split_in_half

HackerRank

Дадено е множество S от цели положителни числа (числата в него могат и да се повтарят). Тегло t(S) на множеството S наричаме сумата от елементите му. Да се раздели множеството на две части S1 и S2 така, че теглата на двете части да бъдат максимално близки, т.е.|t(S1) – t(S2)| да е най-малкото възможно число.

Жокер: Задачата е от динамично програмиране (Братска подялба).
В нашата задача t(S1) = сумата от всички t(S) числа минус t(S2). Като заместим във формулата се получава |t(S1) – t(S2)| = |(t(S) - t(S2)) - t(S2)| = |t(S) - 2 * t(S2)|. Ако приемем, че t(S2) е търсената по-малка сума => t(S2) <= t(S1) и заместим t(S1) = t(S) - t(S2) в неравенството получаваме, че t(S2) <= t(S) - t(S2) => 2 * t(S2) <= t(S) => t(S2) <= t(S) / 2 => |t(S) - 2 * t(S2)| = t(S) - 2 * t(S2). От това че търсим минимума на t(S) - 2 * t(S2) при t(S2) <= t(S) / 2 следва, че t(S2) е най-голямата сума по-малка или равна на половината от сумата на всички числа.
Основната идея на динамичното програмиране е, че имайки помощните стойности (в тази задача това са сумите на всички подмножества с N елемента) от които се е получил отговора на задачата (най-голямата от тези сума, която е по-малка или равна на половината от сумата на всички числа) за дадено N и прилагайки някаква проста рекурента формула можем да получим всички възможни помощни стойности (сумите на всички подмножества с N + 1 елемента), които за нужни за да открием отговора на задачата за N+1 (досегашните N елемента плюс още един нов елемент A_{N+1}). В нашата задача множеството от сумите на всички подмножества с N+1 елемента включва всичките суми от подможества на първите N елемента плюс още толкова елемента, които се генерират, като към всяка от сумите на подмножествата от първите N елемента се прибави новия N+1-ви елемент. Записано формално, казаното изглежда така:
Нека $S_n = \{A_1, A_2, \dots A_{n-1}, A_n\}$ и $SUMS(S_n) = \{SUM_1, SUM_2, \dots , SUM_{p-1}, SUM_p\}$
Добавяме новият елемент A_{n+1}
$S_{n+1} = S_n \cup \{A_{n+1}\} = \{A_1, A_2, \dots A_n, A_{n+1}\}$ =>
=> $SUMS(S_{n+1}) = SUMS(S_n) \cup \{SUM_1+A_{n+1}, SUM_2+A_{n+1}, \dots , SUM_{p-1}+A_{n+1}, SUM_p+A_{n+1}\} = \{SUM_1, SUM_2, \dots , SUM_{p-1}, SUM_p, SUM_1+_{An+1}, Sum_2+A_{n+1}, \dots , SUM_{p-1}+A_{n+1}, SUM_p+A_{n+1})\}$ - това е рекурентната формула за тази задача.
За да разберем по-добре Динамичното програмиране можем да направим аналогия между него и метода на индукцията в математиката - чрез него обикновено се доказват рекурентни формули (Примерно: Фибоначи(N) = Фибоначи(N-1) + Фибоначи(N-2)), които пък са основен инструмент при динамичното програмиране.
Да видим как действа практически рекурентната формула за тази задача.
1) В нашия случай при Ni = 0 нямаме никакви елементи и можем да приемем, че множеството от сумите на всички подмножества е {0}.
Ni = 0, S0={} => SUMS(S0) = {0}. Това се явява стартовия граничен случай на динамичното програмиране при който ние инициализираме стартовото множество с помощни стойности интерпретирайки логически правилно условието на задачата.
2) Ni = 1, S1={A1} => SUMS(S1) = SUMS(S0) U (SUM1 + A1) = {0, 0 + A1} = {0, A1}
3) Ni = 2, S2={A1, A2} => SUMS(S2) = SUMS(S1) U (SUM1 + A2, SUM2 + A2) = {0, A1} U {0 + A2, A1 + A2} = {0, A1, A2, A1 + A2}
4) Ni = 3, S3={A1, A2, A3} => SUMS(S3) = SUMS(S2) U (SUM1 + A3, SUM2 + A3, SUM3 + A3, SUM4 + A3) = {0, A1, A2, A1 + A2} U {0 + A3, A1 + A3, A2 + A3, A1 + A2 + A3} = {0, A1, A2, A1 + A2, A3, A1 + A3, A2 + A3, A1 + A2 + A3}
Тези изчисления продължаваме докато Ni = N(въведено от конзолата за дадения пример).
Виждате че при този подход (копирайки предните суми и добавяйки към тях новото Ai) се получават всички възможни суми от подмножества на елементите Ai (те са N-те числа за всеки пример).
Тук възниква един проблем - множеството с всички суми за Ni елемента удвоява размера си двойно на всяка стъпка. При Ni = MaxN = 1000 то би трябвало да съдържа 2^1000 елемента, а това си е множко при положение че атомите във Вселената би трябвало да са около 2^80. Но този проблем се решава много лесно, като в

текущото множество със сумите държим само уникалните суми. Колко максимум са те? Най-голямото число е 9999, числата са максимум 1000, така че максималната сума е 9999 * 1000 = 9999000. Минималната е 2 (две числа по 1). Всички суми са цели числа, следователно максималнит бой на сумите е 9999000 - 2 + 1 = 9998999, което е около 10,000,000. Спокойно можем да си заделим булев масив с 10,000,000 елемента и в него да маркираме дали сумата е била получена при предни изчисления на суми на подмножества и ако не е в този масив да маркираме с true, че сумата е вече генерирана. Множеството от всички текущи суми можем да реализираме с обикновен масив, който за Ni= 0 инициализираме с един нулев елемент. За дадено Ni итерираме всички негови суми от i-1 елемента (до запомнената текуща дължина на масива преди да започнем да комбинираме с новото Ai). Към всяка от тях добавяме Ai. Ако в булевия масив не е била генерирана тази сума (false на позиция сумата) я маркираме с true, а към края на масива със всички текущи суми добавяме тази нова генерирана сума. Едновременно с това проверяваме дали тя е най-добрия отговор (<=SumAll/2 и по-голяма от предишната най-добра) и го обноваваме, ако тя е по-добра.
Оптимизирайте си кода за скорост, иначе може да не влезете по време (ако използвате std::set за уникалните суми вместо горе-описаната таблица с 10,000,000, може да не влезете по време). Внимавайте за излишни if-ове в най-вътрешния цикъл.

Input Format

Всеки пример започва с числото N – броя на елементите на множеството. Следват самите стойности на елементите – цели положтелни числа, по-малки от 10000. Входът съдържа множество примери. Този път нямаме броя на примерите на първия ред, нито 0 след края на последния. Индикация за края на примерите е, че не може повече да се чете от конзолата: cin.eof() == true или feof(stdin) != 0. За нещастие този начин не е сигурен, тъй като, ако след последното число има интервал или нов ред (\r и/или \n), то cin.eof() == true или feof(stdin) != 0 няма да сработи. По-добрият вариант е да инициализирате N с невъзможна стойност (примерно 0 - по условие, че $2 \le N \le 1000$). Ако след "cin >> N", N има все още същата невъзможна стойност (0) то входа от конзолата е свършил и няма други числа в него.

Constraints

$2 \le N \le 1000$ $0 < Ai < 10000$ Множеството (:)) тестове не са повече от 12.

Output Format

За всеки пример от входа на нов ред да се отпечата теглото на по-лекото множество.

Sample Input 0

```
10
3 2 3 2 2 77 89 23 90 11
```

Sample Output 0

```
136
```