Korešpondenčný seminár z programovania
XX. ročník, 2002/2003
Katedra vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporuje MICROSTEP-MIS spol. s r.o.

Vzorové riešenia 1. kola letnej časti

Milí riešitelia!
Ako už možno viete, v piatok 16. mája 2003 organizovali KSPáci ďalší ročník celosvetovej (nielen) stredoškolskej programátorskej súťaže IPSC. Tento rok sa prihlásilo vyše 600 tímov z celého sveta a nás teší, že sa slovenskí riešitelia v tejto konkurencií nestratili. Zo slovenských tímov obstáli najlepšie tím J2 (15. miesto celkovo) z matfyzu a IdeaZ (16.) z UPJŠ v Košiciach. Najlepší náš stredoškolský tím bol 4Dimensional GJH team, ktorý skončil medzi stredoškolskými tímami na 20. mieste. Všetkým zúčastneným blahoželáme k dosiahnutým výsledkom. A už ostáva len dodať, že najlepšou cestou k úspechu v takejto silnej konkurencií je čítať vzoráky KSP. Hor sa na to.

Gn fvsen wr gh yra gnx, nol fgr fn arahqvyv.

KSPáci

1. O snaživom Tomášovi

opravoval Dávidko
(max. 15 bodov)

Išlo o veľmi populárny príklad, ktorý riešila hŕba z vás. Vaše riešenia možno rozdeliť v zásade do niekoľkých skupín, podľa čoho som udeľoval aj body.

- časová žložitosť $O(N \log N)$, rovnako ako vzorák – max. **15 bodov**
- časová zložitosť $O(N^2)$ – max. **12 bodov**
- riešenia ktoré predpokladali, že časy sú celé čísla v malom rozsahu – max. **10 bodov**
- exponenciálna a horšia časová zložitosť, backtracking a podobné – max. **8 bodov**
- nefuknčné riešenia – max. **3 body**

Bod som stŕhal za neuvedenie (správnej) časovej zložitosti.

Vzorové riešenie: Úloha budeme riešiť metódou **dynamického programovania.** Najprv brigády utriedime vzostupne podľa času kedy končia. Odteraz, keď povieme i-ta brigáda, má sa na mysli i-ta v tomto utriedenom poradí. Z technickým príčin pridajme ešte nultú brigádu s nulovým zárobkom, ktorá končí skôr alebo rovnako skoro, ako začína ľubovoľná iná brigáda.

Teraz nastupuje veľkolepá paradigma dynamického programovania: „Rátaj hodnoty, ktoré na sú na prvý pohľad zbytočné.“ My sa teda budeme pre každé $i = 0, 1, \ldots, N$ snažiť vypočítať, koľko najviac peňazí, môže náš snaživý Tomáš zarobiť, keby šiel len na (niektoré) brigády spomedzi prvých i. Označme tieto hodnoty S_i pre $i = 0, 1, \ldots, N$. Hodnota, o ktorú nám v skutočnosti ide je len S_N. Je jasné, že na prvých nula brigádach nič nezarobí, a teda $S_0 = 0$. Prepokladajme, že máme vyrátané hodnoty $S_0, S_1, \ldots, S_{i-1}$ a chceme vyrátať S_i. Ako na to?

Snaživý Tomáš má dve pricipiálne možnosti: Buď pôjde na i-tu brigádu alebo nepôjde. Ak nepôjde, tak najviac zarobí práve toľko, ako keby i-ta brigáda ani neexistovala, teda presne S_{i-1}. Ak na ňu pôjde, tak nemôže stihnúť posledných niekoľko (možno nula) brigád. Presnejšie, ak ℓ $(0 \leq \ell \leq i-1)$ je posledná taká brigáda, že končí skôr alebo rovnako ako i-ta začína, tak určite nestihne brigády $\ell+1, \ell+2, \ldots, i-1$. Ostatné brigády ale stihnúť môže, a preto pri tejto možnosti Tomáš zarobí $S_\ell + c_i$, kde c_i je suma, ktorú zarobí na i-tej brigáde.

(Treba si uvedomiť, že také ℓ existuje a je jediné. Tohto sme dosiahli práve pridaním nultej brigády.) Ak teda je teda Tomáš taký snaživý, tak si vyberie tú z týchto dvoch možností, na ktorej zarobí najviac a teda zarobí presne $S_i = \max(S_{i-1}, S_\ell + c_i)$.

Implementácia je rovnako priamočiara ako výklad riešenia. Budeme mať jedno pole pre brigády a jedno pole pre hodnoty S. Načítame brigády a utriedime ich podľa konca nejakým „rýchlym“ triediacim algoritmom napr. Quicksortom. Vyrobíme nultú brigádu. Položíme $S_0 := 0$ a následne v cykle od 1 po N rátame hodnoty S_i. Pre každé i potrebujeme nájsť príslušnú brigádu ℓ. Tu sa nesmieme zlákať predstavou prehľadávať všetky brigády od 0 po $i-1$. Ak si uvedomíme, že sú utriedené podľa konca, tak s výhodou použijeme binárne vyhľadávanie.

Pamäťová zložitosť je zjavne $O(N)$. Na načítanie potrebujeme čas $O(N)$, na utriedenie $O(N \log N)$. Na vyrátanie hodnoty nejakej jednej hodnoty S_i je potrebné jedno binárne vyhľadávanie, ktoré má časovú zložitosť $O(\log i) = O(\log N)$. Výsledná časová zložitosť celého algoritmu bude teda $O(N \log N)$.

Listing programu:

```
Program O_snazivom_Tomasovi;

type brigada = record s,t,c:real; end; { zaciatok, koniec a zarobena suma }
var a:array[0..1000] of brigada; { zoznam brigad }
{ maximalna suma, ktoru Tomas je schopny zarobit za prvych i brigad }
    S:array[0..1000] of real;
    N,i,l,r,m:integer;

{ triedenie quicksortom podla konca brigady }
procedure qsort(l,r:integer);
var i,j:integer;
    p:real;
    b:brigada;
begin
     if r<=l then exit; { neni co triedit }
     i:=l; j:=r;
     p:=a[(i+j) div 2].t; { pivot }
     repeat
           while a[i].t<p do inc(i);
           while a[j].t>p do dec(j);
           if i<=j then begin
                b:=a[i]; a[i]:=a[j]; a[j]:=b;
                inc(i); dec(j);
           end;
     until i>j;
     qsort(l,j); qsort(i,r);
end;


begin
     assign(input,'prikl1.in'); reset(input);
     { nacitame brigady }
     readln(N); for i:=1 to N do with a[i] do readln(s,t,c);
     { utriedime ich podla konca }
     qsort(1,N);
     { nulta brigada, ktora skonci skor ako hocijaka ina zacne }
     a[0].t:=a[1].s;
     for i:=1 to N do if a[i].s < a[0].s then a[0].s:=a[i].s;
     S[0]:=0;
     { dynamicke programovanie }
     for i:=1 to N do begin
```

```
          { na i brigadach mozme zarobit aspon tolko ako na i-1}
          S[i]:=S[i-1];
          { binarne vyhladavanie }
          l:=0; r:=i-1;
          while r>l do begin
              m:=(l+r) div 2;
              if a[m].t <= a[i].s
              then if a[m+1].t <= a[i].s then l:=m+1
                                        else begin r:=m; l:=m; break; end
              else r:=m-1;
          end;
          { l-ta brigada, je posledna taka, co skonci skor ako i-ta zacina }
          if S[l] + a[i].c > S[i] then S[i]:=S[l] + a[i].c;
     end;
     writeln('Tomas moze zarobit najviac ',S[N]:0:2,'.');
end.
```

opravoval Goober
(max. 15 bodov)

2. O dovolenke

Tento príklad ma veľmi potešil – okrem toho, že takmer všetky vaše riešenia boli správne, písali ste aj pekné komentáre. Skrátka, radosť opravovať. Tak aj hodnotenie bolo o dosť benevolentnejšie ako obyčajne. Ale dosť bolo rečí, pozrime sa na vzorové riešenie.

Väčšina z vás na základný trik prišla – že stačí uvažovať len také lety helikoptéry, pri ktorých sa najprv stúpa, potom ide horizontálne a na konci klesá. Navyše budeme predpokladať, že počas letu môže helikoptéra klesať (*len* klesať) aj cez kocky vyplnené zemou. Toto si môžeme dovoliť preto, že ak nájdeme cestu, kde helikoptéra klesá cez kocky vyplnené zemou, vieme, že existuje rovnako drahá cesta, kde si helikoptéra „nechá všetko klesanie nakoniec“ a že táto cesta je už v poriadku. Pre každý let L helikoptéry si označme $H(L)$ maximálnu výšku, v ktorej sa nachádzala.

Zadanie si teraz prevedieme na graf s $M \times N$ vrcholmi, ktoré budú zodpovedať jednotlivým políčkam výškovej mapy (písmenká A, B budú odteraz označovať aj príslušné vrcholy nášho grafu). Pre každý vrchol X budeme mať päticu čísel $(x_X, y_X, z_X, P_X, Q_X)$, kde x_X, y_X sú jeho súradnice, z_X je výška, do ktorej siaha zem nad miestom $[x_X, y_X]$. Nech L je najlacnejší let z kocky A do kocky $[x_X, y_X, z_B]$. (T.j. do kocky so súradnicami $[x_X, y_X]$ a výškou, v ktorej chceme skončiť. Uvedomte si, že táto kocka môže byť tvorená zemou.) Potom P_X bude cena tohoto letu a Q_X bude $H(L)$. Hrany grafu budú orientované a povedú vždy medzi dvojicami stranou susediacich políčok. Ešte ich ohodnotíme pomocou veľmi prefíkaného vzťahu (U, V sú dva susedné vrcholy, a, b, c sú ceny jednotlivých pohybov podľa zadania): $d(U \to V) = c + (a + b) \times \max\{(z_V + 1) - Q_v, 0\}$. Tento vzorček hovorí, koľko stojí upravenie cesty, ktorá pôvodne viedla z A do U na cestu, ktorá povedie z A do U a potom hneď od V. Prvý člen zodpovedá horizontálnemu presunu, druhý zase stúpaniu a následnému (po horizontálnom posune) klesaniu v prípade, že prelietavame na vrchol s vyššou úrovňou zeme.

Rozdeľme teraz vrcholy do dvoch kategórii – D, čiže definitívne, a O, čiže ostatné. Vrcholy X z množiny D sa budú od ostatných líšiť tým, že budeme poznať hodnoty P_X aj Q_X. Množinu D budeme postupne rozširovať, pokiaľ sa v nej nevyskytne aj vrchol B. Na začiatku ešte nevieme takmer nič, tak všetkým vrcholom nastavíme $P_X = \infty$. Výnimku tvorí vrchol A, pre ktorý poznáme presnú hodnotu P_X aj Q_X (ako čitateľ iste ľahko nahliadne). Napriek tomu zvolíme počiatočné D prázdne. V každom „kroku“ spravíme nasledovné – vyberieme z O ten vrchol T, ktorý má najmenšie P_T a presunieme ho do D. Súčasne s tým si pozrieme všetkých jeho susedov Y z O (ktorí sú nanajvýš štyria) a upravíme im P_Y, Q_Y nasledovne – $P_Y = min\{P_Y, P_T + d(T \to Y)\}$, a ak v minime nastal druhý prípad, položíme $Q_Y = max\{z_Y + 1, Q_Y\}$. Tieto dva kroky zabezpečujú, že P_Y bude najlepšia cena, za akú sme schopní sa dostať z A do Y iba cez vrcholy z D.

Treba teraz dokázať, že tento postup bude fungovať. Na to použijeme nasledovný invariant – „Pre všetky X platí, že P_X je dĺžka najlacnejšieho letu L z A do X, ktorý ide iba cez vrcholy z D a navyše $Q_X = H(L)$.“ Na začiatku je tento invariant triviálne splnený, teda stačí overiť, či ho zachováva aj každý „krok“. Jediný vrchol, ktorý by mohol spôsobiť problémy, je vrchol T zaraďovaný v danom kroku do množiny D. Zjavne $P_T \geq P_X$ pre každé $X \in D$, čiže prvá časť (a tým aj druhá) je splnená pre všetky vrcholy z D. Susedov vrcholu T sme vybavili počas „kroku“ a teda sme hotoví, lebo žiaden iný vrchol nie je pridávaním T do D ovplyvnený. Teda po skončení celého postupu bude P_B tá hľadaná dĺžka najlacnejšieho letu z A do B.

Ako ste si iste všimli, tento postup sa nápadne ponáša na Dijkstrov algoritmus na hľadanie najkratšej cesty, a presne tak sa aj bude implementovať. Čo vlastne potrebujeme? Treba mať nejakú štruktúru, ktorá bude vedieť rýchlo realizovať dve operácie – nájdenie minima a úpravu (zníženie) hodnoty prvku. Na toto sa priamo ponúka obyčajná halda (v tejto súvislosti nazývaná aj *prioritný front*). Hľadanie a následné odstránenie minima je priamočiare „prebublávanie nadol“, znižovanie hodnoty prvku je „prebublávanie nahor“. Celková časová zložitosť teda bude $O(MN \log MN)$ a pamäťová bude lineárna od veľkosti vstupu (t.j. $O(MN)$).

Aby sme vedeli aj vypísať nejakú tú najlacnejšiu cestu, zíde sa nám pamätať si pre každý vrchol, odkiaľ sme doň nejakou najlepšou cestou prišli.

Nuž a hodnotenie? To bolo vcelku jednoduché – za riešenie v zložitosti $O(MN \log MN)$ bolo plných **15 bodov**. Za $O((MN)^2)$ ste mohli dostať **12 bodov**, za $O(MNK)$ bolo **10 bodov**.[1] No a horšie riešenia dostávali **8 bodov**. Ako vždy, nejaký ten bod-dva mohli byť strhnuté za chyby v popise.

Listing programu:

```
#include <stdio.h>

#define MAXN 10
#define MAXM 10
#define INFTY 100000

typedef struct point { int x, y, z, P, Q, D, kde; struct point *s; } point;
point points[MAXN+1][MAXM+1];
point *halda[MAXN*MAXM+1];
int haldas;
int a,b,c,m,n,k,xa,ya,za,xb,yb,zb;

int max(int x, int y) { return x>y?x:y; };

int swap(int i, int j) {
  point *t; int k;
  t=halda[i]; halda[i]=halda[j]; halda[j]=t;
  k=halda[i]->kde; halda[i]->kde=halda[j]->kde; halda[j]->kde=k;
}

void bublihore(int kde) {
  int i=kde;
  while(i>1) if (halda[i]->P < halda[i/2]->P) { swap(i,i/2); i=i/2; } else break;
}

int bublidole(int start) {
  int j,i=start;
```

[1] Zamyslite sa, prečo je $O(MNK)$ horšie ako $O((MN)^2)$. Hint: Uvedomte si, ako súvisia čísla M, N, K s dĺžkou vstupu.

```
  while(2*i<=haldas) {
    j=2*i;
    if (j+1<=haldas && halda[j+1]->P < halda[j]->P) j++;
    if (halda[i]->P <= halda[j]->P) break;
    swap(i,j); i=j;
  }
}

point *pop() { point *t=halda[1]; swap(1,haldas--); bublidole(1); return t; }

void update(int x1, int y1, int x2, int y2) {
  point *t1, *t2;
  int hrana;

  if (x2<1 || x2>n || y2<1 || y2>m) return;
  t1=&points[x1][y1];
  t2=&points[x2][y2];
  if (t2->D) return; /* Definitivny sa uz neupravuje */
  if (t2->z>=k) return; /* Taky kopec nepreletime */

  hrana=c+(a+b)*max((t2->z+1)-(t1->Q),0);
  if (t2->P > (t1->P)+hrana) {
    t2->P=(t1->P)+hrana;
    t2->Q=max(t1->Q,t2->z+1);
    t2->s=t1;
    bublihore(t2->kde);
  }
}

void displaybokom(point *t) {
  if (!t->s) return;
  displaybokom(t->s);
  printf("[bokom] (%d %d %d)\n",t->x,t->y,t->Q);
}

int main() {
  int i,j;
  point *t;

  scanf("%d %d %d",&n,&m,&k); scanf("%d %d %d",&a,&b,&c);
  haldas=0;
  for (i=1;i<=n;i++) for (j=1;j<=m;j++) {
    points[i][j].x=i; points[i][j].y=j; scanf("%d",&points[i][j].z);
    points[i][j].P=INFTY; points[i][j].Q=INFTY;
    points[i][j].D=0;        /* Este nie je nikto definitivny... */
    points[i][j].s=NULL;
    halda[++haldas]=&points[i][j];
    points[i][j].kde=haldas;
  }
  scanf("%d %d %d",&xa,&ya,&za); scanf("%d %d %d",&xb,&yb,&zb);

  if (za>=zb) { points[xa][ya].Q=za; points[xa][ya].P=b*(za-zb); }
  else { points[xa][ya].Q=zb; points[xa][ya].P=a*(zb-za); }

  /* Vybudujeme haldu */
  for (i=haldas/2;i>=1;i--) bublidole(i);

  while(haldas>=1) {
    t=pop(); t->D=1;
    update(t->x,t->y,t->x+1,t->y); update(t->x,t->y,t->x-1,t->y);
```

```
    update(t->x,t->y,t->x,t->y+1); update(t->x,t->y,t->x,t->y-1);
  }

  if (points[xb][yb].P==INFTY) printf("Cesta neexistuje\n"); else {
    printf("Celkova cena: %d\n",points[xb][yb].P);
    printf("[start] (%d %d %d)\n",xa,ya,za);
    t=&points[xb][yb]; i=za+1;
    while(i<=t->Q) printf("[nahor] (%d %d %d)\n",xa,ya,i++);
    displaybokom(t);
    t=&points[xb][yb]; i=t->Q-1;
    while(i>=zb) printf("[nadol] (%d %d %d)\n",xb,yb,i--);
  }
  return 0;
}
```

3. O obchodnom cestujucom

opravoval Poko
(max. 15 bodov)

Čo na úvod? Asi nič. Ale predsa. V zadaní pri vzorovom príklade sa vyskytla chyba. Totiž z položených otázok jednoznačne nevyplývalo, o aký mnohouholník ide, lebo do úvahy prichádzali dva. Za túto chybu sa vám ospravedlňujeme. Dúfam, že to niektorých neodradilo od riešenia tejto úlohy. Naopak pochvalu si zaslúžia všetci (dvaja či traja), ktorí si to všimli. Poďme už ale k vzorovému riešeniu.

Ostrovy si na chvíľu predstavme ako graf. Okružná trasa pána Hamiltona je vlastne kružnica prechádzajúca cez každý vrchol práve raz. Takáto kružnica sa zvykne nazývať hamiltonovská. Nájsť najkratšiu hamiltonovskú kružnicu vo všeobecnom grafe je NP-úplný problém. To okrem iného znamená, že nie je známy algoritmus, ktorý by tento problém riešil v polynomiálnom čase.

Čo ale v našom dosť špeciálnom prípade? Podľa zadania ostrovy vlastne predstavujú body v rovine a deduško nám prezrádza skutočné priame vzdialenosti medzi týmito bodmi. Navyše vieme, že tieto body ležia vo vrcholoch nejakého konvexného mnohouholníka. Ukážeme si, že potom najkratšia okružná cesta (alebo ak chcete hamiltonovská kružnica) bude práve tá, ktorá prechádza po obvode tohoto konvexného mnohouholníka. Potom nám už stačí zistiť, v akom poradí ležia body po obvode onoho mnohouholníka. A vypočítať celkovú dĺžku už nebude žiaden problém (v najhoršom sa poradíme s deduškom :-)

Uvažujme ďalej už iba body v rovine namiesto ostrovov. Hľadáme uzavretú krivku, ktorá prejde každým bodom práve raz a bude mať minimálnu dĺžku. Aby tak bolo, každé dva vrcholy, ako idú na tejto krivke po rade, musíme spojiť úsečkou. Každá iná krivka spájajúca dané dva vrcholy je totiž dlhšia. Pojem krivka teda môžeme nahradiť pojmom lomená čiara. Nech úsečka $\overline{AB}$ patrí hľadanej lomenej čiare[2], ale je uhlopriečkou v našom mnohouholníku. Potom priamka AB rozdeluje ostatné vrcholy na dve skupiny. Potom ale existujú vrcholy C a D – každý v inej skupine – ktoré tvoria úsečku patriacu hľadanej lomenej čiare. Ale táto úsečka pretína úsečku $\overline{AB}$, lebo mnohouholník je konvexný. Označme priesečník týchto dvoch uhlopriečok X. Z trojuholníkovej nerovnosti máme $|AX| + |XC| > |AC|$ a $|BX| + |XD| > |BD|$. Sčítaním týchto dvoch nerovníc dostaneme $|AX| + |XC| + |BX| + |XD| = |AB| + |CD| > |AC| + |BD|$. To ale znamená, že keby sme miesto úsečiek $\overline{AB}$ a $\overline{CD}$ zakomponovali do našej lomenej čiary úsečky $\overline{AC}$ a $\overline{BD}$, dostali by sme lomenú čiaru s menšou dĺžkou. To je ale spor s predpokladom, že pôvodná lomená čiara bola najkratšia. Najkratšia lomená čiara teda neobsahuje žiadnu uhlopriečku. Teda najkratšia lomená čiara nutne tvorí obvod mnohouholníka.

Už vieme, čo chceme nájsť, a teraz že ako. Pokúsime sa nájsť presné polohy jednotlivých vrcholov v rovine. Zjavne všetky riešenia tvoria zhodné mnohouholníky, a je teda jedno, ktorý

[2] to tiež znamená, že žiadny iný vrchol na úsečke $\overline{AB}$ neleží

z nich nájdeme my, lebo potrebujeme zistiť iba jeho obvod. Súradnicovú sústavu zvolíme tak, aby prvý vrchol (označme A_1) ležal v jej počiatku (bod $[0,0]$) a druhý vrchol (označme A_2) na kladnej časti osi x. A ako nájdeme súradnice i-teho vrchola A_i? Označme $d = |A_1A_2|$, ďalej $p = |A_1A_i|$ a $q = |A_2A_i|$. Inšpirujeme sa konštrukciou trojuholníka, keď poznáme dĺžky všetkých troch strán. Teda vrchol A_i bude ležať na priesečníku kružníc $k_{i_1}(A_1, p)$ a $k_{i_2}(A_2, q)$. Potom ale platí $A_i = [x_i, y_i]$, pričom $p^2 = x_i^2 + y_i^2$ a $q^2 = (d - x_i)^2 + y_i^2$. Odčítaním druhej rovnice od prvej dostávame vzťah pre x_i a y_i, presnejšie

$$A_i = \left[\frac{p^2 - q^2 + d^2}{2d},\ \pm\sqrt{p^2 - x_i^2} \right]$$

Keď $i = 3$, môžeme si znamienko pri y-ovej súradnici ešte zvoliť, napríklad $+$. Pre $i > 3$ sa musíme ešte spýtať na vzdialenosť $r = |A_iA_3|$ – a tu bola práve chyba v zadaní – a porovnať vypočítanú vzdialenosť bodov A_i a A_3 podľa známeho vzťahu $|A_iA_3| = \sqrt{(x_i - x_3)^2 + (y_i - y_3)^2}$ a prípadne zmeniť znamienko. Rovnosť dvoch reálnych čísel budeme (kvôli nepresnostiam spôsobeným spôsobom ukladania reálnych čísel v pamäti) testovať tak, že zistíme, či je ich rozdiel dostatočne blízky 0 (presnejšie či je jeho absolútna hodnota menšia ako $\varepsilon = 10^{-5}$). Hľadanie súradníc isto zvládneme v čase $O(N)$.

Keď máme súradnice všetkých vrcholov, treba nám zistiť, v akom poradí ležia po obvode mnohouholníka. To sa dá riešiť dvomi spôsobmi, obidva sú pre túto úlohu správne a potrebujeme na to rovnaký čas aj pamäť. Prvá možnosť je nájsť konvexný obal danej množiny bodov. Keďže vrcholy tvoria konvexný mnohouholník, zjavne všetky budú tvoriť konvexný obal. Druhá možnosť je body polárne utriediť podľa nejakého vnútorného bodu mnohouholníka. Využijeme pritom opäť skutočnosť, že vrcholy tvoria konvexný mnohouholník. Takým bodom je napríklad ťažisko T každého trojuholníka tvoreného niektorými tromi vrcholmi. Vypočítame si uhol, ktorý zviera priamka TA_i s osou x. Ak $TA_i \mid\mid x$ a $x_i > x_T$, potom je uhol prislúchajúci bodu A_i rovný 0. Uhol sa zväčšuje proti smeru hodinových ručičiek. Použijeme niektoré triedenie v čase $O(N \log N)$, napríklad Quicksort. Posledná časť úlohy je už jednoduchá, spočítame dĺžku obvodu takto nájdeného mnohouholníka, zrejme v čase $O(N)$.

Najdlhšie zo všetkého trvá utriedenie vrcholov, preto časová zložitosť je $O(N \log N)$. Pamätáme si iba súradnice jednotlivých vrcholov a ich uhol vzhľadom na bod T a os x a okrem toho už iba niekoľko málo údajov (konštantný počet), preto výsledná pamäťová zložitosť je $O(N)$. No a deduškovi položíme $3 + (N - 3) \cdot 3 = O(N)$ otázok.

A ešte bodovanie:

- Ak vaše riešenie bolo správne a malo zložitosť ako vzorové, dostali ste maximálne **15 bodmi**.
- Za riešenie v čase $O(N^2)$ bolo maximálne **11 bodov**.
- Ak ste potrebovali viac času, ale stále iba polynomiálne veľa, máte ešte nejaký ten bodík, dva dolu.
- Za skúšanie všetkých možností (čas $O(N!)$ alebo podobne), dostali ste maximálne **7 bodov**.
- A zlé riešenia dostali najviac **3 body**, podľa toho, do akej miery boli dobré :-)

Pár bodov ste stratili za chýbajúci, či nedostatočný dôkaz, **1 bod** za nedostatočný popis, pár bodov za chýbajúci zdrojový kód, a nejaké bodíky šli dolu aj za chybičky vo vašich programoch.

Listing programu:

```
Program Hamilton;

const MAX = 100;          eps = 10e-5;
```

```
type TBod = record x,y: real; u: real; end;

var A: array[1..MAX] of TBod;
    B: array[0..MAX] of integer;
    v1, v2, v3, a12, ham: real;
    N: integer;
    i: integer;
    C: TBod;

function Vzd(i,j : integer): real; {spyta sa deduska}
  var tmp: real;
begin
  writeln('- Dedusko, dedusko, vy ste snad najmudrejsi na svete,');
  writeln('  vzdialenost ostrovov ',i,' a ',j,' povedat mi neviete?');
  write('- Ale viem synak. Je to '); Readln(tmp); writeln;
  Vzd:=tmp;
end;

function D(P,Q: TBod): real; {vypocita vzdialenost dvoch bodov v rovine}
begin D:=sqrt(sqr(P.x - Q.x) + sqr(P.y - Q.y)); end;

procedure Suradnice(var P: TBod); {zisti suradnice bodu P vzhladom na body A[1] a A[2]}
begin
  with P do
    begin x:=(sqr(v1) - sqr(v2) + sqr(a12)) / (2*a12); y:=sqrt(sqr(v1) - sqr(x)); end;
end;

procedure Overenie(var P: TBod); {overi druhu moznost pre P.y s bodom A[3]}
begin if Abs(D(P,A[3])-v3) > eps then P.y:=-P.y; end;

procedure Uhol(i: integer); {vypocita uhol vzhladom na vnutorny bod C}
begin
  if Abs(A[i].x - C.x) < eps then begin
      if A[i].y > C.y then A[i].u:=Pi/2 else A[i].u:=3*Pi/2;
    end else begin
      A[i].u:=arctan((A[i].y-C.y)/(A[i].x-C.x));
      if A[i].u < 0 then A[i].u:=A[i].u+Pi;
      if A[i].y < C.y then A[i].u:=A[i].u+Pi;
    end;
end;

procedure Swap(var p,q: integer); {vymeni dva prvky}
var tmp: integer; begin tmp:=p; p:=q; q:=tmp; end;

procedure QSort(l,r: integer); {triedi}
  var i,j: integer; p: real;
begin
  i:=l; j:=r; p:=A[B[(l+r) div 2]].u;
  repeat
    while A[B[i]].u < p do Inc(i); while A[B[j]].u > p do Dec(j);
    if i<=j then begin Swap(B[i],B[j]); Inc(i); Dec(j); end;
  until i>j;
  if l<j then QSort(l,j); if i<r then QSort(i,r);
end;

begin
  readln(N);
  {vypocitame polohu bodov A[1], A[2], A[3]}
  with A[1] do begin x:=0;        y:=0; end; with A[2] do begin x:=Vzd(1,2); y:=0; end;
  v1:=Vzd(1,3); v2:=Vzd(2,3); a12:=A[2].x;
```

```
  Suradnice(A[3]);
  {a polohu ostatnych bodov}
  for i:=4 to N do begin
    v1:=Vzd(1,i); v2:=Vzd(2,i); v3:=Vzd(3,i);
    Suradnice(A[i]); Overenie(A[i]);
  end;
  {nejaky vnutorny bod}
  with C do begin x:=(A[1].x + A[2].x + A[3].x)/3; y:=(A[1].y + A[2].y + A[3].y)/3; end;
  {vyratame uhol kazdeho bodu vhladom na C}
  for i:=1 to N do Uhol(i);
  {utriedime body polarne - v B[] je permutacia 1..n}
  for i:=1 to N do B[i]:=i;  QSort(1,N);  B[0]:=B[N];
  {a na zaver vypocitame dlzku cesty}
  ham:=0; for i:=1 to N do ham:=ham+D(A[B[i-1]],A[B[i]]);
  writeln('Dlzka: ',ham:0:5);
end.
```

4. O Miškovi na ľade

opravoval MišoF
(max. 15 bodov)

Na úvod terminológia. Súradnicu rovnobežnú s brehom budeme volať šírka, kolmý smer bude výška.

Možných prístupov bolo veľa, začnime od najpomalších. Čas behu najpomalších riešení závisel od veľkosti čísel na vstupe. (Asi najčastejšia myšlienka: Nájdeme maximum z čísel na vstupe, potom skúšame ako výšku obdĺžnika čísla od 1 po túto hodnotu a zakaždým prejdeme všetky výšky a pozrieme, kde takto vysoký obdĺžnik môže byť. Ale našli sa aj ešte pomalšie riešenia.) Treba si uvedomiť, že takéto riešenie má až exponenciálnu časovú zložitosť v závislosti od dĺžky vstupu, resp. od N.

K tomuto trochu podrobnejšie, keďže to na prvý pohľad asi nie je jasné. Predstavme si vstupný súbor. Ja ho teraz otvorím, nájdem v ňom najväčšie číslo a dopíšem mu na koniec nulu. Veľkosť súboru sa každou takouto zmenou zväčší o 1, dĺžka behu takéhoto programu však každou takouto zmenou vzrastie desaťnásobne, teda porastie exponenciálne od dĺžky vstupu.

Keďže u väčšiny „klasických“ algoritmov ich časová zložitosť nezávisí od veľkosti čísel na vstupe (napr. keď triedime nejaké int-y), môže sa

Takýto pohľad na časovú zložitosť sa môže zdať trochu zvláštny. Treba si ale uvedomiť, že **toto** je korektná definícia časovej zložitosti. To, čo poznáme ako zložitosť „klasických“ algoritmov (ako napr. triedenie celých čísel) je jej špeciálnym prípadom. U „klasických“ algoritmov časová zložitosť nezávisí od veľkosti čísel na vstupe, iba od ich počtu, ktorý je priamo úmerný veľkosti vstupného súboru. V tomto prípade je teda jedno, či berieme časovú zložitosť v závislosti od veľkosti vstupného súboru alebo v závislosti od počtu prvkov na vstupe (a to druhé je prirodzenejšie).

Spomínané riešenia mohli získať najviac **6 bodov**. Prejdime teraz k tým efektívnejším, ktoré bežali v čase polynomiálnom od N (a teda aj od veľkosti vstupného súboru). Na tie už nestačí úplne slepo skúšať možnosti, musíme si o našom obdĺžniku čo-to uvedomiť. V prvom rade (túto myšlienku už používali aj predchádzajúce riešenia) jedna strana obdĺžnika leží na brehu. Totiž smerom ku brehu vieme každý iný obdĺžnik zväčšiť.

Predstavme si teraz, že by sme vedeli, v ktorom stĺpci obdĺžnik začína a v ktorom končí. Akú najväčšiu môže mať výšku? Je to minimum z výšok stĺpcov, v ktorých leží. Toto vedie k triviálnemu riešeniu v čase $O(N^3)$. Priamočiaro ho vieme vylepšiť. Zoberme si pevný ľavý kraj obdĺžnika, pravý budeme posúvať doprava a zároveň si budeme upravovať aj minimum. Takto nemusíme zakaždým minimum odznova rátať a dostávame kvadratické riešenie, za ktoré sa dalo dostať **8 bodov**.

Lepšie riešenie? Uvedomme si, že hľadaný obdĺžnik zapĺňa celý niektorý stĺpec. V cykle vyskúšame všetky možnosti, ktorý to je. Keď sme si zvolili stĺpec, postupne predlžujeme obdĺžnik tejto výšky doľava aj doprava, kým to ide. Takto určite nájdeme najväčší obdĺžnik. Aj toto riešenie má v najhoršom prípade časovú zložitosť $O(N^2)$, v priemerom prípade je však lepšie.

(Nasleduje stručné a nepresné zdôvodnenie. Pre stĺpec s najnižšou hodnotou takto prejdeme celé pole, teda spravíme $O(N)$ operácií. Týmto sa nám ale pole rozdelí na dve nezávislé časti v priemere polovičnej veľkosti. V každej z nich opäť máme nejaký najmenší stĺpec. Pre tieto dva stĺpce prejdeme dokopy najviac celé pole. To sa nám už rozdelilo na 4 časti, atď. V priemere takto dostaneme $O(\log N)$ „úrovní“ a na každej spravíme $O(N)$ operácií. V priemernom prípade má tento algoritmus teda časovú zložitosť $O(N \log N)$.)

Za spomínané riešenie sa dalo získať **9 bodov**. (A ešte jeden navyše ak bolo v popise niečo, čo sa podobalo predchádzajúcemu odseku.)

Existujú riešenia, ktoré potrebujú v najhoršom prípade čas $O(N \log N)$. Jedno je založené na myšlienke: Zotriedime stĺpce podľa výšky, potom ich postupne od najvyššieho spracúvame. Vždy, keď sa dva kusy ľadu pridaním práve spracúvaného stĺpca spoja, pozrieme, či ich spojením nevznikol väčší obdĺžnik (zapĺňajúci práve pridaný stĺpec). Takéto riešenia si zaslúžili **12 bodov**.

Plný počet **15 bodov** si samozrejme zaslúžili len lineárne riešenia. Bolo ich viac typov, naše bude založené na myšlienke použiť zásobník. Odkiaľ sa vezme?

Uvedomme si, že najväčší obdĺžnik musí mať (okrem už spomenutých vlastností) pri ľavej aj pri pravej strane kus neodhrnutého ľadu. Takže nám stačí kontrolovať takéto obdĺžniky, budeme ich volať *kandidáti*. Nazvime *ľavým krajom ľadovej plochy* každú zvislú čiaru, naľavo od ktorej je sneh a napravo ľad. Zjavne hľadaný obdĺžnik musí mať na vrchu ľavej strany kus ľavého kraja ľadovej plochy.

Budeme postupne spracúvať stĺpce v poradí zľava doprava. Pritom si stále budeme pamätať, ktoré kusy ľavého kraja ľadovej plochy je z aktuálneho stĺpca „vidieť“. Na obrázku čiarkovaná čiara prechádza cez práve spracúvaný stĺpec, zvýraznená je časť ľavého kraja, ktorú je vidieť z predchádzajúceho stĺpca.

2cm

Pre každú úsečku ľavého kraja si stačí pamätať číslo stĺpca na ľavej strane ktorého leží a výšku jej horného konca. Nie je ťažké uvedomiť si, že výšky horných koncov týchto úsečiek sú zároveň výšky vhodných stĺpcov. Navyše si uvedomme, že pamätané údaje o ľavom kraji nám budú užitočné pri hľadaní kandidátov.

Napr. v prípade na obrázku by sme si pred spracovaním vyznačeného 14. stĺpca pamätali nasledovné dvojice (výška horného konca, stĺpec): (1,2), (3,4), (4,11), (6,13). Čo tieto údaje hovoria o obdĺžnikoch končiacich v 13. stĺpci? Ak má výšku 6 alebo menej, môže siahať nanajvýš po stĺpec 13, tam narazí na ľavý kraj. Ak má výšku 4 alebo menej, môže siahať nanajvýš po stĺpec 11. Obdĺžnik výšky 3 a menej môže siahať doľava až po stĺpec 4, atď.

Spracovanie stĺpca sa bude skladať z dvoch častí. Spočítame plochy kandidátov končiacich v predchádzajúcom stĺpci a navyše upravíme pamätanú množinu úsečiek ľavého kraja. Kandidáti sú tie obdĺžniky, ktoré už ďalej nemohli pokračovať, teda boli vyššie ako aktuálny stĺpec. Ich ľavý horný roh leží na ľavom kraji. Preto ľavý horný roh každého kandidáta leží na niektorej z úsečiek, ktoré tvoria videnú časť ľavého kraja. Keď teraz budeme tento ľavý horný roh po príslušnej úsečke posúvať dohora, obdĺžnik bude stále celý tvorený ľadom a jeho plocha sa bude zväčšovať.

Zhrnutie: Zaujíma nás obdĺžnik najväčšieho obsahu, ktorý skončil v predchádzajúcom stĺpci. Z vyššie povedaného vyplýva, že ako jeho ľavý horný roh stačí skúšať horné konce pamätaných úsečiek. Navyše stačí uvažovať tie horné konce, ktoré sú vyššie ako je výška práve spracúvaného stĺpca.

V našom prípade má spracúvaný stĺpec výšku 5, teda jediný kandidát je obdĺžnik výšky 6, ktorý začína aj končí v 13. stĺpci.

Videnú časť ľavej strany si budeme pamätať v zásobníku usporiadanú podľa výšky, navrchu bude najvyššie začínajúca úsečka. Vždy totiž budeme pracovať len s ňou, preto je zásobník ideálna dátová štruktúra. Nech teraz h je výška jej začiatku a v výška spracúvaného stĺpca.

Ak $h > v$, tak spočítame plochu kandidáta s ľavým horným vrchom na hornom konci tejto úsečky. Ak je väčšia ako doteraz najväčšia nájdená, zapamätáme si ju. Navyše potrebujeme túto úsečku ľavého kraja skrátiť alebo vyhodiť. Pozrime sa, v akej výške úsečka končí (t.j. v akej výške začína najbližšia nižšia). Ak aj končí vo výške $\geq v$, odteraz ju vôbec nebude vidieť, preto ju vyhodíme zo zásobníka (ale pokračujeme v spracúvaní aktuálneho stĺpca, ešte tu môžu byť ďalší kandidáti). V opačnom prípade ju bude odteraz vidieť len od spodku do výšky v, preto zmenšíme výšku jej začiatku z h na v a ideme spracovať ďalší stĺpec (lebo tu už žiadni ďalší kandidáti nebudú).

Ak $h = v$, nemáme žiadnych kandidátov a navyše sa nezmení ani videná časť ľavej strany. Preto iba prejdeme na ďalší stĺpec.

Ak $h < v$, pribudne do videnej časti ľavej strany nová (najvyššie začínajúca) úsečka. Bude ležať v spracúvanom stĺpci a jej vrchol bude vo výške v. Tým sme skončili spracúvanie stĺpca a prejdeme na ďalší.

Príklad. Po spracovaní 14. stĺpca máme na zásobníku: (1,2), (3,4), (4,11), (5,13). Poďme spracovať 15. stĺpec. Ten má výšku 2. Prvým kandidátom na maximálny obdĺžnik je obdĺžnik výšky 5, ktorý končí v 14. stĺpci a začína v 13. Druhým kandidátom je obdĺžnik výšky 4, ktorý začína v 11. stĺpci. Tretím kandidátom je obdĺžnik výšky 3, ktorý začína v 4. stĺpci. Z tejto úsečky ľavej strany bude naďalej spodnú časť vidieť, preto tu končí spracúvanie 15. stĺpca. V zásobníku zostane: (1,2), (2,4).

Ešte pár detailov. Aby sme nemuseli ošetrovať okrajové prípady, na spodok zásobníka vložíme (0,0) a na koniec vstupu pridáme jeden stĺpec s výškou 0. Uvedomte si, že vyššie uvedený algoritmus sa dá použiť aj pri riešení úlohy z predchádzajúcej série – vždy prečítame riadok vstupu, upravíme výšky odhrnutého ľadu a spustením tohto algoritmu nájdeme najväčší obdĺžnik so spodkom na aktuálnom riadku.

Listing programu:

```
#include <stdio.h>
#include <stdlib.h>

#define TOPV (Stack[pStack-1].vyska)  // makra pre
#define TOPS (Stack[pStack-1].stlpec) // vrch zasobnika
typedef struct { int vyska, stlpec; } tData;

int N,*A;     // vstup
int i,kde,plocha;
tData *Stack; // zasobnik
int pStack=0; // pocet udajov v nom
int max=0;    // vysledok

int main(void){
  // nacitame vstup
  scanf("%d ",&N);
  A=(int *)malloc((N+3)*sizeof(int));     Stack=(tData *)malloc((N+3)*sizeof(tData));
  for (i=0;i<N;i++) scanf("%d ",&A[i]); A[N]=0;
  // dame zarazku na stack
  pStack=1; Stack[0].vyska=Stack[0].stlpec=0; kde=0;
  while (1){
    if (kde>N) break;
```

```
    if (A[kde]>TOPV) { // pridame usecku, ideme na dalsi stlpec
      Stack[pStack].vyska=A[kde]; Stack[pStack].stlpec=kde; pStack++; kde++;
    } else if (A[kde]==TOPV) {
      kde++;  // nic sa nedeje, ideme na dalsi stlpec
    } else {
      // spocitame plochu kandidata
      plocha=TOPV*(kde-TOPS); if (plocha>max) max=plocha;
      // (zmensime hranu a ideme dalej) alebo vyhodime hranu
      pStack--; if (TOPV<A[kde]) { pStack++; TOPV=A[kde]; kde++; }
    }
  }
  free(A); free(Stack);
  printf("Najvacsi obdlznik: %d\n",max);
  return 0;
}
```

5. O celulárnych automatoch III

opravoval Martin
(max. 15 bodov)

Asi polovica z vás mala tento príklad vyriešený správne, no mnohí si neuvedomili, že keď na začiatku výpočtu je najľavejší automat v stave Z, tak najpravejší automat nemá ako zistiť, že sa práve začal výpočet a preto nemôžete očakávať, že pravidla typu $00\$ \to \alpha$ sa použiju práve na začiatku výpočtu.

Za správne riešenie s lineárnou časovou zložitosťou od počtu automatov ste mohli získať maximálne **15 bodov**, za správne riešenie s kvadratickou časovou zložitosťou maximálne **11 bodov**, za zlé riešenie maximálne **4 body**. Za slabý alebo chýbajúci popis vášho riešenia ste mohli získať až **-2 body**. Ak ste mali v riešení nejaké drobné chybičky[3], získali ste za ne po **-1 bode**.

Najjednoduchším riešením je postupné ofarbovanie automatov. Na začiatku výpočtu sú všetky automaty neofarbené. Postupne budeme na striedačku ofarbovať najľavejší a najpravejší neofarbený automat – vždy nejakým špeciálnym stavom „prelezieme“ po všetkých neofarbených automatoch a keď posledný neofarbený ofarbíme. Ak nám na konci zostane len jeden alebo dva neofarbené automaty, označíme tieto automaty ako stred. Ofarbenie jedného automatu trvá $O(n)$ sekúnd, kde n je počet automatov, celková časová zložitosť riešenia je $O(n^2)$.

Pokúsme sa nájsť rýchlejšie riesenie. Predstavme si, že by sme z ľavého kraja poslali dva rôzne rýchle „signály“, jeden z nich trikrát rýchlejší ako druhý. Kým pomalší signál príde do polovice, stihne rýchlejší prísť na koniec, odraziť sa a vrátiť sa späť práve do polovice. Takže tieto dva signály sa stretnú práve v strede. Už ostáva len zapísať túto myšlienku formálne, aby ju pochopili nielen riešitelia, ale aj automaty.

Pri hľadaní stredných automatov použijeme dve sady stavov, ktoré budu rôznymi rýchlosťami postupne putovať po automatoch. V prvej sade budú stavy A_0, A_1 a A_2. Vždy, keď sa nejaký automat dostane do stavu A_0, resp. A_1 v nasledujúcej sekunde prejde do stavu A_1, resp. A_2. Podobne keď sa automat dostane do stavu A_2 v nasledujúcej sekunde prejde do stavu 0[4] a svojho pravého suseda prevedie do stavu A_0. Táto sada stavov bude automatmi prechádzať doprava rýchlosťou 1 automat za tri sekundy. V druhej sade budú stavy R a L. Vždy keď sa niektorý automat dostane do stavu R, v nasledujúcej sekunde prejde do stavu 0 a jeho pravý sused do stavu R. Keď stav R dorazí do najpravejšieho automatu, zmení sa na stav L a ten potom podobne putuje doľava. Táto sada stavov bude putovať automatmi rýchlosťou 1 stav za jednú sekundu najprv doprava, pokým nenarazí na najpravejší automat,

[3] Napríklad, keď vaša prechodová funkcia nebola jednoznačná, keď vám v nej chýbali nejaké nie veľmi podstatné pravidlá, keď vaše riešenie nebolo pre malý počet automatov správne, ...

[4] Stav 0 je stav, v ktorom na začiatku výpočtu boli všetky automaty okrem najľavejšieho.

potom bude putovať rovnakou rýchlosťou doľava. Predstavme si, že stav Z značí, že automat je súčastne v stave A_0 aj v stave R. Potom, ako neskôr dokážeme, popísané sady stavov sa časom naozaj stretnú na stredných automatoch. Nesmieme ešte zabudnúť na okrajový prípad, keď máme iba jediný automat.

Automaty budú nadobúdať stavy z množiny $K = \{Z, S, 0, A_0, A_1, A_2, L, R\}$ a ich prechodová funkcia bude:

(1)	$\$Z\$$	$\to$	S	(8)	$\gamma R0$	$\to$	0				
				(9)	$R0\alpha$	$\to$	R				
(2)	$\$Z0$	$\to$	A_1	(10)	$\gamma R\$$	$\to$	L	α	$\in$	$\{0, \$\}$	
(3)	$Z0\alpha$	$\to$	R	(11)	$\delta L\alpha$	$\to$	0	β	$\in$	$\{0, R\}$	
				(12)	$\gamma 0L$	$\to$	L	γ	$\in$	$\{0, A_1\}$	
(4)	$\alpha A_0\beta$	$\to$	A_1					δ	$\in$	$\{0, A_0\}$	
(5)	$\alpha A_1\beta$	$\to$	A_2	(13)	αA_2L	$\to$	S	ζ	$\in$	$\{0, A_0, A_1, L, \$\}$	
(6)	$\alpha A_2\beta$	$\to$	0	(14)	$A_2L\alpha$	$\to$	S	η	$\in$	$\{0, A_0, A_1, A_2, R, \$\}$	
(7)	$A_20\beta$	$\to$	A_0	(15)	αA_0L	$\to$	S				
				(16)	$\zeta 0\eta$	$\to$	0				

Aby ste videli, ako automat funguje, pozrite si výpočet pre párny počet automatov: $\$Z00000\$ \to \$A_1R0000\$ \to \$A_20R000\$ \to \$0A_00R00\$ \to \$0A_100R0\$ \to \$0A_2000R\$ \to \$00A_00L\$ \to \$00A_10L0\$ \to \$00A_2L00\$ \to \$00SS00\$$.

A pre nepárny počet automatov: $\$Z0000\$ \to \$A_1R000\$ \to \$A_20R00\$ \to \$0A_00R0\$ \to \$0A_100R\$ \to \$0A_200L\$ \to \$00A_0L0\$ \to \$00S00\$$.

Dokážeme[5], že automat nájde všetky stredné automaty. Nech n je počet automatov. Ak $n = 1$, potom tento jediný automat je aj hľadaným stredom a v prvej sekunde výpočtu pravidlom (1) sa zmení jeho stav na S. Predpokladajme, že $n > 1$. Očíslujme si automaty číslami 1 až n tak, aby najľavejší automat mal číslo 1, jeho sused 2, až najpravejší automat číslo n. Ak q je stav a k je celé nezáporné číslo, zápisom q^k budeme označovať k za sebou idúcich automatov v stave q. Konfigurácia automatov v t-tej sekunde ($t \geq 0$) je postupnosť aktuálnych stavov všetkých automatov zapísaných zľava doprava. Zapisujeme ju: $t : \$q_1q_2 \ldots q_n\$$, kde q_i je stav i-tého automatu v t-tej sekunde.

Na začiatku výpočtu (v nultej sekunde) sú všetky automaty okrem prvého v stave 0, ten je v stave Z. Teda konfigurácia automatov na začiatku výpočtu je $0 : \$Z0^{n-1}\$$.

Najprv matematickou indukciou dokážeme, že v t-tej sekunde výpočtu ($1 \leq t \leq n-1$) konfigurácia automatov je

$$t : \$0^{\lfloor t/3 \rfloor} A_{t \bmod 3} 0^{t-\lfloor t/3 \rfloor - 1} R 0^{n-t-1}\$$$

- Nech $t = 1$. Vieme, že v nultej sekunde je konfigurácia automatov $0 : \$Z0^{n-1}\$$. V prvej sekunde prejde prvý automat podľa pravidla (2) do stavu A_1 a druhý automat podľa pravidla (3) do stavu R. Ostatné automaty podľa pravidla (16) ostanú nezmenené. Preto v prvej sekunde $1 = t : \$A_1R0^{n-2}\$ \equiv t : \$0^0A_10^0R0^{n-2}\$ \equiv t : \$0^{\lfloor t/3 \rfloor}A_{t \bmod 3}0^{t-\lfloor t/3 \rfloor-1}R0^{n-t-1}\$$ a teda pre $t = 1$ tvrdenie platí.
- Teraz predpokladajme, že v t-tej ($1 \leq t < n-2$) sekunde výpočtu dokazované tvrdenie platí, dokážeme jeho platnosť aj v $t+1$-vej sekunde. Ak $t+1$ je deliteľné tromi, potom podľa indukčného predpokladu konfigurácia v t-tej sekunde je

$$t : \$0^{\lfloor t/3 \rfloor} A_{t \bmod 3} 0^{t-\lfloor t/3 \rfloor - 1} R 0^{n-t-1}\$ \equiv$$

[5] Samozrejme, že ani tentoraz ste nemuseli písať až takto formálny dôkaz, ale zdôvodnenie „veď to vidno“ ma nepresvedčí. :-)

$$\equiv t : \$0^{(t+1)/3-1}A_2 0^{(t+1)-(t+1)/3-1}R0^{n-(t+1)}\$.$$

Podľa pravidla (6) prejde $(t+1)/3$-tý automat do stavu 0. Keďže $(t+1)-(t+1)/3-1 > 0$, podľa (7) prejde jeho pravý sused do stavu A_0. Ďalej podľa pravidla (8) predje $t+1$-vý automat do stavu 0 a keďže $n-t-1 > 0$ podľa pravidla (9) prejde jeho pravý sused do stavu R. Stavy ostatných automatov sa podľa pravidla (16) nezmenia. Preto v $t+1$-vej sekunde výpočtu bude

$$t+1 : \$0^{(t+1)/3-1}0A_0 0^{(t+1)-(t+1)/3-2}0R0^{n-(t+1)-1}\$ \equiv$$

$$\equiv t+1 : \$0^{\lfloor (t+1)/3 \rfloor}A_{(t+1) \bmod 3}0^{(t+1)-\lfloor (t+1)/3 \rfloor -1}R0^{n-(t+1)-1}\$.$$

Ak $t+1$ nie je deliteľne tromi, podľa indukčného predpokladu konfigurácia v t-tej sekunde je

$$t : \$0^{\lfloor t/3 \rfloor}A_{t \bmod 3}0^{t-\lfloor t/3 \rfloor -1}R0^{n-t-1}\$ \equiv$$

$$\equiv t : \$0^{\lfloor (t+1)/3 \rfloor}A_{t \bmod 3}0^{(t+1)-\lfloor (t+1)/3 \rfloor -2}R0^{n-(t+1)}\$.$$

Keďže $t \bmod 3 \neq 2$, podľa pravidla (4) alebo (5) prejde $\lfloor t/3 \rfloor + 1$-vý automat do stavu $A_{(t+1) \bmod 3}$. Ďalej $t+1$-vý automat a $t+2$-hý automat zmenia svoj stav rovnako ako v predošlom prípade, keď $t+1$ bolo deliteľné tromi. Preto v $t+1$-vej sekunde výpočtu bude

$$t+1 : \$0^{\lfloor (t+1)/3 \rfloor}A_{(t+1) \bmod 3}0^{(t+1)-\lfloor (t+1)/3 \rfloor -1}R0^{n-(t+1)-1}\$.$$

Opať matematickou indukciou ukážeme, že v t-tej sekunde výpočtu ($n \leq t \leq n+\lfloor n/2 \rfloor - 1$) je konfigurácia automatov

$$t : \$0^{\lfloor t/3 \rfloor}A_{t \bmod 3}0^{2n-t-\lfloor t/3 \rfloor -2}L0^{t-n}\$$$

- Nech $t = n$. Vieme, že v $n-1$-vej sekunde je

$$n-1 : \$0^{\lfloor (n-1)/3 \rfloor}A_{(n-1) \bmod 3}0^{(n-1)-\lfloor (n-1)/3 \rfloor -1}R0^{n-(n-1)-1}\$ \equiv$$

$$\equiv n-1 : \$0^{\lfloor (n-1)/3 \rfloor}A_{(n-1) \bmod 3}0^{(n-1)-\lfloor (n-1)/3 \rfloor -1}R\$.$$

V n-tej sekunde prejde n-tý automat podľa pravidla (10) do stavu L. Ak je n deliteľné tromi, potom $n-1 : \$0^{n/3-1}A_2 0^{n-n/3-1}R\$$, $n/3$-tý automat prejde podľa (6) do stavu 0 a keďže $n-n/3-1 > 0$ jeho pravý sused podľa (7) do stavu A_0. Preto

$$n = t : \$0^{n/3-1}0A_0 0^{n-n/3-2}L\$ \equiv$$

$$\equiv t : \$0^{\lfloor n/3 \rfloor}A_{n \bmod 3}0^{2n-n-\lfloor n/3 \rfloor -2}L0^{n-n}\$ \equiv$$

$$\equiv t : \$0^{\lfloor t/3 \rfloor}A_{t \bmod 3}0^{2n-t-\lfloor t/3 \rfloor -2}L0^{t-n}\$.$$

Ak n nie je deliteľne tromi, potom

$$n-1 : \$0^{\lfloor n/3 \rfloor}A_{(n-1) \bmod 3}0^{n-\lfloor n/3 \rfloor -2}R\$,$$

$\lfloor n/3 \rfloor + 1$-vý automat prejde podľa (4) alebo podľa (5) do stavu $A_{n \bmod 3}$. Preto

$$n = t : \$0^{\lfloor n/3 \rfloor}A_{n \bmod 3}0^{2n-n-\lfloor n/3 \rfloor -2}L0^{n-n}\$ \equiv$$

$$\equiv t : \$0^{\lfloor t/3 \rfloor}A_{t \bmod 3}0^{2n-t-\lfloor t/3 \rfloor -2}L0^{t-n}\$.$$

Takže pre $t = n$ tvrdenie platí.

- Predpokladajme, že v t-tej sekunde ($n \leq t < n + \lfloor n/2 \rfloor - 1$) dokazované tvrdenie platí. Dokážeme, že platí aj v $t+1$-vej sekunde výpočtu. Podľa indukčného predpokladu je $t : \$0^{\lfloor t/3 \rfloor} A_{t \bmod 3} 0^{2n-t-\lfloor t/3 \rfloor - 2} L 0^{t-n}\$$. Keďže $2n - t - \lfloor t/3 \rfloor - 2 > 0$, $2n-t$-tý automat prejde podľa (11) do stavu 0 a jeho ľavý sused do stavu L. Ak $t+1$ je deliteľné tromi, potom

$$t : \$0^{(t+1)/3-1} A_2 0^{2n-(t+1)-(t+1)/3} L 0^{(t+1)-n-1}\$,$$

  $(t+1)/3$-tý automat prejde podľa (6) do stavu 0 a keďže $2n - (t+1) - (t+1)/3 > 0$, podľa (7) jeho pravý sused do stavu A_0. Preto

$$t+1 : \$0^{(t+1)/3-1} 0 A_0 0^{2n-(t+1)-(t+1)/3-2} L 0 0^{(t+1)-n-1}\$ \equiv$$

$$\equiv t+1 : \$0^{\lfloor (t+1)/3 \rfloor} A_{(t+1) \bmod 3} 0^{2n-(t+1)-\lfloor (t+1)/3 \rfloor - 2} L 0^{(t+1)-n}\$.$$

  Ak $t+1$ nie je deliteľné tromi, potom

$$t : \$0^{\lfloor (t+1)/3 \rfloor} A_{t \bmod 3} 0^{2n-(t+1)-\lfloor (t+1)/3 \rfloor - 1} L 0^{(t+1)-n-1}\$$$

  a $\lfloor (t+1)/3 \rfloor + 1$-vý automat prejde podľa (4) alebo podľa (5) do stavu $A_{(t+1) \bmod 3}$. Preto

$$t+1 : \$0^{\lfloor (t+1)/3 \rfloor} A_{(t+1) \bmod 3} 0^{2n-(t+1)-\lfloor (t+1)/3 \rfloor - 2} L 0 0^{(t+1)-n-1}\$ \equiv$$

$$\equiv t+1 : \$0^{\lfloor (t+1)/3 \rfloor} A_{(t+1) \bmod 3} 0^{2n-(t+1)-\lfloor (t+1)/3 \rfloor - 2} L 0^{(t+1)-n}\$.$$

  Teda ak platí dokazované tvrdenie pre t ($n \leq t < n + \lfloor n/2 \rfloor - 1$), platí aj pre $t+1$.

Dokázali sme, že v $t = n + \lfloor n/2 \rfloor - 1$-vej sekunde výpočtu je konfigurácia automatov $t : \$0^{\lfloor t/3 \rfloor} A_{t \bmod 3} 0^{2n-t-\lfloor t/3 \rfloor - 2} L 0^{t-n}\$$.

Ak je počet všetkých automatov párny, existuje kladné celé číslo k také, že $n = 2k$, potom v $n + \lfloor n/2 \rfloor - 1$-vej sekunde konfigurácia je

$$n + \lfloor n/2 \rfloor - 1 = 3k - 1 : \$0^{\lfloor (3k-1)/3 \rfloor} A_{(3k-1) \bmod 3} 0^{2(2k)-(3k-1)-\lfloor (3k-1)/3 \rfloor - 2} L 0^{(3k-1)-(2k)}\$ \equiv$$

$$\equiv 3k - 1 : \$0^{k-1} A_2 L 0^{k-1}\$.$$

V ďalšej sekunde k-tý a $k+1$-vý automat prejde podľa (13) a (14) do stavu S a konfigurácia bude $n + \lfloor n/2 \rfloor = 3n/2 : \$0^{k-1} S S 0^{k-1}\$$.

Naopak ak je počet automatov je nepárny, existuje kladné celé číslo k, pre ktoré $n = 2k+1$. Potom v $n + \lfloor n/2 \rfloor - 1$-vej sekunde je

$$n + \lfloor n/2 \rfloor - 1 = 3k : \$0^{\lfloor 3k/3 \rfloor} A_{3k \bmod 3} 0^{2(2k+1)-(3k)-\lfloor (3k)/3 \rfloor - 2} L 0^{(3k)-(2k+1)}\$ \equiv$$

$$\equiv 3k : \$0^k A_0 L 0^{k-1}\$.$$

V nasledujúcej sekunde výpočtu $k+1$-vý automat prejde podľa (15) do stavu S a $k+2$-hý automat podľa (11) do stavu 0, preto $n + \lfloor n/2 \rfloor = 3(n-1)/2 + 1 : \$0^k S 0^k\$$.

V $n + \lfloor n/2 \rfloor$-vej sekunde poprvýkrat od začiatku výpočtu niektoré automaty prejdú do stavu S, preto sa výpočet po $n + \lfloor n/2 \rfloor$ sekundách zastaví. Konečné konfigurácie automatov ($\$0^{k-1} S S 0^{k-1}\$$ a $\$0^k S 0^k\$$, kde $k = \lfloor n/2 \rfloor$) zodpovedajú zadaniu, preto skonštruovaný automat je korektný a rieší zadaný problém.

Z dôkazu nám vyplýva, že každý výpočet trvá práve $n + \lfloor n/2 \rfloor$ sekúnd, kde n je počet automatov. Teda časová zložitosť výpočtu je $O(n)$.

Výsledková listina po 1. kole kategórie KSP

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
1	Jančuška Marek	Gym. Párovská Nitra	3	15	15	14	15	13	72
2	Tekeľ Jakub	Gym. Jura Hronca BA	3	15	12	15	10	15	67
3	Repovský Michal	Gym. Komenského Trebišov	3	15	10	10	15	15	65
4	Nánási Michal	Gym. Jura Hronca BA	2	12	10	15	11	14	62
5	Simančík František	Gym. Grösslingová BA	2	15	10	15	15	4	59
6	Lenhardt Rastislav	Gym. Jura Hronca BA	3	14	10	13	9	10	56
	Poláček Lukáš	Gym. K. Štúra Modra	3	15	10	15	13	3	56
8	Ďuriš Michal	Gym. Grösslingová BA	2	15	10	12	15	3	55
9	Baláž Miroslav	Gym. Jura Hronca BA	3	15	10	12	15	1	53
10	Kuchynárová Mariana	Gym. Jura Hronca BA	3	12	8		15	15	50
11	Kováč Jakub	Gym. Jura Hronca BA	3		15	9	10	15	49
12	Perešíni Peter	Gym. Tajovského B. Bystrica	1	15	10	9	12	2	48
13	Glaus Peter	Gym. Jura Hronca BA	4	10	12	13	10		45
	Petruľák Matúš	Gym. Grösslingová BA	2	8	9	9	6	13	45
15	Štefanek Anton	Gym. Jura Hronca BA	3		15	1	10	14	40
16	Rejda Martin	Gym. Grösslingová BA	3	10	15	0	3	11	39
17	Bernát Marek	Gym. K. Štúra Modra	3	12	10	6	8	2	38
18	Šmitala František	Gym. Cyrila a Metoda Nitra	3	8		2	12	14	36
19	Buštor Stano	Gym. Jura Hronca BA	2	3	10		8	14	35
	Ludha Marek	Gym. Tajovského B. Bystrica	3		8	6	8	13	35
	Smažáková Dana	Gym. Jura Hronca BA	3	12	8		15		35
22	Baník Dušan	Gym. Popradské nábr. Poprad	3	7		9	10	4	30
	Štolc Miroslav	Gym. Párovská Nitra	3	10	12		6	2	30
24	Hruda Juraj	SPŠE Stará Turá	2	8	9	5	6	1	29
25	Blénessy Tibor	Gym. Poštová Košice	3	8	8		9	2	27
	Trejbal Ivan	Gym. Jura Hronca BA	3	7	12		6	2	27
27	Takáč Slavomír	Neznama skola	1	7	7	5	5		24
28	Imriška Jakub	Gym. Jura Hronca BA	1	7	8		8		23
	Kollár Juraj	Gym. Jura Hronca BA	2	10		7	6		23
30	Ružička Peter	Gym. Jura Hronca BA	2	3	6	9	2		20
	Zeman Marek	Gym. Jura Hronca BA	2	12	3		5		20
	Šufliarsky Peter	Gym. Nové Zámky	3	12			8		20
33	Šuška Martin	Gymnázium Levice	4		9	1	6	1	17
34	Plžík Milan	Gym. Ľ. Štúra Zvolen	2	7			9		16
35	Kadák Michal	Gym. Ľudovíta Štúra Trenčín	3	7			8		15
36	Bajtoš Maroš	Gym. Jura Hronca BA	2	7			6		13
	Janík Oliver	Gym. L. Stöckela Bardejov	3	7			6		13
38	Sedlák Milan	Gym. Liptovský Hrádok	3	7			5		12