Hello, World!

Марко Савић

Техника meet-in-the-middle

Техника meet-in-the-middle је начин решавања проблема његовим дељењем на два дела приближно једнаке величине. Идеја је да се за сваки део нађу сва могућа решења, а потом да се на ефикасан начин пробају упарити погодни парови решења (једно из једног, а друго из другог дела) у циљу добијања оптималног решења за почетни проблем.

Слично техници подели-па-владај (divide & conquer), и овде проблем делимо на делове. Међутим, постоји битна разлика између ове две технике, јер овде делови нису еквивалентни почетном проблему и није их могуће решавати истим поступком као и почетни проблем, те није реч о рекурзивном приступу.

Илустроваћемо најпре ову технику следећим проблемом.

Проблем 4-SUM

4-SUM: Дат је скуп M који садржи n природних бројева. Да ли у том скупу постоје четири елемента чији је збир једнак нули? Дозвољено је искористити исти елемент више пута.

На пример, за скуп $M = \{3,5,-7,1,-2\}$ одговор је потврдан, јер је $3 + 1 + (-2) + (-2) = 0$.

Тривијалан алгоритам који решава овај проблем испробава све могуће четворке елемената, па је његова временска сложеност $O(n^4)$.

Да бисмо ефикасније решили овај проблем трансформишимо израз $a + b + c + d = 0$ у израз $a + b = -(c + d)$. Уместо да налазимо збирове свака четири елемента, израчунајмо све могуће збирове нека два елемента (на тај начин смо проблем поделили на "пола"). Питање на које смо свели проблем је да ли међу тим збировима постоје два супротна броја. За ову проверу користићемо хеш табелу. Хеш табела (Hash table) је структура података која представља скуп елемената и омогућава да се брзо добије одговор на питање да ли је неки елемент у скупу. Ако бисмо све збирове ставили у хеш табелу, онда је довољно још једном проћи кроз збирове и проверавати да ли се за сваки од њих супротан број налазу у хеш табели.

Алгоритам 4-sum
Улаз: Скуп A
Излаз: Да ли постоје четири елемента чији је збир 0.
for $a \in M$
    for $b \in M$
        Убаци $a + b$ у хеш табелу.

for $c \in M$
    for $d \in M$
        if хеш табела садржи број $-(c + d)$
            return true

return false

Овај проблем смо решили овако једноставно јер смо дозволили коришћење хеш табеле. Под претпоставком да су убацивање броја у хеш табелу и провера да ли се број већ налази у њој операције које се извршавају у константном времену, тј. готово тренутно, сложеност овог алгоритма је $O(n^2)$. Иако је у пракси често случај да ове операције раде веома брзо, то се теоретски не може гарантовати. Зато у наредним примерима нећемо користити хеш табелу, иако се могу конструисати верзије предложених решења које је користе.

Техника meet-in-the-middle је, ипак, најкориснија за проблеме за чије решавање нам нису познати ефикасни алгоритми (NP-тешки проблеми). Њена примена нам омогућава да дуплирамо величину проблема коју можемо решити у датом времену, али за то углавном плаћамо коришћењем велике количине меморије.

Пре него што дамо неке такве проблеме на којима се ова техника може применити, погледајмо следећи једноставан проблем који ће нам касније послужити као основа за конструисање неких алгоритама.

Проблем 2-SUM

2-SUM: Дата су два низа реалних бројева, A и B, оба дужине n, и реалан број s. Да ли постоје два елемента, један из A, а други из B, чији је збир једнак s?

Тривијално решење овог проблема би било да се испробају сви могући парови елемената, што би резултовало алгоритмом временске сложености $O(n^2)$. Међутим, постоји паметнији начин да се реши овај проблем.

Сортирајмо оба низа неопадајуће. Нека су $a_1 \leq a_2 \leq \cdots \leq a_n$ и $b_1 \leq b_2 \leq \cdots \leq b_n$ елементи низова A, односно B након сортирања. Алгоритам проналази да ли постоји пар (a_i, b_j) елемената за које важи $a_i + b_j = s$. У току извршавања алгоритма у променљивој i чувамо позицију у низу A, а у променљивој j чувамо позицију у низу B. У почетку i поставимо да показује на први елемент низа A, а j да показује на последњи елемент низа B. У свакој итерацији проверавамо да ли је $a_i + b_j$ једнако, мање или веће од s. Уколико је једнако, нашли смо тражени пар, и извршавање престаје и алгоритам враћа потврдан одговор. Уколико је мање, повећавамо i за један, а уколико је веће, смањујемо j за један. Ово понављамо све док i не дође до краја низа A или док j не дође до почетка низа B. Ако у овом процесу није пронађен ни један пар са траженом сумом, такав пар онда не постоји, и алгоритам на крају враћа одричан одговор.

Алгоритам 2-sum
Улаз: Сортирани низови A и B, и број s.
Излаз: Да ли постоји пар (a_i, b_j) такви да је $a_i + b_j = s$.

```
        i ← 1
        j ← n
while i ≤ n ∧ j ≥ 1
                if a_i + b_j = s
                        return true
                if a_i + b_j < s
                        i ← i + 1
                if a_i + b_j > s
                        j ← j − 1

return false
```

Исправност овог алгоритма се заснива на следећем запажању. Ако је $a_i + b_j < s$, тада за свако $k \leq i$ важи $a_k + b_j < s$, и сви елементи a_k могу бити избачени из разматрања. Слично, ако је $a_i + b_j > s$, тада за свако $l \geq j$ важи $a_i + b_l > s$, и сви елементи b_l могу бити избачени из разматрања.

Овај алгоритам извршава $O(n)$ корака, па је укупна временска сложеност, када урачунамо и сортирање, $O(nlogn)$.
Ради једноставности претпоставили смо да су низови исте дужине, али приметимо да исти поступак можемо применити и ако низови нису обавезно исте дужине.

Ако би се захтевало исписивање свих парова за које важи $a_i + b_j = s$, за то би било неопходно $O(n^2)$ време, јер таквих парова може бити n^2 (ако су сви бројеви исти). Међутим, ако се тражи само број таквих парова, могуће је модификовати дати алгоритам тако да добијемо тражени број, а време извршавања остане исто. Остављамо читаоцу за вежбу да то уради.

Проблем SUBSET-SUM

Примену meet-in-the-middle технике свођењем на 2-SUM приказаћемо на следећем проблему.
SUBSET-SUM. Дат је скуп бројева $A = \{a_1, a_2, \dots, a_n\}$ и број s. Да ли постоји подскуп скупа A чији је збир s? (Дозвољавамо да скуп може садржати исте елементе више пута, односно реч је о мултискупу.)
На пример, ако је $A = \{5, 5, 10, 60, 61\}$ и $s = 70$, тражени подскуп је $\{5, 5, 60\}$.

Уколико су сви дати бројеви релативно мали природни бројеви, познато је да се овај проблем може ефикасно решити динамичким програмирањем. Међутим, у општем случају, када бројеви могу бити произвољни реални бројеви, овај проблем је NP-тежак, и за њега не знамо ефикасан алгоритам. Тривијалан алгоритам је пролазак кроз све подскупове и проверавање за сваки да ли му је збир једнак s. Временска сложеност овог тривијалног алгоритма је $O(2^n)$.

Да бисмо проблем решили ефикасније, применом meet-in-the-middle технике, поделићемо скуп A на два скупа, $X = \{a_1, a_2, \dots, a_{\lfloor n/2 \rfloor}\}$ и $Y = \{a_{\lfloor n/2 \rfloor + 1}, \dots, a_n\}$. За сваки од ових скупова израчунајмо скуп који садржи збирове свих могућих подскупова. Укупан број израчунатих збирова је највише $2^{n/2+1}$. Нека су S_X и S_Y скупови са израчунатим збировима подскупова скупова X и Y. Решење SUBSET-SUM проблема постоји ако и само ако постоје $s_X \in S_X$ и $s_Y \in S_Y$ такви да је $s_X + s_Y = s$. Да бисмо нашли такве s_X и s_Y, свешћемо овај проблем на 2-SUM проблем. Улаз за 2-SUM проблем представљају два низа од којих ћемо један добити од елемената скупа S_X, а други од елемената скупа S_Y. Обзиром на то да су низови величине највише $2^{n/2+1}$, време потребно да решимо 2-SUM проблем претходно описаним алгоритмом је $O(2^{n/2} log 2^{n/2}) = O(n2^{n/2})$.

Подсетимо се да је временска сложеност тривијалног алгоритма $O(2^n)$. То у пракси значи да тим алгоритмом за једну секунду могу бити решени примери овог проблема у којима је n највише 30. Међутим, са описаним ефикаснијим алгоритмом, који користи meet-in-the-middle технику, скоро да дуплирамо величину примера решивих у неком разумном времену. Да бисмо

схватили колико је такво побољшање значајно, приметимо да би тривијалном алгоритму време потребно да реши дупло већи пример (у коме је $n = 60$) било око 2^{30}секунди, односно преко 34 године! Побољшаном алгоритму треба свега минут!

Задаци

1. Како бисте решили 4-SUM проблем без употребе хеш табеле? Колика би била временска сложеност таквог алгоритма?
2. Дат је скуп U од n целих бројева. Конструисати што ефикаснији алгоритам који одговара на питање да ли постоје $a, b, c, d, e, f \in U$ такви да важи $(a * b + c)/d - e = f$.
3. Модификовати 2-SUM алгоритам тако да налази број парова за које важи $a_i + b_j = s$. Водити рачуна да се овом модификацијом не повећа временска сложеност.
4. Постоје је m сијалица и n тастера. Сваки тастер делује на неки подскуп сијалица тако што им мења стање из укључена у искључена и обрнуто. У почетку су све сијалице искључене. Конструисати што ефикаснији алгоритам који утврђује да ли је притискањем тастера могуће укључити све сијалице.
5. Дато је n бројева. Конструисати што ефикаснији алгоритам који проверава да ли збир свих бројева може бити нула ако
    a. променимо знак неким бројевима.
    b. променимо знак највише k бројева.

Статијата прв пат е објавена во списанието Тангента на ДМ на Србија во 2016/17 година