Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Fabijan Bošnjak, Nikola Dmitrović, Karlo Franić, Marin Kišić, Josip Klepec, Daniel Paleka, Ivan Paljak i Paula Vidas. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima.

Zadatak: FPS

Pripremio: Karlo Franić
Potrebno znanje: naredba učitavanja i ispisivanja

Za rješenje ovog zadatka potrebno je pretvoriti minute u sekunde ($X \cdot 60$) te dobiveni broj sekundi pomnožiti sa brojem FPS-a (sličica u sekundi).

Programski kod (pisan u `Python 3`):

```
X = int(input())
Y = int(input())
print(X * 60 * Y)
```

Zadatak: Amazon

Pripremili: Fabijan Bošnjak i Nikola Dmitrović
Potrebno znanje: naredba ponavljanja, naredba odlučivanja, pohlepni algoritam

Riješimo najprije prvi podzadatak u vrijednosti od 10 bodova u kojem vrijedi da je broj paketa u skladištu jednak 3. U tom slučaju jedino je potrebno znanje naredbe `if` kojom provjeravamo za svaku kombinaciju nošenja paketa vrijedi li i prema tome zaključujemo koliko puta se dron mora vratiti u skladište.

Programski kod tog dijela zadatka (pisan u `Python 3`):

```
if K[0] + K[1] + K[2] <= N:
  print(1)
elif K[0] + K[1] <= N or K[1] + K[2] <= N:
  print(2)
else:
  print(3)
```

Nadalje, za idućih 10 bodova vrijedilo je da su težine svih paketa jednake, odnosno postoji M paketa težine K u skladištu. Ovdje je bitno primijetiti kako će dron u svakoj dostavi uzeti jednak broj paketa. Naime, ako dron može uzeti paket s oznakom X u nizu u svoju dostavu, ali ne uzme, to znači da će morati uzeti paket X u sljedeću dostavu. Da smo uzeli paket X u trenutnu dostavu u koju je mogao ići, u sljedećoj dostavi nakon što bi dron uzeo paket $X + 1$ bi težina bila K, a budući da ga nismo uzeli težina sljedeće dostave nakon što dron uzme paket $X + 1$ je $2K$. Dakle, optimalno je uzeti maksimalan broj paketa u trenutnoj dostavi jer se ne isplati prenositi dalje.

Broj paketa po dostavi koje dron može uzeti je jednak `N // K`, gdje `//` predstavlja cjelobrojno dijeljenje. Broj polijetanja drona je onda jednak `ceil(M / (broj paketa koje dron može uzeti u jednoj dostavi))`, gdje `ceil` predstavlja naredbu zaokruživanja na više.

Rješenje za preostalih 10 bodova je ujedno i rješenje koje rješava sve dosadašnje podzadatke. Ključna je primjedba koja je već dokazana u gornjem dijelu teksta, a to je da se uvijek isplati uzeti paket u trenutnu dostavu ako ga možemo uzeti. Dakle, trebamo samo prolaziti nizom pomoću `for`-petlje i zbrajati težinu trenutne dostave, a kada zbroj pređe nosivost N, dodati jedan na rješenje i resetirati zbrajanje trenutne dostave na težinu prvog paketa u toj dostavi. Na kraju, ako je težina trenutne dostave veća od 0, to znači da se u trenutnoj dostavi nalaze neki paketi pa je potrebno dodatno povećati brojač za jedan.

Zadatak: Holding

Pripremili: Fabijan Bošnjak i Marin Kišić
Potrebno znanje: dinamičko programiranje, memorijske optimizacije

Rješenje prvog podzadatka je dinamika u kojoj je stanje bitmaska. Razradu tog rješenja prepuštamo čitateljici za vježbu.

Za drugi podzadatak je poznato da $R = N$, što znači da ćemo brojeve na pozicijama unutar intervala L, $L+1, \ldots, R$ moći mijenjati samo s brojevima na pozicijama od 1 do $L-1$ (u ostatku rješenja kada se spomene samo interval se podrazumijeva da je interval $L, L+1, \ldots, R$). Prva bitna primjedba jest da nikada nećemo nekom broju mijenjati poziciju više od jednom. Druga važna primjedba, i puno manje očita od prethodne, jest da je jedino bitno koje smo elemente izabrali za mijenjanje unutar intervala te koje izvan, a da će neovisno o tome koje elemente međusobno mijenjamo zbroj potrošenog novca iz Ivičinog džepa ostati isti. U prijevodu, ako su pozicije brojeva unutar zadanog intervala koje smo odlučili mijenjati i, j; a pozicije brojeva izvan intervala koje smo odlučili mijenjati l, k; posve je svejedno hoće li se zamijeniti i,k te j,l ili i,l te j,k. Formalni dokaz te tvrdnje ostavljamo čitateljici za vježbu.

Sada je očigledno jedino bitno odrediti koje elemente biramo unutar intervala te koje izvan i bitno je da ih je jednak broj. To možemo ostvariti dinamikom dp kojoj su argumenti trenutna pozicija izvan intervala, pozicija unutar intervala i koliko novaca smo potrošili, a pamti koliko je maksimalno moguće smanjiti sumu unutar intervala. Početno stanje dp je $dp(1, L, 0)$, a stanje u kojem je rješenje je $dp(L-1, R, K)$.

$$dp(poz_{out}, poz_{in}, spent) = \max\Big\{dp(poz_{out}-1, poz_{in}, spent), dp(poz_{out}, poz_{in}-1, spent),$$
$$dp(poz_{out}-1, poz_{in}-1, spent-(poz_{in}-poz_{out})) + A[poz_{in}] - A[poz_{out}]\Big\} \quad (1)$$

Prvi prijelaz dinamike govori da ne uzimamo element na poziciji poz_{out}, drugi prijelaz govori da ne uzimamo element na poziciji poz_{in}, dok treći govori da uzimamo oba elementa te ih zamjenjujemo pa zato trošimo $poz_{in} - poz_{out}$ novaca.

Složenost ovog rješenja je $\mathcal{O}(N^2 \cdot K)$.

Taj algoritam je dovoljno brz i za potpuno rješenje, ali ne obuhvaća zamjene s desne strane intervala jer je $R = N$. Moguće je primjetiti da će se uvijek kao konačno rješenje uzeti X elemenata s lijeve strane intervala i Y s desne, te $X + Y$ elemenata unutar intervala. Očigledno je da ćemo, ako sortiramo pozicije brojeva unutar intervala koje smo odabrali za zamjenu, prvih X elemenata zamijeniti s odabranih X s lijeve strane intervala te idućih Y s odabranih Y s desne strane. To nam daje naslutiti da postoji linija između pozicija unutar intervala koja određuje da ćemo s lijeve strane te linije sve odabrane brojeve zamijeniti s odabranim brojevima lijevo od intervala, i obratno za desnu stranu linije. Zato što mi ne znamo gdje se ta linija nalazi i zato što nama nije važno koliko se zamjena obavi sa svake strane intervala, već samo potrošen novac, možemo iskušati gdje se nalazi ta linija za sve pozicije unutar intervala. To možemo sljedećim linijama koda:

```
for i in range (L - 1, R+1):
  for j in range (0, K + 1):
    rj = max(rj, dpL(L - 1, i, j) + dpR(R + 1, i + 1, K - j))
```

Što su dpL i dpR? dpL je ona ista dinamika iz prošlog podzadatka, a dpR je posve identična dpL samo što se odvija sa suprotne strane. Još je bitno napomenuti da je bitno da se po pozicijama unutar intervala kod dpL prelazi od L prema R, a kod dpR od R prema L iz očitih razloga. Složenost dinamike jest $\mathcal{O}(N^2 \cdot K)$, a spajanja dvaju dinamika $\mathcal{O}(N \cdot K)$, dakle složenost ovog rješenja je $\mathcal{O}(N^2 \cdot K)$.

Zašto onda to rješenje ne donosi sve bodove? Jer trodimenzionalno polje oblika `int dp[N][N][K]` zauzima previše memorije za $N = 100$, ali dovoljno za $N = 50$. Dakle još je samo potrebno optimizirati memorijsku

složenost. Naime, ovdje ima više pristupa kako to učiniti, ali vjerojatno najlakši je primjedbom da će u najgorem slučaju za bilo koji N, L i R, maksimalna količina novca potrebna Ivici da izvrši sve moguće promjene biti $\frac{N^2}{4}$. Ako implemetiramo polje `int dp[N][N][N*N/4]` to ne prelazi memorijsko ograničenje od `256 MiB`. Postoji druga optimizacija koja zamjenjuje dimenziju N s malom konstantom, ali za implementacijske detalje te optimizacije proučite službeno rješenje.

Zadatak: Nivelle

Pripremili: Daniel Paleka i Paula Vidas
Potrebno znanje: metoda kliznog prozora, metoda dva pokazivača

Rješenje vremenske složenosti $\mathcal{O}(N^2)$ za svaki podstring računa broj različitih slova. Ako koristimo tzv. *metodu kliznog prozora*, potrebno je brojati koliko puta se svako slovo pojavljuje, te primijetiti svaki put kad se neko slovo počne ili prestane pojavljivati. Za detalje implementacije pogledajte sporije službeno rješenje.

Primijetimo da brojnik izraza koji želimo minimizirati, tj. broj različitih znakova u podstringu, može poprimiti samo vrijednosti 1, 2, ..., 26. Stoga, dovoljno je za fiksnu vrijednost brojnika odrediti najduži podniz koji ima točno taj broj različitih znakova, te usporediti dobivenih 26 razlomaka.

Jednostavna implementacija za svaki mogući početak podstringa računa najduži podstring koji sadrži točno K različitih slova. Ako prethodno za svako slovo i za svaku poziciju izračunamo prvo sljedeće pojavljivanje tog slova, za svaki početak možemo brzo isprobati ≤ 26 stringova, koji idu "*do prvog novog slova*".