n-torke u standardnom poretku.

```
def standardni(n):

    if n == 1:
        return ['0', '1']

    else:
        Pom_lista = standardni(n - 1)
        Rez_lista = []

        for niz in Pom_lista:
            Rez_lista.append('0' + niz)

        for niz in Pom_lista:
            Rez_lista.append('1' + niz)

        return Rez_lista
```

U rekurzivnom algoritmu 2.2 koristimo dvije liste. Pod pojmom *lista* u ovom radu podrazumijevat ćemo uređen skup, odnosno konačan niz nekih objekata. *Pom_lista* sadrži binarne nizove duljine $n-1$ dobivene rekurzivno, a u listu *Rez_lista* spremaju se binarne n-torke. Redoslijed u kojem ovaj algoritam generira binarne n-torke nazivamo *standardnim poretkom*. Isti takav poredak binarnih nizova dobivamo pretvaranjem cijelih brojeva od 0 do $2^n - 1$ u binarni zapis. Napišimo sada algoritam koji generira istu listu, ali tako da pretvara cijele brojeve u binarni zapis.

Algoritam 2.3. Binarne n-torke u standardnom poretku dobivene pretvaranjem cijelih brojeva u binarni zapis.

```
def standardni_cijeli(n):

    for i in range(2 ** n):
        niz = ''

        for j in range(n):

            if i % 2 == 0:
                niz = '0' + niz

            else:
                niz = '1' + niz

            i = i // 2

        print(niz)
```

Algoritam 2.3 izvršava petlju u kojoj cijele brojeve od 0 do $2^n - 1$ uzastopno dijeli s 2 i u prefikse zapisuje ostatke.

Za $n = 5$ algoritmi 2.2 i 2.3 ispisuju binarne nizove prikazane u tablici 1. Uočimo da se u listi binarnih n-torki u standardnom poretku susjedne n-torke razlikuju za od 1 do n bitova. Ako promatramo odgovarajuće podskupove n-članog skupa, broj bitova u kojima se razlikuju dvije n-torke jednak je kardinalitetu simetrične razlike odgovarajućih podskupova. *Simetričnu razliku* skupova X i Y definiramo kao

$$X \triangle Y := (X \cup Y) \setminus (Y \cup X)\,.$$

00000	01000	10000	11000
00001	01001	10001	11001
00010	01010	10010	11010
00011	01011	10011	11011
00100	01100	10100	11100
00101	01101	10101	11101
00110	01110	10110	11110
00111	01111	10111	11111

Tablica 1: Binarne petorke u standardnom poretku.

Cilj nam je generirati listu binarnih n-torki ili podskupova n-članog skupa u kojoj se susjedni elementi razlikuju minimalno, odnosno samo u jednom bitu. Takvu listu zvat ćemo Grayevim kodom.

Definicija 2.4. Grayev kod za binarne n-torke (podskupove n-članog skupa S) *je lista koja sadrži sve binarne n-torke (podskupove od S) takva da se susjedne n-torke razlikuju točno u jednom bitu (simetrična razlika susjednih podskupova je kardinaliteta* 1*).*

Grayev kod je *ciklički* ako se njegova prva i posljednja binarna n-torka (prvi i posljednji podskup) također razlikuju točno u jednom bitu (elementu). U tablicama 2 i 3 prikazani su Grayevi kodovi generirani algoritmima koje ćemo kasnije obraditi. Tablica 2 sadrži listu binarnih nizova koju nazivamo zrcaljeni Grayev kod. Uočimo da se prvi i posljednji niz tog koda razlikuju točno u jednom bitu pa je taj kod ciklički. Monoton Grayev kod prikazan u tablici 3 nije ciklički jer možemo uočiti da se prvi i posljednji niz razlikuju u svih 5 bitova.

00000	01100	11000	10100
00001	01101	11001	10101
00011	01111	11011	10111
00010	01110	11010	10110
00110	01010	11110	10010
00111	01011	11111	10011
00101	01001	11101	10001
00100	01000	11100	10000

Tablica 2: Zrcaljeni Grayev kod za $n = 5$.

00000	11000	01010	11110
00001	10000	01011	11100
00011	10001	01001	11101
00010	10101	01101	11001
00110	10100	00101	11011
00100	10110	00111	10011
01100	10010	01111	10111
01000	11010	01110	11111

Tablica 3: Monoton Grayev kod za $n = 5$.

2.2 Povijest i primjene

Američki fizičar i inženjer Frank Gray (1887.-1969.) iz Bell labaratorija svojim je patentima i izumima doprinio medicini i elektrotehnici. Zajedno s kolegama sudjelovao je u početku digitalne revolucije korištenjem nove metode pretvaranja analognih signala u digitalne u raznim uređajima (televizorima i medicinskim aparatima na osnovi vakuumskih cijevi).

Slika 1: Frank Gray (preuzeto iz [14]).

Ideju po kojoj je danas najpoznatiji patentirao je 1953. godine pod nazivom *Pulsni komunikacijski kod*, a u originalnoj prijavi koristio je i termin *zrcaljeni binarni kod*. Do drugog naziva došao je zbog činjenice da se kod može generirati iz standardnog binarnog koda korištenjem neke vrste zrcaljenja. Kod je kasnije, od strane onih koji su ga koristili, nazvan prema Grayju.

F. Gray opisao je zrcaljeni binarni kod kao kod čiji je cilj minimalizirati pogreške pri pretvaranju analognih signala u digitalne. Pretpostavimo da želimo poslati neki konačan niz bitova koristeći se analognim uređajem. Možemo uzeti niz bitova, izračunati cijeli broj kojeg određuje taj niz bitova i poslati signal. Problem se javlja ako se dogodi mala pogreška u jačini signala. Tada će se pogreška pojaviti i u zaprimljenom cijelom broju. No, mala promjena u cijelom broju može uzrokovati veliku promjenu u njegovom nizu bitova. Pitanje koje se postavlja je kako povezati cijele brojeve s nizovima bitova na način da mala pogreška u cijelom broju uzrokuje samo male pogreške u nizu bitova. Jedan odgovor na to pitanje je da napišemo listu nizova bitova tako da se svaki niz razlikuje od susjednog u jednom bitu. Upravo taj odgovor je ideja zrcaljenog Grayevog koda.

Mnogi uređaji temelje se na prekidačima koji mogu biti u dva stanja (upaljeni i ugašeni). Primijenimo sada prethodno opisanu ideju zrcaljenog Grayevog koda. Kada bi se signali takvih uređaja temeljili na standardnom binarnom kodu (tablica 4.a), binarni nizovi 011 i 100 bili bi na susjednim pozicijama. U prelasku iz jednog stanja u drugo sva tri prekidača koja predstavljaju bitove mijenjaju svoje stanje. No, malo je vjerojatno da će fizički prekidači sinkronizirano mijenjati svoja stanja. U kratkom periodu, kada prekidači mijenjaju svoja stanja, može doći do očitanja pogrešnog signala ili do očitanja prijelaznog stanja između dvije pozicije. Zrcaljeni Grayev kod rješava ovaj problem mijenjajući stanje samo jednog prekidača na prijelazu pozicija.

	a. Standardni binarni kod	b. Zrcaljeni Grayev kod
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Tablica 4: Usporedba standardnog binarnog koda i zrcaljenog Grayevog koda.

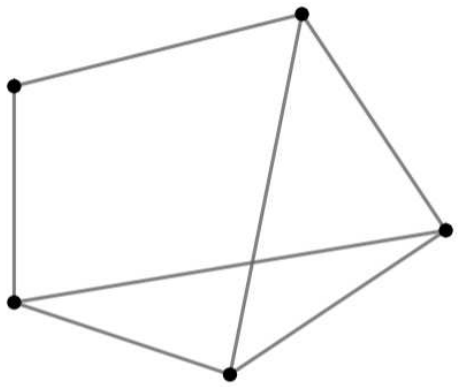
Termin *Grayev kod* prvi se put spominje 1980. godine, a danas se koristi za sve metode kojima se generiraju kombinatorni objekti takvi da se uzastopni objekti razlikuju na neki unaprijed određen način ili, najčešće, minimalno. Koristi se u raznim područjima, na primjer za ispitivanje strujnih krugova, šifriranje signala, sortiranje dokumenata, obradu slika i grafika, i drugdje. Jedna od zanimljivih primjena je svakako u rješavanju zagonetaka kao što su Kineski prstenovi i Hanojski tornjevi. Naime, ideja koja se koristi za generiranje zrcaljenog Grayevog koda primijenjivana je u takvim zagonetkama prije nego li je postala poznata inženjerima.

2.3 Kombinatorna interpretacija

Još jedna interpretacija Grayevih kodova su Hamiltonovi ciklusi u grafovima. Prije definiranja Grayevih kodova kao Hamiltonovih ciklusa, prisjetimo se definicije grafa i nekih povezanih pojmova.

Graf definiramo kao uređeni par $G = (V, E)$, gdje je V skup *vrhova*, a E skup dvočlanih podskupova od V koje nazivamo *bridovima*. Kažemo da su dva vrha u grafu *susjedna* ako postoji brid koji ih sadrži. Primjer jednog grafa prikazan je na slici 2.

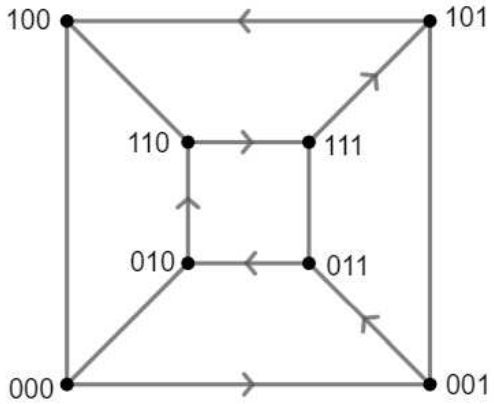
Šetnja u grafu (V, E) je niz vrhova $(v_1, v_2, \ldots, v_n)$, pri čemu su v_i i v_{i+1} susjedni za $i = 1, 2, \ldots, n-1$. *Put* je šetnja u kojoj su vrhovi (osim eventualno prvog i zadnjeg) različiti, a *ciklus* je put kojem su prvi i zadnji elementi jednaki. Put je *Hamiltonov* ako prolazi kroz sve vrhove grafa. Ukoliko je Hamiltonov put zatvoren, govorimo o *Hamiltonovom ciklusu*. Graf koji ima Hamiltonov ciklus nazivamo *Hamiltonovim grafom*.



Slika 2: Primjer grafa s $n = 5$ vrhova.

Definicija 2.5. *Graf Q_n čiji vrhovi odgovaraju skupu svih binarnih n-torki, a dva vrha su susjedna ako i samo ako se odgovarajuće n-torke razlikuju u točno jednom bitu nazivamo* n-kockom.

Promotrimo sada graf Q_3 prikazan na slici 3. Put koji započinje u ishodištu (000), prolazi bridovima kocke i svaki vrh posjećuje točno jednom je Hamiltonov put. Kako se binarne n-torke koje predstavljaju prvi i posljednji posjećeni vrh također razlikuju samo u jednom bitu, taj put je Hamiltonov ciklus. Hamiltonov ciklus možemo primijeniti u n-kocki i tada je to upravo ciklički Grayev kod za binarne n-torke. Ciklus prikazan na slici 3 odgovara zrcaljenom Grayevom kodu prikazanom u tablici 4.



Slika 3: Hamiltonov ciklus u kocki Q_3.

2.4 Algoritmi

U tablicama 2 i 3 prikazali smo petorke zrcaljenog i monotonog Grayevog koda. U ovom poglavlju ćemo definirati te kodove te napisati algoritme koji ih generiraju.

Binarni refleksivni Grayev kod ili *zrcaljeni Grayev kod* osnovni je i najpoznatiji primjer Grayevog koda. Neka je $\mathscr{L}_n$ lista binarnih nizova duljine $n \in \mathbb{N}$ u zrcaljenom Grayevom kodu. Definiramo je rekurzivno:

- Neka je $\mathscr{L}_1 = [0, 1]$.
- Listu $\mathscr{L}_n$ dobivamo od liste $\mathscr{L}_{n-1}$ tako da
  1. ispisujemo listu $\mathscr{L}_{n-1}$ tako da svakom nizu bitova u prefiks dodamo 0,
  2. ispisujemo listu $\mathscr{L}_{n-1}$ u suprotnom poretku tako da svakom nizu u prefiks dodamo 1.

Za $n = 1, 2, 3, 4$ dobivamo sljedeće liste Grayevih kodova:

$\mathscr{L}_1$	$\mathscr{L}_2$	$\mathscr{L}_3$	$\mathscr{L}_4$	
0	00	000	0000	1100
1	01	001	0001	1101
	11	011	0011	1111
	10	010	0010	1110
		110	0110	1010
		111	0111	1011
		101	0101	1001
		100	0100	1000

Uočimo da se, kao i za $n = 5$, prvi i posljednji element liste $\mathscr{L}_n$ također razlikuju u jednom bitu pa riječ je o cikličkom Grayevom kodu. Dokažimo sada da je definirana lista $\mathscr{L}_n$ zaista Grayev kod.

Teorem 2.6. *Za svaki $n \geq 1$, $\mathscr{L}_n$ je ciklički Grayev kod.*

Dokaz. Dokažimo ovaj teorem indukcijom. Za bazu $n = 1$ teorem očito vrijedi jer je $\mathscr{L}_1 = [0, 1]$. Pretpostavimo da je za neki $n \geq 2$ lista $\mathscr{L}_{n-1}$ također Grayev kod.

Lista $\mathscr{L}_n$ sadrži svih 2^n binarnih n-torki. U prvoj polovici liste $\mathscr{L}_n$ nalazi se 2^{n-1} n-torki koje započinju brojem 0, a u drugoj polovici 2^{n-1} n-torki koje započinju brojem 1. Preostaje nam dokazati da se susjedni elementi liste razlikuju u jednom bitu.

Prema pretpostavci indukcije, znamo da se nizovi u prvoj polovici liste $\mathscr{L}_n$ razlikuju u jednom bitu jer su dobiveni dodavanjem broja 0 u prefikse nizova u listi $\mathscr{L}_{n-1}$. No, isto vrijedi i za drugu polovicu liste $\mathscr{L}_n$ jer je ona dobivena dodavanjem broja 1 u prefikse nizova u listi $\mathscr{L}_{n-1}$ u suprotnom poretku. Dva binarna niza koja se nalaze u sredini liste $\mathscr{L}_n$ razlikuju se samo u prvom bitu jer su oba dobivena pomoću posljednjeg elementa liste $\mathscr{L}_{n-1}$, a isto vrijedi i za prvi i posljednji element liste $\mathscr{L}_n$. □

Dokazali smo da zrcaljeni Grayev kod zadovoljava definiciju Grayevog koda za svaki $n \in \mathbb{N}$ i da je taj kod ciklički. Napišimo sada algoritam koji generira binarne n-torke zrcaljenog Grayevog koda.

Algoritam 2.7. Algoritam zrcaljenog Grayevog koda.

```
def zrcaljeni(n):

    if n == 1:
        return ['0', '1']

    else:
        Pom_lista = zrcaljeni(n - 1)
        Grayev_kod = []

        for niz in Pom_lista:
            Grayev_kod.append('0' + niz)

        for niz in list(reversed(Pom_lista)):
            Grayev_kod.append('1' + niz)

        return Grayev_kod
```

Algoritam 2.7 zrcaljenog Grayevog koda je rekurzivan algoritam koji pamti cijelu listu $\mathscr{L}_n$ u memoriji. Iz tog ćemo razloga promotriti nerekurzivan algoritam *pravila sljedbenika* koji pamti samo jedan binarni niz i generira njegovog sljedbenika. Uočit ćemo da su Grayevi kodovi koje generiraju ova dva algoritma isti.

Pretpostavimo da smo došli do određenog binarnog niza u listi $\mathscr{L}_n$ Grayevog koda te želimo pronaći sljedeći niz. Drugim riječima, želimo pronaći j-ti bit binarnog niza koji ćemo zatim promijeniti. Ako je broj jedinica u nizu paran broj, mijenjamo najmanje značajan bit. U suprotnom, tražimo s desna na lijevo u nizu prvu jedinicu koja se javlja te mijenjamo prvi bit lijevo od pojavljivanja jedinice.

Za $n = 4$ Grayev kod generiran pravilom sljedbenika prikazan je u tablici 5. Prvi stupac sadrži Grayev kod, drugi bitove koji su izmijenjeni, a posljednji stupac broj jedinica u pojedinom nizu.

Grayev kod	j	
0000		0
0001	4	1
0011	3	2
0010	4	1
0110	2	2
0111	4	3
0101	3	2
0100	4	1
1100	1	2
1101	4	3
1111	3	4
1110	4	3
1010	2	2
1011	4	3
1001	3	2
1000	4	1

Tablica 5: Grayev kod generiran pravilom sljedbenika za $n = 4$.

Napišimo algoritam koji generira binarne n-torke prema pravilu sljedbenika. Algoritam definira početni binarni niz koji se sastoji od n nula i izvršava petlju $2^n - 1$ puta. Kako se susjedni binarni nizovi razlikuju točno u jednom bitu, lako je uočiti da je parnost broja jedinica alternirajuća. Drugim riječima, dovoljno je provjeriti parnost brojača petlje koja također alternira te ovisno o njoj ispisati idući binarni niz.

Algoritam 2.8. Algoritam pravila sljedbenika.

```
def pravilo_sljedbenika(n):
    niz = '0' * n

    for i in range(2 ** n):
        print(niz)

        if i % 2 == 0:
            indeks = n - 1

        else:
            indeks = niz.rfind('1') - 1

        if niz[indeks] == '0':
            niz = niz[:indeks] + '1' + niz[indeks + 1:]

        else:
            niz = niz[:indeks] + '0' + niz[indeks + 1:]
```

Pokažimo sada još jedan rekurzivan algoritam za generiranje Grayevog koda koji su opisali Savage i Winkler [9], a kasnije i Knuth [3]. Neka je *gustoća* binarnog niza broj jedinica u tom nizu. Kako se susjedni binarni nizovi u Grayevom kodu razlikuju točno u jednom bitu, njihove se gustoće razlikuju za 1. Drugim riječima, za $n \geq 2$ Grayev kod ne može zadovoljiti uvjet da se gustoća binarnih nizova nikada ne smanjuje. Iz tog razloga prilagodit ćemo pravilo za usporedbu gustoća tako da varijablama i i j provjeravamo gustoće binarnih nizova dvaju parova. Grayev kod zovemo *monotonim* ako za sve $0 \leq i < j < n$ svi uzastopni parovi nizova s gustoćama i, $i+1$ koji se pojavlju u kodu prethode uzastopnim parovima s gustoćama j, $j+1$.

Promotrimo tablicu 2 i ispišimo gustoće petorki zrcaljenog Grayevog koda:

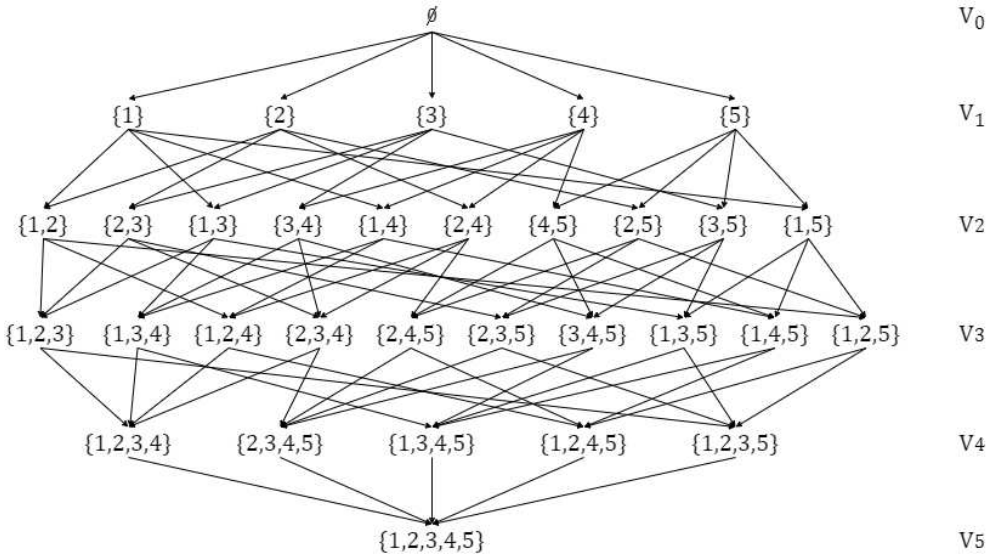
$$0, 1, 2, 1, 2, 3, 2, 1, 2, 3, \ldots.$$

Možemo uočiti da uzastopni par gustoća $2, 3$ prethodi paru $1, 2$, što ne zadovoljava uvjet monotonog Grayevog koda. Drugim riječima, Grayev kod u tablici 2 nije monoton. U tablici 3 već smo ranije prikazali monoton Grayev kod za binarne petorke. Gustoće nizova iz te tablice prikazane su, redom, u tablici 6.

0	2	2	4
1	1	3	3
2	2	2	4
1	3	3	3
2	2	2	4
1	3	3	3
2	2	4	4
1	3	3	5

Tablica 6: Gustoće n-torki monotonog Grayevog koda za $n = 5$.

Za generiranje monotonog Grayevog koda potrebno nam je nekoliko pojmova iz teorije grafova. Neka je S neki skup od n elemenata i neka je $\mathcal{B}_n$ *Booleova rešetka* partitivnog skupa $\mathcal{P}(S)$ s uređajem $\subseteq$. *Hasseov dijagram* $\mathcal{H}_n$ rešetke $\mathcal{B}_n$ je usmjeren graf čiji su vrhovi elementi iz $\mathcal{B}_n$, a bridovi uređeni parovi (X, Y) različitih vrhova iz $\mathcal{B}_n$ takvi da je $X \subset Y$ i ne postoji Z takav da je $X \subset Z \subset Y$. Hasseov dijagram $\mathcal{H}_n$ za $n = 5$ prikazan je na slici 4. Vrhovi u $\mathcal{H}_n$ podijeljeni su u razine $V_0, V_1, \ldots, V_n$, gdje V_i sadrži sve podskupove od S s i elemenata.



Slika 4: Hasseov dijagram $\mathcal{H}_5$.

Monoton Grayev kod za podskupove (binarne n-torke) tada odgovara Hamiltonovom putu u $\mathcal{H}_n$ u kojem bridovi između razina i i $i + 1$ prethode bridovima između razina j i $j + 1$, za $0 \leq i < j < n$. Ideja ovog algoritma je rekurzivno stvoriti puteve $P_{n,j}$ između razina j i $j + 1$.

Neka je $P_{n,j}$, $0 \leq j \leq n$ put između razina j i $j+1$, odnosno lista binarnih nizova u monotonom Grayevom kodu. Definiramo je rekurzivno:

- Neka je $P_{1,0} = [0, 1]$.
- Neka je $P_{n,j} = \emptyset$, za $j < 0$ ili $n \leq j$.
- Listu $P_{n+1,j}$ dobivamo od lista $P_{n,j-1}$ i $P_{n,j}$ tako da
  1. ispisujemo listu $P_{n,j-1}^{\pi_n}$ tako svakom nizu bitova u prefiks dodamo 1,
  2. ispisujemo listu $P_{n,j}$ tako da svakom nizu bitova u prefiks dodamo 0.

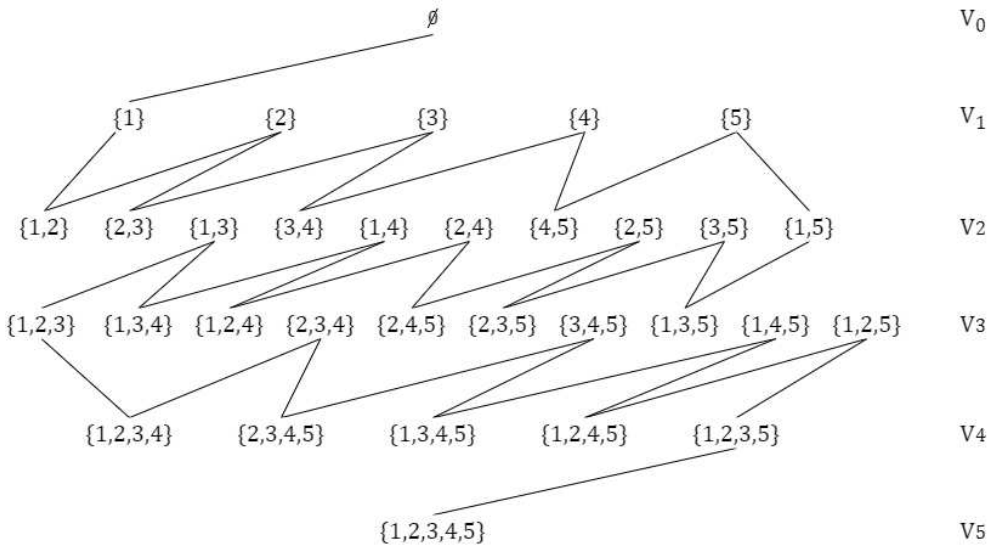
Ovdje je π_n ciklus, a P^π lista P u čijim su nizovima bitovi permutirani za π. Definicije i teoremi vezani uz permutacije nalaze se na početku četvrtog poglavlja. Permutaciju π_n možemo izračunati rekurzivno, prema formuli za $\pi_n = \pi_{n-1}^2 \circ \sigma_n$, gdje je σ_n ciklus $(n \ \ldots 2 \ 1)$. Izračunajmo prvih nekoliko slučajeva:

$$\begin{aligned}
\pi_1 &= (1)\,, \\
\pi_2 &= (1\ 2)\,, \\
\pi_3 &= (1\ 3\ 2)\,, \\
\pi_4 &= (1\ 4)\,, \\
\pi_5 &= (1\ 5\ 4\ 3\ 2)\,.
\end{aligned}$$

Na primjer, ako bitove u binarnom nizu 101 permutiramo za $\pi_3 = (1\ 3\ 2)$ dobit ćemo novi binarni niz 110. Algoritam monotonog Grayevog koda nije jedinstven već daje dva koda zadana s

$$\begin{aligned}
&P_{n,0}\ \overline{P}_{n,1}\ P_{n,2}\ \overline{P}_{n,3} \ldots, \\
&\overline{P}_{n,0}\ P_{n,1}\ \overline{P}_{n,2}\ P_{n,3} \ldots,
\end{aligned}$$

gdje su $\overline{P}_{n,j}$ liste $P_{n,j}$ ispisane u suprotnom poretku. Jedan Hamiltonov put, odnosno monoton Grayev kod za $n = 5$ prikazan je na slici 5 i on odgovara Grayevom kodu u tablici 3.



Slika 5: Hamiltonov put monotonog Grayevog koda za $n = 5$.

Napišimo algoritam monotonog Grayevog koda koji ispisuje binarne nizove prikazane u tablici 3. Algoritam se sastoji od tri funkcije: *monoton*, P i *pi*. Funkcija *monoton* poziva $n+1$ puta funkciju P i sprema dobivene liste u odgovarajućem poretku. Funkcija P je rekurzivna funkcija koja uz varijablu n prima i varijablu j, provjerava uvjete za n i j i vraća listu binarnih nizova. Prvi dio liste koju vraća ova funkcija je lista $P(n-1, j-1)$ (ako postoji) u čijim su binarnim nizovima bitovi permutirani za ciklus pi i s dodanim prefiksom 1. Drugi dio liste je lista $P(n-1, j)$ (ako ona postoji) čijim je binarnim nizovima dodan prefiks 0. Funkcija $pi(n)$ je rekurzivna funkcija koja vraća ciklus π_n.

Algoritam 2.9 (Algoritam monotonog Grayevog koda)**.**

```
def monoton(n):
    Grayev_kod = []

    for j in range(n + 1):
        Pom_lista = P(n, j)

        if j % 2 == 1:
           Pom_lista = Pom_lista[::-1]

        Grayev_kod = Grayev_kod + Pom_lista

    return Grayev_kod
```

```
def P(n, j):

    if n == 1 and j == 0:
        return ['0', '1']

    else:
        Lista = []

        if j > 0 and j < n:
            Pom_lista = P(n - 1, j - 1)
            ciklus = pi(n - 1)

            for niz in Pom_lista:
                perm = 'x' * (n - 1)

                for i in range(n - 1):

                    if i + 1 in ciklus:
                        indeks = ciklus.index(i + 1) + 1

                        if indeks == len(ciklus):
                            indeks = 0

                        k = ciklus[indeks] - 1
                        perm = perm[:i] + niz[k] + perm[i + 1:]

                    else:
                        perm = perm[:i] + niz[i] + perm[i + 1:]

                Lista.append('1' + perm)

        if j >= 0 and j < n - 1:
            Pom_lista = P(n - 1, j)

            for niz in Pom_lista:
                Lista.append('0' + niz)

        return Lista

def pi(n):
```

```
if n == 1:
    return [1]

elif n == 2:
    return [1, 2]

else:
    Pom_pi = pi(n - 1)
    pi_n = []

    sigma = list(range(n, 0, -1))

    s = 1
    t = 1

    for j in range(n):
        indeks = sigma.index(t) + 1

        if indeks == n:
            indeks = 0

        t = sigma[indeks]

        if t in Pom_pi:
            indeks = Pom_pi.index(t) + 2

            if indeks >= len(Pom_pi):
                indeks = indeks - len(Pom_pi)

            t = Pom_pi[indeks]

        if s != t and s not in pi_n:
            pi_n.append(s)
            s = t

        else:
            s += 1
            t = s

    return pi_n
```

3 Grayevi kodovi za kombinacije

Osim za podskupove, Grayevi kodovi primijenjuju se za kombinacije skupova. *Kombinacija* skupa S je svaki podskup sa zadanim brojem elemenata. Katkada se k-člani podskup skupa S naziva *k-kombinacijom* skupa S.

Teorem 3.1. *Neka je S skup od n elemenata. Broj k-kombinacija ($0 \leq k \leq n$) od S je $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.*

Dokaz. Odaberimo k elemenata iz skupa S. Postoji n mogućnosti za odabir "prvog" elementa, $n-1$ mogućnosti za odabir "drugog",..., te $n-(k-1)$ mogućnosti za odabir "k-tog" elementa. Prema principu produkta tada postoji $n \cdot (n-1) \cdots (n-k+1) = \frac{n!}{(n-k)!}$ mogućnosti za odabir k elemenata iz n-članog skupa.

No, kako u podskupu elementi nisu uređeni, ne razlikujemo redoslijed kojim su odabrani. Drugim riječima, dobiveni broj moramo podijeliti brojem različitih redoslijeda k elemenata koje smo izabrali. Ponovimo li postupak s početka ovog dokaza, imamo k izbora za "prvi" element, $k-1$ izbora za "drugi" element, itd. Dakle, ukupno imamo $k!$ izbora za taj redoslijed. Time smo dokazali teorem. □

Kako smo podskupove skupa S ranije povezali s binarnim n-torkama, tako njegove k-kombinacije možemo predstaviti binarnim n-torkama s točno k jedinica.

Definicija 3.2. Grayev kod za k-kombinacije n-članog skupa S (binarne n-torke s točno k jedinica) *je lista koja sadrži sve k-kombinacije od S (binarne n-torke s točno k jedinica) takva da je simetrična razlika susjednih podskupova kardinaliteta 2 (susjedne n-torke razlikuju se za dva bita).*

Grayeve kodove za kombinacije možemo generirati iz zrcaljenog Grayevog koda. Neka je $\mathscr{L}_n$ zrcaljeni Grayev kod iz kojeg uklanjamo sve one n-torke koje nemaju točno k jedinica (ne odgovaraju podskupovima s točno k elemenata). Ono što ostaje je lista svih n-torki s točno k jedinica (podskupova s točno k elemenata) poredanih tako da se susjedne n-torke razlikuju točno u dva bita (simetrična razlika susjednih podskupova je kardinaliteta 2).

Teorem 3.3. *Neka je $\mathscr{L}_n$ zrcaljeni Grayev kod. Lista dobivena uklanjanjem svih n-torki koje nemaju točno k jedinica je Grayev kod za kombinacije.*

Dokaz. Označimo s $\mathscr{L}_{n,k}, 0 \leq k \leq n, n \in \mathbb{N}$ listu dobivenu iz $\mathscr{L}_n$ uklanjanjem svih n-torki koje nemaju točno k jedinica te dokažimo ovaj teorem indukcijom po varijabli n. Baza indukcije je $\mathscr{L}_{1,k}$, gdje je $\mathscr{L}_1 = [0, 1]$, i to je očito Grayev kod za kombinacije za $k = 0, 1$.

Pretpostavimo da je $\mathscr{L}_{n-1,k}$ Grayev kod za kombinacije. Lista $\mathscr{L}_n$ dobivena je rekurzivno, dodavanjem prefiksa 0 i 1 binarnim nizovima iz liste $\mathscr{L}_{n-1}$. Tada, kako bi broj jedinica u binarnim nizovima ostao k, prvi dio liste $\mathscr{L}_{n,k}$ dobivamo dodavanjem prefiksa 0 na binarne nizove iz liste $\mathscr{L}_{n-1,k}$. Drugi dio liste čine nizovi u suprotnom poretku, s prefiksom 1. Drugim riječima, iz liste $\mathscr{L}_{n-1}$ sada uklanjamo nizove koji nemaju točno $k-1$ jedinica i dobivamo listu $\mathscr{L}_{n-1,k-1}$ koja je, prema pretpostavci, Grayev kod za kombinacije.

Sada je potrebno dokazati da se nizovi dobiveni posljednjim elementom liste $\mathscr{L}_{n-1,k}$ i prvim elementom liste $\mathscr{L}_{n-1,k-1}$ razlikuju u točno dva bita. Oba niza dobivena su pomoću posljednjih nizova s k, odnosno $k-1$ jedinica u listi $\mathscr{L}_{n-1}$. Kako je $\mathscr{L}_{n-1}$ Grayev kod za binarne nizove, posljednji nizovi s k i $k-1$ jedinica su uvijek susjedni i razlikuju se u točno jednom bitu. Drugi bit u kojem se susjedni nizovi razlikuju je najznačajniji bit, odnosno dodani prefiks. □

Napišimo algoritam koji poziva funkciju iz algoritma 2.7, provjerava broj jedinica u binarnom nizu i ispisuje one nizove koji imaju točno k jedinica.

Algoritam 3.4. Algoritam Grayevog koda za kombinacije generiran pozivom funkcije iz algoritma 2.7.

```
def k_kombinacije(n, k):

    for niz in zrcaljeni(n):

        if niz.count('1') == k:
            print(niz)
```

Ekvivalentan algoritam za generiranje Grayevog koda za kombinacije je *revolving door* algoritam. Zamislimo da u jednoj prostoriji imamo k osoba, a u drugoj $n-k$ osoba. Između tih dviju prostorija nalaze se rotirajuća vrata kroz koja istovremeno prolaze dvije osobe, po jedna osoba iz svake prostorije. Njihovim prolaskom kroz vrata u svakoj prostoriji ostaje isti broj osoba. Ovaj algoritam možemo shvatiti na analogan način, k osoba u prvoj prostoriji predstavljaju bitove na kojima se nalaze jedinice, a $n-k$ osoba u drugoj prostoriji predstavljaju bitove na kojima se nalaze nule.

Neka je $\mathscr{A}_{n,k}, n \in \mathbb{N}, k \in \mathbb{N}_0, k \leq n$ lista binarnih n-torki s točno k jedinica u revolving door Grayevom kodu. Definiramo je rekurzivno, na sljedeći način:

- Neka je $\mathscr{A}_{1,0} = [0]$.
- Neka je $\mathscr{A}_{1,1} = [1]$.

- Listu $\mathscr{A}_{n,k}$ dobijemo od listi $\mathscr{A}_{n-1,k}$ i $\mathscr{A}_{n-1,k-1}$ tako da

    1. ispisujemo listu $\mathscr{A}_{n-1,k}$ tako da svakom nizu bitova u prefiks dodamo 0,
    2. ispisujemo listu $\mathscr{A}_{n-1,k-1}$ u suprotnom poretku tako da svakom nizu bitova u prefiks dodamo 1.

Dokažimo da je tako dobivena lista zaista Grayev kod za kombinacije. Najprije ćemo dokazati dvije leme koje određuju prvu i posljednju binarnu n-torku liste $\mathscr{A}_{n,k}$.

Lema 3.5. *Za svaki $n \in \mathbb{N}, k \in \mathbb{N}_0, k \leq n$ prvi binarni niz u listi $\mathscr{A}_{n,k}$ ima jedinice na k najmanje značajnih bitova.*

Dokaz. Dokažimo ovu propoziciju indukcijom po varijabli n. Baza indukcije je $\mathscr{A}_{1,0} = [0]$, $\mathscr{A}_{1,1} = [1]$ i ona očito vrijedi. Pretpostavimo da je za fiksiran k takav da je $1 \leq k < n$ prvi binarni niz u listi $\mathscr{A}_{n-1,k}$ oblika $a_1 = 0 \ldots 011 \ldots 1$, odnosno da ima jedinice na k najmanje značajnih bitova.

Budući da se prvi binarni niz u listi $\mathscr{A}_{n,k}$ generira tako da se nizu a_1 u prefiks doda 0, očito je da će i prvi binarni niz liste $\mathscr{A}_{n,k}$ biti oblika $00 \ldots 011 \ldots 1$. □

Lema 3.6. *Za svaki $n \in \mathbb{N}, k \in \mathbb{N}_0, k \leq n$ posljednji binarni niz u listi $\mathscr{A}_{n,k}$ ima jedinicu na najznačajnijem bitu i na $k-1$ najmanje značajnih bitova.*

Dokaz. Za svaki $1 \leq k < n$ posljednji binarni niz u listi $\mathscr{A}_{n,k}$ je prvi binarni niz u listi $\mathscr{A}_{n-1,k-1}$ kojem je dodan prefiks 1. Iz leme 3.5 znamo da prvi binarni niz u listi $\mathscr{A}_{n-1,k-1}$ ima jedinice na $k-1$ najmanje značajnih bitova. Dakle, posljednji element liste $\mathscr{A}_{n,k}$ bit će oblika $100 \ldots 011 \ldots 1$. □

Propozicija 3.7. *Za svaki $n \in \mathbb{N}, k \in \mathbb{N}_0, k \leq n$, $\mathscr{A}_{n,k}$ je Grayev kod za kombinacije.*

Dokaz. Dokažimo ovu propoziciju indukcijom po varijabli n. Baza indukcije je $\mathscr{A}_{1,0} = [0]$, $\mathscr{A}_{1,1} = [1]$ i ona očito vrijedi. Pretpostavimo da je $\mathscr{A}_{n-1,k}$ Grayev kod za kombinacije za neki k takav da $1 \leq k < n$.

Prvi dio liste $\mathscr{A}_{n,k}$ čini $\binom{n-1}{k}$ binarnih nizova koji su dobiveni ispisivanjem liste $\mathscr{A}_{n-1,k}$ s prefiksom 0. Po pretpostavci se susjedni nizovi iz te liste razlikuju točno u dva bita. Nakon toga slijedi $\binom{n-1}{k-1}$ binarnih nizova dobivenih ispisivanjem liste $\mathscr{A}_{n-1,k-1}$ i dodavanjem broja 1 u prefiks. Prema pretpostavci, $\mathscr{A}_{n-1,k}$ je Grayev kod za kombinacije za proizvoljan k pa je onda i za $k-1$. Dakle, susjedni binarni nizovi u drugom dijelu liste $\mathscr{A}_{n,k}$ se također razlikuju točno u dva bita.

Preostaje nam dokazati da se $\binom{n-1}{k}$-ti element i $\left(\binom{n-1}{k}+1\right)$-ti element također razlikuju u dva bita. Označimo ih a_i i a_{i+1}, redom. Iz leme 3.6 znamo da su oblika:

$$a_i = 0100\ldots\underbrace{011\ldots1}_{k-1} \qquad a_{i+1} = 1100\ldots\underbrace{011\ldots1}_{k-2},$$

odnosno razlikuju se u točno dva bita, najznačajnijem bitu i u $(k-1)$.-vom bitu s desne strane. □

Napišimo rekurzivan algoritam koji koristi revolving door pravilo i ispisuje liste prikazane u tablici 7.

Algoritam 3.8. Algoritam revolving door Grayevog koda.

```
def revolving_door(n, k):

    if n == 1 and k == 1:
        return ['1']

    elif n == 1 and k == 0:
        return ['0']

    else:
        Grayev_kod = []

        if n > k and n > 1:
            Pom_lista = revolving_door(n - 1, k)

            for niz in Pom_lista:
                Grayev_kod.append('0' + niz)

        if k > 0:
            Pom_lista = revolving_door(n - 1, k - 1)

            for niz in list(reversed(Pom_lista)):
                Grayev_kod.append('1' + niz)

        return Grayev_kod
```

Algoritam 3.8 poziva rekurzivne funkcije, ovisno o vrijednostima varijabli n i k, a nakon toga dodaje 0 ili 1 u prefikse binarnih nizova.

U tablici 7 prikazani su Grayevi kodovi za kombinacije dobiveni algoritmima 3.4 i 3.8. Iz tablice i lema 3.5 i 3.6 lako se može uočiti da je ovaj Grayev kod ciklički.

$\mathscr{A}_{3,3}$	$\mathscr{A}_{4,3}$	$\mathscr{A}_{5,3}$
111	0111	00111
	1101	01101
	1110	01110
	1011	01011
		11001
		11010
		11100
		10101
		10110
		10011

Tablica 7: Grayevi kodovi za kombinacije generirani algoritmom 3.4 i revolving door algoritmom.

Još jedan algoritam za generiranje Grayevih kodova za kombinacije je algoritam *minimalnih promjena*. Za razliku od revolving door algoritma, ovaj algoritam ima jedan dodatan uvjet za binarne n-torke: između bitova u kojima se susjedne n-torke razlikuju nema jedinica. Neka je $\mathscr{B}_{n,k}, n \in \mathbb{N}, k \in \mathbb{N}_0, k \leq n$ lista binarnih n-torki s točno k jedinica u Grayevom kodu s minimalnim promjenama. Definiramo je rekurzivno:

- Neka je $\mathscr{B}_{n,0}$ lista koja sadrži n-torku s n nula.
- Neka je $\mathscr{B}_{n,n}$ lista koja sadrži n-torku s n jedinica.
- Listu $\mathscr{B}_{n,k}$ dobijemo od listi $\mathscr{B}_{n-1,k}$, $\mathscr{B}_{n-2,k-1}$ i $\mathscr{B}_{n-2,k-2}$ tako da
  1. ispisujemo listu $\mathscr{B}_{n-1,k}$ tako da svakom nizu bitova u prefiks dodamo 0,
  2. ispisujemo listu $\mathscr{B}_{n-2,k-1}$ u suprotnom poretku tako da svakom nizu bitova u prefiks dodamo 10,
  3. ispisujemo listu $\mathscr{B}_{n-2,k-2}$ tako da svakom nizu bitova u prefiks dodamo 11.

Napišimo sada algoritam koji generira Grayev kod za kombinacije s minimalnim promjenama.

Algoritam 3.9. Algoritam Grayevog koda za kombinacije s minimalnim promjenama.

```
def minimalna_promjena(n, k):

    if k == 0:
        return ['0' * n]

    elif n == k:
        return ['1' * n]

    else:
        Grayev_kod = []

        if n > k and n > 1 and k >= 0:
            Pom_lista = minimalna_promjena(n - 1, k)

            for niz in Pom_lista:
                Grayev_kod.append('0' + niz)

        if n > k and n > 1 and k > 0:
            Pom_lista = minimalna_promjena(n - 2, k - 1)

            for niz in list(reversed(Pom_lista)):
                Grayev_kod.append('10' + niz)

        if n >= k and n > 2 and k > 1:
            Pom_lista = minimalna_promjena(n - 2, k - 2)

            for niz in Pom_lista:
                Grayev_kod.append('11' + niz)

        return Grayev_kod
```

Algoritam 3.9, ovisno o vrijednostima varijabli n i k, poziva rekurzivne funkcije, a zatim dobivenim binarnim nizovima u prefiks dodaje 0, 10 ili 11. Grayevi kodovi za kombinacije generirani revolving door algoritmom i algoritmom s minimalnim promjenama prikazani su u tablici 8.

$\mathscr{A}_{5,3}$	$\mathscr{B}_{5,3}$
00111	00111
01101	01011
01110	01101
01011	01110
11001	10110
11010	10101
11100	10011
10101	11001
10110	11010
10011	11100

Tablica 8: Grayev kod za kombinacije $\mathscr{A}_{5,3}$ generiran revolving door algoritmom i Grayev kod za kombinacije $\mathscr{B}_{5,3}$ generiran algoritmom minimalnih promjena.

4 Grayevi kodovi za permutacije

U n-članom skupu S poredak elemenata nije važan. No, elemente skupa uvijek možemo poredati tako da kažemo koji je element prvi, koji je drugi itd. Takve uređene n-torke zovemo permutacijama.

Definicija 4.1. *Neka je S konačan skup.* Permutacija *od S je bijekcija skupa S u samog sebe.*

Permutaciju $\pi : \{a_1, \ldots, a_n\} \to \{a_1, \ldots, a_n\}$ najčešće zapisujemo kao uređenu n-torku

$$(\pi(a_1), \ldots, \pi(a_n)),$$

gdje vrijedi $a_1 \mapsto \pi(a_1)$, $\ldots$, $a_n \mapsto \pi(a_n)$. Katkad koristimo i zapis

$$\begin{pmatrix} a_1 & \ldots & a_n \\ \pi(a_1) & \ldots & \pi(a_n) \end{pmatrix}.$$

Teorem 4.2. *Broj permutacija n-članog skupa je $n!$.*

Dokaz. Dokažimo ovaj teorem kombinatorno. Pretpostavimo da je zadan n-člani skup $\{1, \ldots, n\}$. Tada je permutacija tog skupa uređena n-torka brojeva $1, \ldots, n$. Za izbor prvog elementa postoji n mogućnosti, za izbor drugog elementa $n-1$ mogućnosti, itd.

Konačno, prema principu produkta, broj permutacija n-članog skupa je

$$n \cdot (n-1) \cdots 2 \cdot 1 = n!.$$

□

Drugi način zapisa permutacije je *cikličkи zapis*. Kažemo da je permutacija *ciklus* ili *ciklička permutacija* i pišemo $(a_1\ a_2 \ldots a_k)$ ako preslikava

$$a_1 \mapsto a_2 \mapsto \cdots \mapsto a_k \mapsto a_1,$$

gdje su $a_1, \ldots, a_k \in S$ međusobno različiti, a ostali elementi skupa S su fiksni. Možemo uočiti da u cikličkoj permutaciji nije bitno koji je element prvi, već je bitan samo ciklički poredak elemenata. Vrijedi $a_k \mapsto a_1$ pa je $(a_i \ldots a_k\ a_1 \ldots a_{i-1})$ isti ciklus kao i $(a_1\ a_2 \ldots a_k)$.

Permutacija $(\pi(a_1), \ldots, \pi(a_n))$ je *transpozicija* ako postoje $1 \leq i < j \leq n$, takvi da je $\pi(a_i) = a_j$, $\pi(a_j) = a_i$ i $\pi(a_k) = a_k$ za sve $k \neq i, j$, tj. ako je ciklus duljine dva.

Prikažimo sada na skupu $\{a_1, \ldots, a_n\}$ kompoziciju dviju permutacija.

$$\sigma \circ \pi = \begin{pmatrix} a_1 & \ldots & a_n \\ \sigma(a_1) & \ldots & \sigma(a_n) \end{pmatrix} \circ \begin{pmatrix} a_1 & \ldots & a_n \\ \pi(a_1) & \ldots & \pi(a_n) \end{pmatrix}$$

Prvo djeluje preslikavanje na desnoj strani pa vrijedi $(\sigma \circ \pi)(a_1) = \sigma(\pi(a_1))$, $\ldots$, $(\sigma \circ \pi)(a_n) = \sigma(\pi(a_n))$. Kompoziciju tada možemo napisati na sljedeći način

$$\sigma \circ \pi = \begin{pmatrix} a_1 & \ldots & a_n \\ \sigma(\pi(a_1)) & \ldots & \sigma(\pi(a_n)) \end{pmatrix}.$$

Skup svih permutacija skupa S s obzirom na kompoziciju je grupa. Ta se grupa naziva *simetričnom grupom*.

Svaka se permutacija može zapisati kao kompozicija ciklusa na međusobno disjunktnim skupovima. Taj je zapis jedinstven, do na poredak ciklusa i odabir početnih točaka ciklusa.

Svaku permutaciju možemo zapisati kao kompoziciju transpozicija. Taj zapis nije jedinstven. Na primjer, ciklus (1 2 3) možemo zapisati kao kompoziciju transpozicija na dva načina, (1 2) $\circ$ (2 3) i (2 3) $\circ$ (1 3).

Permutacije nekog skupa najčešće ispisujemo u *leksikografskom poretku*. Kažemo da je permutacija $(a_1, a_2, \ldots, a_n)$ manja od $(b_1, b_2, \ldots, b_n)$ ako je $a_1 < b_1$ ili $a_1 = b_1$, $a_2 = b_2, \ldots, a_{i-1} = b_{i-1}, a_i < b_i$, za neki $i \in \{2, \ldots, n\}$. Pretpostavimo da nam je poznata neka permutacija π skupa $\{1, \ldots, n\}$ te da želimo pronaći permutaciju koja joj slijedi. Algoritam koji pronalazi iduću permutaciju u leksikografskom poretku definiramo na sljedeći način:

- Počevši od predzadnjeg elementa, tražimo najveći i takav da vrijedi

$$\pi(i) < \pi(i+1) > \pi(i+2) > \ldots > \pi(n).$$

- Počevši od zadnjeg elementa, tražimo najveći $j > i$ takav da je $\pi(j) > \pi(i)$. Tada mijenjamo mjesta elementima $\pi(i)$ i $\pi(j)$.

- Elemente $\pi(i+1), \ldots, \pi(n)$ zapisujemo u suprotnom redoslijedu.

Napišimo algoritam koji ispisuje sve permutacije n-članog skupa u leksikografskom poretku. Algoritam 4.3 definira početnu permutaciju kojoj su elementi uređeni u rastućem poretku. Nakon toga izvršava se petlja koja generira i ispisuje iduću permutaciju prema prethodno opisanim koracima, sve dok ne dođe do posljednje permutacije u leksikografskom poretku.

Algoritam 4.3. Algoritam koji ispisuje permutacije skupa u leksikografskom poretku.

```
def permutacije_leksikografski(n):

    pi = list(range(1, n + 1))
    print(pi)
    i = n - 2

    while i >= 0:
        i = n - 2
        j = n - 1

        while pi[i] > pi[i + 1]:
            i -= 1

        if i >= 0:

            while pi[i] > pi[j]:
                j -= 1

            s = pi[i]
            pi[i] = pi[j]
            pi[j] = s

            pomocni = pi[i + 1:]
            pi[i + 1:] = pomocni[::-1]
            print(pi)
```

U tablici 9 prikazana je lista permutacija koju algoritam 4.3 ispisuje za $n = 3$.

$(1, 2, 3)$
$(1, 3, 2)$
$(2, 1, 3)$
$(2, 3, 1)$
$(3, 1, 2)$
$(3, 2, 1)$

Tablica 9: Permutacije skupa $\{1, 2, 3\}$ u leksikografskom poretku.

Permutaciju smo definirali kao bijekciju koja preslikava neki skup u samog sebe pa je očito da se dvije permutacije razlikuju u najmanje dvije pozicije.

Teorem 4.4. *Neka su π_1 i π_2 dvije permutacije skupa S. One se razlikuju u onoliko mjesta koliko je nefiksnih točaka u kompoziciji $\pi_2^{-1} \circ \pi_1$.*

Dokaz. Neka su dane permutacije

$$\pi_1 = \begin{pmatrix} 1 & \dots & n \\ \pi_1(1) & \dots & \pi_1(n) \end{pmatrix} \quad \text{i} \quad \pi_2 = \begin{pmatrix} 1 & \dots & n \\ \pi_2(1) & \dots & \pi_2(n) \end{pmatrix}.$$

Vrijedi $\pi_1(i) = \pi_2(i)$ ako i samo ako vrijedi $\left(\pi_2^{-1} \circ \pi_1\right)(i) = i$, tj. ako je i fiksna točka permutacije $\pi_2^{-1} \circ \pi_1$. □

Ako je kompozicija $\pi_2^{-1} \circ \pi_1$ transpozicija, onda se permutacije π_1 i π_2 razlikuju na točno dva mjesta. Drugim riječima, ako se dvije permutacije razlikuju na točno dva mjesta, onda postoji jedna transpozicija koja zamjenjuje ta dva mjesta. Sada možemo promatrati listu permutacija gdje se susjedne permutacije razlikuju na točno dva mjesta.

Definicija 4.5. Grayev kod za permutacije *je lista koja sadrži sve permutacije n-članog skupa takva da se svake dvije susjedne permutacije razlikuju za točno jednu transpoziciju.*

Jedan od načina generiranja takvog Grayevog koda je da se susjedne permutacije razlikuju do na zamjenu dva susjedna elementa. Da takav Grayev kod postoji zasebno su dokazali Johnson [2] i Trotter [12] pa ćemo način generiranja takvog koda zvati *Johnson-Trotterov* algoritam. Neka je $\mathscr{R}_n, n \in \mathbb{N}$ lista permutacija n-članog skupa u Grayevom kodu generiranom Johnson-Trotterovim algoritmom. Definiramo je rekurzivno:

- Neka je $\mathscr{R}_1 = [1]$.
- Listu $\mathscr{R}_n$ dobivamo od liste $\mathscr{R}_{n-1}$ tako da u svaku permutaciju dodamo element n na svaku poziciju. Prvoj permutaciji dodajemo n na svaku poziciju počevši od desne strane prema lijevoj, drugoj permutaciji dodajemo n na svaku poziciju počevši od lijeve strane prema desnoj, itd.

Napišimo algoritam za generiranje takvog Grayevog koda za permutacije.

Algoritam 4.6. Johnson-Trotterov algoritam Grayevog koda za permutacije.

```
def johnson_trotter(n):

    if n == 1:
        Grayev_kod = [[1]]

    else:
        Pom_lista = johnson_trotter(n - 1)
        Grayev_kod = []
        i = 0

        for perm in Pom_lista:

            if i == 0:
                for j in range(n - 1, -1, -1):
                    pomocna = list(perm)
                    pomocna.insert(j, n)
                    Grayev_kod.append(pomocna)
                i = 1

            else:
                for j in range(n):
                    pomocna = list(perm)
                    pomocna.insert(j, n)
                    Grayev_kod.append(pomocna)
                i = 0

    return Grayev_kod
```

Algoritam 4.6 je rekurzivan algoritam koji za $n = 1$ definira listu [1], a inače poziva funkciju s vrijednosti $n-1$ i za svaku permutaciju u vraćenoj listi izvršava petlju u kojoj se na svaku poziciju u permutaciji dodaje n. Element n dodaje se naizmjenično s desna na lijevo, odnosno s lijeva na desno. Rezultat ovog algoritma za $n = 2, 3, 4$ prikazan je u tablici 10. Možemo uočiti da je ovaj Grayev kod za permutacije ciklički, odnosno da se prva i posljednja permutacija također razlikuju na točno dva mjesta.

$\mathscr{R}_2$	$\mathscr{R}_3$	$\mathscr{R}_4$	
12	123	1234	4321
21	132	1243	3421
	312	1423	3241
	321	4123	3214
	231	4132	2314
	213	1432	2341
		1342	2431
		1324	4231
		3124	4213
		3142	2413
		3412	2143
		4312	2134

Tablica 10: Grayev kod za permutacije generiran Johnson-Trotterovim algoritmom.

Suprotan pristup generiranju svih permutacija n-članog skupa je da se svake dvije susjedne permutacije π_1 i π_2 razlikuju na svakom mjestu, odnosno da permutacija $\pi_2^{-1} \circ \pi_1$ nema fiksnih točaka.

Definicija 4.7. Deranžman *nekog skupa je permutacija tog skupa bez fiksnih točaka.*

Teorem 4.8. *Broj deranžmana n-članog skupa je dan formulom*

$$d(n) = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!} = \left\lfloor \frac{n!}{e} \right\rfloor, n \in \mathbb{N}.$$

Dokaz. Neka je A skup svih permutacija skupa $S = \{1, 2, \ldots, n\}$. Označimo s $A_i,\ i = 1, 2, \ldots, n$ skup svih permutacija $\pi \in S$ za koje je $\pi(i) = i$. Tada je očito $d(n) = \left|\overline{A_1} \cap \overline{A_2} \cap \ldots \cap \overline{A_n}\right|$ pa ćemo koristiti formulu uključivanja i isključivanja.

Permutacije π u skupu A_1 su oblika $(1, a_2, \ldots, a_n)$ i ima ih $|A_1| = (n-1)!$. Analogno vrijedi i $|A_i| = (n-1)!$ za $i = 1, \ldots, n$. Permutacije u skupu $A_1 \cap A_2$ su oblika $(1, 2, a_3 \ldots, a_n)$ i ima ih $|A_1 \cap A_2| = (n-2)!$. Analogno $|A_i \cap A_j| = (n-2)!$ za $1 \leq i < j \leq n$. Općenito, za bilo koju k-kombinaciju $\{a_1, a_2, \ldots, a_k\} \subseteq \{1, 2, \ldots, n\}$ vrijedi $|A_{a_1} \cap A_{a_2} \cap \ldots \cap A_{a_k}| = (n-k)!$.

Kako k-kombinacija u n-članom skupu ima $\binom{n}{k}$, to vrijedi

$$\begin{aligned} d(n) &= \binom{n}{0} n! - \binom{n}{1} (n-1)! + \binom{n}{2} (n-2)! + \cdots + (-1)^n \binom{n}{n} (0)! \\ &= \sum_{k=0}^{n} (-1)^k \binom{n}{k} (n-k)! \\ &= \sum_{k=0}^{n} (-1)^k \frac{n!}{k!} \\ &= n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}. \end{aligned}$$

Preostaje nam dokazati drugi dio jednakosti, $d(n) = \left\lfloor \frac{n!}{e} \right\rfloor$. Vrijedi

$$\frac{n!}{e} - d(n) = n! \left(\sum_{k=0}^{\infty} \frac{(-1)^k}{k!} - \sum_{k=0}^{n} \frac{(-1)^k}{k!} \right) = n! \left(\sum_{k=n+1}^{\infty} \frac{(-1)^k}{k!} \right).$$

Sada je dovoljno dokazati da je $\left| \frac{n!}{e} - d(n) \right| \leq \frac{1}{2}$. Izlučivanjem $\frac{1}{(n+1)!}$ iz sume s desne strane jednakosti, dobivamo

$$\left| \frac{n!}{e} - d(n) \right| = \frac{1}{n+1} \left| 1 - \frac{1}{n+2} + \frac{1}{(n+2)(n+3)} - \cdots \right|.$$

Red unutar apsolutne vrijednosti s desne strane jednakosti alternira i konvergira u sumu strogo manju od 1. Kako je $n \geq 1$, vrijedi

$$\left| \frac{n!}{e} - d(n) \right| < \frac{1}{n+1} \cdot 1 \leq \frac{1}{2}.$$

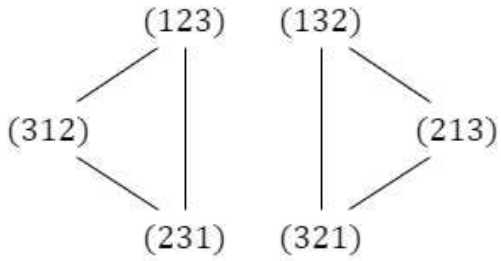
□

Ako je kompozicija $\pi_2^{-1} \circ \pi_1$ deranžman, onda se prema teoremu 4.4 permutacije π_1 i π_2 razlikuju na svih n mjesta. Permutacije n-članog skupa, osim Grayevim kodom za permutacije, sada možemo ispisati i na način da se susjedne permutacije razlikuju na svih n-mjesta.

Definicija 4.9. Kod za permutacije s deranžmanima *je lista koja sadrži sve permutacije n-članog skupa takva da se svake dvije susjedne permutacije razlikuju na svih n mjesta.*

Neka je G_n graf čiji vrhovi odgovaraju permutacijama n-članog skupa, a dva su vrha susjedna ako se permutacije koje ih predstavljaju razlikuju na

svih n mjesta. Kod za permutacije s deranžmanima tada je Hamiltonov put u grafu G_n. Kažemo da je graf *povezan* ako između svaka dva njegova vrha postoji put. Inače je graf *nepovezan*. Na slici 6 prikazan je graf G_3 koji je nepovezan pa je očito da za $n = 3$ nije moguće generirati Hamiltonov put, niti kod za permutacije s deranžmanima.



Slika 6: Graf G_3.

Za povezan graf kažemo da je 2-*povezan* ako izbacivanjem bilo kojeg vrha (zajedno s bridovima koji ga sadrže) ponovno dobivamo povezani graf. *Stupanj* vrha grafa definiramo kao broj bridova koji ga sadrže. Kažemo da je graf k-*regularan* ako je svaki njegov vrh stupnja k. Da bi dokazali da je u grafu $G_n, n \neq 3$, moguće generirati Hamiltonov ciklus, bit će nam potreban sljedeći teorem.

Teorem 4.10 (Jackson)**.** *Svaki* 2*-povezan,* k*-regularan graf s najviše* $3k$ *vrhova je Hamiltonov graf.*

Teorem 4.11 (Wilf [13])**.** *Za* $n \neq 3$ *graf* G_n *dopušta Hamiltonov ciklus.*

Dokaz. Za $n = 1, 2$ teorem očito vrijedi. Pretpostavimo da je $n \geq 4$. Lako je uočiti da je tada graf G_n 2-povezan. Preostaje nam dokazati da su svi vrhovi grafa stupnja k, gdje je $3k \geq n$.

Dva vrha u grafu G_n su susjedna ako se permutacije koje ih predstavljaju razlikuju na svih n mjesta. Tada je, prema teoremu 4.8, svaki vrh grafa stupnja $k = \left\lfloor \frac{n!}{e} \right\rfloor$. Nadalje, vrijedi

$$\left\lfloor \frac{n!}{e} \right\rfloor \geq \frac{n!}{3} \geq \frac{n}{3}, \quad n \geq 1.$$

Sada možemo zaključiti da je za $n \geq 4$ graf G_n 2-povezan i svaki njegov vrh je stupanja k, gdje je $k \geq \frac{n}{3}$. Prema teoremu 4.10 graf G_n je tada Hamiltonov graf, odnosno dopušta Hamiltonov ciklus. □

Jedan od algoritama koji generira kod za permutacije s deranžmanima pripisuje se *Lynn Yarbrough* [7]. Ideja ovog algoritma je da uzima svaku permutaciju iz Johnson-Trotterove liste $\mathscr{R}_{n-1}$ (tablica 10) i dodaje joj element n na posljednje mjesto. Dobivena permutacija zatim se ciklički rotira udesno $n-1$ puta te se zamjenjuju mjesta posljednjim dvijema rotacijama. Permutacija $(a_1, a_2, \ldots, a_n)$ rotira se na sljedeći način:

$$\begin{gathered}
(a_n, a_1, \ldots, a_{n-1}), \\
(a_{n-1}, a_n, a_1, \ldots, a_{n-2}), \\
\vdots \\
(a_4, \ldots, a_n, a_1, a_2, a_3), \\
(a_2, a_3 \ldots, a_n, a_1), \\
(a_3, \ldots, a_n, a_1, a_2).
\end{gathered}$$

Za $n = 1, 2$ je očito da Yarbroughin algoritam generira kod za permutacije s deranžmanima. Neka je π_i permutacija Johnson-Trotterove liste za neki $4 \leq i < (n-1)!$, a $\pi_{i,j}, 1 \leq j \leq n$ permutacija u Yarbroughinoj listi dobivena cikličkom rotacijom π_i. Sve permutacije $\pi_{i,j}$ za neki i tada se očito razlikuju u svakoj poziciji. Preostaje nam dokazati da to vrijedi i kada se prelazi iz jedne Johnson-Trotterove permutacije u drugu. Ranije smo pokazali da se susjedne permutacije Johnson-Trotterove liste razlikuju za transpoziciju uzastopnih elemenata. Zbog zamjene posljednje dvije rotacije, permutacija $\pi_{i,n}$ je dobivena cikličkom rotacijom π_i za dva mjesta ulijevo. Kako je $i \geq 4$, očito je da će se permutacije $\pi_{i,n}$ i $\pi_{i+1,1}$ razlikovati na svih n mjesta. Ovaj je algoritam konstruktivan dokaz teorema 4.11.

Napišimo sada algoritam koji generira kod za permutacije s deranžmanima. Algoritam 4.12 poziva funkciju iz algoritma 4.6 koja generira Grayev kod za permutacije za $n-1$, a zatim izvršava petlju za svaku permutaciju iz te liste. U petlji se dodaje element n na posljednje mjesto u permutaciji i zatim se ta permutacija rotira ciklički udesno za $1, 2, ..., n-3$ mjesta. Predzadnja rotacija je ciklička rotacija udesno za $n-1$ mjesta, a posljednja je rotacija za $n-2$ mjesta.

Algoritam 4.12. Yarbroughin algoritam koda za permutacije s deranžmanima.

```
def yarbrough(n):
    Pom_lista = johnson_trotter(n - 1)

    for perm in Pom_lista:
        perm = perm + [n]

        for i in range(n - 2):
            print(perm)
            perm = rotacija(perm, n)

        print(rotacija(perm, n))
        print(perm)

def rotacija(perm, n):
    return [perm[n - 1]] + perm[:n - 1]
```

Rezultat algoritma 4.12 za $n = 4$ prikazan je u tablici 11.

1234	3124	2314
4123	4312	4231
2341	1243	3142
3412	2431	1423
1324	3214	2134
4132	4321	4213
3241	2143	1342
2413	1432	3421

Tablica 11: Kod za permutacije s deranžmanima generiran Yarbroughinim algoritmom.

5 Grayevi kodovi za particije

Osim Grayevih kodova za podskupove, kombinacije i permutacije, moguće je generirati i Grayev kod za particije.

Definicija 5.1. Particija skupa S *je skup međusobno disjunktnih nepraznih podskupova od* S *koji u uniji daju cijeli* S.

Broj particija n-članog skupa jednak je broju relacija ekvivalencije na n-članom skupu. Elemente particije zvat ćemo *blokovima* i zapisivat ćemo ih u rastućem poretku njihovih najmanjih elemenata. Na primjer, particiju $\pi = \{\{9\}, \{1,2,7\}, \{4,6\}, \{3,5,8\}\}$ zapisat ćemo $\{1,2,7\}$, $\{3,5,8\}$, $\{4,6\}$, $\{9\}$.

Teorem 5.2. *Postoji bijekcija između skupa svih particija skupa* $\{1,2,\ldots,n\}$ *i skupa svih nizova* $a_1a_2\ldots a_n$, *gdje je* $a_1 = 1$ *i* $a_i \leq 1 + max\{a_1, a_2, \ldots, a_{i-1}\}$ *za* $i = 2,\ldots,n$.

Skup svih particija skupa $\{1,2,\ldots,n\}$ označavat ćemo sa P_n, a skup svih nizova $a_1a_2\ldots a_n$ iz teorema 5.2 označavat ćemo sa T_n. Označimo blokove particije iz P_n, redom, $1,2,3,\ldots$. U nizu $a_1a_2\ldots a_n \in T_n$ tada $a_i, i \in \{1,2,\ldots,n\}$ označava blok koji sadrži element i. Niz koji određuje π iz prethodnog primjera je 1 1 2 3 2 3 1 2 4, a za $n = 4$ bijekcija je prikazana u tablici 12.

Definicija 5.3. *Broj particija* n*-članog skupa zovemo* n*-ti* Bellov broj *i označavamo ga* B_n.

Teorem 5.4. *Vrijedi*

$$B_n = \sum_{k=1}^{n} \binom{n-1}{k-1} B_{n-k}, \ \forall n \in \mathbb{N}.$$

Dokaz. Neka je $S = \{1,\ldots,n\}$ i uzmimo neku particiju $\pi_1,\ldots,\pi_l$ od S. Neka je π_j skup koji sadrži element n. Dakle, $\pi_j = S' \cup \{n\}$, gdje je $S' \subseteq \{1,\ldots,n-1\}$. Ostali $\pi_i, i \neq j$ čine particiju skupa $\{1,\ldots,n-1\}\setminus\pi_j$. Skupovi S' i $\pi_i, i \neq j$ potpuno određuju particiju.

Ukoliko je $|S'| = k-1$ postoji $\binom{n-1}{k-1}$ mogućnosti za izbor skupa S', te B_{n-k} mogućnosti za izbor ostalih elemenata particije. Kako za k možemo uzeti bilo koji broj između (i uključivo) 1 i n, slijedi tražena formula. □

S_n	T_n
$\{1,2,3,4\}$	1 1 1 1
$\{1,2,3\}$, $\{4\}$	1 1 1 2
$\{1,2,4\}$, $\{3\}$	1 1 2 1
$\{1,2\}$, $\{3,4\}$	1 1 2 2
$\{1,2\}$, $\{3\}$, $\{4\}$	1 1 2 3
$\{1,3,4\}$, $\{2\}$	1 2 1 1
$\{1,3\}$, $\{2,4\}$	1 2 1 2
$\{1,3\}$, $\{2\}$, $\{4\}$	1 2 1 3
$\{1,4\}$, $\{2,3\}$	1 2 2 1
$\{1\}$, $\{2,3,4\}$	1 2 2 2
$\{1\}$, $\{2,3\}$, $\{4\}$	1 2 2 3
$\{1,4\}$, $\{2\}$, $\{3\}$	1 2 3 1
$\{1\}$, $\{2,4\}$, $\{3\}$	1 2 3 2
$\{1\}$, $\{2\}$, $\{3,4\}$	1 2 3 3
$\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$	1 2 3 4

Tablica 12: Particije skupova u leksikografskom poretku.

Napišimo rekurzivan algoritam koji poziva funkciju s vrijednosti $n-1$ i u sufiks svake particije iz vraćene liste dodaje elemente $a_n = 1, 2, \ldots, 1 + max\{a_1, \ldots, a_{n-1}\}$, redom. Takav algoritam ispisuje nizove iz skupa T_n u leksikografskom poretku koji je prikazan u tablici 12.

Algoritam 5.5. Algoritam koji ispisuje particije u leksikografskom poretku.

```
def particije_leksikografski(n):

    if n == 1:
        return [[1]]

    else:
        Pom_lista = particije_leksikografski(n - 1)
        Part = []

        for pomocna in Pom_lista:

            for i in range(1, max(pomocna) + 2):
                Part.append(pomocna + [i])

        return Part
```

Možemo uočiti da se susjedne particije u leksikografskom poretku razlikuju za zamjenu više elemenata u blokovima. Iz tog ćemo razloga promotriti

kod takav da se susjedne particije razlikuju za premještaj jednog elementa među susjednim blokovima.

Definicija 5.6. Grayev kod za particije skupa *je lista koja sadrži sve particije n-članog skupa takva da se dvije susjedne particije razlikuju samo za premještaj jednog elementa među dva susjedna bloka.*

Jedan takav Grayev kod za particije generirat ćemo *Knuthovim* algoritmom [13]. Neka je $\mathscr{K}_n, n \in \mathbb{N}$ lista particija n-članog skupa u Grayevom kodu za particije generiranom Knuthovim algoritmom. Definiramo je rekurzivno:

- Neka je $\mathscr{K}_n = [1]$.
- Listu $\mathscr{K}_n$ dobivamo od liste $\mathscr{K}_{n-1}$ tako da
  1. Dodajemo element n u svaki blok jedne particije iz liste $\mathscr{K}_{n-1}$, počevši od prvog. Na kraju dodajemo posljednji blok $\{n\}$,
  2. Sljedećoj particiji liste $\mathscr{K}_{n-1}$ dodajemo blok $\{n\}$, a zatim dodajemo element n u svaki blok, počevši od zadnjeg.

     Ponavljamo prethodna dva koraka na preostalim particijama.

Na primjer, particiji $\{\{1,3\},\{2,4,5\}\}$ dodajemo element 6 i dobivamo sljedeće particije: $\{\{1,3,6\},\{2,4,5\}\}$, $\{\{1,3\},\{2,4,5,6\}\}$, $\{\{1,3\},\{2,4,5\},\{6\}\}$. Za $n = 4$ Grayev kod generiran opisanim Knuthovim algoritmom prikazan je u tablici 13.

$\{1,2,3,4\}$	1 1 1 1
$\{1,2,3\}$, $\{4\}$	1 1 1 2
$\{1,2\}$, $\{3\}$, $\{4\}$	1 1 2 3
$\{1,2\}$, $\{3,4\}$	1 1 2 2
$\{1,2,4\}$, $\{3\}$	1 1 2 1
$\{1,4\}$, $\{2\}$, $\{3\}$	1 2 3 1
$\{1\}$, $\{2,4\}$, $\{3\}$	1 2 3 2
$\{1\}$, $\{2\}$, $\{3,4\}$	1 2 3 3
$\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$	1 2 3 4
$\{1\}$, $\{2,3\}$, $\{4\}$	1 2 2 3
$\{1\}$, $\{2,3,4\}$	1 2 2 2
$\{1,4\}$, $\{2,3\}$	1 2 2 1
$\{1,3,4\}$, $\{2\}$	1 2 1 1
$\{1,3\}$, $\{2,4\}$	1 2 1 2
$\{1,3\}$, $\{2\}$, $\{4\}$	1 2 1 3

Tablica 13: Grayev kod za particije skupova generiran Knuthovim algoritmom.

Napišimo Knuthov algoritam koji generira Grayev kod za particije skupova.

Algoritam 5.7. Knuthov algoritam Grayevog koda za particije.

```
def knuth(n):

    if n == 1:
        return [[1]]

    else:
        Pom_lista = knuth(n - 1)
        Part = []
        i = 0

        for pomocna in Pom_lista:

            if i == 0:

                for i in range(1, max(pomocna) + 2):
                    Part.append(pomocna + [i])

                i = 1

            else:

                for i in range(max(pomocna) + 1, 0, -1):
                    Part.append(pomocna + [i])

                i = 0

        return Part
```

Algoritam 5.7 je rekurzivan algoritam koji za $n = 1$ definira listu [1], a inače poziva funkciju s vrijednosti $n - 1$. Za svaku particiju u vraćenoj listi izvršava se petlja koja na prethodno opisan način dodaje element n u sve blokove.

Literatura

[1] M. Bašić, M. Berljafa, M. Božić, M. Erceg, I. Gavran, I. Krijan, M. Marohnić, *Diskretna matematika*, skripta, PMF-Matematički odsjek, Sveučilište u Zagrebu, Zagreb, 2015.

[2] S. M. Johnson, *Generation of permutations by adjacent transposition*, Mathematics of Computation, **17** (1963), 282-285.

[3] D. E. Knuth, *The art of computer programming, Volume 4A: Combinatorial algorithms, Part 1*, Addison-Wesley Professional, New Jersey, 2011.

[4] D. L. Kreher, D. R. Stinson, *Combinatorial algorithms. Generation, enumeration, and search*, CRC Press, Boca Raton, Florida, 1999.

[5] I. Nakić, *Diskretna matematika*, skripta, PMF-Matematički odsjek, Sveučilište u Zagrebu, Zagreb, 2012.

[6] A. Nijenhuis, H. S. Wilf, *Combinatorial algorithms for computers and calculators, 2nd edition*, Academic Press, New York, 1978.

[7] D. F. Rall, P. J. Slater, *Generating all permutations by graphical derangements*, Neobjavljeni rukopis, 1987.

[8] C. Savage, *A survey of combinatorial Gray codes*, SIAM Rev. **39** (1997), 605–629.

[9] C. Savage, P. Winkler, *Monotone Gray codes and the Middle Levels Problem*, J. Combin. Theory Ser. A **70** (1995), 230–248.

[10] S. Skiena, *Implementing discrete mathematics*, Addison–Wesley, 1990.

[11] B. Širola, *Algebarske strukture*, skripta, PMF-Matematički odsjek, Sveučilište u Zagrebu, Zagreb.

[12] H. F. Trotter, *PERM (Algorithm 115)*, Communications of the ACM, **5(8)** (1962), 434-435.

[13] H. S. Wilf, *Combinatorial algorithms: an update*, SIAM, Philadelphia, 1989.

[14] Prabook, *Frank Gray*, dostupno na: `https://prabook.com/web/frank.gray/2439716` (srpanj 2020.).

[15] Wikipedia, *Gray code*, dostupno na: `https://en.wikipedia.org/wiki/Gray_code` (srpanj 2020.).

Sažetak

Tema ovog diplomskog rada su Grayevi kodovi, odnosno metode generiranja kombinatornih objekata takvih da se susjedni objekti razlikuju na neki unaprijed određen način ili, najčešće, minimalno.

Prema patentu Franka Graya definirali smo Grayev kod za binarne nizove i za podskupove nekog skupa. Istu ideju primijenili smo i na druge kombinatorne objekte: kombinacije, permutacije i particije. Uz Grayev kod za permutacije definirali smo dodatno i kod koji mu je suprotan, odnosno u kojem se susjedne permutacije razlikuju na svim mjestima.

Algoritme za generiranje definiranih Grayevih kodova implementirali smo u programskom jeziku Python.

Summary

The topic of this thesis are Gray codes: methods for generating combinatorial objects so that successive objects differ in some pre-specified way or, most often, minimally.

According to Frank Gray's patent, we defined Gray codes for binary arrays and for subsets of a set. We applied the same idea to other combinatorial objects: combinations, permutations, and partitions. In addition to Gray codes for permutations, we have defined a code that is opposite to it, i.e. in which adjacent permutations differ in all places.

We have implemented algorithms for generating Gray codes in the Python programming language.

Životopis

Rođena sam 16. kolovoza 1994. godine u Zagrebu. U Ivanić-Gradu pohađala sam Osnovnu školu Đure Deželića te opću gimnaziju u Srednjoj školi "Ivan Švear" gdje sam maturirala 2013. godine. Iste godine započela sam svoje fakultetsko obrazovanje na Prirodoslovno-matematičkom fakultetu Sveučilišta u Zagrebu, na preddiplomskom studiju Matematika; smjer: nastavnički. Na istom fakultetu 2018. godine upisala sam diplomski studij Matematika i informatika; smjer: nastavnički.