Wycinka drzew (rozwiązanie)

Autor zadania: **Karol Pokorski**
Opracowanie: **Alicja Kluczek, Bartosz Kostka, Jadwiga Pokorska**
Opis rozwiązania: **Lech Duraj**

Olimpiada Informatyczna Juniorów

Rozwiązanie wzorcowe – programowanie dynamiczne

W pierwszej kolejności opiszemy metodą znajdowania **wyłącznie najmniejszego kosztu** wycinki. Później, w sekcji **Odtwarzanie rozwiązania** zostanie pokazane, jak (nieznacznie) zmienić algorytm tak, żeby „przy okazji" obliczał też numery wyciętych drzew.

Rozwiązanie opiera się na jednej z najpopularniejszych technik algorytmicznych – *programowaniu dynamicznym*. Będziemy chcieli stopniowo znajdować rozwiązania dla coraz większych fragmentów danych wejściowych tak, aby w końcu dojść do odpowiedzi dla pełnych danych. Bardziej konkretnie, chcemy dla $k = 1, 2, \ldots, N$ obliczyć następującą wartość:

$koszt[k]$ – *najlepszy możliwy do osiągnięcia wynik, gdyby brać pod uwagę wyłącznie drzewa* $1, 2, \ldots, k$

Dlaczego wybraliśmy akurat taką wielkość? Po pierwsze, rozwiązaniem zadania jest $koszt[N]$, wystarczy więc w obliczeniach dojść do tej wartości. Po drugie, jak się zaraz przekonamy, można wyliczyć $koszt[k]$, jeśli mamy już obliczone wszystkie poprzednie wartości $koszt[1], koszt[2], \ldots, koszt[k-1]$.

Załóżmy więc, że istotnie mamy już poprawnie obliczone wszystkie $koszt[i]$ dla $1 \leq i < k$ i szukamy najlepszego możliwego rozwiązania dla k pierwszych drzew. Są tylko dwie możliwości tego, co możemy zrobić z drzewem numer k:

- Możemy je wyciąć, co będzie nas kosztować C_k zgodnie z założeniami zadania. Jeśli to zrobimy, musimy jeszcze ustalić, jak postąpić z poprzednimi drzewami – ale to już wiemy, bo najlepsze rozwiązanie dla $k-1$ drzew to policzony wcześniej $koszt[k-1]$. W takiej sytuacji całkowity wynik dla k drzew to $C_k + koszt[k-1]$.
- Możemy je pozostawić na miejscu. Wtedy jesteśmy zmuszeni do wycięcia wszystkich poprzednich drzew, które leżą od niego w odległości mniejszej niż D. Innymi słowy, wszystkich drzew o numerach i, dla których zachodzi $X_i > X_k - D$. A zatem robimy w następujący sposób:
  1. Znajdujemy ostatnie drzewo s, dla którego jeszcze zachodzi $X_s \leq X_k - D$.
  2. Ponosimy koszt wycięcia wszystkich drzew od $s+1$ do $k-1$ (czyli $C_{s+1} + C_{s+2} + \ldots + C_{k-1}$).
  3. Dla drzew od 1 do s-tego włącznie używamy znanego już najlepszego rozwiązania, którego wartość obliczyliśmy już wcześniej: $koszt[s]$.

  Zatem wynik dla tego przypadku to $koszt[s] + (C_{s+1} + \ldots + C_{k-1})$. Pozostaje nam jeszcze zrealizować punkty 1. i 2. w dobrej złożoności, czym zajmiemy się poniżej w dalszych sekcjach.

Zauważmy, że dla każdego k wartość $koszt[k]$ jest jedną z dwóch obliczonych przez nas powyżej – skoro naszym celem jest najmniejszy możliwy koszt, wynikiem jest mniejsza z tych dwóch wartości:

$$koszt[k] = \min\{C_k + koszt[k-1], koszt[s] + (C_{s+1} + \ldots + C_{k-1})$$

(gdzie s to numer najdalszego drzewa takiego, że $X_s \leq X_k - D$).

Obliczenia: sumy prefiksowe i wyszukiwanie binarne

Zanim zamienimy powyższy wzór na program, musimy jeszcze rozwiązać dwie kwestie – znalezienie dla każdego k właściwego drzewa s oraz obliczenie sumy $C_{s+1} + \ldots + C_{k-1}$. Drugi z tych problemów jest względnie łatwy: obliczamy tak zwane *sumy prefiksowe*, czyli dla każdego $i = 1, 2, \ldots, N$ zapamiętujemy sumę $P_i = C_1 + C_2 + \ldots + C_i$ (sumę wszystkich wyrazów od początku do i-tego). Przyjmujemy też $P_0 = 0$, a wszystkie te sumy możemy wyliczyć jedną prostą pętlą, jako że $P_i = P_{i-1} + C_i$. Mając już obliczone sumy prefiksowe $P_0, P_1, \ldots, P_N$ zauważamy, że poszukiwana wartość $C_{s+1} + \ldots + C_{k-1} = P_{k-1} - P_s$, a więc każdą potrzebną nam sumę możemy łatwo uzyskać za pomocą jednego odejmowania.

Kod programu w tym momencie możemy zapisać następująco – funkcja `znajdz(X, k, D)` będzie miała za zadanie znalezienie dla danego k odpowiedniego drzewa s.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2019/2020, 2020/2021, 2021/2022".

Główna pętla (bez wyszukiwania i odtwarzania rozwiązania)

```
1
2  int main()
3  {
4      int N, D;
5      cin >> N >> D;                      // Liczby N i D jak w treści zadania.
6      vector<int> X(N+1), C(N+1);         // Ciągi X[i], C[i] jak w treści zadania.
7      vector<long long> koszt(N+1), P(N+1);
8              // koszti i sumy prefiksowe mogą przekroczyć typ int...
9              // ...używamy do nich typu long long.
10     X[0] = 0;                 // X[0] = 0 dla wygody późniejszego wyszukiwania.
11     P[0] = 0;                 // P[0] = 0 zgodnie z ustaleniami.
12     for(int k=1; k<=N; k++)         // Wczytujemy X[1], ..., X[N]
13         cin >> X[k];
14     for(int k=1; k<=N; k++)         // Wczytujemy C[1], ..., C[N]
15     {
16         cin >> C[k];
17         P[k] = P[k-1]+C[k];  // Od razu wyliczamy sumy prefiksowe P[1], ..., P[N].
18     }
19     koszt[0] = 0;             // Zaczynamy pętlę programowania dynamicznego.
20     for(int k=1; k<=N; k++)
21     {
22         int s = znajdz(X, k, D);             // Wyszukujemy drzewo s...
23         koszt[k] = min(koszt[k-1]+C[k],koszt[s]+P[k-1]-P[s]);
24             // Stosujemy wzór z opracowania....
25     }
26     cout << koszt[N] << endl;                // ...i to wszystko.
27 }
```

Pozostaje zaimplementować funkcję `znajdz(X, k, D)`. Zauważmy, że musimy wyszukać w posortowanej tablicy $X[]$ ostatni element, który jest mniejszy lub równy zadanej wartości $(X[k] - D)$. Jest to jeden z typowych problemów, które można rozwiązać za pomocą *wyszukiwania binarnego* – znanej techniki, opisanej np. w kursie podstaw algorytmiki portalu MAIN2: `https://www.main2.edu.pl/main2/courses/show/7/5/`. Odpowiedni element znajdziemy w $O(\log N)$ kroków, a cały algorytm działa zatem w czasie $O(N \log N)$.

Istnieje algorytm szybszy, działający w czasie $O(N)$, korzystający z tzw. metody dwóch wskaźników. Znalezienie go pozostawiamy Czytelnikowi.

Odtwarzanie rozwiązania

Pozostaje jeszcze poprawić nasz algorytm tak, aby oprócz wyniku podawał też, które drzewa są do wycięcia. Osiągniemy to zostawiając w czasie działania algorytmu „podpowiedzi". W kroku k-tym algorytm oblicza

$$koszt[k] = \min\{C_k + koszt[k-1], koszt[s] + P[k-1] - P[s]\}$$

Jeśli z dwóch liczb po prawej mniejsza jest $C_k + koszt[k-1]$, to w optymalnym rozwiązaniu (tym aktualnym, dla k drzew) drzewo numer k jest wycięte, a poprzednie należy ustawić tak, jak w rozwiązaniu poprzednim (dla $k-1$ drzew). Jeśli zaś mniejszą liczbą jest $koszt[s] + P[k-1] - P[s]$, należy wyciąć wszystkie drzewa od $s+1$ do $k-1$, a z drzewami $1, \ldots, s$ postępować tak, jak w rozwiązaniu dla s drzew. Wprowadzimy zatem nową tablicę $podpowiedz[]$ i będziemy zapisywać $podpowiedz[k] = -1$ w pierwszym przypadku (wycinamy k-te drzewo) oraz $podpowiedz[k] = s$ w drugim wypadku (zostawiamy drzewo k, wycinamy od $s+1$ do $k-1$). Po zakończonej pierwszej części algorytmu ustawiamy się (czyli zmienną j) na drzewo N i postępujemy zgodnie z podpowiedziami:

- Jeśli $podpowiedz[j] = -1$, to znaczy że w optymalnym rozwiązaniu trzeba wyciąć drzewo j, a potem odczytać $podpowiedz[j-1]$ (czyli wykonać $j = j-1$).
- Jeśli $podpowiedz[j] = s > 0$, to zostawiamy j, wycinamy drzewa $s+1, \ldots, j-1$, a potem cofamy się do $podpowiedz[s]$ (czyli $j = s$).

Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2019/2020, 2020/2021, 2021/2022".

Cały program z odtwarzaniem

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

// Wyszukiwanie binarne:  znajduje w tablicy X indeks s...
// ...największy taki, że X[s]<=X[k]-D.
int znajdz(vector<int> &X, int k, int D)
{
  int poczatek = 0;
  int koniec = X.size();
  while(poczatek<koniec)
  {
    int srodek = (poczatek+koniec+1)/2;
    if (X[srodek]>X[k]-D)
      koniec = srodek-1;
    else
      poczatek = srodek;
  }
  return poczatek;
}

int main()
{
  // Liczby N i D jak w treści zadania.
  int N, D;
  cin >> N >> D;
  // Ciągi X[i], C[i] jak w treści zadania.
  vector<int> X(N+1), C(N+1);
  // koszti i sumy prefiksowe mogą przekroczyć typ int;
  // Używamy zatem do nich typu long long.
  vector<long long> koszt(N+1), P(N+1);

  vector<int> podpowiedz(N+1);
  X[0] = 0;  // X[0] = 0 dla wygody późniejszego wyszukiwania.
  P[0] = 0;  // P[0] = 0 zgodnie z ustaleniami.
  for(int k=1; k<=N; k++) // Wczytujemy X[1], ..., X[N]
    cin >> X[k];
  for(int k=1; k<=N; k++) {  // Wczytujemy C[1], ..., C[N]
    cin >> C[k];
    // Od razu wyliczamy sumy prefiksowe P[1], ..., P[N].
    P[k] = P[k-1]+C[k];
  }
  koszt[0] = 0;  // Zaczynamy pętlę programowania dynamicznego.
  for(int k=1; k<=N; k++) {
    int s = znajdz(X, k, D);   // Wyszukujemy drzewo s...
    // Porównujemy wielkości z opracowania...
    if (koszt[k-1]+C[k] < koszt[s]+P[k-1]-P[s]) {
      koszt[k] = koszt[k-1]+C[k];
      podpowiedz[k] = -1;  // ...jeśli jesteś przy k-tym drzewie, wytnij je.
    } else {
      koszt[k] = koszt[s]+P[k-1]-P[s]; // ...jeśli jesteśm przy k-tym drzewie...
      podpowiedz[k] = s;               // ...pozostaw je, wytnij kolejne do s-tego.
    }
  }
  cout << koszt[N] << "\n";  // Mamy już wynik...
```


Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2019/2020, 2020/2021, 2021/2022”.

```cpp
  vector<int> wycinka;
  int j = N;  // Odtwarzamy rozwiązanie cofając się od N:
  while (j > 0)
  {
    if (podpowiedz[j]==-1)    // Jeśli podpowiedź to -1...
    {
      wycinka.push_back(j);   // ...j-te drzewo do wycinki
      j--;                    // ...a my cofamy się do drzewa j-1.
    } else
    {
      int s = podpowiedz[j];  // Jeśli podpowiedź to s>0...
      j--;
      while(j>s)              // ...wycinamy od j-1 do s+1...
      {
        wycinka.push_back(j); // ...i cofamy się do s.
        j--;
      }
    }
  }

  cout << wycinka.size() << "\n";
  for(int i=0; i<wycinka.size(); i++)
    cout << wycinka[i] << " ";
  cout << "\n";
}
```

Cały program z odtwarzaniem

```python
# Wyszukiwanie binarne:  znajduje w tablicy X indeks s
# największy taki, że X[s]<=X[k]-D.
def znajdz(X, k, D):
  poczatek = 0
  koniec = len(X)
  while poczatek < koniec:
    srodek = (poczatek + koniec + 1) // 2
    if X[srodek] > X[k] - D:
      koniec = srodek - 1
    else:
      poczatek = srodek
  return poczatek

# Liczby N i D jak w treści zadania.
(N, D) = tuple(map(int, input().split()))

# Ciągi X[i], C[i] jak w treści zadania.
# X[0] = 0 dla wygody późniejszego wyszukiwania.
# A następnie wczytujemy X[1], ..., X[N]
X = [0] + list(map(int, input().split()))
# Wczytujemy C[1], ..., C[N]
C = [0] + list(map(int, input().split()))

# P[0] = 0 zgodnie z ustaleniami.
P = [0 for _ in range(N + 1)]
# Wyliczamy sumy prefiksowe P[1], ..., P[N].
for k in range(1, N + 1):
  P[k] = P[k - 1] + C[k]

koszt = [0 for _ in range(N + 1)]
podpowiedz = [0 for _ in range(N + 1)]
# Zaczynamy pętlę programowania dynamicznego.
```


Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2019/2020, 2020/2021, 2021/2022”.

```
33 for k in range(N + 1):
34   s = znajdz(X, k, D)  # Wyszukujemy drzewo s...
35   # Porównujemy wielkości z opracowania...
36   if koszt[k - 1] + C[k] < koszt[s] + P[k - 1] - P[s]:
37     koszt[k] = koszt[k - 1] + C[k]
38     podpowiedz[k] = -1  # ...jeśli jesteś przy k-tym drzewie, wytnij je.
39   else:
40     # ...jeśli jesteśm przy k-tym drzewie...
41     koszt[k] = koszt[s] + P[k - 1] - P[s]
42     # ...pozostaw je, wytnij kolejne do s-tego.
43     podpowiedz[k] = s
44
45 print(koszt[N])  # Mamy już wynik...
46 wycinka = []
47 j = N            # Odtwarzamy rozwiązanie cofając się od N:
48 while j > 0:
49   if podpowiedz[j] == -1:  # Jeśli podpowiedź to -1...
50     wycinka.append(j)      # ...j-te drzewo do wycinki
51     j -= 1                 # ...a my cofamy się do drzewa j-1.
52   else:
53     s = podpowiedz[j]      # Jeśli podpowiedź to s>0...
54     j -= 1
55     while j > s:           # ...wycinamy od j-1 do s+1...
56       wycinka.append(j)    # ...i cofamy się do s.
57       j -= 1
58
59 print(len(wycinka))
60 for drzewo_do_wyciecia in wycinka:
61   print(drzewo_do_wyciecia, end='␣')
62 print()
```


Olimpiada finansowana jest ze środków Ministerstwa Edukacji i Nauki w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2019/2020, 2020/2021, 2021/2022”.