Druhé domáce kolo 16. sezóny súťaže Palma sa konalo dňa `24.04.2018` v čase `15:00-18:00`. Zúčastnilo sa ho 10 súťažných tímov (celkovo 18 súťažiacich), ktorí riešili 4 úlohy - dve z nich boli rozdelené na ľahšiu a ťažšiu verziu.

1 Pozemok

Táto úloha mala priamočiare riešenie - otestovať všetky možné umiestnenia, teda ľavý horný roh umiestniť na políčko pozemku a skontrolovať či všetky zastavané políčka domu sa nachádzajú na voľnom mieste. Zložitosť takéhoto algoritmu je $O(R.S.RP.SP)$.
Častou chybou nesprávnych riešení bolo, že kontrolovali presnú zhodu znakov a nie len či v prípade zastavaného políčka domu je na pozemku voľné miesto. Napríklad pre nižšie uvedený dom a pozemok je správna odpoveď 4.

dom
```
3 4
.xxx
xx.x
xxxx
```

pozemok
```
4 5
.xxxx
xxxxx
xxxxx
xxxxx
```

2 Ťažisko

Táto úloha je vo všeobecnosti NP-ťažká, t.j. nemáme na jej riešenie polynomiálny algoritmus. Pre dané obmedzenia vstupu stačí vyskúšať vhodné slová a vybrať z nich minimálnu hodnotu. Ako vhodné slová sú všetky také, ktoré vieme povyberať po znakoch z niektorého slova - prvé písmeno z jedného, druhé písmeno z nejakého slova (nie nutne rôzneho od prvého), ... Pre zlepšenie času výpočtu $O(N^{M+1}.M)$ je vhodné každé takéto slovo kontrolovať iba raz (aj keď je možné, že to isté slovo vieme dostať rôznymi spôsobmi výberu znakov). Napríklad vytvorením si množín vyskytujúcich znakov pre každú pozíciu zvlášť.
Greedy algoritmus tu nefunguje, aj keď z príkladov by sa mohlo zdať, že stačí vybrať na každej pozícii najpočetnejšie písmeno. Napríklad pre slová `PALMA`, `SUTAZ`, a `PALMA` vychádza podľa početnosti slovo `PALMA` (s maximálnou vzdialenosťou 5), ale lepším riešením je slovo `PULAZ` (s maximálnou vzdialenosťou iba 3).

3 Parkovanie

Po prečítaní zadania úlohy Parkovanie skúsenejší programátor hneď vidí, že ide o grafovú úlohu - nájdenie najkratšej cesty z jedného miesta do všetkých ostatných, teda Dijkstrov algoritmus. Pracnejšie je spracovanie vstupu a testovanie pri otáčaní. Vrchol grafu reprezentuje polohu prednej časti auta a jeho otočenie. Na riešenie ľahšej úlohy stačí daný algoritmus implementovať v čase $O(n^2) = O((4.R.S)^2)$. Na riešenie ťažšej verzie úlohy je potrebná implementácia v čase $O(n.\log n)$ (použutímm prioritného radu), resp. $O(n)$ (spájané zoznamy pre každú vzdialenosť).

Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu a Európskeho fondu regionálneho rozvoja v rámci Operačného programu Ľudské zdroje

4 Poukážky

Riešenie ľahšej verzie úlohy Poukážky je vyskúšanie všetkých možných nerastúcich postupností dĺžky T so súčtom P. Generovanie všetkých postupností a testovanie monotónnosti a celkového súčtu je triviálne v čase $O(T.P^T)$.

```
#nacitanie jedneho vstupu
T, P = [int(_) for _ in input().split()]
A = [0]*T #postupnost dlzky T

def gen(ind):
        global T, P, A
        #ak je vygenerovana cela postupnost dlzky T
        if (ind == T):
                #kontrola nerastucej postupnosti so suctom P
                if sum(A) == P:
                        if all(x >= y for x,y in zip(A, A[1:])):
                                return 1
                return 0
        #generovanie vsetkych moznosti na pozicii ind
        pocet = 0
        for cislo in range(P+1):
                A[ind] = cislo
                pocet = pocet+gen(ind+1)
        return pocet

print(gen(0))
```

Efektívnejšie riešenie využíva Backtrack, generuje priamo len nerastúce postupnosti a priebežne odpočítava (aby sa ukončilo pri prvom prekročení počtu).

```
T, P = [int(_) for _ in input().split()]

#zostava este naplnit sucet s maximalnou hodnotou maxim
def gen(ind, sucet, maxim):
        #ak je vygenerovana cela postupnost dlzky T
        if ind == T:
                return 1 if sucet == 0 else 0
        if sucet < 0:
                return 0
        #spocitanie vsetkych nerastucich moznosti na pozicii ind
        return sum(
                gen(ind+1, sucet-cislo, cislo)
                for cislo in range(maxim+1)
        )

print(gen(0, P, P))
```


EURÓPSKA ÚNIA
Európsky sociálny fond
Európsky fond regionálneho rozvoja
OPERAČNÝ PROGRAM ĽUDSKÉ ZDROJE
Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu a Európskeho fondu regionálneho rozvoja v rámci Operačného programu Ľudské zdroje

Prípadne môžeme priamo generovať len postupnosti, kde súčet bude P - teda obmedzíme maximálne hodnoty aj podľa aktuálneho súčtu. Takéto riešenie beží približne v čase $O(T.P!)$.

```
T, P = [int(_) for _ in input().split()]

def gen(ind, sucet, maxim):
        global T
        #ak je vygenerovana cela postupnost dlzky T
        if ind == T:
                return 1
        return sum(
                gen(ind+1, sucet-cislo, cislo)
                for cislo in range(1+min(maxim, sucet // (T-ind)))
        )

print(gen(0, P, P))
```

Na riešenie ťažšej verzie úlohy už potrebujeme predchádzajúce riešenie vylepšiť memoizáciou (aby sa zbytočne nepočítali tie isté hodnoty viackrát). Budeme si pamätať pole $OPT[T, P, max]$, čím dostávame zložitosť $O(T.P^3)$ - pole veľkosti $T \times P \times P$ a výpočet hodnoty jedného prvku v čase $O(P)$. V jazyku Python od verzie 3.2 môžete na memoizáciu použiť vstavaný dekorátor.

```
import functools

@functools.lru_cache(maxsize=None)
def gen(ind, sucet, maxim):
        ...
```

Podrobnejším rozborom úlohy je možné vytvoriť riešenie dynamickým programovaním s pamätaním si len dvojrozmerného poľa $OPT[T, P]$. Ak je najmenšie číslo v postupnosti 0, tak ide o $OPT[T-1, P]$ možností. Všeobecnejšie, ak je najmenšie - teda posledné - číslo v postupnosti k, tak súčet je aspoň $k.T$ a rozdeliť zostáva iba zvyšok medzi $T-1$ tímov, teda $OPT[T-1, P-k*T]$. Výsledný vzorec potom je $OPT[T, P] = \sum_{k=0}^{\lfloor \frac{P}{T} \rfloor} OPT[T-1, P-k*T]$ a dáva zložitosť $O(T.P^2)$.

```
T, P = [int(_) for _ in input().split()]
#pole OPT o rozmeroch (T+1)*(P+1), indexovane od nuly
#pre 1 tim je len jedna moznost
OPT = [[1 if i==1 else 0 for j in range(P+1)] for i in range(T+1)]
#dynamicke programovanie pre rastuce P a T
for i in range(2,T+1):
        for j in range(P+1):
                OPT[i][j] = sum(
                        OPT[i-1][j-k*i]
                        for k in range(1+(j // i))
                )
#vysledok
print(OPT[T][P])
```


Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu a Európskeho fondu regionálneho rozvoja v rámci Operačného programu Ľudské zdroje

Ešte efektívnejšie riešenie dostaneme, ak uvažujeme že sú len dve možnosti - buď je najmenšie číslo 0 alebo môžeme z každého čísla 1 odpočítať a možností je potom $OPT[T, P-T]$. Výpočet je teda $OPT[T, P] = OPT[T-1, P] + OPT[T, P-T]$, čím dostávame výslednú zložitosť $O(T.P)$.

```
T, P = [int(_) for _ in input().split()]
#pole OPT o rozmeroch (T+1)*(P+1), indexovane od nuly
#pre 1 tim je len jedna moznost
OPT = [[1 if i==1 else 0 for j in range(P+1)] for i in range(T+1)]
#dynamicke programovanie pre rastuce P a T
for i in range(2,T+1):
        for j in range(P+1):
                OPT[i][j] = OPT[i-1][j]+OPT[i][j-i]
#vysledok
print(OPT[T][P])
```

Keďže je na vstupe viacero testovacích sád a hodnoty poľa OPT sa nemenia, je výhodné si ich počítať iba raz a nie počítať pre každú testovaciu sadu znova. Buď v testovacej sade len dopočítame ďalšie ešte nepočítané hodnoty (ak dostaneme väčšie P a/alebo T), alebo na začiatku vyplníme celú tabuľku 5000×100.

EURÓPSKA ÚNIA
Európsky sociálny fond
Európsky fond regionálneho rozvoja
OPERAČNÝ PROGRAM ĽUDSKÉ ZDROJE
Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu a Európskeho fondu regionálneho rozvoja v rámci Operačného programu Ľudské zdroje