Opisi algoritama

Zadatke, test primjere i rješenja pripremili: Ante Đerek, Ivan Katanić i Gustav Matula. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

Zadatak: Nijansa

Predložio: Ante Đerek
Potrebno znanje: naredba `if`, brojevi s pomičnim zarezom

Zadatak rješavamo direktnom implementacijom niza koraka opisanih u tekstu. Vrijednosti C_{max} i C_{min} svaku računamo pomoću tri `if` naredbe — za svaki od brojeva R, G, B provjerimo je li veći (odnosno manji) od druga dva. Alternativno, možemo koristiti funkcije `max` i `min` u C++-u ili Python-u. Pomoću još dvije `if` naredbe odlučujemo koju formulu za H treba primijeniti. Prilikom računanja vrijednosti H moramo paziti da upotrijebimo dijeljenje brojeva s pomičnim zarezom, primjerice tako da razliku $G - B$ spremimo u varijablu `float` odnosno `double` prije dijeljenja. S još jednom `if` naredbom provjeravamo je li H negativan te ga po potrebi uvećavamo.

Zadatak: Gitara

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

Tablaturu možemo pohraniti u polje od 6 stringova ili u dvodimenzionalno polje znakova. Također, nakon čitanja ulaza je potrebno odrediti duljinu pročitanih stringova primjenom odgovarajuće funkcije (primjerice `len` u Python-u). Sada pomoću jedne `for` petlje prolazimo kroz taktove odnosno stupce slijeva na desno, a pomoću druge `for` kroz žice na instrumentu odnosno retke odozgo prema dolje. Kada u stupcu i, retku j pronađemo znamenku onda:

1. Odredimo redni broj polja k
   (a) Znamenku pretvaramo u broj direktnom aritmetikom u jezicima C i C++ (`k=c-'0'`) odnosno oduzimanjem ASCII kodova u Python-u (`k=ord(c)-ord('0')`).
   (b) Ukoliko je sljedeći znak u retku također znamenka onda je također pretvaramo u broj, te pribrajamo vrijednosti k koju množimo s 10. Na mjesto pročitane znamenke pišemo znak "`-`" kako je ne bi pretvarali u ton kada obrađujemo sljedeći stupac.
2. Pozivamo funkciju koja za redni broj žice j i redni broj polja k vraća string koji sadrži oznaku tona.
   (a) Jedan način za efikasnu implementaciju je da se oznake tonova u oktavi te početni tonovi žice pohrane u konstantna polja.
   (b) Početni ton žice možemo pohraniti kao cijeli broj — broj polustupnjeva udaljenosti od `C1`. Sada je jednostavna artimetika potrebna kako bi se od j i k dobio redni broj tona i broj oktave.
   (c) Oznaku tona dobivamo od rednog broja tona čitanjem iz konstantnog polja. Redni broj oktave dobivamo pretvaranjem broja u string.
3. Ispisujemo oznaku tona.

Zadatak: Cjevovod

Predložio: Ante Đerek
Potrebno znanje: pohlepni algoritam, dinamičko programiranje

Nije teško vidjeti da pohlepna strategija vodi optimalnom rješenju. Instrukcije razmatramo jednu po jednu od prve prema zadnjoj i svaku pokušavamo izvršiti što ranije odnosno ispred nje stavljamo što je moguće manje `NOP` instrukcija kako bi dobili ispravan program.

Kako bi izračunali taj minimalan broj `NOP` instrukcija koje je potrebno dodati, dovoljno je za svaki registar k pamtiti vrijednost $L[k]$ — u kojoj se liniji trenutnog programa (nakon što dodamo `NOP` instrukcije) nalazi zadnja instrukcija kojoj je registar k cilj. Ako trenutni program već ima m linija i razmatramo sljedeću instrukciju A koja je oblika "*op* `ra rb`" onda se zadnja instrukcija o kojoj je instrukcija A ovisna nalazi u liniji $l = \max(L[a], L[b])$. Stoga je potrebno dodati $l + 5 - m$ `NOP` naredbi prije instrukcije A (naravno samo ako je $l + 5 - m$ veće od nule). Sada jednostavnim dinamičkim programiranjem računamo vrijednosti funkcije L te traženu minimalnu duljinu programa.

Zadatak: Voda

Predložio: Gustav Matula
Potrebno znanje: grafovi, pretraživanje u širinu, najkraći put, red

Za djelomične bodove bilo je dovoljno na sve moguće načine pretpostaviti retke koje će bomba odlediti, označiti ta polja odleđenim, te zatim izračunati najkraći put između zadanih polja. Najkraći put je u ovom slučaju najlakše izračunati *0/1 BFS* algoritmom. To je varijanta pretraživanja u širinu (*BFS*) u kojoj postoje bridovi težine nula. Za razliku od običnog pretraživanja u širinu, umjesto običnog reda koristimo red u kojem možemo dodavati element s obje strane: kada prolazimo po bridu težine nula element dodajemo na početak reda, a inače na kraj reda. Ukupna vremenska složenost ovog algoritma je onda $O(N^2K)$.

Za sve bodove primijetimo kako je najkraći put između polja koji prolazi kroz neki $N \times K$ pravokutnik jednak zbroju najkraćih puteva od dvaju polja do samog pravokutnika. Dakle ako znamo brzo izračunati najkraći put od jednog polja do nekog takvog pravokutnika, onda znamo riješiti i cijeli zadatak.

Ako se ledolomac nalazi unutar pravokutnika, najkraći put je jednak nuli. Inače je najkraći put do pravokutnika jednak najkraćem putu do nekog od polja susjednih pravokutniku. U općenitom slučaju, ako se pravokutnik prostire od retka i do retka $i + K - 1$, susjedna su mu sva polja retka $i - 1$ i retka $i + K$ (ako ti retci ne postoje samo ih ignoriramo). Problem smo dakle sveli na traženje najkraćeg puta od ledolomca do nekog zadanog retka matrice.

To možemo riješiti tako da pronađemo najkraći put od polja do svih ostalih polja (već spomenutim *0/1 BFS* algoritmom), i zatim za svaki redak izračunamo minimum. Nakon što sve to izračunamo za početno i završno polje, možemo proći po svim pravokutnicima i pronaći najbolji. Konačna složenost je $O(N^2)$.

Za one koji žele više: Riješite zadatak Led.

Zadatak: Ukulele

Predložili: Ante Đerek
Potrebno znanje: `for` petlja, stringovi

Tablaturu možemo pohraniti u polje od 4 stringa ili u dvodimenzionalno polje znakova. Također, nakon čitanja ulaza je potrebno odrediti duljinu pročitanih stringova primjenom odgovarajuće funkcije (primjerice `len` u Python-u). Sada pomoću jedne `for` petlje prolazimo kroz taktove odnosno stupce slijeva na desno, a pomoću druge `for` kroz žice na instrumentu odnosno retke odozgo prema dolje. Kada u stupcu i, retku j pronađemo znamenku `0` onda ispisujemo broj i.

Zadatak: Turnir

Predložio: Ivan Katanić
Potrebno znanje: `for` petlja, sortiranje, simulacija

Pretpostavimo najprije da unaprijed znamo poredak igrača po snazi, da jači igrač uvijek pobjeđuje slabijega te pokušajmo opisati način konstrukcije stabla turnira na temelju njegovog zadnjeg retka odnosno ždrijeba prvog kola. Stablo turnira možemo pohraniti u dvodimenzionalno polje veličine $n+1$ puta 2^n (veći dio tog polja će doduše biti prazan). Ako se u retku $n+1$ nalazi ždrijeb prvog kola, ostatak tablice konstruiramo tako da s jednom `for` petljom prolazimo kroz kola od prvog prema zadnjemu (odnosno kroz tablicu od zadnjeg retka prema prvom). U retku k pomoću druge `for` petlje prolazimo kroz sve parove i na kraj prethodnog reda stavljamo oznaku jačeg igrača.

Naravno, u zadatku nije zadan poredak igrača po snazi već stablo turnira s pomiješanim oznakama, međutim možemo probati iskoristiti ulaz kako bi dobili istu informaciju. Za svakog igrača pobrojimo koliko se ukupno puta njegova oznaka pojavljuje u stablu turnira: jasno je da se oznaka pobjednika pojavljuje najčešće, zatim slijedi oznaka igrača koji je izgubio u finalu, zatim slijede oznake dvaju igrača koji su izgubili u polufinalu itd. Dakle možemo za svakog igrača izračunati ukupan broj pojavljivanja njegove oznake u ulazu i to koristiti kao njegovu snagu u prethodno opisanom algoritmu.

Za one koji žele više: Neki igračima će se oznake pojavljivati jednak broj puta, zašto to nije problem u našem algoritmu?

Zadatak: Doseg

Predložio: Ante Đerek
Potrebno znanje: stablo, stog, simulacija

Zadatak možemo riješiti na nekoliko različitih načina. Ovdje opisujemo pristup koji nije najefikasniji, ali je zato jednostavan. Definiramo *nivo naredbe* kao broj blokova u kojima je ta naredba ugniježđena. Blok s kojim počinje program sadrži naredbe nivoa 1, blokovi koje on sadrži sadrže naredbe nivoa 2, itd. Primijetite da je vrlo jednostavno izračunati svaki nivo svake naredbe: razmatramo naredbe od početka programa i brojimo `begin` i `end` naredbe. Nivo određene naredbe je razlika između broja do sada viđenih `begin` i `end` naredbi.

Kada razmatramo neku naredbu, reći ćemo da su *aktivni* svi oni blokovi u kojima je ta naredba sadržana. Primijetite da u svakom trenutku aktivni blokovi čine niz – aktivan je jedan blok na nivou 1, jedan blok na nivou 2, i tako dalje do bloka u čijem se tijelu nalazi naredba. Važeća deklaracija varijable se uvijek nalazi u nekom od aktivnih blokova i to u onom na najvećem nivou.

Vrijednosti deklariranih naredbi čuvamo u dvodimenzionalnom polju veličine 1000 puta 26 — $F[l, c]$ je posljednja deklarirana vrijednost varijable c u aktivnom bloku na nivou l (odnosno -1 ako varijabla c nije još definirana u niti jednom aktivnom bloku na nivou l). Naš algoritam održava vrijednosti ovog polja te računa izlaz svake `write` naredbe na sljedeći način:

1. Za naredbu `begin` postavljamo na -1 sve vrijednosti $F[l, c]$ gdje je l nivo bloka koji upravo počinje.
2. Za naredbu `end` postavljamo na -1 sve vrijednosti $F[l, c]$ gdje je l nivo bloka koji je upravo završio.
3. Za naredbu `var` c `=` k postavljamo $F[l, c]$ na vrijednost k gdje je l nivo naredbe.
4. Za naredbu `write` c ispisujemo $F[l, c]$ gdje je l najveći nivo manji ili jednak nivou naredbe za koji ova vrijednost nije -1.

Za one koji žele više: Pronađite rješenje koje radi u vremenu proporcionalnom s brojem naredbi n.

Zadatak: Led

Predložio: Gustav Matula
Potrebno znanje: graf, pretraživanje u širinu, najkraći put, red, stog

Preporučamo na najprije pročitate opis algoritma za zadatak Voda.

Za prvi podzadatak bilo je dovoljno za svaku moguću poziciju kvadrata izračunati najkraći put između dva polja, što je najjednostavnije napraviti *0/1 BFS*-om (modifikacija *BFS*-a u kojoj čvorove težine nula dodajemo na početak, a čvorove težine jedan na kraj reda).

Za drugi podzadatak možemo primijetiti da ako fiksiramo neki kvadrat, najkraći put između dva polja koji prolazi kroz njega jednak je zbroju najkraćih puteva od kvadrata do tih polja.

Kako efikasno izračunati najkraći put od zadanog polja na kojem se recimo nalazi ledolomac do nekog kvadrata? Ako kvadrat sadrži ledolomca duljina je nula. Inače, najkraći put do kvadrata jednak je najkraćem putu do jednog od $4K$ polja susjednih kvadratu. Dakle ako znamo udaljenosti do tih polja, dovoljno je izračunati njihov minimum u $O(K)$. Same udaljenosti dobijemo tako da prije svega izračunamo najkraće puteve od ledolomca do svih drugih polja (u složenosti $O(N^2)$). Kako imamo $O(N^2)$ kvadrata, ukupna složenost je $O(N^2K)$.

Za sve bodove preostaje ubrzati gornji pristup. U općenitom slučaju, polja susjedna kvadratu sastoje se od četiri trake duljine K (ako neka od tih traka izlazi iz matrice jednostavno je ignoriramo). Dakle dovoljno je izračunati najkraći put od ledolomca do svake horizontalne i vertikalne trake duljine K. Minimume po svim takvim trakama nekog retka/stupca možemo izračunati u $O(N)$ koristeći *monotoni red*, strukturu koja se ponaša kao red brojeva i može vrlo efikasno pronaći minimum brojeva koje sadrži. Algoritam se svodi na to da prolazimo kroz redak (ili stupac) održavajući *prozor* od zadnjih K elemenata u monotonom redu, te za svaki od tih prozora pitamo strukturu za minimum. Ako to napravimo za sve retke i stupce (i za svakog ledolomca) lako ćemo pronaći minimume po svim trakama duljine K, a iz toga i najkraće puteve do svih kvadrata. Ukupna složenost ovog pristupa je $O(N^2)$.

Monotoni red možemo efikasno implementirati pomoću dva stoga, ili možemo iskoristi činjenicu da se susjedni brojevi u trakama razlikuju za najviše jedan i implementirati monotoni red tako da održavamo histogram brojeva u prozoru. Detalje efikasne implementacije monotonog reda ostavljamo čitatelju