Univerzitetni programerski maraton

FINALE 2022 – rešitve nalog

Tomaž Hočevar

</upm UNIVERZITETNI PROGRAMERSKI MARATON

Kriptozoologija

Poišči Bigfoot-ovo pot in preštej iztrebke na njej

- najdi eno točko na poti … +
- poišči preostanek poti
  - povezana komponenta
  - sledi poti v največ 2 od 4 smeri
- pazi:
  - Python ima recursion limit 1000

Kriptozoologija

```
from itertools import product

n,m = map(int, input().split())
g = [input() for i in range(n)]

dirs = [(0,1),(0,-1),(1,0),(-1,0)]

def on(i,j):
    return 0<=i<n and 0<=j<m

def track(pi, pj, i, j):
    k = 0
    while True:
        if g[i][j] == '#': k+=1
        found = False
        for di, dj in dirs:
            ni, nj = i+di, j+dj
            if on(ni,nj) and (ni,nj)!=(pi,pj) and g[ni][nj]!='.':
                found = True
                break
        if not found: break
        else: i, j, pi, pj = ni, nj, i, j
    return k

for i,j in product(range(n), range(m)):
    if g[i][j]=='+': break

k = 0
for di,dj in dirs:
    ni, nj = i+di, j+dj
    if on(ni, nj) and g[ni][nj]!='.':
        k += track(i,j,ni,nj)
print(k)
```

</upm UNIVERZITETNI PROGRAMERSKI MARATON

Obojestranska poravnava

Razvrsti besede v vrstice in izračunaj širino presledka, ki vodi do obojestranske poravnave.

- pretvori tabelo širin znakov v primerno obliko
- implementiraj opisan postopek z najmanjšim presledkom (4 pt)
  - ali gre nova beseda še v vrstico?
  - širina vrstice, število besed, zadnja beseda
  - širina vrstica + (presledek) + beseda <= širina lista
- ostanek prostora v vsaki vrstici razdeli med presledke
  - posebni primeri: ena beseda, zadnja vrstica (presledek max 6 pt)
- pazi
  - natančnost!

Obojestranska poravnava

```
words = []
for row in sys.stdin:
    words += row.strip().split()

i = 0
width = 17/0.0352777778 - 1e-4
line, num, last = 0, 0, None
output = []
for word in words:
    gap = (0 if num==0 else 4)
    length = sum(widths[c] for c in word) + (len(word)-1)*1
    if line + gap + length > width:
        output.append((last, "/" if num == 1 else (4 + (width - line) / (num - 1))))
        line, num = length, 1
    else:
        line, num = line + gap + length, num + 1
    last = word
output.append((last, "/" if num == 1 else min(6, (4 + (width - line) / (num - 1)))))
```

</upm UNIVERZITETNI PROGRAMERSKI MARATON

Plus, minus, krat, deljeno

Poišči izraz, ki ga sestaviš iz podanih števil in operatorjev, katerega vrednost se čim bolj približa ciljnemu številu.

- majhen nabor števil in operatorjev
  - 6! permutacij števil, 4^5 operatorjev ... < 10^6 kombinacij
  - brute-force, izbor jezika: Python?

```
5   6   13   7   2   5
  *   +    -   *   /
```

- izračun vrednosti izraza
  - strnjene dele izraza sestavljenih iz operacij + in - zamenjamo z izračunanimi vrednostmi
  - izračunamo rezultat operacij * in /
  - pozor: deljenje z 0

</upm UNIVERZITETNI PROGRAMERSKI MARATON

Šopek marjetic

Izračunaj GCD spreminjajoče se množice števil S.

$2^3\,3^1\,7^2 \quad 2^4\,3^5\,5^2 \quad 2^3\,3^2\,5^2 \quad 2^2\,3^2\,5^1 \quad \rightarrow \quad 2^2\,3^1$

a) praštevilska faktorizacija
   - faktorizacija števil, GCD … min stopnja po skupnih prafaktorjih
   - omejeno število različnih prafaktorjev (< 10 n)
   - majhne stopnje prafaktorjev (<= 23)
   - a[p][d] … količina števil s praštevilom p stopnje vsaj d
   - GCD … praštevila p: a[p][1] = |S|
   - b[u] … seznam prašt. p, ki se pojavijo pri natanko u številih (a[p][1] = u)
   - spremembe: obdelamo vsak prafaktor spremenjenega števila

Šopek marjetic

14, 35, 14 | 12, 18, 9 | 1, 21, 7 | 60, 10

b) korenska dekompozicija
   - seznam števil razdeljen na skupine velikost 400 (√n)
   - GCD posamezne skupine, GCD skupin
   - dodajanje: dodaš število v zadnjo prosto skupino
   - brisanje: poiščeš element in ga zamenjaš z zadnjim v seznamu
   - vsakič popraviš GCD max 2 prizadetih skupin

c) drevesna struktura

Šopek marjetic

```
scanf("%s %d",s,&x);
if (s[0]=='+') {
        int i=a.size(), j=i/K;
        pos[x].PB(i);
        a.PB(x);
        b[j]=gcd(b[j],x);
} else {
        int i=pos[x].back(), j=i/K;
        pos[x].pop_back();
        a[i]=0;
        b[j]=0;
        for (int i=j*K;i<(j+1)*K && i<a.size();i++) {
                b[j]=gcd(b[j],a[i]);
        }
}
int g=0;
for (int j=0;j*K<a.size();j++) {
        g=gcd(g,b[j]);
}
if (g==0) g=1;
printf("%d\n",g);
```

```
#define N 100000
#define K 400
int b[N/K+1];

vector<int> a;
map<int,vector<int>> pos;
```

</upm UNIVERZITETNI PROGRAMERSKI MARATON

Poravnava nizov

Optimalno poravnaj tri nize z vrivanjem zvezdic.

- dinamično programiranje
  - f(i, j, k) = optimalna poravnava predpon $s_1[:i]$, $s_2[:j]$, $s_3[:k]$
    - rezultat = par (vrednost, dolžina)
  - obravnava 7 podmnožic črk v zadnjem stolpcu
  - rekonstrukcija (shranimo optimalno podmnožico)
  - n = 2 … $s_3 = s_2$, f / 2
- pozor
  - minimiziranje dolžine ne minimizira ocene
  - izbira črke za stolpec in požrešno dodeljevanje vseh črk stolpcu ni pravilno

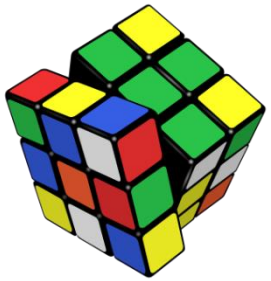
```
požrešno:      boljše:
*ba            *ba
a**            **a
aba            aba
```

</upm UNIVERZITETNI PROGRAMERSKI MARATON

Rubikova kocka

Koliko ponovitev podanih zasukov vrne Rubikovo kocko v prvotno stanje?

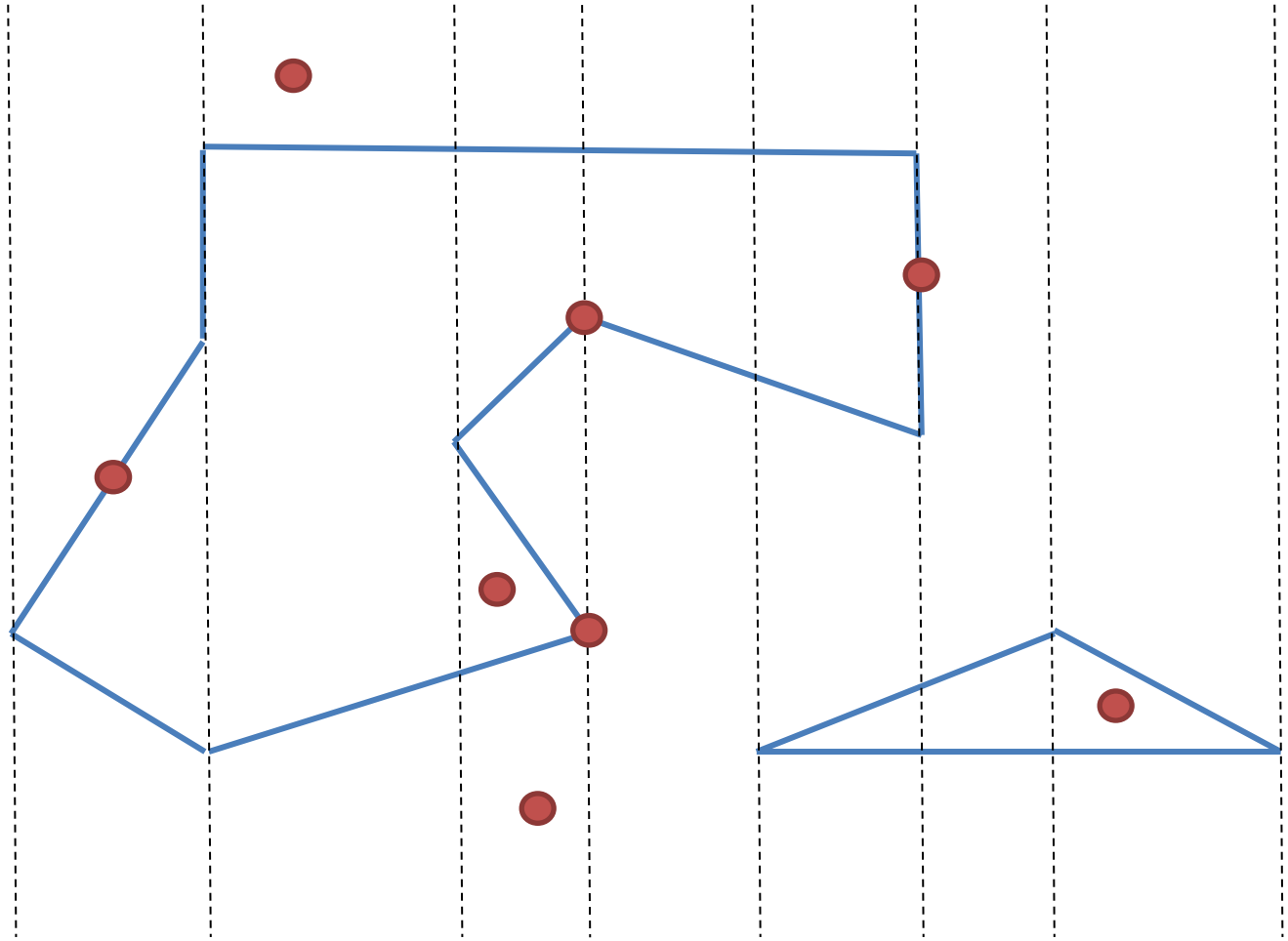
- zaporedje zasukov = permutacija 54 kvadratkov (ploskev)
- izračun nove lokacije vsakega kvadratka
  - koordinatno izhodišče v centru kocke
  - zaporedne rotacije centra kvadratka
- število ponovitev?
  - red permutacije = LCM dolžin ciklov permutacije
  - največ 1260 [1] … lahko enostavno ponavljamo



[1] https://en.wikipedia.org/wiki/Rubik's_Cube_group

Ograde

Za vsak večkotnik preštej točke, ki se nahajajo v njem.



</upm UNIVERZITETNI PROGRAMERSKI MARATON

Ograde

- kvadratna rešitev je prepočasna
- prelet ravnine s premico (sweep line)
- večkotnik -> seznam orientiranih daljic (od leve proti desni)
- aktivne daljice hranimo urejene po y
  - primerjava po y na bolj desnem levem robu daljic
  - liho/sodo št. daljic and točko?
  - drevesna struktura
- pazi:
  - vertikalne daljice (urejen seznam za vsak x)
  - točke na stranicah
  - točke v oglišči (množica oglišč)

Hammingova razdalja

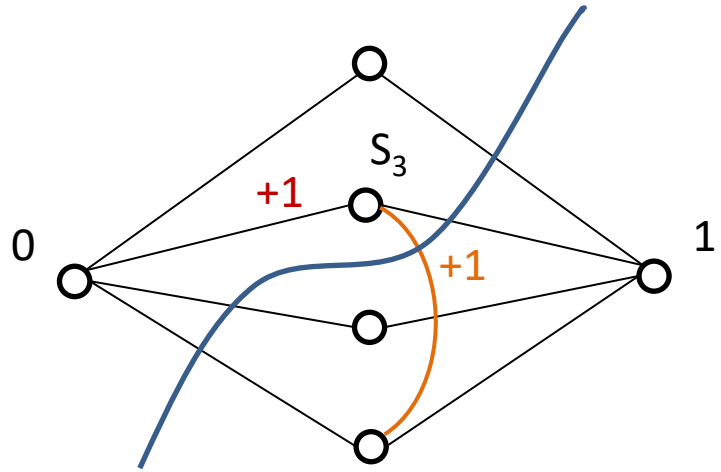
Zamenjaj vprašaje z 0 in 1, da bo vsota Hammingovih razdalj po vseh poravnavah čim manjša.

```
S=?01??        ?01??   ?01??   ?01??
P=1?0          1?0      1?0      1?0
```

- recimo, da v S ni vprašajev
  - za vsak P_i izberemo bolj pogost znak 0/1
- na poravnave med znaki, ki niso '?', ne moremo vplivati
- požrešna strategija?

Hammingova razdalja

- minimalni prerez (izvor, ponor, znaki '?' v S ali P)




```
S=?01??
P= 1?0
```

```
S=?01??
P=  1?0
```

- učinkovit algoritem
  - Ford-Fulkerson (capacity scaling)
  - Dinitz