Budowa algorytmów - programowanie dynamiczne

Wstęp do Informatyki i Programowania

Maciek Gębala

9 stycznia 2025

Notatki

Programowanie dynamiczne

Technika projektowania algorytmów polegająca na rozwiązywaniu podproblemów i zapamiętywaniu ich wyników. W technice tej, podobnie jak w metodzie dziel i zwyciężaj, problem dzielony jest na mniejsze podproblemy (ale nie rozłączne). Wyniki rozwiązywania podproblemów są jednak zapisywane w tabeli, dzięki czemu w przypadku natrafienia na ten sam podproblem nie trzeba go ponownie rozwiązywać.

Notatki

Programowanie dynamiczne

Metoda zstępująca z zapamiętywaniem

Polega na rekurencyjnym wywoływaniu funkcji z zapamiętywaniem wyników. Jeśli rozwiązanie danego problemu jest już w tabeli z wynikami, to należy je po prostu stamtąd odczytać.

Metoda wstępująca

Polega na rozwiązywaniu wszystkich możliwych podproblemów, zaczynając od tych o najmniejszym rozmiarze (w momencie rozwiązywania podproblemu na pewno są już dostępne rozwiązania jego podproblemów). Nie zużywa się pamięci na rekurencyjne wywołania funkcji, ale może się okazać, że część podproblemów została rozwiązana nadmiarowo (nie były one potrzebne do rozwiązania głównego problemu).

Notatki

Własność optymalnej podstruktury

Własność ta oznacza, że optymalne rozwiązanie problemu jest funkcją optymalnych rozwiązań podproblemów (czyli znając optymalne rozwiązania podproblemów można efektywnie wyznaczyć rozwiązanie problemu).

Notatki

Problem plecakowy

Mamy do dyspozycji plecak o maksymalnej nośności $maxW$ oraz zbiór n przedmiotów, każdy o wartości v_i i wadze w_i (wartości całkowite dodatnie).

Szukamy takiego podzbioru przedmiotów $I \subseteq \{1, \ldots, n\}$, że wartość $\sum_{i \in I} v_i$ jest maksymalna i $\sum_{i \in I} w_i \leqslant maxW$.

Notatki

Problem plecakowy

Przegląd zupełny

Przegląd zupełny (bruteforce, metoda siłowa) - przeglądamy wszystkie możliwe podzbiory przedmiotów o wadze nie większej niż $maxW$ i szukamy największej wartości.

Metoda nieefektywna o złożoności $\Theta(2^n)$.

Notatki

Rozwiązanie dynamiczne

Rozwiązanie będziemy budować rozwiązując podproblemy dla coraz większej liczby przedmiotów i wszystkich wag od 0 do $maxW$.

Niech $C(i, w)$ oznacza maksymalną wartość przedmiotów o wadze dokładnie w ze zbioru $\{1, \ldots, i\}$ (jeśli nie istnieje taki zbiór przedmiotów o wadze w, to za $C(i, w)$ przyjmujemy -1.

Dla $i = 0$ przyjmujemy $C(0, 0) = 0$ i $C(0, w) = -1$ dla $w \in \{1, \ldots, maxW\}$.

Notatki

Rozwiązanie dynamiczne

Załóżmy, że mamy policzone wszystkie $C(j, w)$ dla $j \leqslant i$.

Rozpatrujemy teraz $i + 1$-wszy przedmiot i obliczamy $C(i + 1, w)$. Mamy trzy przypadki (własność optymalnej struktury):

1. $w < w_{i+1}$ – przedmiot nie może być w rozwiązaniu o wadze w ponieważ waży więcej, stąd $C(i + 1, w) = C(i, w)$.
2. $w \geqslant w_{i+1}$ i $C(i, w - w_{i+1}) = -1$ – przedmiot mógłby być w rozwiązaniu o wadze w, ale nie ma rozwiązania dla wagi $w - w_{i+1}$, stąd nie możemy wziąć go do rozwiązania o wadze w i ponownie $C(i + 1, w) = C(i, w)$.
3. $w \geqslant w_{i+1}$ i $C(i, w - w_{i+1}) \geqslant 0$ – przedmiot może być w rozwiązaniu i wybieramy czy jego dodanie podnosi wartość rozwiązania, czyli

$$C(i + 1, w) = \max\{C(i, w), C(i, w - w_{i+1}) + v_{i+1}\}$$

Notatki

Rozwiązanie dynamiczne

Optymalne rozwiązanie jest maksymalną wartością $C(n, w)$ dla $w \in \{0, \ldots, maxW\}$.

Odtwarzanie optymalnego zbioru przedmiotów

Aby odtworzyć optymalny zbiór przedmiotów wystarczy cofać się w tabeli i sprawdzać czy w punkcie 3 został dodany nowy przedmiot.

Algorytm działa w czasie $\Theta(n \cdot maxW)$.

Notatki

Implementacja w Adzie

Zakładamy, że funkcja obliczająca rozwiązanie zwróci tablicę wyznaczającą optymalny zbiór przedmiotów.

```
type IntArray is array (Integer range <>) of Integer;
type BoolArray is array (Integer range <>) of Boolean;

function SolveKnapsack (Value, Weight : IntArray;
                        MaxWeight : Integer)
                        return BoolArray is
   N : Integer := Value'Last;
   C : array (0 .. N, 0 .. MaxWeight) of Integer
         := (others => (others => 0));
   S : BoolArray (1 .. N) := (others => False);
   max : Integer;
   l, m : Integer;
begin
   for i in 1 .. MaxWeight loop
      C (0, i) := -1;
   end loop;
```


Notatki

Implementacja w Adzie

```
   for i in 1 .. N loop
      for w in 0 .. MaxWeight loop
         if w < Weight (i)
            or else C (i - 1, w - Weight (i)) < 0
            or else C (i - 1, w) >
                    C (i - 1, w - Weight (i)) + Value (i)
         then
            C (i, w) := C (i - 1, w);
         else
            C (i, w) := C (i - 1, w - Weight (i)) + Value (i);
         end if;
      end loop;
   end loop;
```


Notatki

Implementacja w Adzie

```
   max := 0;
   l := 0;
   for i in 1 .. MaxWeight loop
      if max < C (N, i) then
         max := C (N, i);
         l := i;
      end if;
   end loop;
   for i in reverse 1 .. N loop
      if C (i, l) /= C (i - 1, l) then
         S (i) := True;
         l := l - Weight (i);
      end if;
   end loop;
   return S;
end SolveKnapsack;
```


Notatki

Przykład rozwiązania

```
mgc@diament:~$ ./knapsack2 plecak2-20.txt
 7
    1   10    2
    2    6    3
    3    2    4
    4    3    5
    5    3    1
    6    1    1
Start
    0   -1   10   -1   -1   -1   -1   -1
    0   -1   10    6   -1   16   -1   -1
    0   -1   10    6    2   16   12    8
    0   -1   10    6    2   16   12   13
    0    3   10   13    9   16   19   15
    0    3   10   13   14   16   19   20
Solution
    1    2    5    6
 20
 7
```


Notatki

Problem wydawania reszty

Dany jest (posortowany rosnąco) ciąg nominałów monet $C = (c_1, c_2, \ldots, c_n)$ oraz kwota do wydania r. Należy wyznaczyć takie nieujemne współczynniki $k_1, k_2, \ldots, k_n$, że $\sum_{i=1}^{n} k_i \cdot c_i = r$, a suma $\sum_{i=1}^{n} k_i$ jest jak najmniejsza (lub odpowiedzieć, że rozwiązanie nie istnieje).

Jeśli $c_1 = 1$ to rozwiązanie zawsze istnieje.

Notatki

Rozwiązanie dynamiczne

Niech $C(i)$ oznacza minimalną liczbę monet potrzebnych do wydania reszty i (jeśli rozwiązanie dla i nie istnieje to $C(i) = -1$. Dodatkowo niech $D(i)$ oznacza numer nominału ostatnio dodanej monety (wartość pomocnicza potrzebna do rekonstrukcji rozwiązania).

Na początek mamy $C(0) = 0$.

Teraz

$$C(i) = \min_{j \in \{1,\ldots,n\}} \{C(i - c_j) + 1 \ : \ c_j \geqslant i \wedge C(i - c_j) \neq -1\}$$

lub $C(i) = -1$ jeśli takie minimum nie istnieje, dla kolejnych $i \in \{1, \ldots, n\}$.

Dodatkowo, jeśli $C(i) \neq -1$, to za $D(i)$ podstawiamy numer nominału monety którą użyliśmy.

Notatki

Rozwiązanie dynamiczne

Rekonstrukcja monet składających się na resztę

Wykorzystujemy wartości $D(i)$ do odtwarzania nominałów składających się na resztę.

Algorytm działa w czasie $\Theta(n \cdot r)$.

Notatki

Przykład rozwiązania

```
mgc@diament:~$ more coin2.txt
3
3
5
11
mgc@diament:~$ ./coins coin2.txt 1 2 3 4 5 6 7 8
 1 ==> No solution!
 2 ==> No solution!
 3 ==>  1
 1 x  3
 4 ==> No solution!
 5 ==>  1
 1 x  5
 6 ==>  2
 2 x  3
 7 ==> No solution!
 8 ==>  2
 1 x  3
 1 x  5
```


Notatki

Notatki

Notatki

Notatki