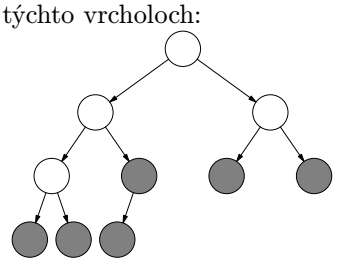
Ukážková písomka: riešenia

1 Pohľad zvnútra: halda [5 bodov]

Druhý najmenší prvok sa môže nachádzať v týchto vrcholoch:



V listoch zjavne môže byť. Vo vyfarbenom nie-liste môže byť, ak je v liste pod ním najmenší prvok. V iných vrcholoch nemôže byť, lebo každý vrchol obsahuje hodnotu väčšiu ako každý z jeho potomkov, a teda ak má ≥ 2 potomkov, už je aspoň tretí najmenší.

2 Pohľad zvnútra: hľadanie nepárneho [10 bodov]

Dokážeme, že existuje vstup, pre ktorý Alfonzov program nedá správnu odpoveď.

Hlavná myšlienka: Ak má jeho program naozaj časovú zložitosť $O(\log N)$, stihne sa pozrieť na nanajvýš $O(\log N)$ políčok poľa. Pre dostatočne veľké N sa teda na niektoré políčka poľa vôbec nepozrie. Oklameme ho teda tak, že jediné nepárne číslo schováme na miesto, kam sa nepozrel.

Dôkaz: Zvoľme si dostatočne veľké N a spustime Alfonzov program na vstupe, kde $\forall i : A[i] = 2i$. Keď jeho program dobehne, pozrieme sa, ako odpovedal. Ak sa pomýlil a povedal „áno“, máme hľadaný vstup. Zostáva uvažovať prípad, kedy Alfonzov program pre toto pole správne odpovie „nie“.
Zároveň vieme, že existuje také x, že na políčko $A[x]$ sa Alfonzov program nikdy nepozrel. Ak teraz spustíme Alfonzov program na vstupe, kde $\forall i : A[i] = 2i$, len $A[x] = 2x - 1$, musí jeho výpočet prebehnúť presne rovnako – a teda dá jeho program aj pre toto pole odpoveď „nie“, čím sme úspešne zostrojili vstup, na ktorom sa Alfonzov program pomýli.

3 Pohľad zvonka: správny pár [10 bodov]

Pole usporiadame v čase $O(N \log N)$ MergeSortom. Následne pre každé i pomocou binárneho vyhľadávania v čase $O(\log N)$ zistíme, či sa niekde v poli nachádza hodnota $S - A[i]$.

Iné riešenie: Po usporiadaní poľa už vieme hľadanú dvojicu nájsť (ak existuje) dokonca v lineárnom čase. Hlavná myšlienka riešenia: Ak $A[1] + A[N] < S$, môžem zahodiť $A[1]$, lebo určite k nemu pár nenájdem. A naopak, ak $A[1] + A[N] > S$, môžem zahodiť $A[N]$. Iterujem kým buď nenájdem dvojicu so správnym súčtom alebo nezahodím všetko.

4 Pohľad zvonka: poradovník u doktora Housa [15 bodov]

Vyzerá to, že potrebujeme prioritnú frontu. Pribudla však operácia navyše: potrebujeme vedieť vymazať ľubovoľný prvok, a na to nám už halda nestačí.
Prioritnú frontu budeme teda implementovať ako vyvážený binárny vyhľadávací strom. Do neho budeme vkladať záznamy (zaujímavosť pacienta, meno pacienta), usporiadané ich budeme mať primárne podľa zaujímavosti.
S takýmto stromom už vieme v logaritmickom čase robiť všetky potrebné operácie, až na jeden detail: vymazať pacienta vieme až vtedy, ak ho v strome nájdeme. A keďže strom je usporiadaný podľa zaujímavosti, na to potrebujeme vedieť nie len jeho meno, ale aj jeho zaujímavosť.
Použijeme teda ešte druhú dátovú štruktúru: asociatívne pole, kde si ku každému menu pacienta zapamätáme jeho zaujímavosť.
V C++ by deklarácia potrebných dátových štruktúr vyzerala nasledovne:

```
set< pair<int,string> > ludia_usporiadani_podla_zaujimavosti;
map< string, int > zaujimavost_cloveka;
```

Jednotlivé operácie potom implementujeme nasledovne:
Príchod človeka. V asociatívnom poli si zaznačíme k jeho menu jeho zaujímavosť. Do usporiadanej množiny čakajúcich pridáme záznam (jeho zaujímavosť, jeho meno).
Odchod človeka. Pomocou mena si v asociatívnom poli nájdeme jeho zaujímavosť. Následne zmažeme záznamy tohto človeka aj z asociatívneho poľa, aj z množiny čakateľov.
Nový diel. V množine nájdeme človeka s najvyššou zaujímavosťou a odstránime jeho záznamy z oboch dátových štruktúr.

Zjavne takto vieme každú z požadovaných operácií spraviť v čase logaritmickom od aktuálneho počtu čakajúcich pacientov.

Iné riešenie. Použijeme klasickú prioritnú frontu implementovanú ako halda. Vymazávania odložíme až do okamihu, keď budeme chcieť natáčať nový diel. Aby sme vedeli, koho zmazať a koho nie, budeme si navyše udržiavať množinu mien ľudí, ktorí medzi časom odišli.
V C++ by deklarácia potrebných dátových štruktúr vyzerala nasledovne:

```
priority_queue< pair<int,string> > ludia_usporiadani_podla_zaujimavosti;
set<string> ludia_ktori_uz_odisli;
```

Jednotlivé operácie potom implementujeme nasledovne:
Príchod človeka. Pridáme nový záznam do prioritnej fronty.
Odchod človeka. Pridáme nový záznam do množiny ľudí, čo už odišli.
Nový diel. V prioritnej fronte nájdeme človeka s najvyššou zaujímavosťou a odstránime jeho záznam. Skontrolujeme, či ešte neodišiel. Ak nie, končíme, ak áno, odstránime ho aj z tej množiny a začneme celé hľadanie odznova.

Takéto riešenie má *amortizovanú* časovú zložitosť $O(\log N)$ na operáciu.
Inými slovami: ak:
– začneme s prázdnou množinou ľudí
– postupne urobíme K operácii,
– celkový počet ľudí, ktorí naraz čakajú na Housa, nikdy neprekročí N
tak celková časová zložitosť bude vždy $O(K \log N)$.
Prečo je to tak? Lebo robíme v podstate to isté, ako v predchádzajúcom riešení, len mazania odkladáme na neskôr. Dokopy teda naša postupnosť operácií bude mať nanajvýš toľko mazaní ako tá v predchádzajúcom riešení.