ДОМАШНО № 1 ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ"
ЗА СПЕЦИАЛНОСТ "ИНФОРМАТИКА", 2. КУРС
(СУ, ФМИ, ЛЕТЕН СЕМЕСТЪР НА 2019 / 2020 УЧ. Г.)

Задача 1. Кореново дърво съдържа 26 върха, включително корена. Всеки връх е обозначен с различна латинска буква и има най-много два преки наследника. Дървото било обходено по два начина — preorder и postorder. Получили се следните редици от латински букви.

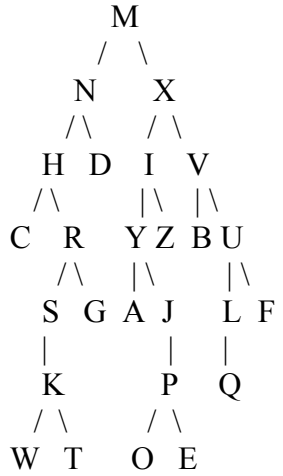
preorder: M N H C R S K W T G D X I Y A J P O E Z V B U L Q F
postorder: C W T K S G R H D N A O E P J Y Z I B Q L F U V X M

Възстановете дървото. **(2 точки)**

Решение: Коренът е M, защото той е пръв при preorder обхождане и последен при postorder. Първият връх при postorder обхождане (тоест C) е най-лявото листо на дървото. Ето защо членовете от M до C в preorder обхождането (M, N, H, C) са резултат от спускане все наляво. Аналогично, последният връх при preorder обхождане (тоест F) е най-дясното листо, затова членовете от F до M в postorder обхождането (F, U, V, X, M) са резултат от изкачване наляво, тоест те образуват най-десния клон на дървото.

Продължаваме разсъжденията стъпка по стъпка, докато възстановим дървото изцяло. Задачата има единствено решение.

Отговор:



Задача 2. В едно предприятие замислили празненство. Организаторите на празненството приписали рейтинг (цяло положително число) на всеки служител: по-висок рейтинг означава по-забавен служител. Йерархията на служителите била представена чрез кореново дърво, като коренът съответствал на директора на предприятието. За да не пречат на забавлението, организаторите взели решение, че на празненството не бива да присъства прекият началник на никого от поканените. Кои служители трябва да бъдат поканени на празненството, ако целта на организаторите е да постигнат най-голям сбор от рейтингите на поканените?

Помогнете на организаторите, като съставите алгоритъм с линейна времева сложност, който по зададени рейтинги и йерархия на служителите отговаря на поставения въпрос. Опишете алгоритъма с думи и го демонстрирайте подробно (стъпка по стъпка) с пример. Дървото от примера трябва да съдържа поне 10 върха и да има височина поне 3. **(3 точки)**

Решение: Задачата се решава с помощта на ***динамично програмиране.*** Да означим с T кореновото дърво, представящо йерархията на служителите. Нека *root* е коренът на T и нека $Adj(u)$ е списъкът от преките наследници на произволен връх u.

Във всеки връх u на T пазим три числа:

1) $u\,.\,rating$ — рейтинг на служителя;
2) $u\,.\,funTaken$ — най-голям сбор от поддървото с корен u, ако служителят u бъде поканен;
3) $u\,.\,funSkipped$ — най-голям сбор от поддървото с корен u, ако u не бъде поканен.

Рейтингите са дадени, а другите две числа алгоритъмът пресмята по следните формули:

$$u\,.\,funSkipped = \sum_{v \in Adj(u)} \max\left\{ v\,.\,funTaken\ ,\ v\,.\,funSkipped \right\};$$

$$u\,.\,funTaken = u\,.\,rating + \sum_{v \in Adj(u)} v\,.\,funSkipped.$$

Изчисленията вървят от листата към корена. За листата сумите във формулите са празни, следователно имат нулеви стойности, тоест $u\,.\,funSkipped = 0$, $u\,.\,funTaken = u\,.\,rating$, ако u е листо. Алгоритъмът връща $\max\left\{ root\,.\,funTaken\ ,\ root\,.\,funSkipped \right\}$: най-големия възможен сбор от рейтинги на поканени служители. Резултатът от това сравнение показва дали директорът (*root*) да бъде поканен на празненството. Ако директорът бъде поканен, то преките му подчинени няма да бъдат поканени, а за техните подчинени ще се извършат аналогични сравнения. Ако не поканим директора, то за преките му подчинени извършваме аналогични сравнения. Тоест определянето на поканените върви от корена към листата — обратно на пресмятането на сборовете от рейтинги.

Псевдокод на алгоритъма:

```
maxSumRating(root) {
   calcDynamicValues(root)
   return printInvited(root, false)
}

calcDynamicValues(u) {
   u.funTaken ← u.rating
   u.funSkipped ← 0
   for v ∈ Adj(u) do
      calcDynamicValues(v)
      u.funTaken ← u.funTaken + v.funSkipped
      best ← max(v.funTaken, v.funSkipped)
      u.funSkipped ← u.funSkipped + best
}

printInvited(u, parentIsInvited) {
   if parentIsInvited or u.funSkipped > u.funTaken
      for v ∈ Adj(u) do
         printInvited(v, false)
      return u.funSkipped
   else
      print u
      for v ∈ Adj(u) do
         printInvited(v, true)
      return u.funTaken
}
```

Задача 3. Предположете, че имате разписанието на БДЖ във вид на масив $A[1...n]$, всеки елемент на който е запис с четири полета: номер на влак, име на гара, час и флаг. Часът е време от денонощието с точност до минута (тоест час и минута). Номерът на влак е цяло положително число, което идентифицира влака, т.е. всеки влак има различен номер. Името е идентификатор на гарата, който в действителност е низ, но за простота приемаме, че е цяло положително число. Флагът показва дали влакът пристига, или заминава от гарата в указания час. Накратко, всеки елемент на масива описва събитие като следните:

Влак № 8601 заминава от гара София в 6:35 ч.
Влак № 8601 пристига в гара Пловдив в 8:48 ч.
Влак № 8601 заминава от гара Пловдив в 9:00 ч.
Влак № 8601 пристига в гара Ямбол в 11:25 ч.
Влак № 19233 заминава от гара Крумово в 22:42 ч.
Влак № 40113 пристига в гара Янтра в 13:30 ч.

За яснота примерните данни по-горе са подредени по номера на влака. В действителност масивът $A[1...n]$ не е подреден по никое поле.

Знаете още къде искате да отидете, къде сте сега и колко е часът. (Например сега е 9:46 ч., намирате се в София, а искате да отидете във Варна.)

Съставете алгоритъм, който за време $O(n \log n)$ в най-лошия случай открива как най-бързо да пристигнете с влак на желаното място. "Най-бързо" значи "в най-ранния възможен час". С други думи, брои се не само времето за пътуване, но и времето за престой по гарите.

Упътване: моделирайте задачата чрез ориентиран граф.

Опишете алгоритъма подробно с думи или на псевдокод. **(3 точки)**

Анализирайте времевата сложност на алгоритъма в най-лошия случай и докажете, че тя удовлетворява поставеното изискване. **(1 точка)**

За улеснение приемете, че цялото пътуване протича в рамките на едно денонощие, тоест завършва най-късно в 24:00 ч. Помислете как можете да се освободите от това ограничение, та алгоритъмът да обработва и информацията за нощните влакове. (Например влак № 8627 заминава от София в 22:55 ч. и пристига в Бургас в 05:43 ч. Часът на пристигане е по-ранен от часа на заминаване, защото по време на пътуването е настъпило ново денонощие.) От практически съображения (размерите на България не са големи) можете да приемете, че пътуването ще трае не повече от 24 часа, тоест може да се застъпи най-много с две последователни денонощия — текущото и следващото. **(1 точка)**

Решение: Най-напред построяваме ориентиран граф. Върховете на графа са n на брой — елементите на масива $A[1...n]$, тоест пристиганията и заминаванията на влаковете. Ребрата са два вида: пътуване и престой. За да получим по-малък граф, слагаме само нужните ребра. Ако например можем да стигнем от София до Карлово и от Карлово до Варна с един влак, излишно е да слагаме ребро от София до Варна.

По-точно, върховете на графа са двойки $\{\text{гара, час}\}$. Броят им не надхвърля n; може да е по-малък от n, ако няколко влака пристигат или заминават едновременно от една гара. Но най-лошият случай е, когато графът е голям, затова ще приемем, че графът има n върха. (Това не е съществено ограничение. То само ни спестява нова променлива за броя върхове.)

Създаваме масив $V[1...n]$, чиито елементи ще бъдат върховете на графа. Всеки връх има три полета: гара, час и списък на излизащите ребра. Отначало списъкът на ребрата е празен.

Групираме елементите на масива $A[1...n]$ по номер на влак. Събитията от всяка група сортираме по час. (Подразбира се, че измежду часовете от една и съща група няма еднакви.) В тази редица флаговете се редуват: "заминава", "пристига", "заминава", "пристига"... Първият флаг винаги е "заминава", а последният — "пристига" (ако няма нощни влакове).

Масивът $A[1...n]$ може да се подреди и така: първо го сортираме по час, а после по влак. За да получим групи (влакове), подредени по час, трябва втората сортировка да е устойчива. Понеже се интересуваме от устойчивост и бързина, но не и от количеството памет, можем да използваме сортиране чрез сливане. Това сортиране прави копие на входните данни, тоест изразходва много памет, но все пак разписанието на БДЖ не е чак толкова голямо, че да не може да се побере в паметта на съвременен компютър. Ако подреждаме масива по първия начин, тогава няма нужда от устойчивост; достатъчно е сортировката да е бърза. За целта можем да използваме пирамидалното сортиране, което пести памет.

Обхождаме така подредения масив $A[1...n]$ по нарастващ ред на индексите; прехвърляме от $A[k]$ във $V[k]$ информацията за гарата и часа (а може и номера на влака, ако преценим, че ще ни бъде нужен по-нататък, например за издаване на билет). Ако номерата на влаковете на $A[k]$ и $A[k-1]$ съвпадат и флагът на $A[k-1]$ е "заминава", а на $A[k]$ — "пристига", добавяме ребро от $V[k-1]$ към $V[k]$ с тегло — разликата на часовете (времето за пътуване).

Дотук построихме единия вид ребра на графа — тези, които съответстват на пътуване. Сега ще опишем как се добавят ребрата, съответстващи на престой. Всеки престой протича в рамките на една гара. Затова първо групираме елементите на масива $V[1...n]$ по гари, после за всяка гара сортираме елементите по час. (Вместо да сортираме всяка отделна група, можем да сортираме масива първо по час, после по гара, с устойчива втора сортировка.)

Обхождаме така подредения масив $V[1...n]$ по нарастващ ред на индексите. Ако гарите на елементите $V[k]$ и $V[k-1]$ съвпадат, т.е. ако елементите са от една група, добавяме ребро от $V[k-1]$ към $V[k]$ с тегло — разликата на часовете (времето за престой). Тази разлика може да бъде нула, ако няколко влака пристигат или заминават едновременно от гарата. Вместо номер на влак в информацията на реброто можем да пазим името на гарата.

Докато обработваме групата, съответстваща на гарата, където се намираме в момента, добавяме един допълнителен връх s с тази гара и с текущия час. Върхът s съответства на нашето начално състояние. Към него не водят ребра. От него излиза ребро на престой към върха, съответстващ на същата гара и следващото по време събитие за тази гара.

След като сме построили графа, търсим най-къс път от върха s до някой от върховете, съответстващи на гарата, където искаме да отидем. Ребрата са с неотрицателни тегла, затова можем да използваме ***алгоритъма на Дейкстра, реализиран с пирамида на Фибоначи.*** Няма нужда да построяваме цялото дърво на най-късите пътища от s до другите върхове. Можем да прекратим алгоритъма веднага щом той затвори връх, чиято гара съвпада с тази, в която искаме да отидем. Тази оптимизация е коректна, защото алгоритъмът на Дейкстра затваря върховете по нарастващ ред на разстоянието им от s. Тя обаче не променя порядъка на времевата сложност: желаният връх ще бъде затворен последен в най-лошия случай.

Анализ на сложността: Четирите сортировки от първия етап изискват време $\Theta(n \log n)$. Сложността на алгоритъма на Дейкстра с избраната реализация е равна на $\Theta(m + n \log n)$. Следователно общото време на двата етапа на нашия алгоритъм е равно на $\Theta(m + n \log n)$. Този анализ е недостатъчен: дължината на входа е n и не зависи от броя m на ребрата, затова и крайният резултат трябва да зависи само от n.

Линейното обхождане на сортираните масиви (през първия етап) става за време $\Theta(n)$. Ребра се добавят само през време на това обхождане, затова броят им $m = O(n)$, а времето на целия алгоритъм е $\Theta(m + n \log n) = O(n) + \Theta(n \log n) = \Theta(n \log n)$.

Можем да забележим, че графът, построен от нас, е ориентиран ацикличен граф, защото ребрата сочат винаги от по-ранен към по-късен час. Ето защо вторият етап на алгоритъма (търсенето на най-къс път) може да се ускори по порядък чрез ***динамично програмиране.*** (То обаче не затваря върховете по нарастващ ред на дължината на пътищата, затова не може да се прекрати предсрочно.) Времето на втория етап спада до $\Theta(m + n) = O(n) + \Theta(n) = \Theta(n)$. Това не променя общата сложност по порядък: тя остава $\Theta(n \log n) + \Theta(n) = \Theta(n \log n)$, но константата пред порядъка намалява. Затова тази оптимизация носи бонус от 3 точки.

Разсъжденията дотук не обхващат пътувания около полунощ: тогава часът на пристигане е по-ранен от часа на заминаване. Например влак № 8627 заминава от Ихтиман в 23:59 ч., а пристига в Костенец в 00:18 ч. Ако просто извадим началния час от крайния, ще получим отрицателна разлика (минус 23 часа и 41 минути). Дори да вземем абсолютната стойност (тоест плюс 23 часа и 41 минути), пак ще получим погрешно време за пътуване. Правилно е да добавим 24 часа към крайния час преди изваждането. Тоест трябва да извадим 23:59 ч. не от 00:18 ч., а от 24:18 ч., за да получим верния резултат: това пътуване трае 19 минути.

Пътуванията около полунощ се разпознават по следния начин: след групиране на $A[1...n]$ по номер на влак и сортиране на всяка група по час флаговете пак се редуват, обаче сега първият флаг е "пристига", а последният флаг е "заминава". В такъв случай добавяме ребро от последния към първия елемент в групата; теглото на това ребро (времето за пътуване) се пресмята, както е описано в предишния абзац.

Престоите около полунощ се отчитат така: след групиране на $V[1...n]$ по гара и сортиране на всяка група по час добавяме ребро от последния елемент към първия. Теглото на реброто (времето за престой) пресмятаме, като към часа на първия елемент от групата добавим 24 ч. и от сбора извадим часа на последния елемент. Времето за престой от 22:43 ч. до 01:17 ч. например е равно на 24:00 + 01:17 – 22:43 = 25:17 – 22:43 = 2:34 ч., тоест 2 часа и 34 минути.

Добавянето на ребрата за пътуване и престой около полунощ прави графа цикличен. Това не е пречка за алгоритъма на Дейкстра, обаче пречи на динамичното програмиране. Ако държим да запазим графа ацикличен (с цел да използваме динамично програмиране), можем да приложим следния трик: правим копие на графа, тоест всеки връх и всяко ребро, което не съдържат момента полунощ, се срещат два пъти. Ребрата, съдържащи полунощ, насочваме от началния връх в екземпляр № 1 на графа към крайния връх в екземпляр № 2. Така графът остава ацикличен, затова можем да използваме динамично програмиране. Това отново е оптимизация на втория етап на алгоритъма (търсенето на най-къс път), защото спестява време от порядък $n \log n$ (поради замяната на един алгоритъм с друг), а губи време от порядък n (за копирането и обхождането на графа).

СХЕМА ЗА ТОЧКУВАНЕ НА ДОМАШНОТО

Цялото домашно носи най-много 10 точки, разпределени по задачи, както е описано по-долу.

Задача 1 носи 2 точки — по 1 точка за всяка от следните стъпки:
— описание на разсъжденията (обосновка);
— получаване на отговора.

Задача 2 носи 3 точки — по 1 точка за всяка от следните стъпки:
— описание на алгоритъма в частта, пресмятаща най-големия възможен сбор от рейтинги;
— описание на алгоритъма в частта, отпечатваща служителите, поканени на празненството;
— подробна демонстрация на алгоритъма.

Задача 3 носи 5 точки, от които:
— 2 точки за описание на построението на графа;
— 1 точка за избор на алгоритъм;
— 1 точка за анализ на времевата сложност;
— 1 точка за обобщаване на алгоритъма.

Задача 3 носи бонус от 3 точки за оптимизиране на втория етап (търсенето на най-къс път): Тези 3 точки не влизат в предвидения максимум от 10 точки, тоест с тях сборът от точките за цялото домашно може да надхвърли 10 точки.