PRASK
II. ročník, 2015/16
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

Vzorové riešenia 2. kola letnej časti

1. Princová dilema

vzorák napísal Prefix
(max. 15 b za riešenie)

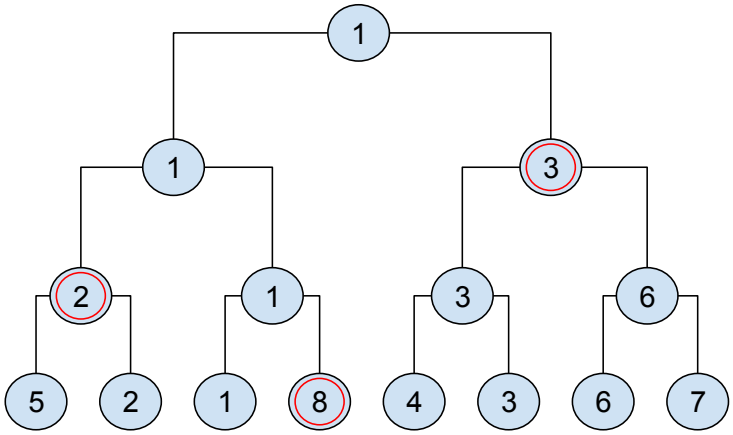
Najľahšia guličkа

Zamyslime sa najprv, ako by sme hľadali najľahšiu guličku.

Prvý spôsob by bol taký, že si vyberieme ľubovoľnú guličku a povieme si, že je to náš favorit na najľahšiu guličku. Teraz favorita porovnáme s inou guličkou. Čo sa môže stať? Nová guličkа je ľahšia ako favorit. Favorit už nemôže byť tou najľahšou guličkou a teda ho musíme zmeniť. Novým favoritom sa stane guličkа ktorá porazila toho predošlého. Ak je ale pôvodný favorit ľahší, znamená to, že stále má šancu stať sa nakoniec tou najľahšou guličkou a teda si ho ponecháme. Naviac guličkа, s ktorou sme ho porovnávali, nemôže byť najľahšia guličkа. Keď toto porovnanie s favoritom a prípadnú výmenu urobíme s každou guličkou, na konci bude favoritom tá najľahšia guličkа zo všetkých.

Druhý spôsob je trochu zložitejší. Usporiadame medzi guličkami turnaj. Najprv všetky guličky rozdelíme na dvojice, medzi nimi usporiadame zápasy (porovnania) a do ďalšieho kola postúpi z každého zápasu tá ľahšia guličkа. Takže zo šestnástich guličiek nám ostane 8. V druhom kole opäť popárujeme tieto zostávajúce guličky, porovnáme zápasiace guličky a postúpia len 4. Toto opakujeme, až kým nám v kole nezostane len jedna guličkа, ktorá je zároveň tou najľahšou, keďže najľahšia guličkа porovnanie nikdy neprehrá.

Koľko porovnávaní potrebujú tieto spôsoby? V prvom spôsobe každú guličku (okrem jednej – tej, ktorá je favorit na začiatku) raz porovnáme a možno vymeníme s favoritom, teda urobíme 15 porovnaní. Keby sme mali n guličiek, použili by sme $n-1$ porovnaní. V druhom spôsobe sme si v každom kole rozdelili guličky do dvojíc a v každej dvojici sme urobili jedno porovnanie. Ak bolo v kole k guličiek, urobili sme v tomto kole $k/2$ porovnaní. Zároveň do ďalšieho kola postúpi len polovica guličiek. V našom príklade by bolo v prvom kole 16 guličiek, v druhom 8, v treťom 4, v štvrtom 2 a potom už máme určeného víťaza. Ak tieto počty vydelíme dvoma a sčítame, dostaneme celkový počet porovnaní, a to je $8+4+2+1=15$. Pre všeobecný prípad s n guličkami to je $n/2+n/4+n/8...+1=n-1$.[1] Takže oba spôsoby sú rovnako dobré.



Na obrázku si môžete pozrieť turnaj s 8 guličkami, čísla v krúžkoch sú váhy guličiek. Červenou sú zvýraznené všetky guličky, ktoré prehrali s najľahšou guličkou – potenciálne druhé najľahšie guličky.

Druhá najľahšia guličkа

Ako nájsť druhú najľahšiu guličku? Skúsime sa odraziť od spôsobov na nájdenie tej najľahšej. V oboch spôsobov sme robili zápasy, ale iným spôsobom. V prvom išiel do zápasu vždy favorit a nová guličkа, v druhom

[1] Toto funguje najlepšie pre počet guličiek 2^k. Vtedy totiž vieme celý čas guličky rozdeľovať do dvojíc. Pre všeobecné n v turnaji guličkа bez páru postupuje automaticky. Napriek tomu však spravíme $n-1$ porovnaní.

sme na to išli kolami. Dôležité pozorovanie je, že druhú najľahšiu guličku musela v zápase poraziť tá najľahšia guličká. Nad ostatnými totiž druhá najľahšia guličká zápas určite vyhrá. Stačí nám teraz nájsť tú najľahšiu guličku spomedzi tých, ktoré porazila tá úplne najľahšia. Koľko najviac takýchto guličiek môže byť?

Ak sme našli najľahšiu guličku prvým spôsobom, v najhoršom prípade sa nám mohlo stať, že sme si na začiatku vybrali ako favorita práve tú najľahšiu guličku, ktorá následne porazila všetky zvyšné guličky. V tomto prípade by sme teda museli nájsť najľahšiu spomedzi 15 guličiek, na čo by sme potrebovali 14 porovnaní. Všobecne, z n guličiek nám zostalo $n-1$ a potrebovali by sme teda $n-2$ porovnaní.

V druhom spôsobe tá najľahšia guličká mohla vyhrať omnoho menej zápasov, keďže v každom kole zápasila len raz. Koľko bolo kôl? V našom prípade bolo kolo so 16 guličkami, 8, 4 a 2. Len 4 kolá, to znamená, že musíme vybrať najľahšiu už len spomedzi štyroch guličiek, na čo nám stačia 3 porovnania. Na to, aby sme určili počet porovnaní pre všeobecný prípad, musíme vypočítať, koľko bude vo všeobecnom prípade kôl. V každom kole sa počet guličiek zmenší na polovicu až kým nezostane len jedna guličká. Takže kôl pre číslo n je toľko, koľko krát by sme vedeli vydeliť n dvojkou. Ak predpokladáme, že $n = 2^k$, tak sa odohrá k kôl.

Pre všeobecné n existuje matematická funkcia **logaritmus**, pre ktorú platí $n = 2^{\log_2(n)}$,[2], teda $\log_2(n) = k$. Počet kôl je teda $\log_2(n)$. Takže musíme nájsť tú najľahšiu spomedzi $\log_2(n)$ guličiek, na čo potrebujeme $\log_2(n) - 1$ porovnaní.

Týmto spôsobom sme použili $n-1$ porovnaní na nájdenie najľahšej guličky, a $\log_2(n) - 1$ na nasledovné hľadanie druhej najľahšej guličky. Dokopy teda použijeme $n + \log_2(n) - 2$ porovnaní, v našom prípade teda 18.

2. Preblšený cirkus sa vracia

vzorák napísal Baklažán
(max. 15 b za riešenie)

Podúloha a)

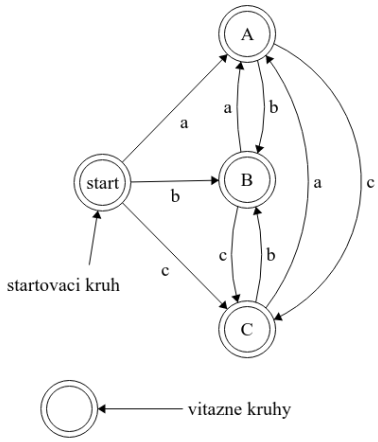
Chceme, aby naša schéma kontrolovala, či sa vo vstupnom texte vyskytujú dve rovnaké písmená po sebe. Na to by sa nám hodilo pamätať si, aké písmeno bolo prečítané ako posledné. Keďže blcha nemá pamäť, budeme túto informáciu musieť zakódovať do pozície blchy. To môžeme urobiť napríklad tak, že budeme mať tri kruhy (označme ich A, B a C), pričom:

- V kruhu A bude blcha práve vtedy, keď posledné prečítané písmeno bolo `„a"` a zatiaľ nemáme dôvod myslieť si, že vstupné slovo nie je víťazné (teda krotiteľ doteraz neprečítal dve rovnaké písmená po sebe, ani neprečítal iné písmeno než `„a"`, `„b"` alebo `„c"`).
- V kruhu B bude blcha vtedy, keď posledné prečítané písmeno bolo `„b"` a vstupné slovo ešte môže byť víťazné.
- V kruhu C bude blcha vtedy, keď posledné prečítané písmeno bolo `„c"` a vstupné slovo ešte môže byť víťazné.

Na začiatku, keď ešte nebolo prečítané žiadne písmeno, by blcha nemala byť v žiadnom z kruhov A, B, C, preto budeme mať okrem týchto troch kruhov štvrtý, štartovací. Zo všetkých kruhov okrem A nakreslíme šípku označenú `„a"` do kruhu A, zo všetkých kruhov okrem B nakreslíme šípku označenú `„b"` do kruhu B a zo všetkých kruhov okrem C nakreslíme šípku označenú `„c"` do kruhu C. Takto sa bude blcha počas predstavenia vždy nachádzať v správnom kruhu podľa toho, čo krotiteľ prečítal ako posledné.

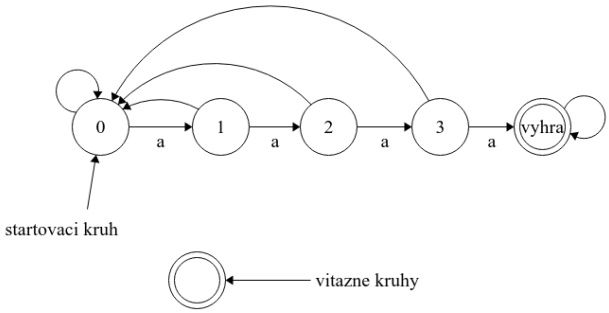
Keď je blcha v kruhu A a krotiteľ prečíta písmeno `„a"`, znamená to, že vo vstupnom texte boli dve `„a"`-čka po sebe. V takom prípade teda vieme, že vstupný text nemá byť víťazný. Preto nemá zmysel pokračovať v predstavení a blcha sa na to môže rovno vykašľať. Z kruhu A teda nenakreslíme žiadnu šípku označnú `„a"` (a ani žiadnu neoznačenú šípku). Podobne nenakreslíme šípku označenú `„b"` z kruhu B, ani šípku `„c"` z kruhu C. Takýmto spôsobom zaručíme, že vo všetkých zlých prípadoch blcha vyskočí von zo schémy – aj v prípade, že krotiteľ prečíta nejaké iné písmeno než `„a"`, `„b"` a `„c"`, aj v prípade, že sú vo vstupnom texte dve rovnaké písmená za sebou. V schéme teda blcha ostane až do konca predstavenia iba pri víťazných textoch, preto všetky štyri kruhy označíme ako víťazné.

[2] Dvojkový logaritmus čísla n je také číslo k, že keď umocníme číslo 2 na k, dostaneme číslo n.



Podúloha b)

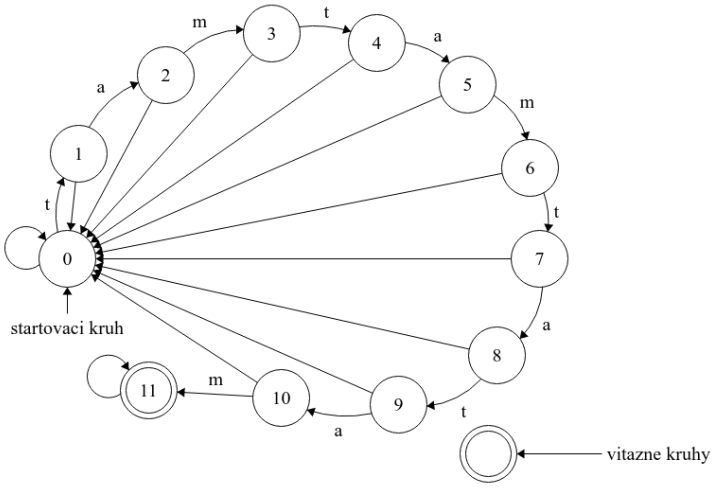
Tentoraz si potrebujeme pamätať, či posledné prečítané písmená boli „a"-čka, a ak áno, koľko za sebou ich bolo. Okrem toho si v prípade, keď krotiteľ prečíta štyri „a"-čka po sebe, potrebujeme zapamätať, že vstupný text je určite víťazný. Nakreslíme si teda 5 kruhov, označme ich $0, 1, 2, 3$ a "výhra". V kruhu s číslom x bude blcha práve vtedy, keď posledných x prečítaných písmen boli „a"-čka a písmeno pred tým nebolo „a". Kruh "výhra" bude víťazný kruh, do ktorého blcha skočí vtedy, keď krotiteľ prečíta 4 „a"-čka po sebe a z tohto kruhu už potom nikdy nevyskočí von. Keď bude blcha v kruhu s nejakým číslom a krotiteľ prečíta „a", znamená to, že počet po sebe prečítaných „a"-čok sa zväčšil o 1, teda blchu necháme skočiť do kruhu s o 1 väčším číslom (z kruhu číslo 3 ju necháme skočiť do kruhu "výhra", lebo štyri „a"-čka znamenajú víťazné slovo). Ak je blcha v číslovanom kruhu a krotiteľ prečíta niečo iné ako „a", znamená to, že v tomto momente je z posledných prečítaných písmen nula „a"-čok, teda ju necháme skočiť do kruhu 0. Kruh 0 bude zároveň aj štartovací.



Predstavenie teda bude vyzerať tak, že vždy, keď bude krotiteľ čítať „a"-čka, blcha bude postupovať smerom ku víťaznému kruhu a vždy, keď prečíta niečo iné, blcha "padne" späť na nulu. Ak krotiteľ niekedy prečíta 4 „a"-čka po sebe, blche sa podarí dostať do víťazného kruhu a tam ostane až do konca predstavenia.

Podúloha c)

Táto podúloha sa dosť podobá na tu predošlú, čo by nás mohlo navádzať na podobné riešenie: nakreslíme si 12 krúžkov očíslovaných 0 až 11 (keďže slovo „tamtamtatam" má jedenásť písmen), pričom v krúžku s číslom x bude blška vtedy, keď sa poslených x prečítaných písmen zhoduje s prvými x písmenami slova „tamtamtatam". Krúžok 11 označíme ako víťazný a nakreslíme mu neoznačenú šípku do neho samého, takže keď sa tam blcha raz dostane, už odtiaľ neujde. Celé by to teda mohlo vyzerať takto:



Mnohí z vás sa vo svojich riešeniach dopracovali presne sem. Toto riešenie však ešte **nie je správne**. Predstavte si, čo by sa stalo, ak by vstupný text bol „`ttamtamtatam`". Po prečítaní prvého „`t`" by blcha skočila do krúžku 1, po prečítaní druhého „`t`" by ale skočila do kruhu 0, kde by ostala aj na najbližšie dve písmená „`am`", následne by počas čítania ďalších 5 písmen „`tamta`" doskákala do kruhu 5, ale potom by pri prečítaní „`t`" padla opäť na 0 a tam by ostala až do konca. Text „`ttamtamtatam`" v našej schéme teda nie je víťazný, napriek tomu, že obsahuje slovo „`tamtamtatam`".

Ako sa s týmto problémom vysporiadať? Prvý spôsob, ktorý by nám mohol napadnúť, je dokresliť zo všetkých krúžkov (okrem kruhu 11), z ktorých nevedie šípka označená „`t`", šípku do kruhu 1 označenú „`t`". To síce pomôže pri vstupnom texte „`ttamtamtatam`", ale nevyrieši to úplne všetko. Skúste si napríklad rozmyslieť, čo by sa v takejto schéme stalo, ak by vstupné slovo bolo „`tamtamtamtatam`". Mohli by sme naše riešenie skúšať plátať ďalej, ale to väčšinou nie je tá správna cesta pri riešení problémov: riešenie by sa stávalo čoraz zložitejším a ľahko by sme mohli na niektoré prípady zabudnúť.

Lepšie je postupovať systematicky. Najprv si *poriadne* definujeme, kedy má byť blcha v ktorom kruhu:

- V kruhu 11 bude blcha vtedy a len vtedy, ak v tom, čo už krotiteľ prečítal, niekde bolo celé slovo „`tamtamtatam`".
- V akomkoľvek inom kruhu číslo x bude blcha vtedy a len vtedy ak je číslo x najväčšie číslo, pre ktoré platí nasledovná vlastnosť:
  - Posledných x písmen, ktoré krotiteľ prečítal sa zhoduje s prvými x písmenami slova „`tamtamtatam`".

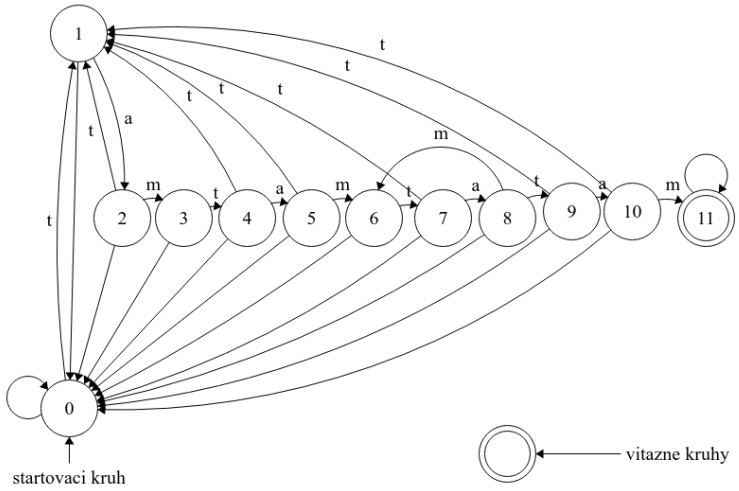
  Ak teda napríklad posledných 10 prečítaných písmen bolo „`ratatamtam`", blcha nebude v kruhu 3, ale v kruhu 6. Je síce pravda, že posledné 3 prečítané písmená sa zhodujú s prvými troma písmenami „`tamtamtatam`", ale aj posledných 6 písmen sa zhoduje a 6 je viac ako 3.

Ako teraz správne nakresliť šípky? Prvá vec, ktorú si môžeme uvedomiť, je nasledovná: ak je blcha v nejakom momente v kruhu číslo x a $x > 0$, o písmeno skôr musela byť v kruhu s číslom minimálne $x - 1$: ak sa totiž teraz posledných x prečítaných písmen zhoduje s prvými x písmenami „`tamtamtatam`", o písmeno skôr sa určite zhodovalo posledných $x - 1$ písmen. To okrem iného znamená, že zo žiadneho kruhu sa nedá na jeden skok dostať do kruhu, ktorého číslo by bolo väčšie o 2 alebo viac. Z každého kruhu teda pôjde jedna šípka do kruhu s o 1 vyšším číslom a všetky ostatné šípky pôjdu do kruhov s menšími číslami. Šípky postupujúce do kruhov s väčšími číslami teda budú vyzerať rovnako ako v pôvodnom riešení. Zaujímavejšie sú šípky vedúce dozadu.

- Ak je blcha v kruhu 1, posledné prečítané písmeno muselo byť „`t`".
  - Ak krotiteľ prečíta „`t`", posledné dve písmená budú „`tt`", čo znamená, že posledné písmeno sa zhoduje s prvým písmenom „`tamtamtatam`", ale viac ako jedno sa určite nezhoduje. Preto by v takom prípade blcha mala skočiť naspäť do kruhu 1.
  - Ak ktoriteľ prečíta čokoľvek iné ako „`t`" alebo „`a`", posledné prečítané písmeno nebude `t`, teda blcha by mala skočiť do kruhu 0.
- Ak je blcha v kruhu 2, posledné dve prečítané písmená boli „`ta`".

  - Ak krotiteľ prečíta „t"m posledné tri písmená budú „tat", čo znamená, že blcha bude musieť skočiť do kruhu 1.
  - Ak krotiteľ prečíta čokoľvek iné ako „t" alebo „m", blcha skočí do kruhu 0.
- Ak je blcha v kruhu 3, posledné tri písmená boli „tam". Ak krotiteľ prečíta čokoľvek iné ako „t", žiadny nenulový počet posledných prečítaných písmen sa nebude zhodovať so začiatkom slova „tamtamtatam". Preto blcha skočí do kruhu 0.
- Pre kruhy 4, 5 a 7 bude situácia podobná ako pre kruh 2, pre kruh 6 zasa ako pre kruh 3. Zaujímavý moment ale príde pri kruhu 8.
- Ak je blcha v kruhu 8, posledné prečítané písmená sú „tamtamta".
  - Ak teraz krotiteľ prečíta „m", posledných 9 písmen bude „tamtamtam", teda až 6 posledných písmen sa bude zhodovať s prvými 6 písmenami „tamtamtatam". Preto vtedy blcha skočí do kruhu 6.
  - Ak krotiteľ prečíta čokoľvek iné ako „t" alebo „m", blcha skočí do kruhu 0.
- Pre kruhy 9 a 10 bude situácia rovnaká ako pri kruhu 2.

Celá schéma teda bude vyzerať nasledovne:



3. Problémové pohostenie

vzorák napísal Andrej
(max. 15 b za riešenie)

Podúloha a)

Riešenie prvej podúlohy je celkom priamočiare, je treba napísať program, ktorý za nás rýchlo nakliká koláče.

AutoHotKey

My sme na riešenie použili funkciu Click, ktorá si ako parameter berie x, čo je číslo, ktoré nám hovorí, koľko krát sa funkcia vykoná. V našom prípade to bude ľubovolné rozumné čislo vačšie alebo rovné 150. Keď skript spúšťate, dajte si pozor, aby ste kurzor mali nastavený na koláči.

Listing programu (Text)

```
e::
Click 150
Return
```

Podúloha b)

V tejto podúlohe opäť nie je potrebné veľa rozmýšľať, ale treba sa hneď zamyslieť, ako riešenie naimplementovať. Všetko, čo potrebujeme, je písať niečo do formulára a preskakovať medzi jednotlivými oknami. Na vypisovanie do okien použijeme funkciu Send, ktorej ako parameter dáme reťazec, ktorý ma vypísať. Ako ale preskakovať medzi oknami? Jednoducho, tabulátorom – to je síce klávesa, ale vieme ho podobne ako písmena

vypísať príkazom `Send, 't` . Posledným problémom môžu byť políčka, ktoré je treba zaškrtnúť, tie vyriešime pomocou príkazu `Send, {Space}` , čím akoby "stlačíme" medzerník. Ten nám posledné dve okná označí. Už stačí len kliknúť do prvého okna a spustiť náš skript.

Listing programu (C++)

```
x::
Send, Andrej
Send, `t
Send, Král
Send, `t
Send, Génius
Send, `t
Send, Lúčka-Potoky 1
Send, `t
Send, Hlohovec
Send, `t
Send, Slovenské
Send, `t
Send, kuchár
Send, `t
Send, Chcem získať body do výsledkovky, naozaj ich potrebujem lebo chcem ísť na sústredko :)
Send, `t
Send, 10000001
Send, `t
Send, 38
Send, `t
Send, {Space}
Send, `t
Send, {Space}
Send, `t
Send, {Enter}
Return
```

Podúloha c)

Táto podúloha má viac možných riešení, my použijeme trochu pokročilejšie pomocou vlastných vytvorených funkcií.

Najprv potrebujeme zistiť súradnice jednotlivých obrázkov, tie si uložíme do globalných premenných.

Urobíme to pomocou funkcie MouseGetPos, ktorá si ako parameter berie 2 premenné a do nich uloži x-ovú a y-ovú súradicu na ktorej je práve kurzor. Tak si teda vytvoríme dve premenné pre každý objekt, Mäso, Cibuľu, Kapustu, Smotanu, Korenie a Hrniec. Spravíme to tak, že postupne myš namierime na každý objekt a zavoláme funkciu `kdeObjekt()` [3].

Tá nam uloží do našich premenných pozície jednotlivých objektov. Keď už máme súradnice, použijeme funkciu MouseClickDrag. Tá si berie 5 parametrov(T, x, y, $x1$, $y1$), prvý hovorí, s ktorým tlačidlom myši budeme pracovať ďalšie 4 sú najprv x-ová a y-ová suradnica odkiaľ a potom kam sa má myšou ťahať. Táto funkcia na súradniciach $[x; y]$ stlačí myš, potiahne ju na nové súradnice $[x1; y1]$ a uvoľní tlačidlo. Pre každý objekt si spravíme funkciu s rovnakým názvom, ktorá nám prenesie objekt typu Objekt do hrnca.

Potom už len nakombinujeme tieto funkcie tak, aby boli v správnom poradí.

Listing programu (C++)

```
global m1, m2, c1, c2, ka1, ka2, s1, s2, ko1, ko2, h1, h2

kdeMaso() {
    MouseGetPos, m1, m2
}

kdeCibula() {
    MouseGetPos, c1, c2
}

kdeKapusta() {
    MouseGetPos, ka1, ka2
}

kdeSmotana() {
    MouseGetPos, s1, s2
}

kdeKorenie() {
    MouseGetPos, ko1, ko2
}

kdeHrniec() {
    MouseGetPos, h1, h2
}

a::
MouseGetPos, m1, m2
```

[3] Pozri implementáciu nižšie.

```
Return

s::
MouseGetPos, c1, c2
Return

d::
MouseGetPos, ka1, ka2
Return

f::
MouseGetPos, s1, s2
Return

g::
MouseGetPos, ko1, ko2
Return

h::
MouseGetPos, h1, h2
Return

Maso() {
    MouseClickDrag, L, %m1%, %m2%, h1, h2
}

Cibula() {
    MouseClickDrag, L, %c1%, %c2%, h1, h2
}

Kapusta() {
    MouseClickDrag, L, %ka1%, %ka2%, h1, h2
}

Smotana() {
    MouseClickDrag, L, %s1%, %s2%, h1, h2
}

Korenie() {
    MouseClickDrag, L, %ko1%, %ko2%, h1, h2
}

x::
Cibula()
Kapusta()
Kapusta()
Maso()
Korenie()
Maso()
Korenie()
Kapusta()
Cibula()
Smotana()
Korenie()
Korenie()
Smotana()
Maso()
Cibula()
Maso()
Cibula()
Smotana()
Return
```

vzorák napísal Žaba
(max. 15 b za riešenie)

4. Programátori a zásuvky

Máme predlžovačku s n zásuvkami, do ktorej chceme vsunúť k normálnych úzkych zástrčiek a m širokých zástrčiek, ktoré po zasunutí zaberú miesto aj v susedných zásuvkách. Ako sa však dalo všimnúť v ukážkovom príklade, tak vieme zastrčiť dve široké zástrčky tak, že je medzi nimi len jedna voľná zásuvka.

Je jasné, že hlavný problém je ako pozastrkávať široké zástrčky tak, aby zabrali čo najmenej miesta. Zdá sa, že vhodné miesto pre širokú zástrčku je na kraji predlžovačky. Ak ju totiž dáme na kraj, zaberie iba dve zásuvky, z jednej strany totiž prečnieva ponad koniec predlžovačky. Jediná susedná zásuvka krajnej zásuvky bude určite zablokovaná, do tretej zásuvky v poradí však môžeme dať aj úzku aj širokú zástrčku. Zbavili sme sa teda jednej širokej zástrčky a našu predlžovačku sme si skrátili na $n - 2$ zásuviek.

Túto myšlienku sa nám však oplatí zopakovať opäť, až kým nevyčerpáme všetky široké zástrčky. Tie sme teda nastrkali čo najviac na jeden kraj predlžovačky čo najbližšie k sebe, každá zabrala dve zásuvky, preto nám zostalo $n - 2m$ zásuviek, do ktorých chceme vložiť k úzkych zástrčiek. Je jasné, že to sa bude dať spraviť iba ak je $u \leq n - 2m$.

Takéto riešenie nie je problém naprogramovať, stačí predsa v čase $O(1)$ overiť túto jedinú podmienku. Teda takmer. Špeciálny prípad nastane, ak bude $k = 0$, teda máme iba široké zástrčky. Vtedy nám postačuje iba $2m - 1$ zásuviek. Takéto riešenie nám už dá na testovači plný počet bodov.

Listing programu (C++)

```cpp
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;

int main () {
  long long n, m, k;
  cin >> n >> m >> k;
  n -= 2*m;
  if (k == 0) {
    n++;
  }
  if (n < k) {
    cout << "nie\n";
  } else {
    cout << "ano\n";
  }
}
```

Riešenie, ktoré sme vymysleli nazývame *pažravé*[4]. Používame tam totiž tvrdenie: *Určite bude najlepšie, ak všetky široké zástrčky dáme vedľa seba na kraj predlžovačky.* Platí to však vždy? To musíme dokázať v popise.

Najlepšie sa takéto tvrdenie dokazuje nasledovne. Zoberieme si nejaké rozmiestnenie zástrčiek do predlžovačky. Ľubovoľné také rozmiestnenie. No a potom ukážeme, že takéto rozmiestnenie vieme upraviť na naše rozmiestnenie, ktoré má široké zástrčky vedľa seba na (ľavom) kraji predlžovačky.

Zoberme si ľubovoľné rozmiestnenie zástrčiek. Prvé čo spravíme je, že posunieme všetky zástrčky čo najviac doľava, aby sme odstránili zbytočné medzery (samozrejme, nejaké zásuvky zostanú voľné lebo budú zakryté širokými zástrčkami). Následne, ak to nevyzerá ako naše riešenie, tak niektoré dve široké zástrčky nie sú pri sebe. Medzi nimi je teda jedna, alebo viac úzkych zástrčiek.

Ak „`o`" označuje prázdnu zásuvku, „`u`" zásuvku s úzkou zástrčkou a „`s`" zásuvku so širokou zástrčkou, tak kus predlžovačky, kde niečo takéto vznikne môže vyzerať „`osouuoso`". Nič však nepokazíme, ak tieto zástrčky prepojíme do tvaru „`ososouuo`". Akurát sme odstránili problém širokých zástrčiek, ktoré neboli vedľa seba. Opakovaním tohto postupu naozaj vytvoríme rovnaké riešenie ako náš program – široké zástrčky na kraji predlžovačky.

Vidíme teda, že ľubovoľné riešenie vieme prerobiť na nami navrhnuté riešenie, preto je takéto riešenie určite správne.

5. Poradie trpaslíkov

vzorák napísal Dávid
(max. 15 b za riešenie)

Hrubá sila

Riešenie hrubou silou vyzerá tak, že postupujeme presne podľa zadania. Postupne si vygenerujeme každú permutáciu cifier zadaného čísla a potom všetky takto vytvorené čísla sčítame. V jazyku `C++` sa to dá robiť pomocou príkazu `next_permutation()`, ktorý generuje ďalšiu permutácie poľa. V Pythone sa na to dá použiť knižnica `itertools`. Ak si počet cifier zadaného čísla označíme k, tak takéto riešenie má časovú zložitosť $O(k!)$ (k faktoriál) a na testovači by malo získať 2 body.

Listing programu (Python)

```python
from itertools import permutations

n = int(input())

for i in range(n):
    c = input()
    print(sum(int(''.join(p)) for p in permutations(c)))
```

Optimálne riešenie

Prvá otázka, ktorú musíme vyriešiť je, koľko rôznych permutácií k cifier vlastne existuje. Ak chceme z k cifier vyrobiť číslo, dôležité je, v akom poradí ich za seba naukladáme. Na prvé miesto môžeme dať ľubovoľnú z cifier k dispozícii, takže máme k možností. Na druhé miesto, ale už nemáme k možností, ale iba $k-1$, lebo jedna cifra (aj keď nevieme ktorá) leží na prvej pozícii. Preto máme pre druhé miesto iba $k-1$ možností, čo je dokopy pre prvé dve miesta $k \cdot (k-1)$ možností.

Neprekvapivo, pre tretie miesto máme $k-2$ možností a tak ďalej. Na poslednom, k-tom mieste máme len 1 možnosť, lebo nám zostala posledná nezaradená cifra. Dokopy máme teda $k \cdot (k-1) \cdot (k-2) \ldots 2 \cdot 1$ možností. Takýto súčin čísel od 1 po k zvykneme tiež označovať $k!$ (k faktoriál).

[4] Po anglicky greedy.

Označme si cifry nášho čísla zľava doprava ako $a_k\ a_{k-1} \ldots a_2\ a_1$. Toto číslo si teda môžeme zapísať aj ako:

$$10^{k-1} \cdot a_k + 10^{k-2} \cdot a_{k-1} + \ldots + 10^1 \cdot a_2 + 10^0 \cdot a_1$$

Je jasné, že to koľko zaváži daná cifra je dané aj jej poradím v čísle. Skúsme teraz vypočítať, akú hodnotu pridá cifra a_1 do súčtu všetkých permutácií. Ak dáme cifru a_1 na prvú pozíciu tak nám do súčtu pridá $10^{k-1}a_1$. Potrebujeme už len zistiť, v koľkých permutáciách bude a_1 na prvom mieste. Keď si však takýmto spôsobom zafixujeme prvú cifru, ostane nám $k-1$ cifier, ktoré chceme uložiť na $k-1$ pozícií, teda máme $(k-1)!$ rôznych možností.

Ak dáme cifru a_1 na druhú pozíciu, bude pridávať do súčtu hodnotu $10^{k-2}a_1$ a opäť $(k-1)!$ krát, lebo zafixovaíím druhej pozície nám opäť ostane $k-1$ cifier na $k-1$ pozícií. A to isté bude platiť pre ľubovoľné miesto, kam našu cifru a_1 uložíme. Ak toto všetko sčítame, dostaneme celkovú hodnotu, ktorú do súčtu permutácií pridá cifra a_1 a táto hodnota bude:

$$10^{k-1}a_1(k-1)! + 10^{k-2}a_1(k-1)! + \ldots + 10^1a_1(k-1)! + 10^0a_1(k-1)!$$

Po vyňatí a_1 a $(k-1)!$ sa nám to upraví na jednoduchší tvar

$$a_1(k-1)!(10^{k-1} + 10^{k-2} + \ldots 10 + 1)$$

Táto istá úvaha však platí pre ľubovoľnú cifru, nie len pre a_1. A cifry sú od seba nezávislé, preto celkový súčet permutácií bude určite rovný súčtu hodnôt, ktoré pridajú jednotlivé cifry. Výsledný súčet všetkých permutácií preto bude

$$a_1(k-1)!(10^{k-1}+10^{k-2}+\ldots 10+1)+a_2(k-1)!(10^{k-1}+10^{k-2}+\ldots 10+1)+\ldots+a_k(k-1)!(10^{k-1}+10^{k-2}+\ldots 10+1)$$

čo môžeme opäť upraviť na oveľa jednoduchší tvar

$$(a_1 + a_2 + \ldots + a_k)(k-1)!(10^{k-1} + 10^{k-2} + \ldots 10 + 1)$$

Výsledok teda vieme vypočítať ako súčin troch členov. Prvý z nich je súčet cifier, druhý je $(k-1)!$, čo je $(k-1)(k-2)\ldots 1$ a tretí súčet mocnín 10. Každý z týchto členov vieme vypočítať v čase $O(k)$, čo je teda výsledná časová zložitosť nášho programu. Pamäťová zložitosť je dokonca konštantná $O(1)$, keďže cifry zadaného čísla vieme počítať postupne jeho delením 10.

Listing programu (C++)

```
#include <iostream>
#include <cmath>

using namespace std;

int main(){
    int n;
    cin>>n;
    for (int i=0; i<n; i++){
        long long a;
        cin>>a;
        int pocet=0; //počet cifier
        int sucet=0; //súčet cifier
        long long desiatky=0; //sucet mocnin 10
        long long desat=1;
        long long faktorial=1;
        while(a>0){
            desiatky+=desat;
            desat*=10;
            pocet++;
            faktorial*=pocet;
            sucet+=a%10; //a%10 nam da poslednu cifru cisla a
            a/=10; //a/10 nam odstrani poslednu cifru cisla a
        }
        cout<<faktorial*desiatky*sucet<<endl;
    }
}
```