Korešpondenčný seminár z programovania
XXI. ročník, 2003/04
Katedra vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

KSP finančne podporujú: aSc – Applied Software Consultants spol. s r.o.
MICROSTEP-MIS spol. s r.o.

Vzorové riešenia 2. kola zimnej časti

Milí riešitelia!
KSP vám opäť raz prináša pár riadkov o tom, ako sa dali ľahko, rýchlo a správne vyriešiť zadané úlohy. Dúfame, že si ich so záujmom prečítate a budete ešte múdrejší ako už ste :-) A samozrejme na tých najlepších z vás sa tešíme, stretneme sa na sústredení... A aj vy sa máte na čo tešiť, sústredenie bude ešte lepšie ako obvykle!

Keď to nejde podobrom, musí to ísť nad obrom!

KSPáci

1. O povrchovej ťažbe zlata

opravoval Paľo
(max. 15 bodov)

Najprv niekoľko slov k hodnoteniu. Riešenia s časovou zložitosťou $O(N^2M)$ mohli získať maximálne 15 bodov. Riešenia v čase $O(N^2M^2)$ maximálne 12 bodov. Riešenia v čase $O(N^3M^3)$ maximálne 9 bodov. Nefunkčné riešenia mohli dostať najviac 2 body. Body ste stratili hlavne za zlý odhad časovej a pamäťovej zložitosti. Ale poďme sa rýchlo pozrieť na vzorové riešenie.

Načítajme si mapu do matice A výšky N a šírky M. Odčítajme od každého prvku matice $A_{i,j}$ cenu d za jedno jutro. Skoro všetci ste si všimli, že v takto vzniknutej matici stačí nájsť obdĺžnik s najväčším súčtom a vyhlásiť ho za riešenie. Najjednoduchšie riešenie ako nájsť taký obdĺžnik je vyskúšať všetky možnosti. Každý obdĺžnik je jednoznačne určený svojim ľavým horným a pravým dolným rohom. Takže vyskúšame všetky možnosti výskytu ľavého horného a pravého dolného rohu - tých je maximálne N^2M^2. Každý z týchto obdĺžnikov je určite menší ako celá mapa, takže obsah každého z nich je maximálne $O(NM)$. Našli sme riešenie s časovou zložitosťou $O(N^3M^3)$.

Toto sa dá zlepšiť nasledovnou fintou. Označme $B_{[i,j][k,l]}$ súčet obdĺžnika s ľavým horným rohom $[i,j]$ a s pravým dolným rohom $[k,l]$. Označme $S_{i,j}$ súčet obdĺžnika s ľavým horným rohom $[1,1]$ a s pravým dolným rohom $[i,j]$. Predstavme si, že by sme vedeli všetky $S_{i,j}$. Hľadajme $B_{[i,j][k,l]}$. Ľahko nahliadneme, že $B_{[i,j][k,l]} = S_{k,l} - S_{k,j-1} - S_{i-1,l} + S_{i-1,j-1}$. Takto vieme v konštantom čase zistiť súčet ľubovolného obdĺžnika. Stačí nám teda vypočítať všekty $S_{i,j}$. Predstavme si, že už vieme všetky $S_{k,l}$, kde $k < i$ alebo $l < j$. Teraz už vieme vypočítať aj $S_{i,j} = A_{i,j} + S_{i-1,j} + S_{i,j-1} - S_{i-1,j-1}$. Takže náš algoritmus bude fungovať tak, že si najprv vypočítame všetky $S_{i,j}$ v čase $O(NM)$ a potom vyskúšame všetky možnosti polohy obdĺžnika v čase $O(N^2M^2)$ a teda náš vylepšený algoritmus má časovú zložitosť $O(N^2M^2)$.

Jedno z riešení v čase $O(N^2M)$ funguje nasledovne: Pozrime sa najprv na podobný problém v jednej dimenzii. Máme nájsť v nejakej postupnosti B_i dĺžky N úsek s najväčším súčtom. Toto vieme urobiť v lineárnom čase. Označme C_i súčet takého úseku, ktorého pravý koniec sa nachádza na i-tom mieste a má zo všetkých takých najväčší súčet. Stačí nám vypočítať všetky C_i a zistiť maximum z nich. Vieme, že $C_1 = B_1$. Predstavme si, že chceme vypočítať C_i a poznáme už všetky C_j, kde $j < i$. Rozlíšime dva prípady. Prvý prípad nastane, ak dĺžka C_i je 1. Vtedy $C_i = B_i$. V druhom prípade je dĺžka C_i väčšia ako 1. Ľahko nahliadneme, že v tomto prípade $C_i = B_i + C_{i-1}$. Takže stačí zvoliť C_i ako maximum z čísel B_i a $B_i + C_{i-1}$. Všimnime si, že si nemusíme pamätať všetky prvky C_i. Stačí si pamätať iba

posledné dva a vždy keď vypočítame C_i, porovnáme ho s doterajším maximom a prípadne ho prehlásime za nové maximum.

Teraz sa pokúsime využiť tento spôsob v dvojrozmernej matici. Označme $D_{i,j}$ súčet takého obdĺžnika, ktorého výška je i, y-ová súradnica ľavého horného rohu je j a má zo všetkých takých najväčší súčet. Stačí nám nájsť všetky $D_{i,j}$ a potom zistiť maximum z nich. Hľadajme hodnotu $D_{i,j}$. Vytvorme postupnosť B_k tak, že $B_k = A_{j,k} + A_{j+1,k} + \ldots + A_{j+i-1,k}$, teda B_k je súčet i prvkov v k-tom stĺpci nachádzajúcich sa na pozíciach $j, j+1, \ldots, j+i-1$. Nájdime v tejto postupnosti úsek U_1 s najväčším súčtom. Nech ten úsek má tvar $B_a, B_{a+1}, \ldots, B_b$. Ukážeme sporom, že obdĺžnik s ľavým horným rohom $[j, a]$ a s pravým dolným rohom $[j+i-1, b]$ je hľadaný obdĺžnik. Nech by existoval obdĺžnik s väčším súčtom taký, že má ľavý horný roh $[j, c]$ a pravý dolný roh $[j+i-1, d]$, kde $c \leq d$. Potom by ale úsek $B_c, B_{c+1}, \ldots, B_d$ mal väčší súčet ako U_1, čo je v spore s predpokladom, že U_1 má najväčší súčet. Všetkých $D_{i,j}$ je $O(N^2)$. Každé B_k vieme zistiť v konštantnom čase vyššie uvedenou fintou. U_1 vieme zistiť lineárne v závislosti od dĺžky postupnosti B_k, t.j. v čase $O(M)$. Teda aj každé $D_{i,j}$ vieme zistiť v čase $O(M)$. Celková časová zložitosť je $O(NM + N^2M) = O(N^2M)$. Potrebujeme si pamätať niekoľko polí velkosti $O(NM)$, teda pamäťová zložitosť je $O(NM)$.

Myšlienka ešte raz slovami: Zvolíme pevne riadky, v ktorých začína a končí hľadaný obdĺžnik. Medzi týmito riadkami je niekoľko „pásikov políčok“ rozmeru $1 \times i$. Hľadaný obdĺžnik sa musí skladať z niekoľkých po sebe idúcich pásikov. Súčty políčok v jednotlivých pásikoch vieme spočítať[1] v konštantnom čase na pásik. V tejto postupnosti potrebujeme nájsť súvislý úsek s najväčším súčtom. To vieme spraviť v lineárnom čase, teda $O(M)$. Dvojíc riadkov je $O(N^2)$, preto výsledná časová zložitosť je $O(N^2M)$.

Listing programu:

```
const INF=32767;

var i,j,k: integer;
    d,n,m: integer;
    mapa: array[1..100, 1..100] of integer;
    sum: array[0..100, 0..100] of integer;
    akt,ax: integer;
    max,mx1,mx2,my1,my2: integer;

{sucet obddlznika s lavym hornym rohom y1,x1 a pravym dolnym rohom y2,x2}
function sucet(x1,y1,x2,y2: integer): integer;
begin sucet := sum[y2,x2]-sum[y2,x1-1]-sum[y1-1,x2]+sum[y1-1,x1-1]; end;

begin
    read(d,n,m);
    for i := 1 to n do for j := 1 to m do begin
        read(mapa[i,j]);
        mapa[i,j] := mapa[i,j]-d;
    end;

    for i := 0 to m do sum[0,i] := 0;
    for i := 0 to n do sum[i,0] := 0;
    for i := 1 to n do for j := 1 to m do
        sum[i,j] := sum[i-1,j]+sum[i,j-1]-sum[i-1,j-1]+mapa[i,j];

    max := -INF;
    for k := 1 to n do begin
        for i := 1 to n - k + 1 do begin
            akt := -INF;
```

[1] Napr. pomocou hodnôt $S_{i,j}$, ale stačilo by len mať spočítané čiastočné súčty v každom stĺpci.

```
          for j := 1 to m do begin
              if akt+sucet(j,i,j,i+k-1) > sucet(j,i,j,i+k-1) then begin
                  akt := akt+sucet(j,i,j,i+k-1);
              end else begin
                  akt := sucet(j,i,j,i+k-1);
                  ax := j;
              end;
              if akt > max then begin
                  max := akt;
                  mx1 := ax;    my1 := i;
                  mx2 := j;     my2 := i+k-1;
              end;
          end;
      end;
  end;

  write('Najlepsie je kupit pozemok s rohmi [', my1, ', ', mx1, ']');
  writeln(' a [', my2, ', ', mx2, ']. Zisk bude ', max, ' dolarov.');
end.
```

2. O plnom prasiatku

Opravoval Braňo
(max. 15 bodov)

Na chvíľu si predstavme, že máme len také mince, kde ľubovolná minca je vždy násobkom menších mincí (napr. nemáme 25centovky ale 50centovky). Za takýchto podmienok ľahko vidno, že väčšiu mincu vieme rozmeniť na menšie mince vždy, keď majú aspoň hodnotu väčšej mince. Zadaný príklad by sa nám riešil veľmi ľahko tzv. greedy. Išli by sme od najväčšej mince k menším a zaplatili by sme ňou presne toľko, aby sme menšími dokázali zaplatiť zvyšok.

Ešte raz a pomalšie. Napr. mincu s hodnotou 10 sa nám neoplatí použiť, ak máme ešte dosť menších mincí na to, aby sme nimi zaplatili hodnotu 10. Vyššie uvedeným postupom vieme teda ľahko zostrojiť optimálne zaplatenie pomocou mincí s hodnotou 1, 5 a 10.

Čo však s 25centovými mincami, ktoré nie sú násobkom 10centových mincí? Je jasné, že s nimi musíme zaplatiť aspoň toľko, aby sme menšími mincami zaplatili zvyšok. Ale čo ak je výhodnejšie použiť viac 25centových mincí? Niekedy sa nám môže oplatiť použiť o jednu 25centovú mincu viac ako je nutné. (Napr. ak chceme pomocou 1, 1, 1, 1, 1, 10, 10, 10 a 25 zaplatiť 30.) Ukážeme, že nikdy sa nám neoplatí použiť viac 25centových mincí.

Nech sme pri platení použili aspoň o 2 25centové mince viac ako bolo nutné. Tieto dve 25centové mince vieme určite rozmeniť na drobnejšie (25+25=50 - nasobok 10), ktorých je dosť, lebo zvyšnú sumu sme nimi vedeli zaplatiť aj bez týchto dvoch 25centoviek. Takto dostaneme zaplatenie hľadanej sumy pomocou viac mincí.

Máme hotové riešenie.[2] Zistíme, koľko 25centoviek musíme použiť, aby sme zvyšok ešte vedeli zaplatit menšími mincami. Vyskúšame použiť tento minimálny počet 25centoviek a vyskúšame použiť o jednu 25centovku viac. Zvyšnú sumu zaplatíme už raz spomínaným greedy (pažravým) spôsobom.

Uvedený algoritmus má časovú aj pamäťovú zložitosť $O(1)$. Za takto rýchly algoritmus ste mohli získať 15 bodov. Za riešenia v lineárnom čase sa dalo získať 9 bodov, za pomalšie 7 a za nefunkčné najviac 4.

[2] Kiež by boli aj všetky váše riešenia také stručné a prehľadné

Listing programu:

```
#include <stdio.h>

//vrati maximum dvoch argumentov
int max(int a, int b) { return a>b ? a : b; }

//zaplati najvacsou sumou len tolko, aby mensimi zaplatil zvysok
//mince 1, 5, 10
int greedy(int K, int L, int M, int S)
{
	int used1, used5, used10, sum;

	if (K+L*5+M*10<S) return -1;	//neda sa zaplatit

	//pouzijeme najmensi mozny pocet 10centoviek
	sum=S-K*1-L*5;		//mensimi toto uz nezaplatime
	sum=max( 0, sum);	//ak vieme mensimi preplatit, nepouzijeme
	used10= (sum+9)/10;	//horna cela cast sum/10
	S-=used10*10;		//zmensime sumu na zaplatenie

	//to iste pre 5, 1centovky
	sum=max( 0, S-K*1);	used5= (sum+4)/5;	S-=used5*5;
	sum=max( 0, S);		used1= (sum+0)/1;	S-=used1*1;
	return used10 + used5 + used1;
}

//zaplati najvacsou sumou len tolko, aby mensimi zaplatil zvysok
//a vyskusame pridat este jednu mincu
int greedy25(int K, int L, int M, int N, int S)
{
	int used25, sum;

	if (K+L*5+M*10+N*25<S) return -1;	//neda sa zaplatit

	//greedy odhad pre 25centovky
	sum=max( 0, S-K*1-L*5-M*10);	used25= (sum+24)/25;	S-=used25*25;
	return max(
		greedy(K, L, M, S) + used25,
		greedy(K, L, M, S-25) + used25 + 1
	);
}

int main(void)
{
	int K, L, M, N, S, x;

	scanf("%d %d %d %d %d", &K, &L, &M, &N, &S);

	x =  greedy25(K, L, M, N, S);

	if ( x<0 ) printf("Suma sa neda zaplatit\n");
	else printf("Suma da zaplatit najviac %d mincami\n", x);

	return 0;
}
```

opravoval Žužu

3. O trpaslíkoch a diaľnici

(max. 15 bodov)

Fantastické!! Veľmi ste ma potešili. Zdanlivo neľahký príklad má celkom pekné jednoduché riešenie, a veď vlastne všetky prislé riešenia boli správne. Tak moji milí, nebojte sa tangensov či sínusov a vstúpte do tajov vzorového riešenia.

Vzorové riešenie: Geometricky si trpasličiu diaľnicu predstavíme ako nekonečný pás danej šírky a stromy ako body. Úlohou je určiť, či existuje nekonečný pás šírky D, ktorého stredová priamka prechádza bodom $[0,0]$ a neobsahuje žiadny zo zadaných N bodov. Všimnime si, že hľadaný pás predstavuje objekt, ktorý má plochu (pokrýva časť roviny), naopak body plochu nemajú. Lepšie ako hľadať *„plošný“ pás* medzi *„neplošnými“ bodmi* bude hľadať niečo *neplošné* medzi niečím *plošným*.[3]

Myšlienka: Neváhajme a nafúknime každý zadaný bod $[x_i, y_i]$ do kružnice so stredom $[x_i, y_i]$ a polomerom D. Teraz nám stačí hľadať priamku prechádzajúcu bodom $[0,0]$, ktorá sa nepretína so žiadnou kružnicou(zdôvodnite!)[4]. Ak takúto priamku nájdeme, tak pás šírky $2D$, ktorého je stredovou priamkou, vyhovuje podmienkam úlohy (viď. obrázok č.1).

4cm Obrázok č. 1: Priamka p medzi kružnicami určuje pás medzi bodmi.

Každú priamku prechádzajúcu bodom $[0,0]$ jednoznačne charakterizuje uhol, ktorý zviera s x-ovou osou. Hľadáme teda uhol α, ktorému zodpovedá priamka, ktorá nepretína žiadnu kružnicu. Všimnime si, ako nám vo výbere uhlu α bránia kružnice. Každá kružnica ohraničuje (svojími dotyčnicami z bodu $[0,0]$) dva symetrické intervaly uhlov, v ktorých náš hľadaný uhol α nebude (viď. obrázok č.2).

Vieme teda, že v intervaloch, prislúchajucich kružniciam, uhol α byť nemôže. Teraz si stačí uvedomiť, že všetky ostatné uhly vyhovujú našim požiadavkám. Riešenie preto spočíva v zistení, či intervaly, ktoré prislúchajú dotyčniciam ku kružniciam, pokrývajú celých 2π radiánov okolo bodu $[0,0]$.

Ak intervaly pokrývajú celé zorné pole okolo bodu $[0,0]$ – každá priamka prechádzajúca bodom $[0,0]$ pretína nejakú kružnicu (a diaľnica sa nedá postaviť), v opačnom prípade taká priamka existuje (a nej zodpovedajúca diaľnica vyhovuje zadaniu).

4cm Obrázok č. 2: Dotyčnice ku kružnici určujú (stredovo súmerné) intervaly uhlov.

Keďže hľadáme priamku, stačí uvažovať interval uhlov $I =< -\frac{\pi}{2}, \frac{\pi}{2})$. Problémom teda zostáva, keď máme dané intervaly $< a_i, b_i >$ $(a_i, b_i \in I)$ (intervaly, ktoré presahujú uhol $-\frac{\pi}{2}$ alebo $\frac{\pi}{2}$ ošetríme tak, že množstvo, ktoré nejaký interval na svojej strane (intervalu I) presahuje, utvorí nový interval na opačnej strane (intervalu I)) zistiť či pokrývajú celý interval I.

Toto určíme ľahko nasledovným postupom. Ukotvíme polpriamku pod uhlom $-\frac{\pi}{2}$ v bode $[0,0]$ a začneme ju pomaly otáčať smerom k uhlu $\frac{\pi}{2}$. Ak sa počas otáčania nachádzala (pod uhlom α) mimo všetkých intervalov, tak sme našli hľadaný uhol α. V opačnom prípade intervaly pokrývajú celý interval I.

Implementácia: Vyššie uvedené myšlienky potrebujeme efektívne implementovať. Výpočet uhlov dotyčníc sa ľahko zrealizuje funkciami *arctg* a *arcsin* (premyslite si!)[5]. Pre N daných bodov (kružníc) teda dostaneme najviac $2N$ intervalov (v prípade ak by všetky boli na rozhraní intervalu I a museli by sme každý deliť na dva).

Teraz potrebujeme zistiť, či intervaly pokrývajú všetky možné uhly. Využijeme dátovú štruktúru haldu[6]. Každý interval tvoria dva koncové body, ktoré vložíme do haldy (v ktorej

[3] Aj sýteho medzi hladnými nájdete ťažšie ako hladného medzi sýtymi, no nie?

[4] Naviac, pre $\forall\delta$ $(0 \leq \delta \leq D)$ platí: ak by sme body nafúkli do kružníc s polomerom δ, museli by sme hľadať pás šírky $2(D-\delta)$). Pre $D = \delta$ je to priamka, čo je výhodné.

[5] Alebo nahliadnite do zdrojových kódov.

[6] Ak ju nepoznáte, treba viac študovať múdre knižky!

sú body zotriedené vzostupne podľa uhla, ktorý zvierajú s x-ovou osou) spolu s informáciou, či sa jedná o koncový alebo začiatočný bod príslušného intervalu.

Po vložení všetkých bodov budeme z haldy prvky vyberať. Poradie vyberania simuluje otáčanie polpriamky ukotvenej v bode $[0, 0]$, ako sme už naznačili vyššie. Počas tohto procesu si budeme udržiavať hodnodu h (reprezentujúcu v koľkých intervaloch sa aktuálne nachádza polpriamka – na začiatku nastavená na 0), ktorú vždy pri výbere bodu – začiatku intervalu – zvýšime o 1 a pri výbere bodu – konca intervalu – zasa znížime o 1 (pričom v halde, medzi prvkami s rovnakým uhlom uprednostníme začiatočné body intervalov pred koncovými). Ak sa nám počas tohto prechodu dostala hodnota h na 0, vieme, že sme narazili na interval nepokrytý žiadnym zo zadaných.

Poznámka: Ešte pripomenieme, že interval, ktorý presahuje okraj intervalu I netreba rozďeľovať na dva, ale môžeme na neho úplne zabudnúť s tým, že si pamätáme aká veľká časť od okraja $-\frac{\pi}{2}$ a okraja $\frac{\pi}{2}$ je takýmito presahujúcimi intervalmi pokrytá a vyššie uvedený algoritmus spustíme už pre zúžený interval I bez okrajov, ktoré sú pokryté kúskami presahujúcich intervalov.

Zložitosť: Priestorová je bez debaty lineárna vzhľadom na počet (označme ho N) zadaných stromov, časová je zrejme závislá od vykonania $4N$ operácií ($2N$ vložení a $2N$ odstránení) na hlade, ktoré pri štandardnej implementácií[7], trvajú $O(N \log N)$.

Na zamyslenie: Popremýšľajte, ako by ste hľadali konvexný obal množiny kružníc rovnakého polomeru a skúste vymyslieť ďalšie úlohy, v ktorých sa dá pekne využiť metóda „nafukovanie bodov do kružníc“, príp. „spľasnutia kružníc do bodov“. Niekedy sa o tom porozprávame, dobre?

Bodovanie: Prišlé riešenia by sa dali zaškatuľkovať do troch rôznych typov podľa časovej zložitosti[8]:

- Čas $O(N \log N)$ – riešenia založené na popísaných myšlienkach vzorového riešenia, **15 bodov**.
- Čas $O(N^2)$ – riešenia založené buď na myšlienke intervalov uhlov (ktoré avšak nedoriešili poslednú fázu pokrývania optimálne), alebo na systematickom prikladaní diaľnice tesne ku každému z N stromov (overenie správnosti jednej takejto diaľnice trvá čas $O(N)$), **8 bodov**.
- Iné riešenia, väčšinou inžinierskeho typu, ktoré prehľadávali uhly s určitou presnosťou, **4 body**.

Navyše, zo subjektívnych (nespravodlivých) dôvodov som strhával **1-2 body** vtedy, ak som usúdil, že popis je nedostačujúci, alebo program nedôstojne napísaný. Ale z celkového hľadiska, boli riešenia tohto príkladu vypracované pekne. Len tak ďalej!

Keďže sa jazyk C teší medzi riešiteľmi stále väčšej popularite (väčšina riešení tohto príkladu prišla v jazyku C), uvádzame vzorové riešenia naprogramované v jayzku C.

Listing programu:

```
/* 21-2-3, O trpaslikoch a dialnici, Zuzu */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define EPSILON 0.0001
#define MAX_N10000

typedef struct { double x, y; } POINT;
```

[7] Aj tak ju v zdrojákoch neprehliadnite, určite sa čo-to naučíte!

[8] Lineárnu priestorovú zložitosť si netrúfol prekročiť nik.

```
double d;
int n;

/* prvok haldy */
typedef struct { double a; int delta; } ITEM;

/* usporiadanie v halde */
int item_cmp(ITEM *i1, ITEM *i2) {
  if (i1->a>i2->a+EPSILON) return 1;
  if (i2->a>i1->a+EPSILON) return -1;
  if (i1->delta<i2->delta) return 1;
  return -1;
}

ITEM item[2*MAX_N+1];
int h[2*MAX_N+1];
int nh=0;

/* oprav hladu smerom hore */
void up(int i) {
  int j=h[i];

  while (i>1&&item_cmp(&item[j],&item[h[i>>1]])<0) { h[i]=h[i>>1]; i>>=1; }
  h[i]=j;
}

/* oprav smerom dole */
void down(int i) {
  int j=h[i];

  while ((i<<1)<=nh) {
    i<<=1;
    if (i+1<=nh&&item_cmp(&item[h[i+1]],&item[h[i]])<0) i++;
    if (item_cmp(&item[h[i]],&item[j])>0) { i>>=1; break; }
    h[i>>1]=h[i];
  }
  h[i]=j;
}

/* vlozit prvok do hlady */
void insert(int i) { h[++nh]=i; up(nh); }

/* odstranit prvok s najmensim uhlom */
int delete(void) { int i=h[1]; h[1]=h[nh--]; down(1); return i; }

/* pridaj zaciatok/koniec intervalu do haldy */
void event(int i, double a, int d) { item[i].a=a; item[i].delta=d; insert(i); }

int main(void) {
  double px, py, a, b, min=-M_PI/2.0, max=M_PI/2.0;
  int i, j;

  scanf("%d %lf", &n, &d);
  for(i=0;i<n;i++) {
    scanf("%lf %lf", &px, &py);
    if ((px*px+py*py)<(d*d)) { printf("Neda sa postavit.\n"); return 0; }
    if (fabs(px)<EPSILON) { if(py>0) a=M_PI/2.0; else a=-M_PI/2.0; }
      else a=atan(py/px);

    b=fabs(asin(d/sqrt(px*px+py*py)));
```

```
    /* min <- ako vela od -pi/2 je uz pokrytych, max <- detto, ale od pi/2 */
    if (a+b>M_PI/2.0) {
      if (a-b<max) max=a-b;
      if (a+b-M_PI>min) min=a+b-M_PI;
    } else if (a-b<-M_PI/2.0) {
      if (a+b>min) min=a+b;
      if (a-b+M_PI<max) max=a-b+M_PI;
    } else {
      event(i+i,a-b,1);
      event(i+i+1,a+b,-1);
    }
  }

  i=0;
  /* ak je este zaciatok "najprvsieho" intervalu
  nepokryty, tak uhol "kusok" pred nim vyhovuje */
  if (item[h[1]].a>min) { printf("Da sa postavit.\n"); return 0; }
  while (nh) {
    j=delete();
    a=item[j].a;
    i+=item[j].delta;
    if (i==0) break;
  }
  /* to iste s "najposlednejsim" intervalom */
  if(max>a+EPSILON) { printf("Da sa postavit.\n"); return 0; }

  /* intervaly uhlov pokryvaju cele zorne pole */
  printf("Neda as postavit.\n");
  return 0;
}
```

opravoval Dávidko
(max. 15 bodov)

4. O slovnej hádanke

Riešenia boli v zásade dvoch druhov. Jedny boli podobné vzorovému riešeniu, t.j. lineárne. Za tie som dával plný počet bodov. Druhá kopa riešení bola pomalších, zväčša kvadratických a za tie som udeľoval 10 bodov. Vyskytli sa riešenia, ktoré nefungovali, alebo o ktorých funkčnosti mám pochybnosti. Za tie úmerne menej, podľa funkčnosti. Tradične som stŕhal jeden až dva body za popis, chyby v programe, prípadne odhady zložitosti.

Vzorové riešenie. Dĺžku slova označme n a hľadané slovo nech je $w = w_1 w_2 \dots w_n$. Vstupnú permutáciu suffixov[9] označme $a = (a_1, a_2, \dots, a_n)$ a k nej inverznú $p = (p_1, p_2, \dots, p_2)$ (t.j. $p_{a_i} = a_{p_i} = i$ pre všetky i od 1 po n). Ďalej označme suffix začínajúci písmenom w_i ako s_i, teda $s_i = w_i w_{i+1} \dots w_n$. (Z technických príčin s_{n+1} bude prázdne slovo a $a_{n+1} = -1$.)

Ľahko si uvedomíme, že písmená postupne na pozíciách $p_1, p_2, \dots, p_n$ idú v neklesajúcom abecednom poradí. Pozrime sa na pozíciu, kde je na vstupe číslo 1. (Je to pozícia p_1.) Na tejto pozícii je suffix ležiaci prvý v abecede. Preto môžme hravo dať na túto pozíciu písmeno a. Vezmime si teraz pozíciu, kde leží číslo 2 t.j. pozíciu p_2. Tu začína suffix ležiaci druhý v abecede. Ten bude začínať zrejme písmenom a alebo písmenom b. Ak začína písmenom a, tak musia slová vzniknuvšie zo suffixov s_{p_1} a s_{p_2} odtrhnutím prvých písmen ísť v abecede v správnom poradí. Lenže (a tu je geniálnosť celej myšlienky) tieto slová sú tiež suffixy a

[9] Suffix je cudzím slovom koniec slova.

ich poradie v abecede predsa vieme! (Sú to suffixy s_{p_1+1} a s_{p_2+1}.) Takže písmeno a bude na pozícii p_2 práve vtedy, keď $a_{p_1+1} < a_{p_2+1}$. Inak, tam pochopiteľne musí byť písmeno b.

Všeobecne na pozícii p_i bude to isté písmeno ako na pozícii p_{i-1} alebo o jedna väčšie. To isté tam bude práve vtedy, keď $a_{p_i-1+1} < a_{p_i+1}$, inak o jedna väčšie.

Zrátať inverznú permutáciu ide triviálne v lineárnom čase. Zvyšok funguje presne podľa popisu. Algoritmus je zrejme lineárny v čase aj pämati t.j. $O(N)$. Kto neverí nech si prečíta vskutku krátky program.

Listing programu:

```
var a,p,w:array[1..50] of integer; { pole koncov, inverzna permutacia, slovo }
   N,i:integer;

begin
    { nacitame pole koncov a vyrobime inverznu permutaciu }
    i:=0;
    repeat
        inc(i); read(a[i]);
        if a[i]=-1 then break;
        p[a[i]]:=i;
    until false;
    N:=i-1;
    { prebiehame suffixy v abecednom poradi }
    w[p[1]]:=0;
    for i:=2 to N do
       if a[p[i-1]+1] < a[p[i]+1] then w[p[i]]:=w[p[i-1]] else w[p[i]]:=w[p[i-1]]+1;
    { vypiseme slovo }
    for i:=1 to N do write(chr(w[i]+ord('a'))); writeln;
end.
```

opravoval MišoF.
(max. 15 bodov)

5. O mravcoch II

Úvodom tohto riešenia udeľujem pochvalu pred nastúpenou čatou Mirovi Cickovi, ktorého riešenie bolo krásne napísané (vrátane elegantného vyriešenia synchronizácie počítačov) a navyše upozornil na drobnú chybu vo vzorákoch minulého kola.

Úlohou bolo preusporiadať pole tak, aby na začiatku boli prvky menšie ako `Mem[1]`$=X$, potom X a na konci prvky väčšie ako X. Sekvenčne vieme túto úlohu riešiť v lineárnom čase algoritmom od pána Hoareho. Väčšina z vás ho pozná z triedenia QuickSort. Dá sa implementovať napríklad nasledovne: Spočítame si počet prvkov menších ako X. Umiestnime X na správne miesto. Tým sa nám pole rozdelí na dve časti. Obe postupne prechádzame. Vždy, keď v prvej časti nájdeme prvok väčší ako X a v druhej prvok menší ako X, vymeníme ich. Za takéto riešenie (správne implementované a s popisom) sa dalo získať 7 bodov.

Ešte stále sa našlo zopár riešení, ktoré na dosiahnutie lineárneho času potrebovali viac ako 1 počítač. Tieto riešenia boli ohodnotené skromnejším počtom bodov.

Pripomeňme si, že v minulých vzorových riešeniach sme si ukázali algoritmus na spočítanie čiastočných súčtov postupnosti. S úspechom ho použijeme pri našom riešení tejto úlohy.

Myšlienka riešenia bude nasledovná: Prvky menšie ako X si očíslujeme číslami od 1 do k, prvky väčšie ako X očíslujeme číslami od 1 do $n-k-1$. Prvky menšie ako X umiestnime na políčka 1 až k (podľa čísla, ktoré dostanú), prvok X na políčko $k+1$ a väčšie prvky (opäť podľa čísla) na políčka $k+2$ až n.

Začneme tým, že nájdeme všetky prvky menšie ako X. Ak `Mem[i]`$<X$, do `Mem[N+i]` zapíšeme 1, inak 0. Pre každý z týchto prvkov by sme si chceli spočítať, kam ho máme presunúť. To spravíme jednoducho tak, že spočítame čiastočné súčty postupnosti uloženej v `Mem[N+1]` až `Mem[2N]`. Teraz pre každé i také, že `Mem[i]`$<X$ máme v `Mem[N+i]` uložené, koľký prvok menší ako X je v `Mem[i]`. Toto bude zároveň políčko, kam tento prvok umiestnime vo výslednom poli.

Príklad výpočtu.											
Pôvodné pole:	5	11	7	1	12	9	0	0	0	0	0
Zapíšeme 0/1:	5	11	**7**	**1**	12	**9**	0	1	1	0	1
Nájdeme čiast. súčty:	5	11	7	1	12	9	0	**1**	**2**	2	**3**
Výsledok: prvok 7 pôjde do `Mem[1]`, 1 do `Mem[2]`, 9 do `Mem[3]`											

Analogicky do `Mem[2N+i]` zapíšeme 1, ak `Mem[i]`$>X$, 0 inak. Spočítame čiastočné súčty aj v tomto poli. Teraz už vieme umiestniť všetky prvky:

- ak `Mem[i]`$<X$, chceme presunúť obsah `Mem[i]` do `Mem[N+i]`
- ak `Mem[i]`$=X$ (teda $i = 1$), chceme presunúť obsah `Mem[i]` do `Mem[2N]+1`
- ak `Mem[i]`$>X$, chceme presunúť obsah `Mem[i]` do `Mem[2N]+1+Mem[2N+i]`

Podľa týchto myšlienok vieme ľahko napísať program, ktorý úlohu vyrieši v čase $O(\log N)$ na N počítačoch – zapísať hodnoty 0/1 vieme v konštantnom čase, spočítať čiastočné súčty v čase $O(\log N)$. Teraz si každý počítač vezme jeden prvok a spočíta si, kam ho presunúť. Nakoniec všetky počítače naraz zapíšu svoje prvky na správne miesto v poli a vyhrali sme.

Ľubovoľné riešenie, ktoré dosiahlo pomocou N počítačov čas $O(\log N)$, mohlo získať 15 bodov. Práve popísané riešenie chce čo najnázornejšie ukázať postup, akým zo správnej myšlienky dostať zaručene fungujúce riešenie. Ukážeme si teraz riešenie, ktoré bude trochu viac trikové, ale ľahšie sa bude implementovať.

Všimnime si, že hodnoty v `Mem[2N+1]` až `Mem[3N]` vlastne nemusíme počítať – totiž `Mem[2N+k]=k-1-Mem[N+k]`. Inými slovami, ak je medzi prvými k prvkami `Mem[N+k]` menších ako X a jedno rovné X, väčších ako X je zjavne `k-1-Mem[N+k]`. Tiež si všimnime, že nemusíme samostatne ošetrovať hodnotu X, stačí sa zaoberať dvoma skupinami prvkov – menšími ako X a ostatnými. Keďže X je na začiatku výpočtu prvý z ostatných prvkov, bude prvý z ostatných aj na konci. No a presne tam aj má skončiť. Posledná drobná úprava bude, že hneď na začiatku si každý počítač zapamätá v lokálnej premennej „svoju“ hodnotu a čiastočné súčty spočítame priamo na políčkach `Mem[1]` až `Mem[N]`. A teraz už očakávaný program:

```
  var N, X, prvok, kam, dva_na_k : integer;

  if (cpuid > Mem[0]) then halt;
  N:=Mem[0];
  X:=Mem[1];
  prvok:=Mem[cpuid];
  if (prvok<X) then Mem[cpuid]:=1 else Mem[cpuid]:=0;

  dva_na_k := 1;
1: if (cpuid > dva_na_k) then Mem[cpuid] := Mem[cpuid] + Mem[cpuid - dva_na_k];
  dva_na_k := 2 * dva_na_k;
  if (dva_na_k < N) then goto 1;

  if (prvok<X) then kam:=Mem[cpuid] else kam:=Mem[N]+cpuid-Mem[cpuid];
  Mem[kam]:=prvok;
```

Tí, ktorí poriadne čítali vzorové riešenia minulej série, si pravdepodobne všimli na konci pár odsekov o *práci* paralelného algoritmu. Tam sme si povedali, že *práca* je čas behu algoritmu krát počet potrebných počítačov – inými slovami je to práca, ktorú vykoná sekvenčný algoritmus, keby chcel náš paralelný algoritmus simulovať. Optimálny[10] by bol taký paralelný algoritmus, ktorý by mal čo najlepšiu časovú zložitosť a pritom prácu rovnakú, ako sekvenčný algoritmus pre danú úlohu.

Minule sme si ukázali, ako spočítať čiastočné súčty v čase $O(\log N)$ pomocou $N/\log N$ počítačov (teda s lineárnou prácou). Teraz ukážeme, že aj na riešenie našej úlohy nám stačí $N/\log N$ počítačov. Každý počítač bude „obsluhovať“ úsek približne $\log N$ prvkov poľa. Zapísanie hodnôt 0/1 nám bude trvať už nie konštantný čas, ale až $O(\log N)$ – každý počítač musí porovnať hodnoty zo svojho úseku s hodnotou X. Teraz v $O(\log N)$ spočítame čiastočné súčty. Opäť v $O(\log N)$ si každý z počítačov uloží napr. do lokálnej pamäte svoje prvky a spočíta si, kam ich má presunúť. Nakoniec ich tam v čase $O(\log N)$ presunie.

Toto riešenie je (aj podľa kritérií uvedených v zadaní!) lepšie od riešenia s N počítačmi. Keby ho bol niekto implementoval, dostal by aj viac ako 15 bodov. Nestalo sa.

[10] Napr. kvôli distribuovaným výpočtom, viď minulé vzoráky

Výsledková listina po 2. kole kategórie KSP

	Meno a priezvisko	Škola	Ročník		21	22	23	24	25	Σ
1	Perešíni Peter	Gym. Tajovského B. Bystrica	2	67	15	15	14	15	15	141
2	Jančuška Marek	Gym. Párovská Nitra	4	71	15	15	8	15	15	139
3	Cicko Miroslav	Gym. Tajovského B. Bystrica	3	63	15	15	15	15	15	138
4	Poláček Lukáš	Gym. K. Štúra Modra	4	60	15	15	15	15	15	135
5	Imriška Jakub	Gym. Jura Hronca BA	2	61	15	15	15	10	15	131
6	Simančík František	Gym. Grösslingová BA	3	62	15	9	15	15	14	130
7	Lenhardt Rastislav	Gym. Jura Hronca BA	4	64	15	15	8	15	12	129
8	Bernát Marek	Gym. K. Štúra Modra	4	62	15	4	15	15	15	126
9	Glaus Peter	Gym. Jura Hronca BA	4	65	12	9	8	15	15	124
10	Nánási Michal	Gym. Jura Hronca BA	3	61	12	4	15	15	15	122
11	Kováč Jakub	Gym. Jura Hronca BA	4	47	15	15	15	15		107
12	Buštor Stano	Gym. Jura Hronca BA	3	46	12	4	15	13	13	103
12	Hrinčár Peter	ZŠ Jarná Poprad	0	46	12	7	15	10	13	103
14	Rejda Martin	Gym. Grösslingová BA	4	42	15	15	9	15	6	102
15	Smažáková Dana	Gym. Jura Hronca BA	4	59			15	15		89
16	Kuchynárová Mariana	Gym. Jura Hronca BA	4	56		15		15		86
17	Plžík Milan	Gym. Ľ. Štúra Zvolen	3	51	2	4	8	8	12	85
18	Ludha Marek	Gym. Tajovského B. Bystrica	4	32	15	9		15	7	78
19	Kuštárová Tamara	Bilingválne gym. Sučany	3	36	2	4	15	5	15	77
19	Tekeľ Jakub	Gym. Jura Hronca BA	4	47			15		15	77
21	Bundala Daniel	Gym. Jura Hronca BA	2	35	12	7		15	7	76
21	Petruľák Matúš	Gym. Grösslingová BA	3	34	15	4	15		8	76
23	Beleš Lukáš	Gym. Čadca	3	32	11	15	3	5	7	73
24	Zeman Marek	Gym. Jura Hronca BA	3	39	12	4		10	6	71
25	Hrobár Miso	Gym. Jura Hronca BA	4	12	12	15	15	15		69
26	Trejbal Ivan	Gym. Jura Hronca BA	4	39	12	4	8		5	68
27	Kundrák Ľubomír	Ev. lýceum BA	1	29	10	7	1	8	6	61
28	Štefanek Anton	Gym. Jura Hronca BA	4	0	15	15	15	15		60
29	Schlosáriková Eva	Gym. SNP Piešťany	3	30	8	4	2	10	3	57
30	Kevický Michal	Gym. Grösslingová BA	4	50						50
31	Baláž Miroslav	Gym. Jura Hronca BA	4	48						48
32	Hanulová Anna	Gym. Jura Hronca BA	4	24		4	15			43
32	Repovský Michal	Gym. Komenského Trebišov	4	43						43
34	Trenkler Pavol	Gym. Zbrojničná Košice	4	37						37
35	Vadkerti Miroslav	SPŠE Nové Zámky	4	18	10	7				35
36	Dzetkulič Michal	Gym. P. Horova Michalovce	3	34						34
36	Šomlo Ivan	Gym. Mládežnícka Šahy	4	34						34
38	Krchniak Matej	Gym. Jura Hronca BA	2	18	2	4		5	2	31
38	Pančík Andrej	Gym. Tajovského B. Bystrica	1	12		4		15		31
40	Mindek Peter	Škola pre mim. nadané deti BA	2	20	9					29
41	Mikuláš Ján	Gym. Haličská Lučenec	2	27						27
42	Štolc Miroslav	Gym. Párovská Nitra	4	24						24
43	Gottweis Martin	Gym. Jura Hronca BA	2	23						23
44	Bodnár Jozef	Gym. Fiľakovo	3	20						20
45	Ďurfina Lukáš	Gym. Golianova Nitra	3	2				8		10
46	Ragány Peter	SPŠE Prešov		8						8
46	Sedlák Štefan	Gym. Daxnera V.n. Topľou	3	8						8
48	Oremus Vladimír	Gym. Jura Hronca BA	2	1						1
49	Morihladko Peter	Gym. Školská Spiš. Nová Ves	3	0				..		0