Изпит по дизайн и анализ на алгоритми (28.VIII.2022 г.)
за специалност "Компютърни науки", първи поток — СУ, ФМИ

Алгоритми и структури от данни от лекциите по ДАА и сложностите им по време да се ползват наготово. Българската терминология е задължителна!

Задача 1. Съставете и опишете словесно алгоритъм, който за време O(n) при всякакви входни данни разделя дадени $2n$ реални числа на n двойки с най-голям сбор от абсолютните разлики. Входът е числов масив $A\,[1\ldots 2n]$, а изходът е пермутация $i_1\,,\, j_1\,,\, i_2\,,\, j_2\,,\, \ldots\,,\, i_n\,,\, j_n$ на $1\,,\, 2\,,\, 3\,,\, \ldots\,,\, 2n$, за която

$$\left|A\,[\,i_1\,] - A\,[\,j_1\,]\right| + \left|A\,[\,i_2\,] - A\,[\,j_2\,]\right| + \cdots + \left|A\,[\,i_n\,] - A\,[\,j_n\,]\right| = \max.$$

Задача 2. Неориентиран свързан граф без кратни ребра и без примки е зададен чрез списъци на съседствата. Графът притежава n върха и m ребра. Всяко ребро на графа е оцветено в бяло или в черно (оцветяването е дадено). Съставете алгоритъм с времева сложност $\Theta(m + n)$ при най-лоши входни данни, който намира покриващо дърво с възможно най-много бели ребра. Предложете словесно описание, псевдокод и анализ на алгоритъма и структурите от данни. Линейният порядък на времевата сложност трябва да е точен, не приблизителен.

Време за работа: два астрономични часа (сто и двайсет минути).

Оценката = 2 + точките. За всяка пълно решена задача се дават две точки: първата — за съставяне на коректен и бърз алгоритъм; втората — за доказване на тези свойства (тоест за анализ на алгоритъма).

Решения на задачите
от изпита по дизайн и анализ на алгоритми (28. VIII. 2022 г.)
за специалност "Компютърни науки", първи поток — СУ, ФМИ

Задача 1. Както и да разкриваме модулите, от всеки модул ще излиза едно събираемо със знак плюс и едно със знак минус. Общо за целия израз ще има точно n събираеми със знак плюс и точно n събираеми със знак минус. Полученият израз (алгебричен сбор) ще притежава най-голяма стойност тогава и само тогава, когато събираемите със знак плюс са възможно най-големи, а събираемите със знак минус са възможно най-малки.

И така, трябва да разделим дадените числа на големи и малки, което може да се извърши с линейна времева сложност (разделяне по Ломуто или по Хоор), стига да е известен разделителят. Щом искаме да има n големи и n малки числа (т.е. равен брой), то за разделител трябва да използваме медианата на масива. Тя се намира с линейна времева сложност посредством алгоритъма PICK.

Окончателно, нашият алгоритъм се състои от три стъпки:

1) Чрез алгоритъма PICK намираме медианата на дадения масив $A[1\ ...\ n]$.

2) Разделяме масива $A[1\ ...\ n]$ относно медианата на малки и големи числа.

3) Групираме едно малко и едно голямо число, без значение как точно. Например групираме първия с последния елемент, втория с предпоследния и тъй нататък.

Група наричаме всяка двойка елементи, чиято разлика се пресмята някъде в алгебричния израз. Тоест група образуват елементите с индекси i_k и j_k според обозначенията от условието на задачата. За да можем да отпечатаме пермутацията на индексите, трябва да поддържаме един масив от индекси:

1) На първата стъпка инициализираме този масив с числата от 1 до $2n$.

2) На втората стъпка (при разделянето на масива A относно медианата), когато разместваме елементи на масива A, разместваме и съответните елементи на допълнителния масив.

3) На третата стъпка отпечатваме елементите на допълнителния масив: първия елемент, последния, втория, предпоследния и т.н. (Има и други начини; важно е само да редуваме малките и големите елементи на дадения масив.)

Всички стъпки — инициализирането на помощния масив от индекси и намирането на медианата на дадения числов масив, разделянето на числата на големи и малки (относно медианата), отпечатването на индексите — имат линейни времеви сложности, ето защо същото важи за целия алгоритъм: времевата му сложност е $\Theta(n)$ при всякакви входни данни.

Тази задача (в малко по-различна формулировка) се съдържа под номер 1 в публикуваното примерно второ контролно по дизайн и анализ на алгоритми от летния семестър на 2017 / 2018 уч.г.

Задача 2. Понеже покриващите дървета имат еднакъв брой ребра ($n - 1$), то белите ребра са най-много, когато черните са най-малко. С други думи, търсим минимално покриващо дърво: черните ребра имат тегло 1, белите — 0. За целта използваме алгоритъма на Прим—Ярник или алгоритъма на Крускал. Техните времеви сложности не са линейни, обаче и при двата алгоритъма можем да постигнем значително ускорение.

При алгоритъма на Прим—Ярник няма нужда от приоритетна опашка: стигат ни три списъка от върхове (със стойности нула, единица и безкрайност). Използваме двусвързани списъци, за да можем лесно да трием посочен елемент (можем да минем и с едносвързани списъци, но с цената на изкуствен трик: когато искаме да посочим елемент от списък, трябва да използваме указател не към желания от нас елемент, а към неговия предшественик в списъка). В началото на алгоритъма слагаме всички върхове в списъка "безкрайност" с изключение на един връх (избран произволно) — той отива в списъка "нула" и става начален връх за алгоритъма; списъкът "единица" е празен отначало.

Извличаме произволен връх u от списъка "нула"; в случай че той е празен, извличаме произволен връх u от списъка "единица"; ако и този списък е празен, алгоритъмът приключва работа: успешно, тоест намерил е покриващо дърво с минимален брой черни ребра — ако списъкът "безкрайност" е празен също; неуспешно (няма покриващо дърво) — ако списъкът "безкрайност" не е празен. Последният случай се отнася само за несвързан граф и е изключен по условие.

Извлеченият връх u става текущ връх. Обхождат се всички ребра $\{u\ ;\ v\}$. Ако върхът v е затворен, т.е. присъединен е към дървото, той не се обработва. (Отначало няма затворени върхове.) Иначе правим опит за релаксация на v:
— ако v е в списъка "нула", релаксация не е възможна;
— ако v е в списъка "единица" и реброто $\{u\ ;\ v\}$ е бяло, преместваме върха v в списъка "нула" и караме v да запомни реброто $\{u\ ;\ v\}$ (всеки връх помни по едно входящо ребро, чрез което ще се присъедини към нарастващото дърво);
— ако v е в списъка "безкрайност", преместваме v в някой от другите списъци: в списъка "нула" — ако $\{u\ ;\ v\}$ е бяло ребро; в "единица" — ако е черно; и в двата случая караме v да запомни реброто $\{u\ ;\ v\}$.

След това обявяваме u за затворен връх и го добавяме към дървото заедно с реброто, запомнено от u. Началният връх не притежава такова ребро. Отначало дървото е празно. Нараства постепенно, започвайки от началния връх и присъединявайки все нови и нови върхове и ребра. Накрая се получава минимално покриващо дърво или съобщение, че такова дърво няма.

За да знае всеки връх в кой списък се намира, пазим още един масив от състоянията на всички върхове:

0 — върхът се намира в списъка "нула";
1 — върхът се намира в списъка "единица";
2 — върхът се намира в списъка "безкрайност";
състояние -1 означава, че върхът е затворен (присъединен към дървото).

Псевдокод на алгоритъма:

```
ALG(Adj[1...n]) // Върхове са целите числа 1, 2, ... , n.
 // Adj[u] е списък на ребрата, излизащи от върха u;
 // всяко ребро пази другия връх и цвят (бял или черен).
 π[1...n]: предшествениците на върховете в дървото.
 σ[1...n]: състояния на върховете.
 L0, L1, L2: двусвързани списъци от върхове.
 L0 ← ∅
 L1 ← ∅
 L2 ← ∅
 for v ← 2 to n do
    π[v] ← 0
    σ[v] ← 2
    добави v към L2
 π[1] ← 0
 σ[1] ← 0
 добави 1 към L0
 while L0 ≠ ∅ or L1 ≠ ∅ do
    if L0 ≠ ∅
       u ← първия връх от L0
       премахни u от L0
    else
       u ← първия връх от L1
       премахни u от L1
    σ[u] ← -1 // Затваряне на върха u.
    for v ∈ Adj[u] do
       if σ[v] > {u; v}.colour // 0 — бяло; 1 — черно
          π[v] ← u
          премахни v от Lσ[v]
          σ[v] ← {u; v}.colour
          добави v към Lσ[v]
 if L2 ≠ ∅
    return NULL // Няма покриващо дърво.
 return π[1...n] // Има покриващо дърво.
```

Анализ на времевата сложност на променения алгоритъм на Прим—Ярник: С всеки връх на графа се извършват следните операции: добавяне към списък в началото на алгоритъма, най-много две релаксации (от ∞ към 1 и от 1 към 0), извличане от списък като текущ връх и затваряне — между три и пет операции за всеки отделен връх на графа. За всички върхове на графа: $\Theta(n)$ операции. Освен това всяко ребро се обработва най-много два пъти — по веднъж за всеки от краищата си, когато стават текущи върхове. За всички ребра: $\Theta(m)$ операции. Цялото време на алгоритъма е от порядък $\Theta(m + n)$ при всякакви входни данни. Този алгоритъм отговаря на всички изисквания и носи пълен брой точки.

По принцип задачата може да бъде решена и с алгоритъма на Крускал, но желаният порядък е достижим само приблизително. Сортирането отпада: просто разделяме ребрата по цвят. Това изисква време $\Theta(m)$, не $\Theta(m \log m)$. Така ускоряваме първия, по-бавния етап от алгоритъма. Не е ясно дали и как можем да ускорим втория етап. Разделянето на ребрата по цвят не ни помага при проверката дали дадено ребро затваря цикъл. Не е известен по-бърз начин от използването на структурата Union-Find, а тя изразходва време $\Theta(m\,\alpha(n))$. Теоретично погледнато, то не е линейно: функцията $\alpha(n)$ расте неограничено. Но тя расте толкова бавно, че $\alpha(n) < 5$ за всички практически възможни n. Тоест това решение е практически бързо, обаче не отговаря на изискването за точен, а не приблизителен асимптотичен порядък на времевата сложност, поради което не носи точки.

Тази задача е публикувана под номер 2 във второто домашно по ДАА от зимния семестър на 2020 / 2021 уч.г.

Двете задачи са публикувани на страницата с учебни материали по ДАА, обявена в началото на летния семестър 2021 / 2022.