Zadanie 1 (15 pkt).

Napisz program, który otworzy w trybie tekstowym pliki o nazwach podanych w argumentach wywołania i – zakładając, że ich kolejne wiersze są uporządkowane (zgodnie ze strcmp) – scali je w tym samym porządku i wypisze na standardowe wyjście. (Taką funkcjonalność, choć dużo bardziej rozbudowaną, ma uniksowy program sort z flagą -m.)

Program powinien otworzyć wszystkie pliki naraz, wczytać po jednym wierszu z każdego z nich do jakiejś struktury, a następnie w pętli: usuwać z niej (i wypisywać) najwcześniejszy alfabetycznie wiersz, oraz dodać do niej kolejny wiersz pochodzący z tego samego pliku, co ten wypisany. Kiedy jakiś plik zostanie doczytany do końca przed innymi, zamknij go od razu, tj. nie czekając na koniec wykonania programu.

Zaimplementuj – w dwóch osobnych plikach źródłowych, ale oczywiście z jednym plikiem nagłówkowym – dwie takie struktury:

- zwykłą tabelę, w której do wsortowywania nowo wczytywanych wierszy wykorzystasz przeszukiwanie binarne (najlepiej bsearch z stdlib.h), ale za to konieczne będzie przesuwanie potencjalnie wielu jej elementów o 1 pozycję;
- jednokierunkową listę wiązaną, w której znalezienie właściwej pozycji będzie wymagało sekwencyjnego jej przeczytania, ale za to wstawienie elementu na właściwe miejsce będzie w czasie stałym.

Wyraźnie zaznacz w komentarzach, które pliki źródłowe należy skompilować, żeby otrzymać program wykorzystujący daną implementację.

Postaraj się wykorzystywać już przydzieloną pamięć, zgodnie z obserwacją, że nowy wiersz wczytywany jest zaraz po wypisaniu starego, więc można wykorzystać "jego" bufor (być może po realokacji, jeśli jest za krótki), element listy, itp.

Napisz nową funkcję do wczytywania całych wierszy z pliku – najpierw czytaj jego zawartość tylko po to, by określić, ile znaków ma wiersz, a potem przy użyciu fseek (a wcześniej ftell) cofnij się, by faktycznie zapisać treść. (To podejście ma wady: nie na wszystkich strumieniach można wykonać fseek – np. stdin – ale za to pozwala przydzielać pamięć tylko raz, bez skomplikowanych realokacji bufora.)

Wiersze, jak zwykle, kończą się kodem końca wiersza '\n' i mogą być puste. (Założenie, że czytane pliki są już posortowane oznacza, że puste wiersze mogą być tylko na początku, ale nie korzystaj z tej obserwacji – program ma zrobić coś w miarę sensownego, a przynajmniej nie niepoprawnego, wywołany dla dowolnych plików.) Jeśli plik kończy się od razu po kodzie końca wiersza, to nie należy tego traktować jako kolejnego, pustego wiersza; podobnie, gdy plik jest pusty, tj. nie zawiera nic, nawet kodu końca wiersza. Jeśli natomiast po ew. ostatnim kodzie końca wiersza są jeszcze jakieś znaki, to nie należy ich gubić, a wypisując – "uzupełnić" brakujący koniec wiersza. (Warto robić to już przy wczytywaniu, żeby uniknąć niepotrzebnego rozstrzygania przypadków np. przy porządkowaniu wierszy.)

Każdy problem m.in. z działaniami na plikach powinien skutkować: wypisaniem odpowiedniego komunikatu (na standardowy strumień błędów), natychmiastowym zakończeniem działania programu (chyba że błąd dotyczy "tylko" zamknięcia pliku), i wyjściem z niezerową wartością. Wartość ta powinna nieść informację o napotkanych problemach – przemyśl, ile różnych kodów musisz uwzględnić, i opisz je w komentarzu w kodzie programu.

Jako rozwiązanie możesz przesłać 8 plików, czyli chyba o (co najmniej) dwa więcej, niż wydaje się to potrzebne. Jeśli masz realny powód, by podzielić rozwiązanie jeszcze bardziej, opowiedz o nim osobie prowadzącej zajęcia, żeby zmienić to ograniczenie.

Zadanie 2 (15 pkt).

Na potrzeby tego zadania dobrym drzewem będziemy nazywali takie, ktore ma korzeń o stopniu 1 (**nie uwzględniając** rodzica, bo go nie ma), a wszystkie pozostałe jego wierzchołki mają stopień (**uwzględniając** rodzica) 1 (gdy są liśćmi), albo co najmniej 3. Innymi słowy, drzewo "rozgałęzia się" w każdym wierzchołku poza korzeniem (który ma dokładnie jedno dziecko) oraz liśćmi. Każdy wierzchołek naszego drzewa ma indeks, a każda krawędź – nieujemną całkowitą wagę.

Na początku drzewo składa się z korzenia (o indeksie 0) połączonego z jedynym liściem (o indeksie 1) krawędzią o wadze 0. Drzewo rośnie, czasem jakaś gałąź odpadnie. Odpowiadają temu następujące operacje (które pojawią się na standardowym wejściu):

- z **liścia** o indeksie *idx* może wyrosnąć *k* nowych gałęzi o kolejnych nieużywanych jeszcze indeksach i odległościach do wierzchołka *idx* równych 0,
- wierzchołek o indeksie *idx* może oddalić się od swojego ojca w drzewie poprzez wydłużenie się jego gałęzi do ojca o x,
- wierzchołek o indeksie *idx* może wraz ze swoją gałęzią do ojca i swoim poddrzewem odpaść. W tym przypadku może się zdarzyć, że drzewo przestało być dobre. Należy wtedy usunąć wierzchołek ojca, a jego jedyne już dziecko połączyć z (dotychczasowym) dziadkiem krawędzią o długości odpowiadającej ich odległości w ścieżce idącej przez ojca (czyli będącej sumą długości dwóch "łączących się" w tym momencie krawędzi).

Powyższe operacje będą na wejściu oznaczane literkami 'f' (fork), 'g' (grow), 'd' (delete). Żadna operacja nigdy nie będzie dotyczyć wierzchołka 0, jak również operacja 'd' nigdy nie będzie dotyczyć jego dziecka (może zdarzyć się, że będzie mieć indeks inny niż 1 po wykonaniu operacji 'd' wykonanej na wnuku korzenia).

Operacje należy wczytywać i wykonywać tak długo, jak się pojawiają na wejściu. Na koniec wypisz drzewo następująco:

- pomiń korzeń (wierzchołek o indeksie 0),
- kolejne wierzchołki w drzewie wypisuj w kolejności w jakiej odwiedziłby je algorytm przeszukiwania wgłąb (*depth-first search*) tj. wchodząc do poddrzew i rekurencyjnie wypisując je w całości przed przejściem do kolejnego poddrzewa),
- dla danego wierzchołka odwiedzaj jego dzieci w kolejności od najmniejszego indeksu,
- każdy wierzchołek wypisz w osobnym wierszu razem z jego dystansem do korzenia (wierzchołka o indeksie 0).

Wejście

Na wejściu pojawiają się kolejne etapy ewolucji drzewa w postaci:

- 'f' *idx k*
- 'g' *idx x*
- 'd' *idx*

Wartości *idx* są indeksami wierzchołków znajdujących się w drzewie.

Wyjście

Wyjście powinno składać się z tylu wierszy, ile jest obecnie wierzchołków w drzewie różnych od korzenia 0. W wierszu powinny znaleźć się dwie liczby, kolejno indeks wierzchołka oraz jego odległość od korzenia wierzchołka o indeksie 0.

Dodatkowe informacje

W pierwszych czterech testach nie występuje operacja usuwania wierzchołków. W pięciu pierwszych testach operacja 'f' jest wykonywana zawsze z *k=2*. Liczba aktywnych (nieusuniętych) indeksów nie przekracza 30000.

Suma wartości *k* we wszystkich operacjach 'f' nie przekracza 100000. Wartości *x* nie przekraczają 100000, dodatkowo wagi krawędzi końcowego drzewa są nie większe niż 2 000 000 000. Limit pamięci w tym zadaniu wynosi 6 MB.

Przykłady

Przykład A

Wejście

```
f 1 2
f 2 2
f 3 2
f 4 5
g 5 1
g 3 4
g 6 2
```

Wyjście

```
1 0
2 0
4 0
8 0
9 0
10 0
11 0
12 0
5 1
3 4
6 6
7 4
```

Przykład B

Wejście

```
f 1 2
g 1 3
g 2 5
g 3 4
d 3
```

Wyjście

```
2 8
```

Przykład C

Wejście

```
f 1 5
f 3 2
f 7 3
f 5 2
d 8
```

Wyjście

1 0
2 0
4 0
5 0
12 0
13 0
6 0
7 0
9 0
10 0
11 0