Zadatke, test primjere i rješenja pripremili: Alen Rakipović, Frane Kurtović, Ivan Mandura, Ivo Sluganović, Tomislav Gudlek, i Ante Đerek. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

JOKER

Predložio: Frane Kurtović
Potrebno znanje: naredba if, for petlja, razlaganje broja na znamenke, stringovi

Zadatak se rješava direktnom simulacijom postupka: najprije pronađeno Joker broj te nakon toga uspoređujemo znamenke sa svakim od tri zadana serijska broja.

Jedan jednostavni način implementacije je da i Joker i serijske brojeve pohranimo kao stringove. Joker broj računamo tako da pomoću for petlje prođemo kroz sve izvučene brojeve, odredimo zadnju znamenku (to je ostatak pri dijeljenju sa deset) te dodamo odgovarajući znak na kraj stringa koji čuva Joker broj.

Određivanje vrste dobitka također radimo pomoću jedne for petlje. Krenemo od zadnje, šeste, znamenke, uspoređujemo Joker i serijski broj te prekidamo for petlju kada nađemo prvu razliku. Vrstu dobitka određujemo prema tome koliko puta se izvršilo tijelo te for petlje.

Za one koji žele više

Napišite program koji za zadani serijski broj i vrstu dobitka računa broj različitih načina da se dobije taj dobitak. Da li je za svaki serijski broj jednaka vjerojatnost dobitka?

FLASH

Predložio: Ante Đerek
Potrebno znanje: polja, stringovi, for petlja

Primijetimo da zadatak možemo riješiti tako da razmatramo jedan po jedan blok, za svaki blok nađemo vrijeme potrebno da od početnog stanja dobijemo traženo te zbrojimo sva ta vremena. Naime, svaka operacija mijenja samo jedan blok pa je jasno da je ovo ispravan pristup.

Za pojedini blok postupak je sljedeći:

1. Ako postoji bit koji je na početku 0, a treba biti 1 onda nemamo izbora – moramo cijeli blok najprije pretvoriti u 1.
2. Za svaki bit koji je 1 a treba biti 0 pretvorimo ga u nulu.

Ovaj algoritam se može jednostavno implementirati tako da se vrijednost memorije čuva u stringu te se sa jednom for petljom prolazi kroz sve blokove, a drugom kroz pojedine bitove u bloku. Na temelju indeksa bloka i indeksa bita unutar bloka se lagano izračuna indeks u stringovima koji sadrže bitove.

PITON

Predložio: Ante Đerek
Potrebno znanje: polja, stringovi, rekurzija, stog (stack)

Za svaku pojedinu naredbu definiramo njen *kontekst* kao niz svih F naredbi u čijem je tijelu (bloku) ta naredba sadržana - i to poredanih od van prema unutra, odnosno onim redom kojim F naredbe dolaze u programu. Na primjer, u programu danom kao primjer u zadatku, kontekst naredbe `Pd` iz sedme linije čine naredbe `F3` i `F2` iz prve odnosno šeste linije programa. Za pojedini kontekst je jasno da se naredbe unutar njega izvršavaju onoliko puta koliko je umnožak pojedinih brojača iz konteksta. Dakle, spomenuta naredba `Pd` se izvršava šest puta.

```
F3
..Pa
..Pb
..F4
...Pc
..F2
.....Pd
.....Pe
..Pa
```

```
for (int i=0; i<3; i++) {
 printf("a");
 printf("b");
 for (int j=0; j<4; j++) {
   printf("c");
 }
 for (int j=0; j<2; j++) {
   printf("d");
   printf("e");
 }
 printf("a");
}
```

```
FOR i := 1 TO 3 DO BEGIN
 write('a');
 write('b');
 FOR j := 1 TO 4 DO BEGIN
   write('c');
 END;
 FOR j := 1 TO 2 DO BEGIN
   write('d');
   write('e');
 END;
 write('a');
END;
```

Okvirno, algoritam je sljedeći:

1. Čitamo naredbu po naredbu i u svakom trenutku održavamo trenutni kontekst
2. Za svaku P naredbu pribrojimo rješenju za odgovarajuće slovo umnožak brojača iz konteksta

Problem se sada svodi na održavanje trenutnog konteksta. U jezicima poput Pascala-a to je jednostavno – BEGIN nakon FOR dodaje novi element na kraj konteksta, a END miče zadnji element sa kraja. U Pitonu je to samo nešto teže te se kontekst održava prema tome koliko je naredba uvučena:

- Kontekst je niz parova (Indent, Mult) gdje je prvi element broj točaka ispred odgovarajuće F naredbe a Mult vrijednost njenog brojača
- Kada čitamo novu naredbu (bilo F bilo P), najprije možda moramo izbrisati neke elemente sa kraja konteksta jer su možda neke petlje zatvorene (ako se smanjio broj točaka na početku). Točnije nakon svake pročitane naredbe sa kraja konteksta skidamo sve parove čiji je Indent veći ili jednak od broja točaka trenutno pročitane naredbe.
- Ako je nova naredba tipa F, stavljamo novi par na kraj konteksta Indent je količina točaka, a Mult vrijednost brojača pročitane naredbe.
- Ako je nova naredba tipa P rješenju za odgovarajuće slovo pribrajamo umnožak svih brojeva Mult u kontekstu.

Ovaj algoritam možemo implementirati tako da kontekst pamtimo ili stogom parova ili pomoću dva obična polja.

Za one koji žele više

- Ova vrsta sintakse se naziva *pravilo zaleđa* (off-side rule) te je koriste mnogi programski jezici. Ovaj zadatak je nešto lakši od općenitog parsera jer su natjecatelji mogli pretpostaviti da je ulaz ispravan. Napravi program koji prepoznaje neispravne programe u Pitonu.
- Riješite zadatak rekurzivnim algoritmom.

KOCKE

Predložio: Tomislav Gudlek
Potrebno znanje: simulacija, polja

Trenutno stanje kocke možemo opisati sa 6 brojeva: broj naprijed, broj desno, broj gore, broj dolje, broj lijevo, broj iza. Naravno nisu svi oni potrebni jer se na temelju prva dva broja mogu odrediti svi ostali ali nam je ovako jednostavnije opisati rješenje.

Primijetite da sa ovako odabranim redoslijedom početno stanje se izražava šestorkom (1, 2, 3, 4, 5, 6). Svaki potez također opisujemo sa šest brojeva kojih kaže kako izgleda kocka u početnom stanju kada na nju primijenimo taj potez. Na primjer nakon poteza D kocka iz početnog stanja prelazi u (3, 2, 6, 1, 5, 4).

Kada zapišemo sve poteze kao konstante, algoritam je jednostavan, za svaki korak primijenimo odgovarajuće poteze na Mirkovu i Slavkovu kocku te usporedimo brojeve na vrhu.

Za one koji žele više

Riješite zadatak Kocka sa DMIH-a 2001.

KAMIONI

Predložio: Frane Kurtović
Potrebno znanje: dinamičko programiranje

Jedan od načina za riješiti zadatak je iterirati po parkirnim mjestima s lijeva na desno, te na svakom mjestu isprobati dvije mogućnosti, postaviti kamion ili ne postaviti ništa i samo prijeći na iduće parkirno mjesto. Primijetite da u trenutku odlučivanja između te dvije mogućnosti nije bitno kako su raspoređeni kamioni na parkirnim mjestima prije tog trenutka, već je jedino bitno koliko je kamiona iskorišteno do sada. To znači da je moguće definirati funkciju $cijena(n,k)$ koja vraća koliko je najmanje automobila potrebno maknuti s parkinga da se uspješno rasporedi k kamiona na zadnjih n parkirnih mjesta. Ta funkcija se računa isprobavajući dvije mogućnosti, postaviti ili ne postaviti kamion.

$$cijena(n,k) = min\{cijena(n+L, k-1) + koliko(n), cijena(n+1,k)\},$$

gdje $koliko(n)$ označava koliko je zauzetih parkirnih mjesta u unutar sljedećih L mjesta, tj. u intervalu $[P-n, P-n+L>$.

Broj stanja ove funkcije je O(PK), a prijelaz je konstante vremenske složenosti što znači da je ukupna vremenska i memorijska složenost O(PK).

Za one koji žele više

Riješite ovaj zadatak rekurzivno i iterativno (računajući stanja funkcije pravim redoslijedom). Iterativnim postupkom je moguće koristiti O(P) memorije jer se za računanje funkcije cijena(n, k) koriste samo vrijednosti funkcije za trenutni k i za k - 1, što je lako vidljivo iz formule.