Задачи за подмножества

Петър Петров

February 25, 2020

Задача 1. *Дадени са n числа - $a_1, ..., a_n$. За всяко подмножество разглеждаме сумата от неговите елементи. Намерете сумата на тези числа.*
Ограничения: $1 \leq n \leq 10^6, 1 \leq a_i \leq 10^9$.

Задача 2 (https://www.urionlinejudge.com.br/judge/en/problems/view/1690)**.** *Дадени са n положителни числа - $a_1, ..., a_n$. Намерете най-малкото положително число, което не може да се представи като сума на числа от тази редица, без да ги повтаряме.*
Ограничения: $1 \leq n \leq 10^6, 1 \leq a_i \leq 10^9$.

Задача 3. *Дадени са n положителни числа - $a_1, ..., a_n$. Колко различни числа могат да се представят като суми на числа от тази редица.*
Ограничения: $1 \leq n \leq 10^3, 1 \leq a_i \leq 10^4$.

Задача 4 (Codeforces)**.** *Дадени са n числа със сума $2n$. Може ли да намерим подмножество със сума равна на k.*
Ограничения: $1 \leq n \leq 10^6, 1 \leq k \leq 2n$.

Задача 5 (Zero sum subset)**.** *Дадени са n числа - $a_1, ..., a_n$. Може ли с k от тези числа да образуваме сума нула, като едно число може да участва няколко пъти в сумата?*
Ограничения: $1 \leq n \leq 10^5, 1 \leq k \leq 10^5, -10^5 \leq a_i \leq 10^5$.

Задача 6 (https://arena.olimpiici.com/#/catalog/406/problem/1127)**.**

Задача 7 (https://codeforces.com/contest/1270/problem/G)**.** *Дадена е редица от n числа - $a_1, ..., a_n$, такива че за всяко i и изпълнено $i-n \leq a_i \leq i-1$. Намерете непразно подмножество от тези числа със сума нула.*
Ограничения: $1 \leq n \leq 10^6$.

Задача 8 (https://www.codechef.com/problems/ANUCBC)**.** *Дадени са n числа и q заявки. На всяка заявка е дадено число m и трябва да намерите броя подмножества на n със сума кратна на m.*
Ограничения: $1 \leq n \leq 10^5, 1 \leq q \leq 30, 1 \leq m \leq 100$.

Задача 9 (https://csacademy.com/contest/round-79/task/smallest-subsets/statement/)**.** *Дадени са n числа - $a_1, ..., a_n$. Разглеждаме всички 2^n подмножества подредени по тяхната сума(празното подмножество има сума нула). Отпечатайте сумата от елементите на първите k подмножества.*
Ограничения: $1 \leq n \leq 10^5, 1 \leq k \leq min(10^5, 2^n), -10^9 \leq a_i \leq 10^9$.

Задача 10. *Дадени са n предмета, i-ят от тях с тегло w_i и стойност v_i. Разполагате с раница, която побира тегло W. Трябва да сложите някой предмети в раницата, така че общото им тегло да не надвишава W. Каква е най-голямата стойност която може да получите.*
Ограничения: $1 \leq n \leq 10^3, 1 \leq W \leq 10^5, 1 \leq w_i \leq 10^9, 1 \leq v_i \leq 10^9$.

Задача 11 (USACO Training: Subset Sums)**.** *Дадени са числата от 1 до n. По колко различни начина може да ги разделим на две групи с равни суми, като всички числа са в една от двете групи.*
Ограничения: $1 \leq n \leq 39$.

Задача 12. *Дадени са n предмета, i-ят от тях с тегло w_i, стойност v_i и разполагане с c_i копия на този предмет. Разполагате с раница, която побира тегло W. Трябва да сложите някой предмети в раницата, така че общото им тегло да не надвишава W. Каква е най-голямата стойност която може да получите.*
Ограничения: $1 \leq n \leq 10^3, 1 \leq W \leq 10^5, 1 \leq w_i \leq 10^5, 1 \leq v_i \leq 10^9, 1 \leq c_i \leq 10^5$.

Решение. Най-тривиалното решение би било да разглеждаме всяко копие като отделен предмет. Така ще имаме задачата за раница като броя предмети ще бъде $N * C$. Така получаваме решение $N * C * W$.
Трябва да се опитаме да оптимизираме нещата. Един вариант е да пробваме да намалим копията. Идеята е пак да гледаме един предмет няколко пъти, но разликата е че не искаме да имаме само единични копия. Важно е като вземем различните комбинации от предмети на които сме го разделили да може да получим всички първоначални варианти. Например ако i-ят предмет се среща 7 пъти може да го заменим с три предмета с по едно копие - (w_i, v_i), $(2w_i, 2v_i)$ и $(4w_i, 4v_i)$. Така с избор на един, два или три от новите предмети може да получим като резултат всеки от седемте първоначални варианта. Как да комбинираме копията оптимално, че да имаме всички първоначални варианти? Решението е грийди - винаги комбинираме толкова копия колкото не може да получим до момента. В началото ни трябва едно копие. После взимаме 2 копия. С 1 и 2 копия може да направим 1, 2 и 3. Сега ни трябва да комбинираме 4 копия и т.н. Получаваме степените на двойките. Така ако имаме 26 ще направим комбинации от 1, 2, 4 и 8. В резултат ще останат 11 предмета които не се побират в следващата степен. За това тях ги гледаме отделно. Така 26 го разбиваме на групи от 1, 2, 4, 8 и 11 предмета. С тях може да направим всички бройки от 1 до 26 и нито една повече. Така със степените на двойките от c_i предмета получаваме $\lg c_i$. Това е съществено подобрение на първоначалния алгоритъм от $O(N * C * W)$ на $O(N * \lg C * W)$.

```
#include <bits/stdc++.h>
using namespace std;

long long dp[1000001];

void fillKnapsack(long long w, long long v, int W) {
    for (int i = W; i >= w; i--) {
        if (dp[i-w] == -1) continue;
        dp[i] = max(dp[i], dp[i-w]+v);
    }
}

void fillWithCombinations(long long w, long long v, long long c, int W) {
    long long cnt = 1;
    while (c != 0) {
        fillKnapsack(w*cnt, v*cnt, W);
        c -= cnt;
```

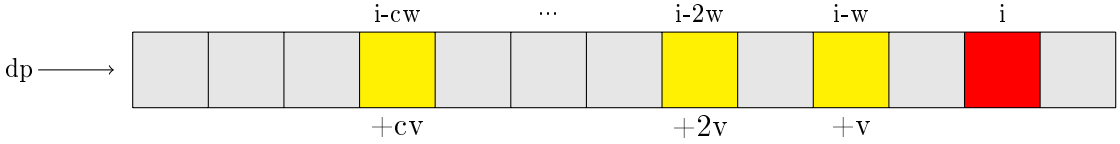
```
        cnt = min(2*cnt, c);
    }
}

int main() {
    int N, W; cin >> N >> W;

    for (int i = 1; i <= W; i++) dp[i] = -1;
    for (int i = 1; i <= N; i++) {
        long long w, v, c; cin >> w >> v >> c;
        fillWithCombinations(w, v, c, W);
    }
    cout << dp[W] << endl;

    return 0;
}
```

Може ли да постигнем още по-добро решение? Остава да проверяваме всички копия на един предмет едновременно. Да разгледаме какво ни трябва да запълним $dp[i]$, когато разглеждаме предмет със тегло, цени и бройки - w, v, c.



Интересуваме се от най-голямата стойност измежду $dp[i-w]+v, dp[i-2w]+2v, ...dp[i-cw]+cv$.

$$dp[i] = max(dp[i], dp[i-w]+v, dp[i-2w]+2v, ...dp[i-cw]+cv)$$

Да разгледаме и $dp[i-w]$.

$$dp[i-w] = max(dp[i-w], dp[i-2w]+v, dp[i-3w]+2v, ...dp[i-(c+1)w]+cv)$$

Основният проблем е, че всяко събираемо зависи от позицията си спрямо i и когато i се промени трябва да търсим максимум от съвсем различни числа. Например да видим $dp[i-2w]$ - първият път ни интересува $dp[i-2w]+2v$, а после $dp[i-2w]+v$.
Има вариант да съберем/извадим всички елементи с едно и също число в началото, така че тази сума да не зависи от индекса. В случая е достатъчно от всяко число в началото $dp[i]$ да извадим $\lfloor i/w \rfloor * v$. Нека да разгледаме масива $upd[i] = dp[i] - \lfloor i/w \rfloor * v$ и да разгледаме как изглеждат формулите с него като заместим $dp[i] = upd[i] + \lfloor i/w \rfloor * v$.

$$dp[i] = max(dp[i], dp[i-w]+v, dp[i-2w]+2v, ...dp[i-cw]+cv)$$

$$dp[i] = max(upd[i]+\lfloor i/w \rfloor * v, upd[i-w]+\lfloor (i-w)/w \rfloor * v+v, upd[i-2w]+\lfloor (i-2w)/w \rfloor * v+2v, ...upd[i-cw]+\lfloor (i-cw)/w \rfloor$$

$$dp[i] = max(upd[i]+\lfloor i/w \rfloor * v, upd[i-w]+\lfloor i/w \rfloor * v, upd[i-2w]+\lfloor i/w \rfloor * v, ...upd[i-cw]+\lfloor i/w \rfloor * v)$$

$$dp[i] = max(upd[i], upd[i-w], upd[i-2w], ...upd[i-cw]) + \lfloor i/w \rfloor * v$$

Сега за всяко $dp[i]$ трябва да намерим най-голямата стойност от няколко числа от масива

upd. Когато сменим на $dp[i-w]$ ще трябва да търсим максималното число от същите, като евентуално променим само по едно в двата края.
За това ще смятаме $dp[i]$ по групи за различните остатъци при деление на w. Например тези с остатък нула са - $dp[0], dp[w], dp[2w], dp[3w], ...$ Сега остава да търсим в максимума в прозорец от c числа. За целта може да ползваме стандартния алгоритъм, които използва дек.

```
#include <bits/stdc++.h>
using namespace std;

long long w, v, c;
long long dp[1000001];
deque<pair<long long, long long>> d;

void addToDeque(long long weight) {
    if (weight < 0 || dp[weight] == -1) return;

    long long dVal = dp[weight]-weight/w*v;
    if (!d.empty() && d.front().second <= dVal) d.pop_front();
    d.push_front({weight, dVal});
}

int main() {
    int N, W; cin >> N >> W;

    for (int i = 1; i <= W; i++) dp[i] = -1;

    for (int i = 1; i <= N; i++) {
        cin >> w >> v >> c;
        for (int j = W; j > max(0ll, W-w); j--) {
            for (int k = 1; k <= c; k++) {
                long long nextW = j-k*w;
                if (nextW < 0) break;
                addToDeque(nextW);
            }
            for (int k = j; k >= 0; k -= w) {
                if (!d.empty()) dp[k] = max(dp[k], dp[d.back().first]+(k-d.back().first)/w*v);
                while (!d.empty() && d.back().first >= k-w) d.pop_back();
                long long nextW = k-(c+1)*w;
                addToDeque(nextW);
            }
        }
    }
    cout << dp[W] << endl;

    return 0;
}
```

Бързина в действителност

Въпреки, че двата алгоритъма имат сложности съответно $O(N * \lg C * W)$ и $N * W$, втория алгоритъм има по-висока константа и по-лесният за писане, всъщност е доста по-практичен за състезателни задачи.

□

Задача 13. *http://codeforces.com/contest/95/problem/E*