**Big files finder**

*Vlastimil Brečka*

Zadaním úlohy bolo nájsť N najväčších súborov vo zvolených adresároch.

Aplikácia sa skladá z 2 architekturálnych častí

1. **MainActivity**

Predstavuje „klientskú“ časť, obsahuje GUI ktoré ovláda vstupy pre servisu.

Skladá sa z 4 fragmentov

* 1. *WelcomeFragment*

Predstavuje len úvodnú obrazovku, ukážka komplexnejších animačných techník a material designu

* 1. *FolderPickerFragment*

UI pre výber adresárov, je možné prechádzať aktuálnu súborovú štruktúru a prídávať si adresáre, ktoré budú prehľadávané. Zvolené adresáre su zobrazené v FileChipsView, čo je subclassa FlowLayout-u, ktorá je obdobou LinearLayoutu taká, že ak sa View zmestí do riadku tak ho dá do toho istého riadku, ak nie tak na nový riadok.

Ako list je použitý RecyclerView, ktorý sa lepšie chová s Transition animáciami ako ListView. V pravej časti riadku je taktiež znázornený stav súboru (zašktrnutie / plus)

* 1. *CountPickerFragment*

Zvolenie počtu najväčších hľadaných súborov

* 1. *ResultsFragment*

Reaguje na broadcasty FinderServicu, a zobrazuje progressbar/výsledky/chybu výpočtu.

Kde každý z nich predstavuje logický krok zadávania vstupov pre výpočet.

1. **FinderService**

Predstavuje „serverovú“ časť, ktorá vykonáva samotné prehľadávanie. Service bol zvolený preto lebo aktivita je „len GUI“, ktoré po odchode do pozadia,môže kedykoľvek zaniknúť a preto nie je vhodným rodičom pre potenciálne dlhotrvajúce thready výpočtu. Tzn, výpočeť je nezávislý od GUI, a to je dobre. Komunikácia medzi nimi prebieha pomocou intentov (broadcastov)

Vstupom pre service je počet koľko najvačších súborov hľadáme a cesty na adresáre, ktoré majú byť prehľadávané.

Po očístení o neplatné vstupy, service optimalizuje zadané cesty adresárov vyhodením duplikátov a ciest ktoré su podadresárom niektorých z ostatných ciest a nemá zmysel ich prehľadávať (v princípe, algoritmus kontroluje prefixy ciest)

Potom sa pre každý prekonvertovaný File, vytvára FindLargestFilesTask asynchrónny task, ktorý beží na vlastnom Thread-e. Tasky sú spúšťané na exekútore, tzn. všetky thready bežia paralelne per-file. V tele tasku sa prehľadáva cesta rekurzívne a ak súbor nie je adresár je pridaný do kolekcie.

Zvolená kolekcia FilesBoundedPriorityQueue, ktorá predstavuje subclass-u PriorityQueue, čo je štandardná implementácia heap-u, ktorá nám zaručí polo-zoradenie súborov s garantovaným najmenším súborom na vrchole. Pri pridávaní je avšak zvolená optimalizácia, kedže vieme koľko max. súborov hľadáme, teda nemá zmysel, aby v heap-e boli všetky súbory ale len najvačších N v danom momente. Teda pri pridávaní sa kontroluje, čí je vstup väčší ako súbor na vrchole (najmenší z heap-u), taký súbor je do heap-u pridaný a vrchol odstránený. Takto časom dostaneme N najvačších súborov. (ešte podotknúť, že heap-a je zoradená vzostupne, tzn najmenší je na vrchole, teda je po vybratí prvkov z nej do listu, je potrebné list otočiť). Taktiež metóda add() je synchronizovaná a garantuje thread-safety, keďže heap je zdieľaný medzi thread-mi.

Po dokončení úlohy, je referencia naň odobratá zo Set-u bežiacich úloh. Ak je set prázdny, boli vykonané všetky úlohy, teda výsledky vraciame cez broadcast naspať do UI a service ukončujeme. (Samozrejme o začatí a skončení výpočtov notifikuje systémová notifikácia s progress-barom, resp. jednorázová o ukončení všetkých výpočtov.). Pre prenos výsledkov bol použítý ParcelableFile, čo je len POJO cesty, názvu a veľkosti súboru, keďže systémový File neimplementuje Parcelable interface a nie je možné preposlať ArrayList File-ov cez intent.

*Aplikácia bola testovaná na Nexus 5, Android 5.1 Lolipop*