KSI

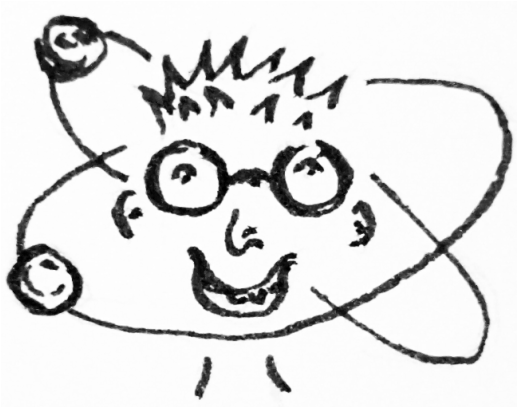
Korespondenční Seminář z Informatiky

Mílí řešitelé,
věříme, že se vám druhá sada líbila a že jste se nemohli dočkat dalších úloh. Tématem této sady je vyhledávání. V úvodníku se seznámíte se základními algoritmy a připravili jsme si pro vás i video, kde je hezky vysvětlené lineární a binární vyhledávání.

Vyhledávání

Pokud vám nic neříká pojem časová složitost, tak vám silně doporučujeme, abyste si přečetli článek na našem webu, protože tentokrát to budete opravdu potřebovat:
`http://ksi.fi.muni.cz/clanek?id=5`

Poté bychom vám doporučili podívat se na video zde:
`http://www.youtube.com/watch?v=i2oBKUScQb0`.



Lineární vyhledávaní

Lineární vyhledávání používáme, pokud potřebujeme prohledat pole nebo jinou datovou strukturu, o které nic nevíme, nebo víme, že není seřazená. Princip je triviální, prostě postupně procházíme všechny prvky a kontrolujeme, jestli jsme našli hledaný prvek. Proto je složitost tohoto algoritmu $O(n)$.

```
function linearniHledani (pole, hledanaHodnota) {
  i = 0
  while (i < delka(pole)) {
    if (pole[i] == hledanaHodnota) {
      return true
    }
    i++
  }
  return false
}
```

Binární vyhledávání (metoda půlení intervalů)

Binární vyhledávání se hodí na prohledávání seřazených polí. Jak už název napovídá, v každém kroku pole rozdělíme v polovině na dvě části a tak pokaždé vyřadíme polovinu hodnot, které nemusíme prohledat a postupně tak na konci dostaneme hledané číslo. Složitost je $O(log_2 n)$.

```
function binarniHledani (pole, levyIndex, pravyIndex, hledanaHodnota) {
  //pokud jsme dosli na konec a zbyla hodnota neni ta, co hledame,
  //pole hodnotu vubec neobsahuje
  if (levyIndex == pravyIndex AND pole[levyIndex] != hledanaHodnota) {
    return false
  }
  //najdeme prostredek pole
  stred = levyIndex + (pravyIndex - levyIndex) / 2;
  //pokud je hodnota uprostred ta co hledame, mame uspesne hotovo
  if (pole[stred] == hledanaHodnota) {
    return true
  } else if (pole[stred] > hledanaHodnota) {
    //pokud je prostredni hodnota vetsi nez hledana hodnota,
    //levou cast pole zahodime a hledame znovu napravo od
    //stredu
    return binarniHledani(pole, stred + 1, pravyIndex, hledanaHodnota);
  } else {
    //v tomto pripade je prostredni hodnota mensi nez hledana,
    //takze obdobne jako v predchozi podmince polovinu pole
    //zahodime a hledame v te spravne
    return binarniHledani (pole, leftIndex, Math.max(leftIndex, stred - 1),
       hledanaHodnota);
  }
}
```

Interpolační vyhledávání

To je vylepšené binární vyhledávání, které používáme na seřazená pole, která mají navíc rovnoměrně rozložené prvky. Na rozdíl od binárního vyhledávání nerozděluje pole v polovině, ale v blízkosti pozice prvku, který hledáme. Složitost tohoto algoritmu je $O(log(logn))$.

```
function interpolacniVyhledavani (pole, hledanaHodnota, od, do) {
  if (pole[od] == hledanaHodnota) {
    return true
  } else if (od == do OR pole[od] == array[do]) {
    //zarazka proti rekurzi
    return false
  }

  //odhadneme pozici hledaneho prvku
  odhad = od + ((do - od) / (pole[do] - pole[od])) * (hledanaHodnota -
     pole[od])

  if (pole[odhad] == hledanaHodnota) { // nalezeno
    return true
  }

  //pokud jsme nenalezli, pokracujeme stejne jako u binarniho
  //vyhledavani
  else if (pole[odhad] < hledanaHodnota) {
    return interpolacniVyhledavani (pole, hledanaHodnota, stred + 1, do)
  } else {
    return interpolacniVyhledavani (pole, hledanaHodnota, od, stred - 1);
  }
}
```

Zadání 3. sady úloh KSI (termín odevzdání: 6. 1. 2014)

Řešení zasílejte pomocí internetového systému na adrese `http://ksi.fi.muni.cz`.

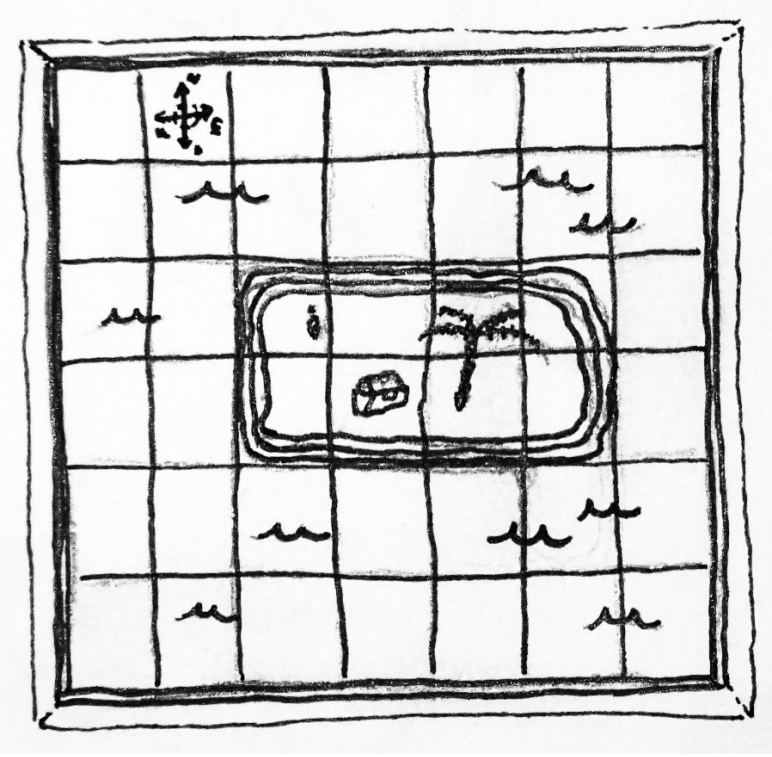
Příklad 1: Zapomenutý poklad (10 bodů)

Pirát, který je ve světě znám jako Zelená noha, si kdysi dávno zakopal do truhly na pustém ostrově část svého největšího lupu. Nyní po několika letech se mu však pokladu zachtělo zpátky. Vytáhl tedy mapu ostrova a začal hledat velké červené X označující polohu pokladu. Zírá do mapy, ale na obdélníkovém ostrově po křížku ani památky. V samé radosti z velkého lupu asi tehdy zapomněl do mapy zaznačit polohu pokladu.

Jediný, kdo s ním při zakopávání pokladu byl, je jeho mluvící papoušek. Ten je však umíněný a polohu pokladu nechce prozradit. Náš pirát tedy zkusil ukázat do mapy a zeptal se, jestli je poklad zde. Papoušek mu neodpověděl. Zkusil ukázat na nové místo na mapě a papoušek mu po dlouhém přemýšlení řekl, že je blíže než ukazoval předtím. Toho se pirát chytl a zkusil ukázat jinam. Opět po dlouhém přemýšlení papoušek řekl, že je tentokrát dále, než ukazoval předtím.

Poraďte našemu pirátovi, jak má poklad najít! Pirát papouškovi vždy ukáže místo na mapě ve čtvercové síti (viz. ilustrační mapa pod zadáním) a papoušek mu řekne, jestli ukazuje blíže nebo dále, než ukazoval v předcházejícím pokusu („tahu“). Ostrov má tvar obdélníku a leží uprostřed moře. Pirát je netrpělivý, políček na mapě je mnoho a navíc papoušek dlouho přemýšlí, takže projít všechna pole nepřipadá v úvahu.

Vymyslete a popište co nejefektivnější algoritmus, dle kterého bude náš pirát schopen najít poklad v co nejkratším čas (tedy na co nejméně pokusů). Do řešení algoritmus slovně popište, případně uveďte pseudokód. Nezapomeňte také na zdůvodnění jeho časové složitosti.



Příklad 2: Hledání mediánu (10 bodů)

Organizátoři KSI se rozhodli, že si začnou vést podrobné statistiky o tom, kolik řešitelé získávají bodů. Jako první by chtěli začít s průměrem a s mediánem. Jak asi víte, *průměr* množiny čísel je definován jako jejich součet vydělený jejich počtem. Pro nalezení *mediánu* je potřeba čísla setřídit podle velikosti. Je-li čísel lichý počet, medián je to prostřední. Je-li jich sudý počet, je to průměr prostředních dvou. Jak efektivně naprogramovat počítání průměru jednoho z organizátorů napadlo poměrně rychle. Na medián ale nikdo nemohl přijít [1], tak se rozhodli, že zavolají na pomoc řešitele.

Vaším úkolem je navrhnout algoritmus, který bude ze vstupu postupně načítat přirozená čísla a po každém načtení vypíše medián všech předchozích (všech doposud načtených). Přestože v KSI bodujeme úlohy jen od 0 do 10, váš algoritmus by měl fungovat i pro libovolně velká čísla. Pokud bychom například postupně zadávali čísla 4, 1, 3, 2, 8, 9, 8, 10, váš program by měl postupně vypsat 4, 2.5, 3, 2.5, 3, 3.5, 4, 6.

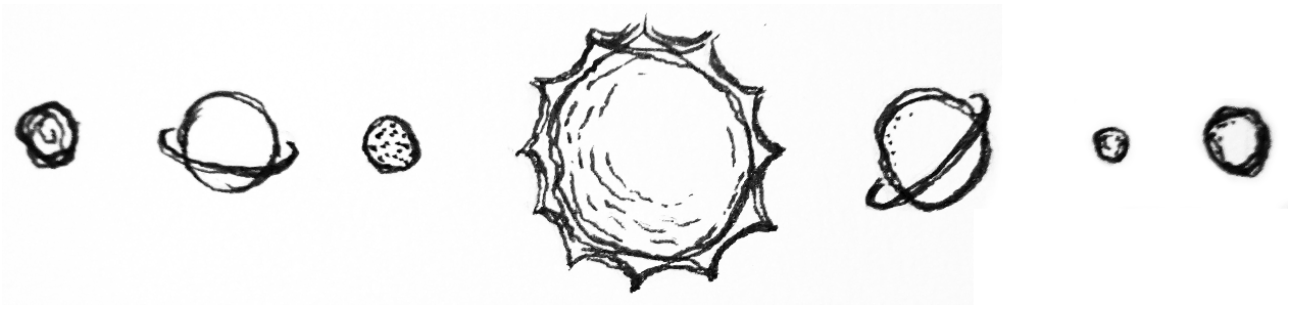
Jelikož KSI mohou v budoucnu řešit miliony lidí, je potřeba udělat algoritmus co nejefektivnější. Konkrétně požadujeme, aby složitost zpracování nového čísla byla logaritmická vzhledem k počtu již načtených čísel. Řečeno matematicky přesně, měla by existovat konstanta c taková, že počet operací, které váš program vykoná mezi načtením n. a $(n+1)$. čísla nebude nikdy větší než $c \cdot \log n$. Pokud se vám to takto rychle nepodaří, klidně nám napište i méně efektivní algoritmus. Pokud bude fungovat, určitě to oceníme.

[1] Jeden z organizátorů si myslí, že by mohla pomoct halda, ale neví, jak přesně ji použít.

Příklad 3: Astronomické problémy (10 bodů)

Prednedávnom astronómovia z celého sveta zoskupili informácie o planétach a poskytli ich verejnosti. Pár dní od zverejnenia však prišli na to, že ich systém nie je dosť pružný a nestíha odpovedať na množstvo požiadaviek. Po hlbšom skúmaní zistili príčinu: skoro všetci boli zvedaví na najmenšiu a najväčšiu planétu. To by nebolo samo o sebe veľkým problémom, lenže planét boli miliardy, požiadavky prichádzali rýchlo a z dátovej štruktúry, v ktorej boli planéty udržiavané, sa dali tieto údaje vyhľadať len s lineárnou časovou zložitosťou.

Navrhnite takú dátovú štruktúru, kde by sa maximum a minimum dalo nájsť v konštantnom čase. Keďže je pre astronómov rovnako dôležité rýchlo aktualizovať tieto informácie, vloženie a zmazanie prvku zo štruktúry by malo mať lepšiu časovú zložitosť ako je lineárna. Taktiež algoritmicky popíšte, ako by také vkladanie/zmazanie prvku vo vašej dátovej štruktúre vyzeralo.



Příklad 4: Vyhľadávanie slov (10 bodů)

Našli sme stránku textu z knihy a radi by sme zistili, či sa tam nachádza nejaké vopred určené slovo a označili ho. Slovo sa, prirodzene, môže v texte nachádzať viackrát, vtedy chceme, aby sa označili všetky. Keďže sme leniví informatici, našli sme aj starý program, ktorý toto spravil za nás. Ako sme si až neskôr všimli, tak program nefungoval správne a nenašiel všetky slová, ktoré sa na stránke nachádzali.

Vašou úlohou je preto analyzovať, kde program robí chybu (kde sa nachádzajú slová, ktoré nenájde a prečo), opraviť ho a napísať zjednodušene / efektívnejšie, aby sme sa v ňom všetci vyznali.

```
function tomato (cucumber, j)
  cloud := 1

  while ((cucumber <> ' ') and (cloud <= length (word)))
    if (cucumber = word [cloud])
      inc (j)
      inc(cloud)
      cucumber := char [j]
    endif

    if (cucumber <> word [cloud])
      j := j + lenght (word)
    endif
  endwhile

  if (whole_word_found)
    mark_it
  endif

  return parameters (' ', j)
endfunction
```

```
function wanna_find_a_word
  x := random (int)
  y := random (int) + x

  for a := 1 to 100
    x := (x + y) div 2
  endfor

  if ((x*x*x) > 0)
    wanna_find_a_word := true
  else
    if ((x + y) <= haha)
      wanna_find_a_word := true
    else
      wanna_find_a_word := false
    endif
  endif
endfunction


function haha
  fork := 1
  date := number_of_page

  for z := 1 to date
    fork := fork * z
  endfor

  if z < 10
    z := z * 2
  endif

  if z > 0
    haha := 0
  endif
endfunction


function sun
  sun := 0

  for hallo := 1 to number_of_chars_on_page
    sun := sun - 1
  endfor

  sun := abs_value (sun)
endfunction
```

```
function potato
  if (wanna_find_a_word = true)
    ignore_endline_newline
  endif

  tshirt := sun
  for i := 1 to tshirt
    apple := char [i]

    if (apple = ' ')
      inc (i)
      apple := char [i]
    endif

    tomato (apple, i)
  endfor
endfunction
```

Příklad 5: Sánkovačka (10 bodů)

Konečne nasnežilo a orgovia sa chcú ísť sánkovať. Zobrali si so sebou mapu a chcú nájsť ten najširší kopec, aby sa šmýkali čo najdlhšie. Mapa je veľká $n * n$ a je rozdelená na štvorcové políčka. Každé políčko obsahuje svoju nadmorskú výšku. Vrchol kopca je také políčko, pre ktoré platí: keď sa vzďaľujeme od vrcholu (po osi X alebo Y), tak klesá výška. Platí to pre každé políčko, ktoré je v maximálnej vzdialenosti k políčok od daného vrcholu vo všetkých smeroch, pričom k je prirodzené číslo , kroré vyjadruje šírku kopca. Inak povedané - ak máme vrchol so šírkou k a vyrežeme z mapy štvorec o strane $2k+1$ tak, že daný vrchol je v strede, tak platí, že so vzďaľovaním od vrcholu klesá nadmorská výška. Pomôžte orgom nájsť najširší kopec dostatočne efektívne voči šírke mapy tak, aby im zostal vôbec nejaký čas na sánkovanie.

A to je z třetí sady KSI vše. Přejeme ti hodně úspěchů při řešení, a když budeš mít jakékoliv otázky, neváhej se na nás obrátit e-mailem na adresu `ksi@fi.muni.cz` nebo v diskuzním fóru na webových stránkách.

Termín odevzdání 3. sady úloh KSI: 6. 1. 2014

`http://ksi.fi.muni.cz`