-Popis myšlienky:

Algoritmus si odzadu vypočíta aký dlhý rad sa dá vytvoriť od pozície každého čísla v liste pole, uloží si výskyt jednotlivých síl do slovníka aby sa jednoduchšie prehľadávali a postupne pridáva Adama za každé číslo a nastavuje jeho silu na silu čísla pred ním+1 a skúša spojiť 2 postupnosti. Aby nemusel počítať(od začiatku) aká najdlhšia postupnosť sa dá spraviť do pozície čísla pred Adamom tak si tieto hodnoty ukladá do druhého pola.

-Popis dátových štruktúr:

Inty n(počet ľudí), fma(max sila), ma(max dĺžka postupnosti=výsledok), a2(pozicia cloveka s silou o 2 vacsiou ako sila aktualneho cloveka), slovníky di(slovnik s poslednym vyskytom cisla ktoré je kľúč),d(kľúč je sila a obsahuje pole pozícií ľudí kt. majú túto silu), polia pole(obsahuje vstup=sila i-teho človeka), po(pole s max dĺžkou postupnosti po každú hodnotu), po_otocene(pole s max dĺžkou postupnosti od každej hodnoty po koniec)

-Popis algoritmu:

Algoritmus načíta vstup, vytvorí polia po a po_otocene o dĺžke n, vytvorí slovník di. Algoritmus si bude pamätať posledné výskyty síl a odzadu doplní pole po_otocene s hodnotami max dĺžky postupnosti po koniec(riadok 7-12). Resetuje slovník di a vytvorí slovník d a int ma. Do slovníka d pod kľúč kt. je sila zapíše pole s pozíciami ľudí s týmito silami (riadok 17-21). V hlavnom cykle bude skúšať dať Adama na každú pozíciu, vypočítať jedno políčko pola po (24-28) a určiť dĺžku postupnosti s Adamom. Teraz algoritmus overí či teoreticky existuje 2. postupnosť kt. môže Adam spojiť(29). Ak nie tak ma nastaví na max z ma a postupnosti+1(Adam)(42,43). Ak áno(29,41) tak tak skúsi či je index začiatku 2. postupnosti za koncom 1. a ak áno tak napíše dĺžku týchto postupností spojených Adamom(40,41). Ak takýto index nie je tak vypíše 1. postupnosť+1(Adam)(38,39). Aby nemusel iterovať cez postupnosti kt. začínajú pred aktuálnou pozíciou zas a znova tak ich vymazáva(35,37). Na konci keď vyskúšal všetky možnosti vypíše ma(max dĺžka)(44).

-Zdôvodnenie správnosti:

Algoritmus vždy nájde správnu odpoveď preto že skúša všetky možnosti a nič nevynecháva.

-Odhad časovej zložitosti:

Pretože všetky funkcie v cykloch okrem pop(0) majú konštantnú zložitosť a funkcia pop(0) sa udeje n krát s lineárnou časovou zložitosťou závislou od dĺžku pola v slovníku d ,tak je časová najhoršia zložitosť kvadratická a závislá od n a jednotlivých počtov ľudí s rovnakou silou.

-Odhad pamäťovej zložitosti:

Pamäťová zložitosť je lineárna a závislá od n.

Kód:

```
n,fma=map(int,input().split(" ")) #nacitanie vstupu
pole=list(map(int,input().split(" "))) #nacianie vstupu z silami
po = [0 for i in range(n)] #pole v ktorom budu hodnoty max dlzky postupnosti po tuto
hodnotu = dlzka postupnosti potialto
po_otocene=[0 for i in range(n)] #pole v ktorom budu hodnoty dlzky postupnosti po
koniec=od tialto sa da spravit postupnost takejto dlzky
di ={} #slovnik s poslednym vyskytom cisla

for i in range(n-1,-1,-1): #vytvaranie pola s dlzkami postupnosti od i po koniec,
algoritmus postupuje od zadu
    if pole[i] + 1 in di: #ked sa nachadza vacsie cislo v slovniku
```

```
        po_otocene[i] = 1 + po_otocene[di[pole[i] + 1]]#pridanie hodnoty o 1 vacsej ako
hodnota cisla o 1 vacsieho
    else:
        po_otocene[i] = 1 #koniec postupnosti, uz nieje o 1 vacsie cislo, ! ideme od zadu
!
    di[pole[i]] = i  #nastavenie posledneho vyskytu cisla pole[i] na i

di={} #resetovanie slovniku
d={}  #vytvorenie nového slovniku kde kluc=cislo v poli-sila (pole) a pod nim je list
indexov tohto cisla=pozicie tochto cisla, potom nebude musiet algoritmus hladat v celom
poli
ma=0 #premenna s max dlzkov radu
for i in range(n): #vytvorenie slovnika s poziciami
    if pole[i] in d: #ak uz bol clovek s rovnakou silou
        d[pole[i]].append(i) #prida sa jeho index do pola pod kluc=sile
    else: #ked je tento clovek prvy s silou pole[i]
        d[pole[i]] = [i] #nastavi sa jeho index pod kluc jeho hodnoty

for i in range(n):#hlavny cyklus skusa pridavat adama na kazdu poziciu, i= pozicia adama,
popri tom tvori pole po s dlzkami postupnosti po cislo na indexe i (pole[i])
    if pole[i] - 1 in di: #ked toto nie je zaciatok postupnosti
        po[i] = 1 + po[di[pole[i] - 1]] #nastavi sa dlzka postupnosti na 1+dlzka
postupnosti po pole[i]-1
    else:#ked nie je v slovniku element o 1 mensi=postupnost zacina odtialto
        po[i] = 1 #nastavi sa dlzka postupnosti na 1
    di[pole[i]]=i #do slovnika sa da posledny vyskyt hodnoty pole[i]
    if pole[i]+2 in d:#ked existuje hodnota kt. je o 2 vacsia teda adam spoji 2
postupnosti
        a2=-1 #pozicia cloveka s silou o 2 vacsiou ako pole[i] kde pole[i]+1 je adam,
nastavena na -1 kedby takyto clovek za poziciou i nebol
        for index in d[pole[i]+2]:#v slovniku najde index cloveka s touto silou ktory je
za aktualnou poziciou i
            if index>=i:
                a2=index #nastavenie a2 na tento index
                break #loop skonci, index sa uz nasiel
        while len(d[pole[i]+2])>0 and d[pole[i]+2][0]<=i:#ked je este v slovniku pod
rovnakym klucom nejaka hodnota
            #ked je tato hodnota(index) mensia ako i tak ju vymazeme lebo ju uz nebude
treba a zbytocne by sa cez nu iterovalo
            d[pole[i]+2].pop(0)#vymazanie prvej hodnoty z pola v slovniku
        if a2==-1:#clovek s silou pole[i]+2 za poziciou i nie je
            ma=max(ma,po[i]+1)#ma sa nastavi na max z ma a dlzku postupnosti po
pole[i]+1(jedna je adam)
        else:#je tu dalsi clovek s silou pole[i]+2 a ten moze byt sucastou dalsej
postupnosti
            ma=max(ma,po[i]+1+po_otocene[a2])# ma sa nastavi na max z ma a dlzku
postupnosti po pole[i]+1(adam)+dlzku postupnosti od pole[1]+2
    else: #ked nie
        ma=max(ma,po[i]+1) #ma sa nastavi na max z ma a dlzku postupnosti po
pole[i]+1(jedna je adam)
print(ma) #vypise sa dlzka najdlhsej postupnosti
```