Korešpondenčný seminár z programovania
XXXIII. ročník, 2015/16
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

Návody k úlohám 1. kola letnej časti kategórie T

V kategórii T neuvádzame vzorové riešenia ale skôr návody na riešenie úloh. Ich cieľom je prezradiť vám hlavnú myšlienku riešenia, aby ste podľa návodu mohli riešenie domyslieť sami. Občas teda vynecháme niektoré drobné detaily, neuvádzame implementácie dátových štruktúr a nerozpisujeme niektoré kroky.

Neuvádzame ani vzorové programy, pretože chceme, aby ste po prečítaní návodu naprogramovali tie úlohy, ktoré ste nespravili počas trvania série. Keď si tieto riešenia sami naprogramujete, naučíte sa tým oveľa viac ako pozeraním sa na zdrojové kódy.

Implementácie všeobecne známych algoritmov a dátových štruktúr môžete nájsť vo vzorových riešeniach kategórií Z a O starších ročníkov KSP alebo aj na internete.

1. Tuším sme už raz ochraňovali pred povodňami

návod písal Mário
(max. 0 b za popis, 20 b za program)

V tejto úlohe nám pomôže to, čo ste viacerí skúšali už pri riešení pôvodnej KSP úlohy Ochrana pred povodňami – prehľadávanie, pričom sa pomocou union-findu snažíme detekovať, kedy sme uzavreli nejaké väčšie územie a kedy nie. Pri stavaní väčšiny múrov sa totiž zväčší chránená plocha len o 1, prípadne o 0, ak ho postavíme v už chránenom území.

Predstavme si, že kúsky múra sú vrcholy grafu spojené hranami s okolitými najviac 4 kúskami múra. Môžeme si pamätať súvislé komponenty múrov a vieme, že nejakú väčšiu plochu ochránime len vtedy, keď okolo nej uzavrieme múr – keď spojíme jeden múrový komponent *sám so sebou*. Zisťovanie, v akom súvislom komponente je políčko, a spájanie dvoch komponentov v čase (dokonca menšom ako) $O(\log n)$ vieme implementovať štruktúrou union-find.

Keď uzavrieme oblasť, stačí nám spustiť prehľadávanie z políčka, na ktoré sme práve pridali múr. Toto prehľadávanie musí zistiť, ktorými smermi sa z tohto políčka dá dostať na kraj (teda sú vonku/nechránené) a ktrorými smermi zostaneme vovnútri chránenej oblasti. Následne stačí označiť vnútro oblasti ako ochránené. Ako ale zistiť, čo je vonku a čo vnútri?

Pridaním ľubovoľného kúska múru vieme rozdeliť plochu na 1-4 územia. To, na koľko území plochu rozdelíme však vieme zistiť len z údajov o tom, v ktorých komponentoch súvislosti sú múry na susedných 8 políčkach.

```
. . .  2 2 2  . 1 .  . 1 .
1 X 1  . X .  3 X 4  1 X 1
. . .  1 1 1  . 2 .  . 1 .
```

Vo vyššie uvedených príkladoch reprezentuje `.` prázdne políčko a `1-4` políčka, na ktorých sú múry, pričom tieto múry patria do komponentov `1-4`. Pridaním múra na políčko `X` rozdelíme plochu na 2, 1 (teda nič nerozdelíme), 1 a 4 územia. Zistiť počet rôznych území (a pre každé územie prázdne políčka, ktoré sú jeho súčasťou) vám prenechávame ako implementačnú výzvu. Neodporúčame *hardcodovať* všetky možnosti.

Vieme teda, na koľko území je rozdelené okolie políčka `X`. Niektoré z území budú vnútorné a niektoré vonkajšie. Pokiaľ sme múr nepostavili v už ochránenej ploche, vdžy bude aspoň jedno územie vonkajšie (dokážte!). Pokiaľ všetku vodu okolo ostrova pokladáme za jednu oblasť, vždy bude dokonca práve jedno z oddelených území vonkajšie (dokážte!) a 0-3 vnútorné – ochránené.

Ak by sme pustili prehľadávania postupne do všetkých plôch, mohlo by sa nám stať, že na vodu narazíme, až keď prehľadáme väčšinu mapy. Ak by sme ale pustili prehľadávania paralelne ...

Nech plochu rozdelíme na 2 územia. Pustíme 2 prehľadávania, do každého územia jedno. Prehľadávať budeme tak, že vždy preskúmame jedno políčko v jednom území a jedno v druhom. V každom čase budeme mať teda prehľadané rovnako veľké časti oboch území. Zastaneme, keď narazíme na vodu – vtedy stačí prehľadať druhé územie, alebo keď vyplníme nejakú celú oblasť – vtedy už nemusíme prehľadávať druhé územie, lebo vieme, že je vonkajšie.

Ak sme postavením múra ochránili oblasť veľkosti s, prehľadáme tak najviac $2s$ políčok. Úvahy sa dajú zovšeobecniť aj pre prípady, keď rozdelíme plochu na **viac** oblastí (spúšťame najviac 4 paralelné prehľadávania do najviac 4 území a každé prehľadávanie zastavíme, ak sme v ňom narazili na vodu alebo ak sme už vyplnili

všetky ostatné územia). Rozmyslite si, že ak pridaním múra plochu rozdelíme iba na 1 oblasť, tak si nemôžeme dovoliť púšťať prehľadávanie. To ale nevadí, nakoľko táto jediná oblasť musí byť vonkajšia.

Každé políčko označíme najviac raz ako ochránené – počas nejakého prehľadávania. Celkovo teda prehľadáme najviac $2 \cdot w \cdot h$ políčok. Naše online riešenie má teda celkovú časovú zložitosť $O(w \cdot h + q)$.

návod písal Jano

2. Teória relativity

(max. 0 b za popis, 20 b za program)

Toto bola jedna z ľahších úloh.

Všimnime si, že sa posun o k dá vždy spraviť na tri obrátenia intervalu. Najprv obrátime knihy na pozíciách 1 až k, potom knihy $k+1$ až n a napokon 1 až n.

Ostáva teda zistiť, kedy sa to dá na menej, čiže kedy je odpoveď 0, 1 alebo 2. Odpoveď je 0 práve vtedy, keď k je 0. Pokiaľ k je 1 alebo $n-1$ alebo oboje, tak prvé, druhé alebo oboje obrátenia nemusíme robiť a odpoveď je 1 alebo 2.

Dôkaz toho, že sa to v týchto prípadoch nedá spraviť na menej obrátení, prenechávame ako jednoduché cvičenie pre čitateľa.

Časová zložitosť riešenia je konštantná.

návod písal Jano

3. Troška čokolády

(max. 0 b za popis, 20 b za program)

Kľúčová časť riešenia je, dokázať rýchlo odpovedať na takúto otázku: "ak by sme začali čokoládu rozdávať od i-teho dieťaťa až po j-te dieťa vystačí nám čokoláda?"

Totiž, ak by sme to vedeli, vieme úlohu riešiť takzvanou technikou dvoch bežcov. Zoberieme dvoch bežcov a oboch posadíme na začiatok kruhu. Bežci sa budú volať Ivan a Jožo a budú na pozíciách i a j. Vždy, keď dokážeme nakŕmiť deti od pozície i po pozíciu j, pohne sa Jožo na ďalšie políčko v kruhu. V opačnom prípade sa pohne Ivan. Všimnime si, že Ivan nikdy nepredbehne Jožu, pretože nula detí dokážeme nakŕmiť. Algoritmus skončí, keď Jožo dvakrát obehne celý kruh a riešenie úlohy bude najväčšia možná vzdialenosť bežcov počas behu algoritmu. Rozmyslite si, prečo je tomu tak.

Výnimkou je, že ak by bola odpoveď väčšia ako počet detí, vypíšeme len počet detí.

Obaja bežci spravia najviac 2ℓ krokov, kde ℓ je veľkosť kruhu (dĺžka reťazca Z), vďaka čomu je algoritmus podstatne rýchlejší, ako keby sme skúšali všetkých $O(\ell^2)$ dvojíc (i, j).

Ako teda zistiť, či sa nejaký úsek detí dá nakŕmiť?

Najskôr si otočíme čokoládu, tak aby bola aspoň taká vysoká ako široká. Túto vlastnosť potom vieme zachovať počas celej konzumácie – chlapec vždy zje stĺpec a čokoládu zúži. Dievča ak dostane čokoládu, ktorá je vyššia ako širšia, zje riadok. Ak však dievča zje čokoládu, ktorá je rovnako vysoká ako široká, zje stĺpec.

Pozrime sa na úsek detí od i po j. Predstavme si, že i-te dieťa bolo prvé čo jedlo čokoládu. Označme si c počet chlapcov v tomto úseku, d počet dievčat a e počet dievčat, ktoré zjedli stĺpec čokolády. Potom, čokoláda rozmerov $r \times s$, kde $(r \leq s)$ vystačí pre deti od i po j, pokiaľ $e + c \leq s$ a zároveň $d - e \leq r$. To je inak to isté, ako overovať $e + c \leq s$ a $c + d \leq r + s$.

Zistiť c a d pre nejaký úsek kruhu je ľahké. Jediná ťažká časť tejto úlohy bolo efektívne zistiť e.

Všimnime si, že keď dievča je čokoládu po chlapcovi, tak sa určite nezapočíta do e. Podobne, ak máme 8 dievčat po 8 chlapcoch. Intuitívne, na to, aby bolo veľké e, potrebujeme, aby veľa dievčat jedlo čokoládu bez toho aby pred nimi jedlo čokoládu veľa chlapcov. Presnejšie, zoberme postupnosti detí od pozície i, po všetky možné pozície k, $k \leq j$, v každej postupnosti spočítame dievčatá a chlapcov, a označme f najväčší z rozdielov počtu dievčat mínus počet chlapcov. Potom $e = max(0, (s - r + f + 1)//2)$. (Rozmyslite si, prečo.)

Čiže už len potrebujeme zistiť, ako spočítať f. To sa dá spraviť, tak, že dokopy na výpočet všetkých f spotrebujeme $O(\ell)$ času, stačí však jednoduchšie riešenie pomocou intervalového stromu v čase $O(\ell \log \ell)$, prípade pomocou multimnožiny s rovnakou zložitosťou.

Najskôr si do samostatného poľa uložíme na každú pozíciu rozdiel počtu dievčat a chlapcov od začiatku kruhu po danú pozíciu. Keď budeme chcieť zistiť f, potrebujeme vedieť maximum intervalu od i po j v tomto poli, a od toho odpočítať hodnotu na pozícii $i - 1$.

návod písal Buj

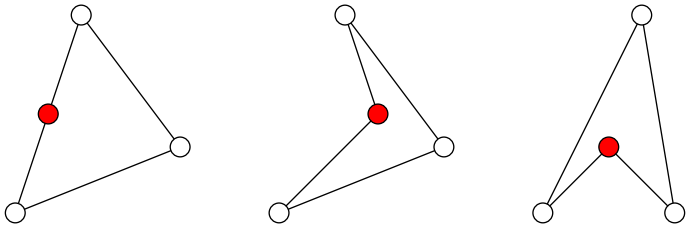
4. Trúba, štvoroká kyklopka

(max. 0 b za popis, 20 b za program)

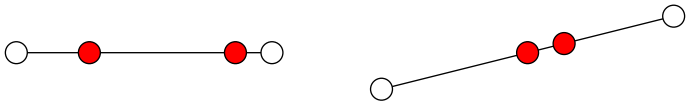
Budeme počítať počet tých štvoríc bodov, ktoré **netvoria** konvexný štvoruholník. Takúto štvoricu budeme

nazývať *nekonvexná*. Ak počet nekonvexných štvoríc je p, potom odpoveď na zadaný problém je $\binom{n}{4} - p$ (kedže všetkých štvoríc bodov je $\binom{n}{4}$).

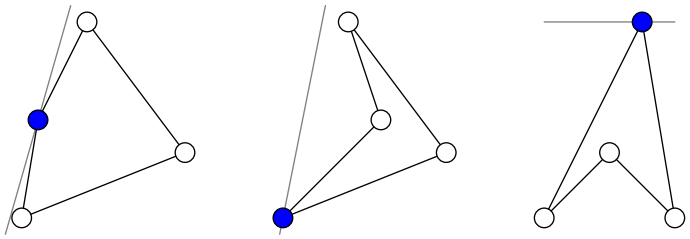
Zoberme si nekonvexnú štvoricu bodov takú, že jej body neležia všetky na jednej priamke. Potom tejto štvorici vieme prirodzene priradiť *stred* a tri *ramená* – stred je ten vrchol spomedzi tej štvorice, ktorý leží vo vnútri trojuholníka tvoreného ostatnými tromi vrcholmi. Ramená sú zvyšné tri vrcholy. V nasledujúcich obrázkoch je stred vyznačený červenou farbou, ramená bielou:



Ak tie štyri body ležia na jednej priamke, tak jej vieme stred a ramená priradiť dvomi spôsobmi.



Všimnime si, že neexistuje priamka vedúca cez stred taká, že všetky tri ramená ležia na jednej strane tejto priamky. Zoberme si teraz nejakého kandidáta na stred A, a troch kandidátov na ramená B, C, D. Potom [$ABCD$ netvorí konvexný štvoruholník a A je stredom štvorice A, B, C, D] platí práve vtedy, keď neexistuje priamka vedúca cez A taká, že B, C aj D sú na jednej strane tejto priamky. (Ak takáto priamka existuje, tak síce $ABCD$ môže netvoriť konvexný štvoruholník, ale A už nebude stredom tejto štvorice.)



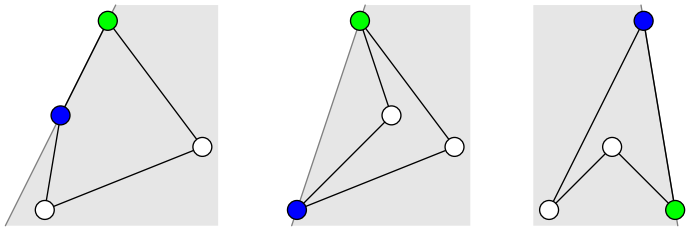
Náš algoritmus bude pracovať nasledovne: vyskúšame každý bod ako stred. Pre aktuálne zvoleného kandidáta na stred A zistíme, koľko trojíc ostatných bodov spĺňa: neexistuje priamka vedúca cez A taká, že tie tri body sú na jednej strane od tej priamky. Takto ale zarátame dvakrát štvorice bodov také, že ležia na jednej priamke. Preto ešte odčítame počet takých štvoríc bodov, a dostaneme tak vytúžené p (počet nekonvexných štvoríc bodov).

Počet štvoríc bodov ležiacich na jednej priamke vieme vypočítať ľahko – pre každý bod A spočítame, koľko trojíc bodov B, C, D je takých, že A, B, C, D ležia na jednej priamke a pritom A je krajný bod tejto štvorice (teda na priamke ležia v poradí A, a potom ostatné body). Usporiadame si všetky body X primárne podľa toho, aký orientovaný uhol zviera polpriamka $\overrightarrow{AX}$ s vodorovnou osou, a sekundárne podľa dĺžky $|AX|$. Ďalej to zvládnete aj sami...

Zaujímavejšia časť je pre aktuálneho kandidáta na stred A zistiť počet vyhovujúcich trojíc. Opäť si pomôžeme tým, že budeme namiesto vyhovujúcich trojíc počítať nevyhovujúce – teda trojice bodov B, C, D také, že **existuje** priamka q taká, že B, C aj D ležia na jednej strane q.

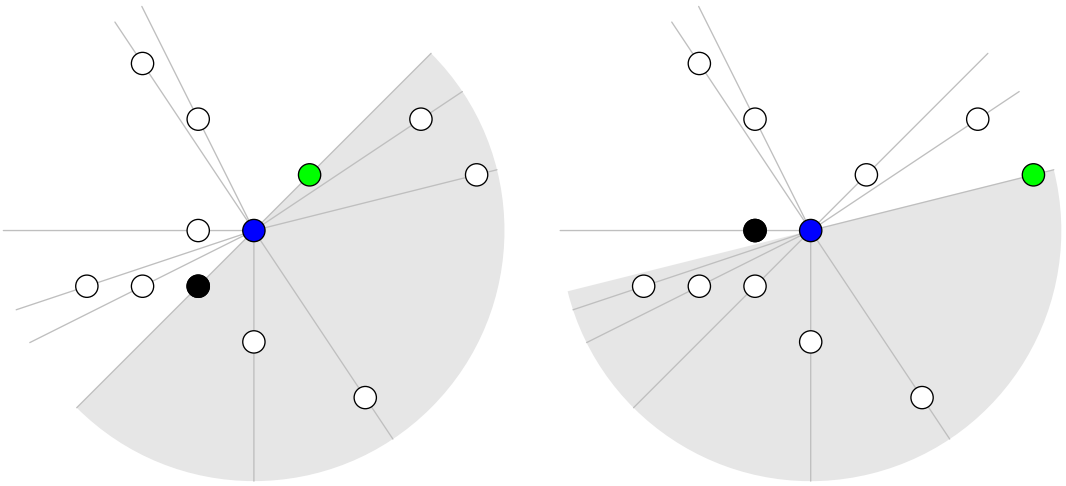
Všimnime si, že každej vyhovujúcej trojici bodov vieme prirodzene priradiť jeden špecifický bod – ten, ktorý je v smere hodinových ručičiek "prvý". Nech to je napríklad B. Ten má navyše tú vlastnosť, že priamka AB (tj. spojnica stredu A a tohto význačného bodu B) skoro má požadovanú vlastnosť – C aj D ležia na jednej strane od tejto priamky, a B leží na nej. Nakoľko iba jeden z bodov B, C, D na nej leží, vieme priamku o máličko otočiť tak, že všetky tri body budú na jednej strane od tej priamky.

V nasledujúcom obrázku je stred označený modrou, "prvý" bod označený zelenou.



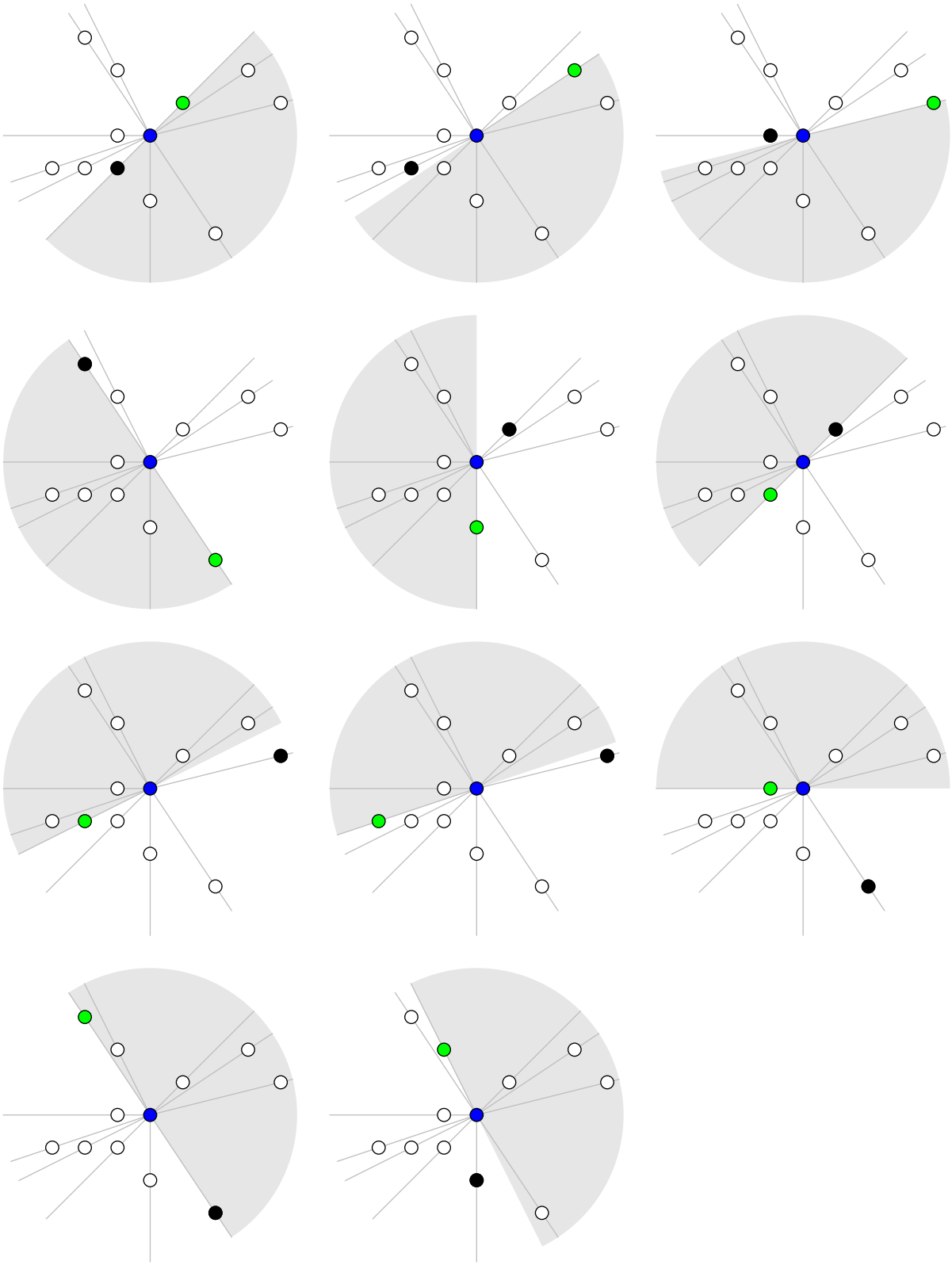
Na základe týchto pozorovaní vieme pre konkrétny stred spočítať počet nevyhovujúcich trojíc nasledovne: najprv utriedime všetky body okrem stredu S polárne. Následne si zvolíme počiatočného kandidáta X na "prvý" bod. Potom nájdeme prvý bod v smere hodinových ručičiek taký, že neleží napravo od polpriamky $\overrightarrow{SX}$. Pritom si počítame, koľko bodov sme vyskúšali – o nich vieme, že sú to všetky body ležiace napravo od $\overrightarrow{SX}$. Ak ich počet označíme k, vieme, že pre aktuálne zvolený "prvý" bod máme $\binom{k}{2}$ možností vybrať zvyšné dva body tak, aby ležali napravo od $\overrightarrow{SX}$.

V nasledujúcom obrázku je stred označený modrou a "prvý" bod zelenou. Prvý bod v smere hodinových ručičiek, ktorý nie je napravo od $\overrightarrow{SX}$, je čiernou.



Postupne v smere hodinových ručičiek vyskúšame všetky body ako "prvé". Pritom si všimneme, že keď "prvý" bod posunieme v smere hodinových ručičiek, tak prvý nevyhovujúci bod sa tiež pohne v smere hodinových ručičiek. A ľahko odtiaľ vidno, že keď vyskúšame všetky body ako "prvé", prvý nevyhovujúci bod sa posunie najviac toľkokrát, koľko je všetkých bodov.

Nasledujúci obrázok pekne demonštruje tento proces "posúvania".



návod písal Buj

5. Tarantulu za klobásu a tri dukáty

(max. 0 b za popis, 20 b za program)

Než sa pustíte to čítania, odporúčam si naštudovať silno súvislé komponenty.

Preveďme teda našu úlohu do grafovej terminológie. Zadaný je ohodnotený orientovaný graf taký, že každý sled začínajúci a končiaci v tom istom vrchole má celkové ohodnotenie 0. (Sled je, neformálne povedané, ľubovoľná prechádzka v grafe.) Úlohou je nájsť najväčšie číslo také, že v grafe existuje sled s takým celkovým

ohodnotením.

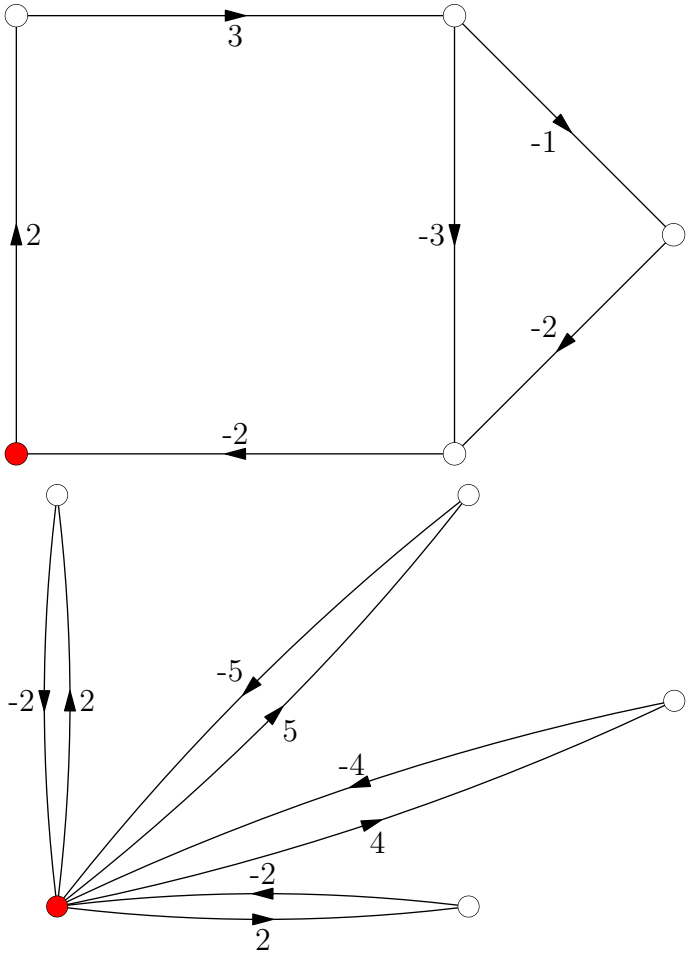
Zrejme nám stačí v grafe hľadať **cestu** s najväčším celkovým ohodnotením – ak sme totiž nejaký vrchol navštívili dvakrát, tú časť sledu môžeme vynechať, nakoľko má celkové ohodnotenie 0. (Cesta je sled, v ktorom každý vrchol navštívime najviac raz.)

Jedným z možných riešení je Floyd-Warshallov algoritmus. Ten môžeme použiť, nakoľko graf neobsahuje cykly s kladným celkovým ohodnotením. Je ale pomalé a zbytočne silné – nehľadá totižto len dĺžku najdlhšej cesty, ale hľadá dĺžky najdlhších ciest medzi každou dvojicou vrcholov.

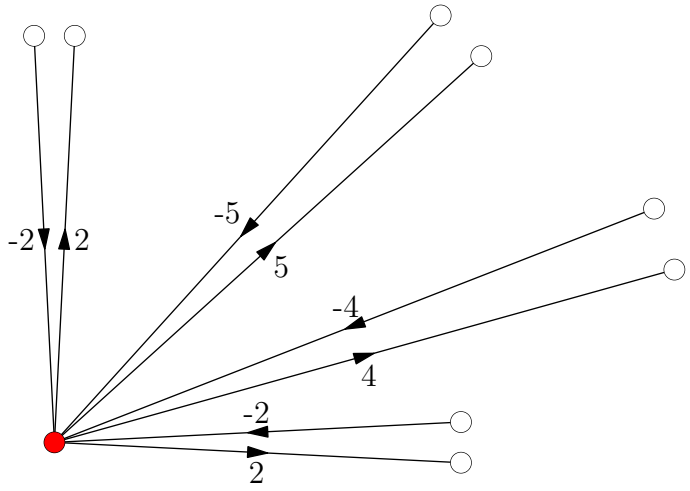
Ďalej sa budeme snažiť zredukovať zadaný problém na hľadanie cesty z najväčším ohodnotením v *acyklickom grafe*. Tento problém sa totižto dá jednoducho riešiť pomocou rekurzie s memoizáciou, alebo pomocou dynamického programovania.

Všimneme si nasledovné pozorovanie: nech existuje cesta z A do B, a tiež nech existuje cesta z B do A s celkovým ohodnotením q. Potom každá cesta z A do B má rovnaké celkové ohodnotenie rovné $-q$.

Z predchádzajúceho pozorovania vyplýva: nech A, B, C patria do toho istého silno súvislého komponentu. Potom namiesto toho, aby sme uvažovali cestu z A do C, nám stačí uvažovať cestu z A do B a cestu z B do C – tie majú dokopy rovnaké ohodnotenie, ako pôvodne zamýšľaná cesta z A do C. Naskytá sa nám tak nasledujúca úvaha: z každého silno súvislého komponentu $\{A_1, \ldots, A_n\}$ vyberieme jedného reprezentanta. Nech to je A_1. Pre všetký ostatné vrcholy A_i vypočítame ohodnotenie cesty z A_1 do A_i. Nech je to ohodnotenie rovné q. Vytvoríme hranu s hodnotou q vedúcu z A_1 do A_i, a hranu s hodnotou $-q$ z A_i do A_1. Pôvodné hrany súvislého komponentu zmažeme.



Ľahko nahliadneme, že odpoveď pre takto zostrojený graf je rovnaká, ako odpoveď pre pôvodný graf. Tento graf ale ešte nie je acyklický – na to si ešte všimneme, že nikdy nepotrebujeme navštíviť reprezentanta dvakrát. Rozdelíme teda každý vrchol A silno súvislého komponentu, ktorý nie je reprezentant, na dve – vstupný vrchol A_{in} a výstupný vrchol A_{out}. Zo vstupného vrcholu vedie jediná hrana z A_{in} do reprezentanta, a vedú doňho rovnaké hrany, ako viedli do pôvodného vrcholu A. Do výstupného vrcholu vedie jediná hrana z reprezentanta do A_{out}, a vychádzajú z neho rovnaké hrany, ako vychádzali z pôvodného vrcholu A.



Toto spravíme pre každý silno súvislý komponent. Silno súvislé komponenty vieme nájsť napríklad pomocou Tajranovho algoritmu na hľadanie silno súvislých komponentov. Vzniknutý graf je zrejme acyklický, tak na ňom spustíme riešenie pre acyklické grafy.