Riešenia kategórie B

B-I-1 Teploty

Na zisk 6 bodov stačilo naprogramovať najjednoduchšie možné riešenie: každý z $n - t + 1$ riadkov výstupu vyrobíme tak, že prejdeme všetky zodpovedajúce záznamy, nájdeme ich minimum, súčet a maximum a na záver priemer vypočítame ako súčet deleno t. Časová zložitosť takéhoto riešenia je teda približne priamo úmerná hodnote $t(n - t)$. (Matematicky to zapisujeme: časová zložitosť je $O(t(n - t))$.)

Takéto riešenie dokonca mohlo **trpezlivému** riešiteľovi stačiť aj na 10 bodov. Odhadnime si totiž, ako dlho budeme musieť čakať na výsledok pre najväčší vstup. Výstup pre vstup `5.txt` má 500 001 riadkov, a pre každý z nich potrebujeme spracovať 500 000 hodnôt. To znamená, že náš program spraví rádovo $500\,000^2 \approx 2.5 \cdot 10^{11}$ logických krokov.

Pri súčasných počítačoch je celkom dobrý odhad, že za sekundu stihne vykonať asi miliardu inštrukcií, čo zhruba zodpovedá 10^8 jednoduchým logickým krokom. Náš hrubý odhad teda hovorí, že na priemernom súčasnom počítači by malo vyriešenie vstupu `5.txt` byť zvládnuteľné rádovo za hodinu. (Od efektívnosti počítača, zvoleného programovacieho jazyka a šikovnosti implementácie závisí, či to bude 13 minút alebo 11 hodín. Ale tak či onak to bude dostatočne rýchlo.)

Trpezlivosti sa dalo pomôcť rôznymi vylepšeniami. Niektoré z nich fungujú len občas – napríklad riešenie, v ktorom minimum/maximum prerátame len vtedy, keď to treba (hodnota, ktorá práve opustila spracúvaný interval, je rovná dovtedajšiemu minimu/maximu). V tomto vzorovom riešení si však popíšeme niekoľko algoritmov, u ktorých máme istotu, že budú vždy efektívnejšie ako riešenie „hrubou silou“.

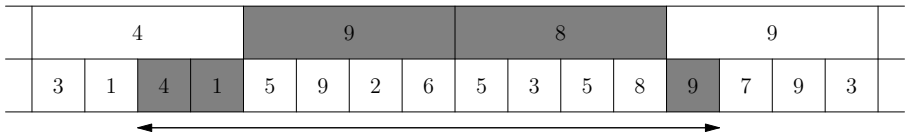
Zaručene lepšie riešenie pre priemer

Priemernú teplotu vieme ľahko počítať omnoho efektívnejšie. Predstavme si, že sme práve spočítali priemer teplôt $a_1, \ldots, a_t$. Teda poznáme súčet $s = a_1 + \cdots + a_t$. Ako vypočítame priemer pre nasledujúce meranie? Na ten potrebujeme poznať súčet $a_2 + \cdots + a_{t+1}$. Lenže ten vieme vypočítať v konštantnom čase: ako $(s - a_1 + a_{t+1})$.

Zaručene lepšie riešenie pre minimum a maximum

Ako zrýchliť výpočet minima a maxima? Treba nejako zabrániť tomu, aby sme zakaždým prechádzali všetkých t posledných meraní. Jednou možnou cestou je rozdeliť si vstup na bloky nejakej pevne zvolenej veľkosti. Každému bloku len raz spočítame minimum a maximum a zapamätáme si ich. A potom vždy, keď potrebujeme nájsť napr. maximum spomedzi nejakého úseku hodnôt, tak sa pozrieme na maximá blokov, ktoré sú celé vo vnútri nášho úseku. Po jednom prvku potrebujeme prejsť maximálne jeden blok na začiatku a jeden na konci.

Celé to je názorne ukázané na nasledujúcom obrázku:



V spodnom riadku sú jednotlivé namerané teploty. Toto pole sme si „nasekali“ na bloky dĺžky 4. Tie sú znázornené v hornom riadku – pre každý blok máme jedno políčko a v ňom maximum spomedzi hodnôt v danom bloku. Šípkami je znázornený interval, ktorého maximum nás zaujíma. Na zistenie tohoto maxima sa stačí pozrieť na farebne zvýraznené políčka.

Ako zvoliť dĺžku bloku? Ak si zvolíme primalé bloky, bude ich treba prejsť veľa. A ak si zvolíme bloky priveľké, bude na začiatku aj na konci úseku treba pozerať veľa jednotlivých prvkov. Dobrou voľbou je zlatá stredná cesta: hodnota $b = \lfloor \sqrt{t} \rfloor$. Úsek dĺžky t bude obsahovať najviac b celých blokov, a taktiež bude treba prezrieť najviac $b - 1$ prvkov na začiatku a najviac $b - 1$ prvkov na konci za posledným celým blokom. Dokopy nám teda na nájdenie maxima pre konkrétny úsek stačí pozrieť na menej ako $3\sqrt{t}$ hodnôt.

Dostávame teda riešenie s časovou zložitosťou $O(\sqrt{t}(n-t))$. Toto riešenie už aj najväčší zo zadaných vstupov vyrieši za pár sekúnd.

Listing programu:

```
var n, t, bl : longint; { pocet merani, dlzka useku, dlzka bloku }
    teploty, bloky_max, bloky_min : array of longint;

function max(x,y : longint) : longint; begin if x>y then max:=x else max:=y; end;
function min(x,y : longint) : longint; begin if x<y then min:=x else min:=y; end;

procedure nacitaj;
var i : longint;
    x : double;
begin
   read(n,t);
   bl := trunc(sqrt(t));
   setlength(teploty,n);
   setlength(bloky_max,(n div bl)+1);
   setlength(bloky_min,(n div bl)+1);
   for i:=0 to n-1 do begin
      read(x); teploty[i]:=round(x*100);
      if i mod bl=0 then begin
         bloky_max[i div bl] := teploty[i];
         bloky_min[i div bl] := teploty[i];
      end else begin
         bloky_max[i div bl] := max( bloky_max[i div bl], teploty[i] );
         bloky_min[i div bl] := min( bloky_min[i div bl], teploty[i] );
      end;
   end;
end;

procedure max_min_useku(zac, kon : longint; var odpoved_max, odpoved_min : longint);
begin
   odpoved_max := -10000; odpoved_min := 10000;
   { spracujeme prvky po najblizsi zaciatok bloku }
   while (zac<kon) and (zac mod bl > 0) do begin
      odpoved_max := max( odpoved_max, teploty[zac] );
      odpoved_min := min( odpoved_min, teploty[zac] );
      inc(zac);
   end;
   { spracuvame cele bloky }
   while (zac+bl <= kon) do begin
      odpoved_max := max( odpoved_max, bloky_max[zac div bl] );
      odpoved_min := min( odpoved_min, bloky_min[zac div bl] );
      inc(zac,bl);
   end;
   { spracujeme prvky po koniec useku }
   while (zac<kon) do begin
      odpoved_max := max( odpoved_max, teploty[zac] );
      odpoved_min := min( odpoved_min, teploty[zac] );
      inc(zac);
   end;
end;

procedure vyries;
var i,mx,mn : longint;
    sucet : int64;
begin
   sucet := 0; for i:=0 to t-2 do sucet:=sucet+teploty[i];
   for i:=0 to n-t do begin
      { spocitame odpovede pre usek zacinajuci na indexe i }
      sucet:=sucet+teploty[i+t-1];
      if i>0 then sucet:=sucet-teploty[i-1];
      max_min_useku(i,i+t,mx,mn);
      writeln( (mn/100):0:2, ' ', (sucet/(t*100)):0:4, ' ', (mx/100):0:2 );
   end;
end;

begin
   nacitaj;
   vyries;
end.
```

Iné ľahké vylepšenie

Taktiež sa na urýchlenie výpočtu minima/maxima dalo využiť to, že hodnoty na vstupe sú len z malého rozsahu. Keď všetky teploty vynásobíme číslom 100, dostaneme celé čísla od -8000 do 6000. A tie môžeme použiť ako indexy do poľa, v ktorom si budeme pamätať, ktorú hodnotu koľkokrát práve spracúvaný úsek obsahuje. Takto dosiahneme situáciu, kedy väčšinou vieme minimum/maximum povedať v konštantnom čase. Len občas (práve vtedy, keď dovtedajšie minimum/maximum prestane ležať v aktuálne spracúvanom úseku) potrebujeme prechádzať naším pomocným poľom a nájsť najbližšiu väčšiu/menšiu hodnotu, ktorá sa v aktuálnom úseku vyskytuje. (Všimnite si, že počet krokov, ktoré pri hľadaní nového minima/maxima spravíme, nezávisí od t.)

Nad rámec riešenia tejto úlohy

Pokročilé dátové štruktúry

Veľmi stručnú a zároveň efektívnu implementáciu vieme dosiahnuť v programovacích jazykoch, ktoré nám ponúkajú dátovú štruktúru „usporiadaná množina“. (Napr. `set` v C++, `TreeSet` v Jave.) Táto dátová štruktúra je zväčša implementovaná ako vyvažovaný binárny strom. Ak si v nej budeme pamätať t aktuálne spracúvaných meraní, budeme vedieť v čase $O(\log t)$ zistiť aktuálne minimum/maximum, a taktiež v čase $O(\log t)$ prejsť na nasledujúci úsek meraní.

Listing programu:

```
#include <cmath>
#include <iostream>
#include <set>
#include <vector>
using namespace std;

int get() { double x; cin >> x; return int(round(100*x)); }

int main() {
    int N, T; cin >> N >> T;
    vector<int> vstup;
    for (int n=0; n<N; ++n) vstup.push_back(get());
    multiset<int> usek;
    long long sucet = 0;
    for (int t=0; t<T-1; ++t) { usek.insert(vstup[t]); sucet += vstup[t]; }
    for (int t=T-1; t<N; ++t) {
        usek.insert(vstup[t]); sucet += vstup[t];
        if (t>=T) { usek.erase(usek.find(vstup[t-T])); sucet -= vstup[t-T]; }
        cout << *usek.begin()/100. << " " << sucet/(100.*T) << " " << *usek.rbegin()/100. << "\n";
    }
}
```

Rovnako efektívne riešenie vieme dostať pomocou intervalového stromu. Záujemcom odporúčame prečítať si riešenie úlohy A-I-1 z 25. ročníka OI. Ide vlastne o vylepšenie vyššie popísaného riešenia s blokmi: pri intervalovom strome vstup rozdelíme na $n/2$ blokov veľkosti 2, tie pospájame do $n/4$ väčších blokov veľkosti 4, tie do blokov veľkosti 8, a tak ďalej.

Optimálne riešenia

Existuje dokonca aj riešenie, ktoré má aj v najhoršom možnom prípade časovú zložitosť $O(n)$ – každý možný vstup teda spracuje v čase lineárnom od jeho veľkosti. Detaily tohoto riešenia nebudeme uvádzať. Prípadným záujemcom poradíme, že je myšlienkovo príbuzné riešeniu z časti „Iné ľahké vylepšenie“. Namiesto poľa so všetkými hodnotami si ale treba vo vhodnej dátovej štruktúre (deque, teda tzv. obojsmerná fronta) pamätať len tie hodnoty, ktoré majú teoretickú šancu niekedy sa stať maximom úseku.

Iné riešenie s celkovou časovou zložitosťou $O(n)$ vyzerá nasledovne: rozdelíme si vstup na bloky dĺžky t a v každom bloku pre každý prefix aj pre každý sufix spočítame jeho minimum/maximum. Pomocou týchto údajov vieme teraz minimum ľubovoľného úseku dĺžky t určiť v konštantnom čase.

B-I-2 Sudoku

Riešenie tejto úlohy neobsahuje žiadne hlboké myšlienky. Na získanie bodov stačilo správne pochopiť, zovšeobecniť a implementovať pravidlá uvedené v zadaní. Popíšeme si, ako to malo celé vyzerať.

Prvé pravidlo je zbytočné

To samozrejme nie je úplne pravda. Implementuje sa najľahšie a pár bodov sa zaň dalo získať. Ak ale plánujeme získať všetkých 10 bodov, nemá zmysel strácať čas implementovaním prvého pravidla. Prečo? Preto, že vždy, keď niečo vieme odvodiť prvým pravidlom, vieme to isté odvodiť aj druhým pravidlom – ak sú v danom riadku/stĺpci/štvorci plné všetky políčka okrem jedného, keď sa na dotyčné prázdne políčko pozrieme, budeme už mať len jednu možnosť.

Implementácia druhého a tretieho pravidla

Tieto pravidlá sa síce dajú implementovať každé zvlášť, avšak môžeme si ich obe zjednodušiť tým, že si budeme pamätať vhodné pomocné údaje. Presnejšie, pre každé políčko si budeme pamätať, ktoré cifry sa na ňom ešte môžu vyskytnúť bez toho, aby vznikol priamy konflikt. Na toto je najjednoduchšie použiť pole boolovských premenných – presnejšie teda trojrozmerné pole: pre každý riadok r, stĺpec s a cifru c si budeme pamätať, či môžeme cifru c umiestniť na pozíciu (r, s).

(Pre už zaplnené políčka nastavíme všetkým cifrám, vrátane tej na ňom použitej, hodnotu `false`. Takto ich odlíšime od políčok, kde už ostáva len jedna možnosť, ale ešte sme ju tam neumiestnili.)

Hodnoty v poli je ľahké udržiavať: vždy, keď umiestnime nejakú cifru, stačí prejsť jej riadok, stĺpec a štvorec 3×3 a „vyškrtať ju“ – teda zakázať jej výskyty na všetkých spomínaných políčkach.

Ako bude vyzerať inicializácia nášho poľa? To bude tiež jednoduché: stačí na začiatku nastaviť, že každá cifra môže byť všade, a potom postupne po jednej popridávať cifry zo vstupu.

Druhé pravidlo teraz môžeme implementovať nasledovne: zaradom prejdeme všetky políčka, pre každé z nich skontrolujeme, či náhodou nemá práve jednu možnú cifru, a ak áno, tak ju tam umiestnime. No a takisto pri treťom pravidle sa stačí pozerať do nášho pomocného poľa a skontrolovať, či má daná cifra v danom riadku/stĺpci/štvorci už len jednu možnú pozíciu.

Listing programu:

```
program sudoku;

var moze : array[1..9,1..9,1..9] of boolean; { moze byt na pozicii A,B cifra C? }
    riesenie : array[1..9,1..9] of longint;

procedure inicializuj;
var i,j,k : longint;
begin
   for i:=1 to 9 do for j:=1 to 9 do for k:=1 to 9 do moze[i,j,k]:=true;
   for i:=1 to 9 do for j:=1 to 9 do riesenie[i,j]:=0;
end;

procedure umiestni_cifru(r,s,c : longint);
var i,j,br,bs : longint;
begin
   riesenie[r,s]:=c;
   for i:=1 to 9 do begin moze[i,s,c]:=false; moze[r,i,c]:=false; moze[r,s,i]:=false; end;
   br:=(r-1) div 3; bs:=(s-1) div 3;
   for i:=1 to 3 do for j:=1 to 3 do moze[3*br+i,3*bs+j,c]:=false;
end;

procedure nacitaj;
var i,j : longint;
    s : string;
begin
   for i:=1 to 9 do begin
      readln(s);
      for j:=1 to 9 do if s[j]<>'.' then umiestni_cifru(i,j,ord(s[j])-48);
   end;
end;

procedure vypis;
```

```
var i,j : longint;
begin
   for i:=1 to 9 do begin
      for j:=1 to 9 do write(riesenie[i,j]);
      writeln;
   end;
end;

function hotovo : boolean;
var i,j : longint;
begin
   hotovo := true;
   for i:=1 to 9 do for j:=1 to 9 do if riesenie[i,j]=0 then hotovo := false;
end;

procedure druhe_pravidlo;
var r,s,c,kolko,co : longint;
begin
   for r:=1 to 9 do for s:=1 to 9 do begin
      kolko := 0;
      for c:=1 to 9 do if moze[r,s,c] then begin inc(kolko); co:=c; end;
      if kolko=1 then umiestni_cifru(r,s,co);
   end;
end;

procedure tretie_pravidlo;
var rr,ss,r,s,c,kolko,kder,kdes : longint;
begin
   for c:=1 to 9 do begin
      { riadky }
      for r:=1 to 9 do begin
         kolko := 0;
         for s:=1 to 9 do if moze[r,s,c] then begin inc(kolko); kdes:=s; end;
         if kolko=1 then umiestni_cifru(r,kdes,c);
      end;
      { stlpce }
      for s:=1 to 9 do begin
         kolko := 0;
         for r:=1 to 9 do if moze[r,s,c] then begin inc(kolko); kder:=r; end;
         if kolko=1 then umiestni_cifru(kder,s,c);
      end;
      { stvorce 3x3 }
      for rr:=0 to 2 do for ss:=0 to 2 do begin
         kolko := 0;
         for r:=1 to 3 do for s:=1 to 3 do begin
            if moze[3*rr+r,3*ss+s,c] then begin inc(kolko); kder:=3*rr+r; kdes:=3*ss+s; end;
         end;
         if kolko=1 then umiestni_cifru(kder,kdes,c);
      end;
   end;
end;

var t : longint;

begin
   readln(t);
   while t>0 do begin
      dec(t);
      inicializuj;
      nacitaj;
      while not hotovo do begin druhe_pravidlo; tretie_pravidlo; end;
      vypis;
   end;
end.
```

B-I-3 Uhádni deň

Na úvod riešenia si treba uvedomiť, že sa zaobídeme aj bez komplikovaných otázok ako táto: „Mal si tento rok narodeniny v utorok?“ Namiesto nej totiž môžeme položiť rovnocennú otázku: „Máš narodeniny v jeden z nasledovných dní: 3, 10, 17, 24, 31?“ Každú otázku vieme preformulovať na ekvivalentnú otázku o číslach dní. Stačí nám teda riešiť jednoduchšiu úlohu: na čo najmenej otázok uhádnuť číslo od 1 do 31.

Podúloha a)

Dobrá stratégia pre Aničku (neskôr si dokonca ukážeme, že najlepšia možná) je pýtať sa tak, aby sa jej po každej otázke približne na polovicu zmenšil počet možností, ktoré ešte pripadajú do úvahy.

Začať môže napríklad otázkou: „Je deň tvojich narodenín číslo od 1 do 15?“

Ak dostane odpoveď „áno“, zmenšil sa počet možností z 31 na 15. A ak dostane odpoveď „nie“, prichádzajú už do úvahy len čísla 16 až 31, čo je 16 možností.

A rovnako môže Anička pokračovať ďalej – vždy rozdelí čísla, ktoré ešte prichádzajú do úvahy, na dve rovnako veľké kôpky a opýta sa, či je hľadané číslo na prvej kôpke. (Ak má nepárny počet možností, prvá kôpka bude mať o číslo menej ako druhá.) Ak napríklad začala vyššie uvedenou otázkou a dostala odpoveď „nie“, v druhom kole môže položiť otázku „Je deň tvojich narodenín číslo od 16 do 23?“.

Ako veľa otázok bude Aničke stačiť pri tomto postupe? Po prvej otázke jej ostávalo najviac 16 možností. Po druhej tento počet klesne na 8, po tretej na 4, po štvrtej na 2 a po prípadnej piatej otázke (ak ju ešte treba) už určite ostala len jedna možnosť – tá správna.

Ak teda Miško dovolí Aničke položiť päť otázok, má Anička pri použití našej stratégie istotu, že jeho deň narodenín uhádne.

Podúloha b)

Napriek tomu, že je Miškova situácia ťažšia, aj jemu stačí položiť len päť otázok. Ako to má spraviť? Jednoducho sa zakaždým šikovne opýta naraz všetky otázky, ktoré by sa v danú chvíľu mohla opýtať Anička.

Prvú otázku položí Miško rovnakú ako Anička.

Pozrime sa teraz na Aničkinu druhú otázku. Tá mohla vyzerať dvomi spôsobmi: ak na prvú otázku dostala odpoveď „áno“, opýtala by sa druhú otázku „Je deň tvojich narodenín číslo od 1 do 7?“, ak nie, tak otázku „Je deň tvojich narodenín číslo od 16 do 23?“.

Miškova druhá otázka teraz môže vyzerať nasledovne: „Je deň tvojich narodenín medzi číslami (1 až 7) a (16 až 23)?“

A rovnako pokračujeme aj ďalej. Tretia Miškova otázka by vyzerala takto: „Je deň tvojich narodenín medzi číslami (1 až 3), (8 až 11), (16 až 19) a (24 až 27)?“

Z odpovedí na všetkých päť otázok vie Miško zakaždým jednoznačne zostrojiť hľadané číslo.

Iný pohľad na toto isté riešenie:

Číslo x, ktoré sa Miško snaží uhádnuť, vieme zapísať v dvojkovej sústave. Keďže je z rozsahu od 1 do 31, určite nám na to stačí päť bitov. (Kratšie čísla si doplníme zľava nulami, teda napr. 6 zapíšeme ako 00110, 18 zapíšeme ako 10010.)

No a Miškovu stratégiu, ktorú sme vyššie popísali, môžeme teraz formulovať jednoduchšie: každou otázkou sa dozvieme jeden bit hľadaného čísla.

Napríklad otázku „Je deň tvojich narodenín medzi číslami (1 až 7) a (16 až 23)?“ môžeme preformulovať takto: „Má číslo x druhý bit zľava rovný nule?“

Podúloha c)

Teraz potrebujeme dokázať, že Miškovi nemôže stačiť menej ako päť otázok. Dôkaz vôbec nebude závisieť na tom, ako vyzerajú Miškove otázky – využijeme len to, že na primálo otázok dostaneme primálo odpovedí.

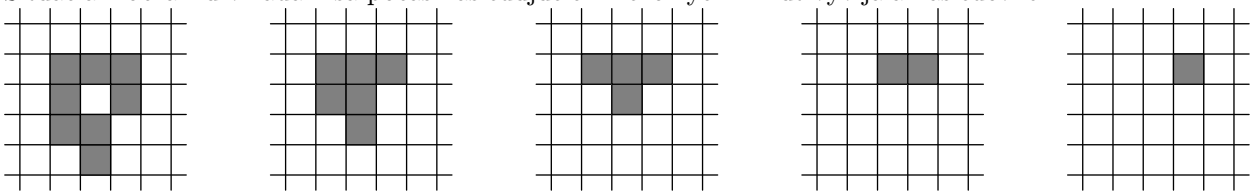
Predstavme si, že Miško napísal na papier len štyri otázky. Anička teraz zobrala jeho papier a ide k nim napísať svoje odpovede. V tejto chvíli je len $2^4 = 16$ možností, aké odpovede Anička napíše. Lenže dní, ktoré pripadajú do úvahy, je až 31, no a $31 > 16$. Preto určite (podľa Dirichletovho princípu) budú existovať dva rôzne dni, pre ktoré Anička napíše na Miškov papier tú istú postupnosť odpovedí. No a práve tým vzniká problém – ak sa Anička narodila v jeden z týchto dní, tak Miškovi ostane viac ako jedna možnosť. Miško teda nebude mať istotu, že vie, kedy sa Anička narodila.

(Rovnako by dôkaz fungoval, ak by Miško položil menej ako štyri otázky. Rozmyslite si tiež, že z tohto dôkazu vyplýva aj to, že aj Anička na uhádnutie Miškovych narodenín potrebuje aspoň päť otázok.)

B-I-4 Čierne štvorce

Podúloha a)

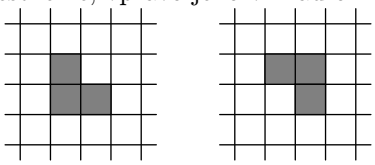
Situácia z obrázku v zadaní sa počas nasledujúcich niekoľkých minút vyvíjala nasledovne:



Po piatich minútach už boli všetky štvorce biele.

Podúloha b)

Po príklad takého rozmiestnenia netreba chodiť ďaleko – najmenšie má len tri štvorce!
Na obrázku vľavo je pôvodné rozmiestnenie, vpravo jeho vzhľad o minútu neskôr.



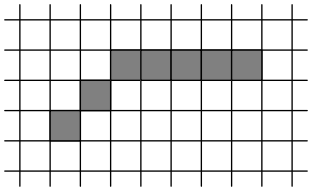
Podúloha c)

Po chvíli experimentovania ľahko odhalíme viaceré také rozmiestnenia štvorcov. Asi najjednoduchším je jednoducho 7 čiernych štvorcov v rade vedľa seba.

Podúloha d)

Dá sa to pre každé n a k. Z predchádzajúcej podúlohy už vieme, že ak umiestnime k štvorcov do radu, práve po k minútach dostaneme všetky štvorce biele. Následne stačí správne pridať ďalších $n-k$ štvorcov tak, aby to celé držalo pokope a aby ich pridanie neovplyvnilo tých prvých k. Na to ich napríklad stačí vhodne diagonálne pripojiť k prvým k štvorcom. Potom hneď po prvej minúte všetky zmiznú a ostane len $k-1$ z pôvodných k štvorcov.

Na nasledujúcom obrázku je príklad pre $n = 7$ a $k = 5$.



DVADSIATY SIEDMY ROČNÍK OLYMPIÁDY V INFORMATIKE

Autori úloh: Vladimír Boža, Michal Forišek, Peter Fulla, Ján Katrenič
Recenzent: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2011