Korešpondenčný seminár z programovania
XXIII. ročník, 2005/06
Katedra základov a 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.
Gnome spol. s r.o.

Vzorové riešenia 1. kola letnej časti, kat. Z

Mili nasi riesitenia, mile nase riesitelky

Takze tu mame priebezne poradie letnej casti KSP-Z. Gratulujem priebeznym vitazom, ale opakujem, toto nie je konecne poradie, takze vsetko sa este moze zmenit. Tak sa pustite do studovania vzorakov, nech ste o kus mudrejsi...

1. Zase tie podvody...

opravoval Mic
(max. 15 bodov)

Najskôr si asi povieme, ako som bodoval.

- Za časovú zložitosť $O(M)$ som udeľoval 15 bodov.
- Riešenia bežiace v čase $O(M^2)$ dostali najviac 12 bodov.
- Za časovú zložitosť $O(MN)$ som udeľoval najviac 10 bodov.
- Nefunkčné riešenie mohlo dostať najviac 4 body.
- Za chýbajúci alebo chybný odhad zložitosti som strhával po bode.
- Za chýbajúci alebo nedostačujúci popis som strhával aj 4 body.
- Riešeniam písaným v Befunge som strhával body podľa nálady.

No a teraz sa už môžeme pozrieť na vzorové riešenie...

Najprv spravíme jedno dôležité pozorovanie. Vždy dávame karty iba na vrch kôpok, takže ak sa nejaká karta nedostane na spodok v prvom rozdávaní, nedostane sa tam už nikdy. Vieme tiež, že po prvom rozdaní bude na spodku i-tej kopy karta číslo i. Povedané inak, stačí zistiť, ktorá kôpka vydrží najdlhšie – z ktorej nebudeme nikdy karty rozdávať – a budeme mať výsledok.

Ako vyzerá jedno rozdanie kariet? Máme na ruke N kariet a začíname rozdávať na M kôp. Kopy budeme číslovať od 0 po $M-1$, ako uvidíme neskôr, takéto číslovanie sa nám bude hodiť. Začíname teda rozdávať od nultej kopy. Ak máme na ruke miniálne M kariet, tak na každú kopu dáme jednu kartu a z ruky nám ubudne M kariet. Toto budeme robiť, až kým nebudeme mať na ruke menej ako M kariet, ich počet označíme a. Teraz ideme rozdávať týchto a kariet, a teda poslednú kartu položíme na kopu s číslom $a-1$. Kopa, ktorú zrušíme (a začneme z nej rozdávať) bude mať teda číslo a.

Koľko bude mať tá kopa kariet? A ako jednoducho vypočítame číslo a? Predtým, než sme začali rozdávať posledné kolo (to, v ktorom sme rozdali už iba a kariet), mala každá kopa rovnako veľa kariet, konkrétne N/M (delenie je celočíselné[1]). No a číslo a sa rovná zvyšku po delení N/M ($N \bmod M$). Takže tieto dve čísla vieme vyrátať v konštantnom čase. Ako bude teraz vyzerať náš program?

V každom kroku vyhodíme jednu kopu, zapamätáme si jej číslo, všetkým kopám, ktoré majú číslo väčšie ako číslo vyhodenej kopy znížime číslo kopy o 1 (ak sme mali k kôp, teraz budeme mať $k-1$ kôp očíslovaných od 0 po $k-2$). Toto opakujeme, až kým nezostane jediná kopa. Tá bude mať, pochopiteľne, číslo 0. Ale aké mala pôvodne číslo? Niekedy sme

[1] karty predsa trhať nebudeme

jej možno číslo zmenšili! Bolo to vtedy, ked jej číslo bolo väčšie ako číslo vyhodenej kopy. Jej nové číslo teda bolo väčšie alebo rovné číslu poslednej vyhodenej kopy. To znamená, že ak aktuálne číslo zostavájucej kopy je väčšie alebo rovné číslu poslednej vyhodenej kopy, tak pred jej vyhodením mala naša kopa číslo o jedna väčšie. Takto prechádzame postupne kopy od poslednej vyhodenej po prvú a upravujeme číslo vyhodenej kopy.

Teraz to už len treba rozumne implementovať. Postačí nám jedno pole veľkosti M (pole *odpad*), v ktorom si budeme pamätať akurát čísla kôp, ktoré sme vyhodil. Na začiatku si nastavíme veľkost kopy na N. Potom $(M-1)$-krát vyhodíme jednu kopu. Vyrátame si, ktorú kopu pôjdeme vyhodiť a koľko má tá kopa na sebe kariet. Potom namiesto toho, aby sme začali rozdávať z nasledujúcej kopy, tak tie karty, ktoré sme rozdali v poslednom kole, zoberieme naspäť a začneme rozdávať od začiatku (ináč povedané prirátame číslo kopy k počtu kariet, ktoré na sebe mala). Teraz nám už iba ostáva znížiť počet kôp o 1. (Všimnite si, že nikde neposúvame karty, ktoré boli za vyhodenou kopou, v skutočnosti si čísla kôp, tak ako boli pôvodne, ani nikde nepamätáme. A ani to nie je treba.) Ak nám už ostane iba jedna kopa (pochopiteľne, s číslom 0), začneme prechádzať čísla vyhodených kôp od konca. Ak je číslo vyhodenej kopy menšie alebo rovné číslu zostavájúcej kopy, tak zvýšime číslo zostavájúcej kopy o 1. Potom nám na konci už neostáva nič iné, iba toto číslo vypísať.

Aká bude časová zložitosť tohto algoritmu? Vykonávame M-krát konštantný počet operácií, takže to bude $O(M)$. Pamäťové nároky sú $O(M)$.

A teraz si bežte prečítať kód...

Listing programu:

```
program Karty;
  const MaxN=100000;
var N,M,kopa,ostala,i:integer;
    odpad:array[1..MaxN]of integer;
begin
  readln(N,M);
  kopa:=N;
  for i:=0 to M-2 do begin
    odpad[i]:=kopa mod (M-i);
    kopa:=N div (M-i) + odpad[i];
  end;
  ostala:=0;
  for i:=M-2 downto 0 do if(odpad[i]<=ostala)then Inc(ostala);
  writeln('Na spodku kopy ostane karta cislo ',ostala+1);
end.
```

2. Zrovnoprávnenie guličiek

opravoval Miňo
(max. 15 bodov)

Riešenie zrovnoprávnenia guličiek bolo vcelku jednoduché a krátke. Finta bola však v tom, ako naň prísť, čo sa už nepodarilo každému. Poďme však pekne po poriadku.

Z riešení, ktoré prišli, by sa dali vyrobiť dve základné kôpky. Prvá z nich boli zvyčajne riešenia v $O(N^2)$, využívajúce nejakú simuláciu prehadzovania guličiek s rozumnými predpokladmi. Ďalšia kôpka boli riešenia pracujúce v $O(N)$, na ktoré už bolo treba trošku prísť. Tu sa dajú vyrobiť ešte dve podkôpky a to riešenia s pamäťovou zložitosťou $O(N)$ a $O(1)$.

Prideľovanie bodíkov sa riadilo takýmito pravidlami: Riešenia s lineárnou časovou zložitosťou získali maximálne 15 bodov. Pokiaľ niekto mal lineárne obe zložitosti, skóre bolo najviac 12 bodov. Za $O(N^2)$ sa dalo získať maximálne 9 bodov. Za chýbajúci alebo chybný odhad zložitosti sa strhávalo po bode za jednu zložitosť. Chýbajúci dôkaz správnosti (ktorý

bol hrubo zvýraznený aj v zadaní) sa oceňoval jedným strhnutým bodom. Chýbajúci popis riešenia mal zrážku ďalšie dva body. Chybné riešenia dostávali najviac 4 body :-).

Ešte zopár slov k popisom – v niektorých riešeniach boli dosť slabé. Popis nemá byť iba prepis algoritmu – má vedieť vysvetliť človeku, ktorý váš kód nevidel a nepozná myšlienku riešenia, ako presne funguje a prečo tak funguje. Netreba zabudnúť, že popis čítajú tiež ľudia a ak ho napíšete nezrozumiteľne, človek na druhom konci mu nebude rozumieť. Takže, pre rýchlo opravené riešenia treba pekne písať popisy, ale nezabudnúť zase ani na programy.

Vzorové riešenie funguje v lineárnej časovej a konštantnej pamäťovej zložitosti. Hlavná myšlienka je asi takáto: predstavme si, že stojíme medzi i-tou a $(i+1)$-vou jamkou a zatiaľ sme nepohli žiadnou guličkou. Naľavo od nás je q_i guličiek, potom napravo musí byť $n - q_i$ guličiek. Naľavo od nás je i miest, teda $i - q_i$ guličiek je nazvyš. Ak je $i < q_i$, $q_i - i$ guličiek na tejto strane chýba. V prvom prípade prejde $i - q_i$ guličiek z i-teho miesta na $(i+1)$-vé, v druhom prípade $q_i - i$ guličiek opačným smerom. Alebo, ak chcete, $|i - q_i|$ guličiek prejde cez medzeru medzi i-tou a $(i+1)$-tou jamkou. Týmto sme zistili dolný odhad pre potrebný počet presunov. Stačí teda pre každú medzeru zistiť hodnotu q_i a spočítať súčet hodnôt $|i - q_i|$.

Ukážeme, že tento dolný odhad je zároveň aj horným odhadom – v každom kroku nám stačí so sebou niesť $|i - q_i|$ guličiek (ak je $i - q_i$ záporné, tak si to predstavte tak, že ich prenášame opačným smerom). V každom kroku si budeme pamätať rozdiel $i - q_i$. Na začiatku pre $i = 0$ je $i - q_i = 0$. Pri presune na ďalšiu jamku najskôr zoberieme všetky guličky z tejto jamky – zvýšime hodnotu q_i. Zároveň sa zvýši hodnota i o jedna, takže by sme tu mali jednu guličku nechať. Ak je $q_i > i$, tak si situáciu vieme ľahko predstaviť – v ruke máme $q_i - i$ guličiek, jednu z nich necháme v jamke, ostatné nesieme so sebou. V opačnom prípade nemáme v ruke žiadne guličky a musíme si požičať. Predstavme si, že vykonávame rovnaký postup, ale z druhej strany. Potom namiesto q_i máme $n - q_i$ guličiek a namiesto i máme $n - i$ jamiek. Teda keby sme išli z druhej strany, máme v ruke $(n - q_i) - (n - i) = i - q_i$ guličiek. Kedže je $q_i \leq i$, potom $i - q_i \geq 0$ a teda v ruke budú nejaké guličky a nemusíme si požičiavať. Znova tu jednu guličku necháme a ostatné berieme so sebou.

Vo vzoráku sa používa jedna celočíselná premenná – ak je kladná, máme guličky navyše, ak záporná, požičané. Absolútna hodnota tejto premennej je potom počet ťahov, ktorý vykonáme pri prechode na ďalšie políčko.

Listing programu:

```
var naviac, tahov, n, i: Integer;

begin
    naviac := 0; tahov := 0; n := 0; i := 0;
    read(n);
    while n > 0 do
    begin
        read(i);
        Inc(naviac, i-1);
        Inc(tahov, abs(naviac));
        Dec(n);
    end;
    writeln(tahov);
end.
```

3. Zaujímavé a ešte zaujímavejšie predmety

opravoval Zemčo
(max. 15 bodov)

Pri opravovaní tohto príkladu som sa opäť raz presvedčil, že vaša fantázia je pomerne bujará. Poslali ste totiž vcelku dosť riešení rôznych zložitostí, ktoré mali väčšinou aspoň tú vlastnosť, že boli korektné. Najskôr si ale uzrejmime písmenká – v ďalšom texte budeme veľkosť rozvrhu značiť N a počet predmetov P. Medzi tie pomalšie z vašich kreácií patrili napríklad riešenia bežiace v čase $O(NP)$, ale ako som naznačil, dalo sa vymyslieť aj všeličo iné. Snáď najčastejšie sa asi vyskytoval čas $O(N + P \log P)$.

Pri tomto riešení sa musíme zastaviť. Fungovalo tak, že sa načítali predmety, zotriedili sa podľa priority a potom sa postupne nahadzovali do rozvrhu – poľa veľkosti N. Keď prišlo k situácii, že sa vkladalo do rozvrhu na obsadené miesto, označil sa vkladaný predmet za výsledok. S dobrým pocitom ste takéto riešenia označovali ako $O(P \log P)$, čo je náročnosť triediaceho algoritmu (mnohí ste použili Quicksort, ale vyskytli sa aj iné). Táto klasifikácia žiaľ nie je správna. Súvisí to s odvekým problémom ľudstva – časová zložitosť je minimálne taká, aká je pamäťová náročnosť algoritmu[2]. Alokácia pamäte nás totiž stojí čas úmerný jej veľkosti. Už len tým, že ju napríklad musíme nulovať (prípadne použiť `fillchar` alebo `memset`). Niekedy sa to síce robí automaticky, ale to neznamená, že sa to vykonáva v konštantnom čase. Pamätajte, že každý slušný programátor si na to dáva veľký pozor. A kedže používame pole veľkosti N, uvádzame to aj v časovej zložitosti. N môže byť ale o dosť viac ako $P \log P$, takže časová zložitosť má výslednú podobu $O(N + P \log P)$.

A takto sa dostávame k vzorovým riešeniam. Tie sú tento raz dve, pretože o nich nevieme jednoznačne povedať, ktoré je lepšie. Sú to riešenia s časom $O(P \log P)$ a $O(N+P)$. V zadaní bolo napísané, že $P, N < 10000$, pričom o vzťahu N a P nevieme nič. Keď si porovnáme uvedené dve zložitosti, vidíme, že pri väčšine vstupov je menej $N + P$, ale nie pre všetky. Ak máme najväčšie možné $N = 10000$, tak pre P rádovo 1000 a menej platí, že $P \log P < N+P$. Takže obidve tieto riešenia budeme označovať ako optimálne a obidve si spomenieme, aj keď $N + P$ je predsa len vo väčšine prípadov lepšie, preto ho označíme ako „hodnotnejšie optimálne riešenie“ a budeme mu venovať aj viac pozornosti.

Takže, ako funguje riešenie v čase $O(P \log P)$? My sa vlastne len potrebujeme nejako zbaviť toho poľa s rozvrhom. Budeme postupovať takto – na začiatku si predmety zotriedime (teda, až po tom ako ich načítame). Nebudeme však triediť podľa priority, ale podľa času, v ktorom začínajú. Potom jedným alebo niekoľkými málo prechodmi vieme určiť, ktoré predmety nemôžeme navštevovať, a ktorý z nich je najväčší smoliar. Využijeme, že máme predmety z rovnakých časov spolu.

Riešenie v čase $O(N + P)$ je o niečo jednoduchšie. Pri ňom postupujeme tak, že máme pole veľkosti N a ako predmety načítavame, tak ich do tohto poľa označujúceho rozvrh zapisujeme. Pri tomto nám môže nastať niekoľko situácií. V prípade, že predmet nemôže byť do rozvrhu umiestnený, musíme ho zahodiť, pričom samozrejme pozeráme, či nie je prioritnejší ako ten najväčší doteraz zistený smoliar. Ak je na príslušnom mieste v rozvrhu nejaký neprioritný predmet, zahadzujeme ten, pričom opäť porovnávame s tým, ktorý je v danej chvíli evidovaný ako najväčší smoliar. Tu môže prísť k zaujímavému prípadu – vyhodíme predmet, ktorý je menej prioritný ako práve umiestňovaný, ale zavadzia nám len jedným políčkom. Čo spravíme s tým druhým, ktoré nám momentálne nezavadzia a teda ho neprepisujeme? Ono je už neaktívne, t.j. na to miesto pokojne môže ísť nejaký iný, menej prioritný predmet. Nenarobí nám to neporiadok? No, možno niekoho prekvapí odpoveď – vôbec sa oňho nemusíme zaujímať. Poďme si dokázať/ukázať, prečo. Ak by to neaktívne políčko mal zabrať nejaký prioritnejší predmet, tak nás to aj tak neovplyvní. Čo ak by ale ten nevyhodený záznam o neaktívnom predmete mal spôsobiť, že sa do rozvrhu nedostane nejaký predmet, ktorý tam patrí? To je vyriešené tým, že ten nespravodlivo vyradený predmet bude

[2] minimálne v takomto 'štandardnom' programovaní to platí

mať menšiu prioritu ako ten neaktívny, a preto určite nebude výsledkom nášho programu – bude ním ten, ktorý je neaktívny, ale má tam okienko (alebo nejaký iný – prioritnejší). Pamätajme, že žiadne predmety nemajú rovnakú prioritu.

Po spracovaní všetkých predmetov máme hotový aj výsledok. Tým že spracovávame pri načítavaní, nemusíme si predmety pamätať, a preto máme veľmi lichotivú pamäť $O(N)$. A nemali by sme, samozrejme, zabudnúť na prípad, že všetky predmety navštíviť môžeme.

Táák bubáčikos [3], toľko k riešeniu, teraz niečo k bodovaniu. Za riešenie $O(NP)$, alebo $O(P^2)$ sa udeľovalo najviac 10 bodov. Za riešenia $O(N+P\log P)$ ste mohli získať 13 bodov a nakoniec za niektoré z optimálnych 15. Za rôzne chyby ste mohli stratiť rôzny počet bodov a za nekompletný popis neboli kompletné ani body. Mnohí ste zabudli alebo uviedli nesprávny odhad časovej a/alebo pamäťovej zložitosti, čo sa tiež trestalo stratou bodu.

Listing programu:

```
var priority,rozvrh:array[1..10000]of integer;
    x,y,toppriorita,topsmoliar,dlzka,predmetov,i:integer;

begin
  toppriorita:=0; topsmoliar:=0; readln(dlzka,predmetov);

  for i:=1 to dlzka do begin rozvrh[i]:=0; priority[i]:=0; end;
  for i:=1 to predmetov do begin
    readln(x,y);
    if (priority[x]<y) and (priority[x+1]<y) then begin
      if priority[x]>priority[x+1] then begin
        if priority[x]>toppriorita then begin
          toppriorita:=priority[x];
          topsmoliar:=rozvrh[x];
        end;
      end else begin
        if priority[x+1]>toppriorita then begin
          toppriorita:=priority[x+1];
          topsmoliar:=rozvrh[x+1];
        end;
      end;
      rozvrh[x]:=i;
      rozvrh[x+1]:=i;
      priority[x]:=y;
      priority[x+1]:=y;
    end else if toppriorita<y then begin
        toppriorita:=y; topsmoliar:=i;
    end;
  end;

  if topsmoliar=0 then writeln('Mozme stihnut vsetky predmety')
  else writeln(topsmoliar);
end.
```

[3] ako by povedal istý nemenovaný učiteľ matematickej analýzy.

4. Znížené ceny, zľavy ...

opravoval Peťo
(max. 15 bodov)

Lenivci! Väčšina z vás sa nezamyslela nad niečím *rádovo* rýchlejším ako obyčajné porovnávanie každej ceny s každou. A pritom existujú dve skvelé riešenia v čase $O(m \log m)$. Podľa zložitosti sa dá vytušiť[4], že najprv ceny utriedime vhodným triediacim algoritmom.

Prvé z nich si po utriedení pozrie každú cenu. No a ku každej cene C *binárnym vyhľadávaním* zisťuje, či sa medzi cenami nachádza k nej „vhodná“ cena $N - C$. Tu sa však binárnemu vyhľadávaniu venovať nebudeme. Jediné, čo povieme je, že *binárne vyhľadať* nejakú cenu trvá $O(\log m)$ a robíme to pre všetky ceny, teda celková zložitosť je $O(m \log m + m \log m) = O(m \log m)$.

Druhé, ešte lepšie riešenie začne po utriedení pozerať ceny po dvojiciach. Dôležité je ako. Budeme mať dva ukazovatele, ktoré budú ukazovať na nejaké miesta v poli. Na začiatku bude jeden úplne „naľavo“ na najmenšej cene a druhý úplne „napravo“ na najväčšej cene.

Teraz v každom kroku urobíme toto: skontrolujeme, či dve ceny, na ktoré sa pozeráme, nemajú súčet N, ak áno, tak môžeme skončiť. Ak nie, posunieme jeden z tých ukazovateľov bližšie k tomu druhému. Ako ho posunieme? No ten ukazovateľ, čo je naľavo, posúvame vždy len doprava a ten, čo je napravo, vždy len doľava. Ak je aktuálny súčet cien S, na ktoré ukazujeme, menší ako nami hľadaná cena N, tak posunieme ukazovateľ, čo je naľavo, o jedno miesto doprava. Tým sa nám cena S určite nejako zvýši[5]. Ak je $S > N$, posunieme ten ukazovateľ, čo je napravo, o jedno miesto doľava. Takto pokračujeme, až kým sa nám ukazovatele neprekrížia, alebo kým sme nenašli správny súčet. Ak sa prekrížia, môžeme prehlásiť, že problém nemá riešenie.

Teraz je otázka, či to naozaj funguje, a teda, či vždy nájdeme riešenie, ak sa tam nachádza. Ja tvrdím, že áno. Ak by sme riešenie nenašli, znamenalo by to, že sme sa ani raz nepozerali na obe vyhovujúce ceny naraz. Dôležité je si všimnúť, že na každú cenu sa pozrieme aspoň raz. Zoberme si už utriedenú postupnosť. Máme dva ukazovatele l a p a dva indexy $i \leq j$ také, že $cena_i + cena_j = N$. Na začiatku $l = 1$, $p = m$, takže $l \leq i \leq j \leq p$. Náš algoritmus postupne zväčšuje l alebo zmenšuje p. Na oba indexy i, j sa určite niekedy pozrieme. Zatiaľ sa tvárime, že to nebude v ten istý moment, lebo sme akože nenašli riešenie a na jeden z indexov sa musíme teda pozrieť skôr ako na druhý. Nech je to BUNV[6] index i. Potom $l = i \leq j < p$. Vďaka utriedeniu cien vieme, že $N = cena_i + cena_j \leq cena_l + cena_p$. Ak by platilo, že $cena_i + cena_j = cena_l + cena_p$, tak sme práve našli riešenie, takže platí $cena_i + cena_j < cena_l + cena_p$. V takomto prípade náš algoritmus posúva vždy len pravý ukazovateľ smerom doľava. Takže p sa približuje k j. No a keďže náš algoritmus posúva ukazovatele vždy po jednotlivých cenách, musí nastať moment keď $l = i \wedge p = j \Rightarrow cena_l + cena_p = cena_i + cena_j = N$. Teda sme našli riešenie.

Pamäťová zložitosť je zrejmá. Pamätáme si všetky ceny, n, m, a nejaké pomocné premenné, čo znamená $O(m)$, lebo m je počet cien. Časová zložitosť druhého riešenia nehľadiac na utriedenie cien je $O(m)$. Pretože v najhoršom prípade, keď tam nie je riešenie, vykonávame cyklus, ktorý pozerá všetky ceny, kým sa ukazovatele neprekrížia. Cien je stále m a v každom kroku posunieme jeden ukazovateľ, teda preskočíme jednu cenu.

Pokiaľ nie je vyslovene napísané, že ceny sú celé čísla v nejakom rozsahu tak používame nejaké štandardné triedenie s časom $O(m \log m)$, napr. heapsort, quicksort alebo mergesort. Teda celková časová zložitosť nášho druhého riešenia je $O((m \log m) + m) = O(m \log m)$.

Ako sa hodnotilo:

- časová zložitosť $O(m \log m)$ – 15 bodov

[4] ak nájdem ľahké riešenie $O(m^2)$, tak skoro určite existuje niečo rýchlejšie. Najbližšia možnosť je $O(m \log m)$. No a to mi rovno napovedá, že sa asi bude triediť a...

[5] lebo ceny sú utriedené zľava doprava, od najmenšej po najväčšiu, jedine že by ostala rovnaká, ak máme dve rovnaké ceny

[6] Bez ujmy na všeobecnosti

- časová zložitosť $O(m^2)$ – 9 bodov
- za slabý až žiadny popis som strhával 1 až 3 body
- za nesprávny alebo žiadny odhad zložitosti ste mohli získať -1 bod

Listing programu:

```
const maxM=100;
var ceny:array[1..maxM] of integer;

procedure vymen(a,b:integer);
begin
   if (a=b) then exit;
   ceny[a]:=ceny[a]+ceny[b];
   ceny[b]:=ceny[a]-ceny[b];
   ceny[a]:=ceny[a]-ceny[b];
end;

procedure halduj(otec,m:integer);
var syn:integer;
begin
   while (2*otec <= m) do begin
      syn:=2*otec;
      if (syn<m) and (ceny[syn+1] > ceny[syn]) then inc(syn);
      if (ceny[syn] > ceny[otec]) then vymen(otec,syn);
      otec:=syn;
   end;
end;

var n,m,i,lavy,pravy:integer;
begin
   readln(n,m);
   for i:=1 to m do read(ceny[i]);
   {nazaciatok si ceny utriedime. budem pouzivat nami vybrany a oblubeny heapsort}
   for i:=(m div 2) downto 1 do halduj(i,m);
   for i:=m downto 1 do begin vymen(1,i); halduj(1,i-1); end;
   {samotne pozeranie cien:}
   lavy:=1;
   pravy:=m;
   while(lavy <= pravy)do begin
      if(ceny[lavy]+ceny[pravy] = n)then break;
      if(ceny[lavy]+ceny[pravy] > n)then dec(pravy)
      else inc(lavy);
   end;
   {jednoduche, nie?}
   if(ceny[lavy]+ceny[pravy] = n)then writeln('Ano kupuj ',ceny[lavy],' +
',ceny[pravy])
   else writeln('Pridi nabuduce');
end.
```

5. Zaútočme na chladničku

opravoval Stano
(max. 15 bodov)

Nuž, táto úloha bola zaujímavá :-). Vzhľadom na to, že varenie je skôr umenie než práca pre počítač, optimálne riešenie na najchutnejší recept z hocijakých surovín pravdepodobne neexistuje. Ale väčšinou ste to zvládli celkom dobre, recepty boli zaujímavé, takisto ako vaše rozličné prístupy k nim.

Najjednoduchšie riešenia boli založené na nejakom konštantnom recepte, ku ktorému sa pripisovali jednotlivé suroviny. Niektoré boli vskutku minimalistické, niektoré boli trochu krajšie.

Lepšie riešenia by sa dali zaradiť do 2 kategórií: také, čo sa snažili na základe analýzy vstupných surovín rozdeliť ich do kategórií a podľa toho, čo práve máme v chladničke sa pokúsili vygenerovať čo najchutnejšie jedlo a druhý spôsob spočíval v udržovaní databázy receptov, pričom sa vyhľadávalo podľa zadaných surovín čo najlepší recept. Oba tieto postupy som považoval za dosť dobré, a podľa toho, ako boli prepracované sa dali považovať za skoro vzorové riešenie.

Bodíky sa dali získať za dobrý popis (ktorý často nebol taký, ako by sa patrilo), výstup z príkladov, ktoré sme dali v zadaní, pekný výstup programu (respektíve inteligentný – niektoré programy vedeli skloňovať, alebo si pamätali pre každú surovinu spôsob jej prípravy ...). Hlavné kritérium bol spôsob, akým ste pristupovali k tvorbe receptu. Za absenciu programu som takisto ubral bodíky.

Ako by malo vyzerať vzorové riešenie? Ideálne by bolo, keby malo databázu receptov, v ktorých by mohlo vyhľadať recept, na ktorý potrebujem najviac surovín, alebo recept, ktorý má najväčšiu prioritu (chutí najlepšie `:)`). Ak by žiadny recept podmienkam nevyhovoval, mohlo by na základe surovín, ktoré máme, určiť čo najlepší recept (napríklad, keď máme veci na cesto, môžeme spraviť pizzu... Tá sa dá jesť naozaj so všetkým...). A samozrejme dobrý popis.

Výsledková listina po 1. kole kategórie KSP-Z

	Meno a priezvisko	Škola	Trieda	11	12	13	14	15	Σ
1	Brada Miroslav	Gym. Varšavská Žilina - Vlčince	2	10	14	14	15	9	62
1	Hapák Samuel	Gym. Grösslingová BA	2	15	12	10	13	12	62
3	Nagy Kristián	Gym. Jura Hronca BA	1	12	11	13	9	13	58
4	Vojtko Jakub	Gym. Jura Hronca BA	1	12	15	15	9	6	57
5	Herencsár Albert	Gym. maď. Galanta	1	11	8	14	8	13	54
6	Fecko Stanislav	Gym. Pankúchova Bratislava	3	10	8	10	9	14	51
6	Kostolányi Peter	Gym. Jura Hronca BA	2	12	11	15	13		51
8	Uherčík Tomáš	Gym. P. de Coubertina Piešťany	3	12	8	10	7	6	43
9	Korcsok Peter	Gym. Mládežnícka Šahy	2	12	9	12	9		42
9	Synak Peter	Gym. Jura Hronca BA	2	12	9	12	9		42
11	Blaho Pavol	Gym. Jura Hronca BA	2		14	13	13		40
12	Košinárová Alena	Gym. Grösslingová BA	2			15	12	10	37
13	Hlavatý Peter	Gym. Jura Hronca BA	2	10	7	8	9	12	36
13	Košdy Martin	Gym. Jura Hronca BA	1	9		13	14		36
13	Kvasnička Igor	Gym. Jura Hronca BA	2	3	6	13	7	7	36
13	Magyar Vladimír	SPŠE Nové Zámky	2	10	7	10	9		36
13	Sucha Martin	Gym. Jura Hronca BA	2		9	13	14		36
13	Švec Marcel	Gym. Jura Hronca BA	3	12	15		9		36
19	Fedáková Dominika	Gym. Stará Ľubovňa	2	9	7	10	9		35
19	Hanes Filip	Gym. Tajovského B. Bystrica	3	12	14		9		35
19	Svitek Andrej	Gym. Jura Hronca BA	0	9	5	14	7		35
22	Kočiský Tomáš	Gym. Grösslingová BA	2	9	3	6	14		32
22	Sabo Michal	Gym. Jura Hronca BA	2	9	6	9	8		32
24	Křemenová Lucia	Neznama skola	0	10	10		5	3	28
25	Beňo Fratišek	SPŠ Humenné	2		1	9	8	8	26
25	Jenis Jakub	Gym. Cyrila a Metoda Nitra	1		9	9	8		26
25	Mészároš Roman	SPŠ Levice	3	11	7		8		26
28	Kučin Alexander	Gym. Lipany	2		6	10	7		23
29	Nikodem Peter	Gym. Školská Spiš. Nová Ves	2		8	6	8		22
30	Jančigová Jarmila	Gym. Jura Hronca BA	2		6	8	7		21
31	Dullová Veronika	Gym. Ul. L. Sáru BA	3			10	9		19
31	Truben Martin	Gym. Jura Hronca BA	2	7	6		6		19
33	Tóth Ladislav	SPŠE Nové Zámky	3	10			8		18
34	Bachratý Martin	ZŠ Gaštanová Žilina	0		3	5	8		16
34	Simanová Lucia	Gym. Grösslingová BA	2		3		13		16
36	Saleh Adam	Gym. Jura Hronca BA	2			9	6		15
37	Popovič Viktor	Gym. Mudroňova Prešov	0	11					11
38	Hozza Michal	Gym. Jura Hronca BA	1				9		9
39	Dittrich Andrej	Gym. Jura Hronca BA	2		1		7		8
39	Trnovec Matúš	Gym. Jura Hronca BA	2		1		7		8
41	Buchovecká Simona	Gym. Medzilaborce	3				3	4	7