Techniki konstruowania algorytmów

Metoda dziel i zwyciężaj

Technika „dziel i zwyciężaj”

- Aby rozwiązać problem techniką „dziel i zwyciężaj” musi on wykazywać własność podstruktury – rozwiązanie problemu można wyliczyć na podstawie rozwiązań podproblemów.
- Idea: dla małych danych rozwiązanie wyliczamy ad hoc; duży problem dzielimy na podproblemy, rozwiązujemy je (rekurencyjnie) a na końcu obliczamy wynik korzystając z rozwiązań podproblemów.
- Dziel i zwyciężaj to podejście zstępujące do rozwiązywania zadań.

Technika „dziel i zwyciężaj”

- Znane przykłady stosowania techniki „dziel i zwyciężaj”:
  - sortowanie przez scalanie,
  - sortowanie przez podział (sortowanie szybkie).

Maksymalny zysk na giełdzie

- Problem maksymalnego zysku na giełdzie:
  - dane są ceny jakiegoś towaru w kolejnych dniach;
  - należy wyznaczyć taką parę dni, pomiędzy którymi jest największa dodatnia różnica cen tego towaru.
- Rozwiązanie naiwne: sprawdzamy każdą możliwą parę najpierw kupna a potem sprzedaży – czas $O(n^2)$.
- Rozwiązanie techniką „dziel i zwyciężaj”: rozwiązujemy podproblem dla pierwszej połowy danych, potem dla drugiej połowy danych, następnie obliczamy rozwiązanie leżące na styku obu połówek i wyciągamy maksimum.
- Złożoność czasowa (liczba obliczeń max) – $O(n)$.
- Złożoność pamięciowa (głębokość rekursji) – $O(\log n)$.
- Wynik: maksymalny zysk w danym fragmencie, wartość minimalna i wartość maksymalna.

Maksymalny zysk na giełdzie

```
f(C[0…n-1]) -> (z, i, j) {
  if (n jest małe) {
     sprawdź wszystkie możliwe pary;
     return optymalne rozwiązanie;
  }
  m := ⌊n/2⌋;
  (a, b, c) := f(C[0…m-1]);
  (d, e, f) := f(C[m…n-1]);
  y := max{a, d, f-b};
  return (y, min{b, e}, max{c, f});
}
```

Mnożenie liczb zespolonych

- Problem mnożenia liczb zespolonych:
  - dane są dwie liczby zespolone;
  - należy obliczyć ich iloczyn.
- Rozwiązanie naiwne: dla liczb $x=(a + bi)$ i $y=(c + di)$ wyliczamy $x \cdot y = ((ac-bd) + (bc+ad) \cdot i)$
  - 4 mnożenia i 2 dodawania.
- Rozwiązanie techniką „dziel i zwyciężaj”:
  $m := (a–b) \cdot (c+d)$
  $f := ad$
  $g := bc$
  $x \cdot y = (m–f+g) + (f+g) \cdot i$
  - 3 mnożenia i 5 dodawań.

Metoda rekurencji uniwersalnej

- Uproszczona metoda rekurencji uniwersalnej to sposób rozwiązania równań rekurencyjnych postaci:
  $T(n) = c \quad \text{dla } n = 1$
  $T(n) = a \cdot T(n/b) + c \cdot n \quad \text{dla } n > 1$
  - $T(n) \in \Theta(n) \quad \text{dla } a < b$
  - $T(n) \in \Theta(n \cdot \log n) \quad \text{dla } a = b$
  - $T(n) \in \Theta(n^{\log_b a}) \quad \text{dla } a > b$

Mnożenie długich liczb

- Problem mnożenia długich liczb:
  - dane są dwie n-cyfrowe liczby;
  - należy obliczyć ich iloczyn;
  - Umiemy dodawać i odejmować długie liczby w czasie liniowym.
- Rozwiązanie tradycyjne: mnożenie pisemne metodą 4 Rosjan – czas $O(n^2)$.
- Rozwiązanie techniką „dziel i zwyciężaj” w sposób naiwny:
  $m := \lfloor n/2 \rfloor$;
  $x = (x_{n-1}\, x_{n-2} \ldots x_1\, x_0) = 2^m \cdot (x_{n-1} \ldots x_m) + (x_{m-1} \ldots x_0)$
  $= 2^m \cdot x_1 + x_0$
  $y = (y_{n-1}\, y_{n-2} \ldots y_1\, y_0) = 2^m \cdot (y_{n-1} \ldots y_m) + (y_{m-1} \ldots y_0)$
  $= 2^m \cdot y_1 + y_0$
  $x \cdot y = (2^m \cdot x_1 + x_0) \cdot (2^m \cdot y_1 + y_0) =$
  $4^m \cdot x_1 \cdot y_1 + 2^m \cdot (x_0 \cdot y_1 + x_1 \cdot y_0) + x_0 \cdot y_0$
  - 4 mnożenia i 3 dodawania : czas $O(n^2)$

Mnożenie długich liczb

- Rozwiązanie techniką „dziel i zwyciężaj” algorytmem Karacuby:

  $m := \lceil n/2 \rceil$;

  $x = (x_{n-1}\, x_{n-2} \ldots x_1\, x_0) = 2^m \cdot (x_{n-1} \ldots x_m) + (x_{m-1} \ldots x_0)$

  $= 2^m \cdot x_1 + x_0$

  $y = (y_{n-1}\, y_{n-2} \ldots y_1\, y_0) = 2^m \cdot (y_{n-1} \ldots y_m) + (y_{m-1} \ldots y_0)$

  $= 2^m \cdot y_1 + y_0$

  $a := x_1 \cdot y_1$

  $c := x_0 \cdot y_0$

  $b := (x_1 + x_0) \cdot (y_1 + y_0) - a - b$

  $x \cdot y = 4^m \cdot a + 2^m \cdot b + c$

  - 3 mnożenia i 6 dodawań: czas $O(n^{\log 3})$

Techniki konstruowania algorytmów

Metoda redukcji

Technika redukcji

- Technika redukcji jest szczególnym przypadkiem techniki „dziel i zwyciężaj”.
- Aby rozwiązać problem techniką redukcji musi on wykazywać własność pojedynczej podstruktury – rozwiązanie problemu można wyliczyć na podstawie rozwiązania jednego podproblemu.
- Idea: dla małych danych rozwiązanie wyliczamy ad hoc; duży problem redukujemy do mniejszego podproblemu, rozwiązujemy go (rekurencyjnie) a na końcu obliczamy wynik korzystając z rozwiązania podproblemu.

Technika redukcji

- Znane przykłady stosowania techniki redukcji:
  - wyszukiwanie binarne,
  - szybkie potęgowanie,
  - obliczanie silni.

Techniki konstruowania algorytmów

Programowanie dynamiczne

Programowanie dynamiczne

- Programowanie dynamiczne stosuje się głównie do problemów optymalizacyjnych.
- Aby rozwiązać problem techniką programowania dynamicznego musi on wykazywać własności:
  - podstruktury – rozwiązanie problemu można wyliczyć na podstawie rozwiązań podproblemów;
  - wspólnych podproblemów – rozwiązując rekurencyjnie takie zadanie wielokrotnie obliczalibyśmy wyniki dla tych samych danych.
- Idea: dla małych danych rozwiązanie wyliczamy ad hoc; duży problem dzielimy na podproblemy, rozwiązujemy je wstępująco zaczynając od najmniejszych podzadań i kończąc na zadaniu głównym (w bieżących obliczeniach korzystamy z wcześniej obliczonych wyników).
- Programowanie dynamiczne to podejście wstępujące rozwiązywania zadań.

Odtwarzanie optymalnego rozwiązania

- Aby odtworzyć optymalne rozwiązanie potrzebujemy dodatkowo pamiętać skąd się wziął optymalny wynik dla określonego podproblemu – też go zapamiętujemy w tablicy pomocniczej.
- Optymalne rozwiązanie odtwarzamy od końca (optymalnych rozwiązań może być wiele)

Stokrotki

- Problem przejścia przez pole ze stokrotkami:
  - dane jest prostokątne pole podzielone na mniejsze prostokąty zwane kwartałami w liczbie n x m (n wierszy, m kolumn); w każdym kwartale rośnie określona liczba stokrotek;
  - należy przejść z pola w lewym górnym rogu do pola w prawym dolnym rogu tak, aby zebrać po drodze jak największą liczbę stokrotek;
  - można się poruszać tylko w prawo i w dół.

Stokrotki

- Rozwiązanie rekurencyjne:
  suma(i, j) =
  kwartał(i, j) + max{suma(i-1, j), suma(i, j-1)}
  - Złożoność: pamięć O(n+m), czas wykładniczy.
- Rozwiązanie dynamiczne: wypełniamy całą tabelę wartościami zysków wg wzoru rekurencyjnego.
  - Złożoność: pamięć O(n·m), czas O(n·m).
- Tabelę można wypełnić wierszami lub kolumnami, pamiętając tylko dwa wiersze lub dwie kolumny.
  - Złożoność: pamięć O(min{n, m}), czas O(n·m).
- Aby odtworzyć optymalną drogę, należy zapamiętywać skąd przyszliśmy.
  - Złożoność: pamięć O(n·m), czas O(n·m).

Stokrotki

Kwartały ze stokrotkami

2	0	3	3
4	7	1	4
6	4	3	5
1	5	4	2
2	2	6	5

Tablica z rozwiązaniem

2	2	5	8
6	13	14	18
12	17	20	25
13	22	26	28
15	24	32	37

Najdłuższy wspólny podciąg

- Problem najdłuższego wspólnego podciągu (ang. Longest Common Subsequence, LCS):
  - dane są dwa ciągi znaków $X = (x_1,...,x_m)$ i $Y = (y_1,..., y_n)$;
  - należy znaleźć długość najdłuższego z możliwych ciągów Z takiego, że Z jest podciągiem X i Z jest podciągiem Y.
- Zastosowanie:
  - wykrywanie zmian w dokumentach;
  - weryfikacja antyplagiatowa.

Najdłuższy wspólny podciąg

- Oznaczenie. Niech $Z = (z_1,...,z_n)$ będzie n-znakowym ciągiem. Wówczas przez Z_i będziemy oznaczali i-znakowy prefiks ciągu Z dla $i \in \{0,...,n\}$ (Z_0 będzie ciągiem pustym).
- Lemat. Niech $Z = (z_1,...,z_k)$ będzie najdłuższym wspólnym podciągiem ciągu $X = (x_1,...,x_m)$ i $Y = (y_1,..., y_n)$. Wówczas:
  - (1) jeżeli $x_m = y_n$, to $z_k = x_m = y_n$ oraz $Z_{k-1} = lcs(X_{m-1}, Y_{n-1})$;
  - (2) jeżeli $x_m \neq y_n$ i $z_k \neq x_m$, to $Z = lcs(X_{m-1}, Y)$;
  - (3) jeżeli $x_m \neq y_n$ i $z_k \neq y_n$, to $Z = lcs(X, Y_{n-1})$.
- Lemat pozwala na napisanie prostej funkcji rekurencyjnej rozwiązującej ten problem.

Najdłuższy wspólny podciąg

- Niech c(i, j) oznacza długość najdłuższego wspólnego podciągu dla X_i, Y_j. Zależności rekurencyjne:
  $c(i, j) = 0$ gdy $i = 0$ lub $j = 0$
  $c(i, j) = c(i-1, j-1) + 1$ gdy $i, j > 0$ oraz $x_i = y_j$
  $c(i, j) = \max\{c(i, j-1), c(i-1, j)\}$ gdy $i, j > 0$ oraz $x_i \neq y_j$
- Rozwiązanie dynamiczne: wypełniamy całą tabelę długościami wg wzoru rekurencyjnego.
  - Złożoność: pamięć $O(n \cdot m)$, czas $O(n \cdot m)$.
- Tabelę można wypełnić wierszami lub kolumnami, pamiętając tylko dwa wiersze lub dwie kolumny.
  - Złożoność: pamięć $O(\min\{n, m\})$, czas $O(n \cdot m)$.
- Aby odtworzyć najdłuższy wspólny podciąg, należy zapamiętywać wg której reguły wypełniliśmy dane pole.
  - Złożoność: pamięć $O(n \cdot m)$, czas $O(n \cdot m)$.

Najdłuższy podciąg rosnący

- Problem najdłuższego podciągu rosnącego (ang. Longest Increasing Subsequence, LIS):
  - dany jest ciąg liczb $X = (x_1,...,x_n)$;
  - należy znaleźć długość najdłuższego z możliwych ciągów Z takiego, że Z jest podciągiem X i Z jest rosnący.
- Zastosowanie:
  - w matematyce i fizyce.

Najdłuższy podciąg rosnący

- Oznaczenie. Niech $Z = (z_1,...,z_n)$ będzie n-liczbowym ciągiem. Wówczas przez Z_i będziemy oznaczali i-liczbowy prefiks ciągu Z dla $i \in \{0,...,n\}$ (Z_0 będzie ciągiem pustym).
- Lemat. Niech $Z = (z_1,...,z_k)$ będzie najdłuższym podciągiem rosnącym ciągu $X = (x_1,...,x_m)$. Wówczas:
  - (1) jeżeli $x_m = z_k$, to $Z_{k-1} = lis(X_{m-1}) + 1$;
  - (2) jeżeli $x_m \neq z_k$, to $Z = lis(X_{m-1})$.
- Lemat pozwala na napisanie funkcji rekurencyjnej rozwiązującej ten problem.

Najdłuższy podciąg rosnący

- Niech c(i) oznacza długość najdłuższego podciągu rosnącego dla X_i, zawierającego liczbę x_i. Zależności rekurencyjne:
  $c(i) = 1 \quad \text{gdy } x_i = \min_{j=1\ldots i}\{x_j\}$
  $c(i) = \max_{j=1\ldots i-1}\{c_j : x_j < x_i\} + 1 \quad \text{gdy } x_i > \min_{j=1\ldots i}\{x_j\}$
- Rozwiązanie dynamiczne: wypełniamy całą tabelę długościami wg wzoru rekurencyjnego.
  - Złożoność: pamięć O(n), czas $O(n^2)$.
- Rozwiązanie dynamiczne ulepszone o wyszukiwanie binarne.
  - Złożoność: pamięć O(n), czas $O(n \cdot \log n)$.
- Aby odtworzyć najdłuższy podciąg rosnący, należy zapamiętywać przedostatni element którym przedłużamy podciąg rosnący.
  - Złożoność: pamięć O(n), czas $O(n \cdot \log n)$.

Techniki konstruowania algorytmów

Rekurencja ze spamiętywaniem

Rekurencja ze spamiętywaniem

- Rekurencja ze spamiętywaniem jest szczególnym przypadkiem programowania dynamicznego.
- Aby rozwiązać problem za pomocą rekurencji ze spamiętywaniem musi on się dać rozwiązać dynamicznie, czyli wykazywać własności:
  - optymalnej podstruktury;
  - wspólnych podproblemów.
- Idea: dla małych danych rozwiązanie wyliczamy ad hoc; duży problem dzielimy na podproblemy i wyliczamy ich rozwiązania; jeśli dany podproblem był już wcześniej obliczony to odczytujemy gotowy wynik z tablicy pomocniczej, w przeciwnym przypadku rozwiązujemy go rekurencyjnie; wyliczony wynik dla problemu zapamiętujemy w tablicy pomocniczej, aby w przyszłości nie liczyć go ponownie, tylko szybko odczytać (rozwiązany problem może być przecież częścią większego problemu).

Rekurencja ze spamiętywaniem

- Znane przykłady stosowania rekurencji ze spamiętywaniem:
  - współczynnik dwumianowy,
  - ciąg Fibonacciego.

Współczynniki dwumianowe

- Problem wyznaczenia k-tego wyrazu w n-tym wierszu trójkąta Pascala:
  - dane są liczby naturalne n i k, takie że $0 \leq k \leq n$;
  - należy obliczyć współczynnik dwumianowy $\binom{n}{k}$.
- Rozwiązanie tradycyjne: budujemy trójkątną tablicę i wypełniamy ją wierszami od góry:
  A[0][k] = A[k][0] = 1,
  A[k][j] = A[k-1][j-1] + A[k-1][j] dla k=1…n-1.
  Na końcu odczytujemy A[n][k].
  - Złożoność: pamięć $O(n^2)$, czas $O(n^2)$.

Współczynniki dwumianowe

- Rozwiązanie rekurencyjne: wypełniam po kolei tablicę współczynnikami dwumianowymi;
- Rekurencja:
  Pascal(n, 0) = 1
  Pascal(n, n) = 1
  Pascal(n, k) = Pascal(n-1, k-1) + Pascal(n-1, k)
  - Złożoność: pamięć $O(n^2)$, czas wykładniczy.
- Rekurencja ze spamiętywaniem wyników.
  - Złożoność: pamięć $O(n^2)$, czas $O(n^2)$.
- Rozwiązanie dynamiczne: przechowuję tylko ostatnio wyliczony wiersz trójkąta Pacsala; na jego podstawie obliczam następny.
  - Złożoność: pamięć $O(n)$, czas $O(n^2)$.

Współczynniki dwumianowe

- Algorytm ze spamiętywaniem:
  - Tablica globalna
    int p[n+1][n+1] = {0,…};
  - Funkcja rekurencyjna

```
function Pascal(n, k) {
    if (p[n][k] = 0) return p[n][k];
    if (k = 0 or  k = n) p[n][k] = 1;
    else p[n][k] = Pascal(n-1, k-1) + Pascal(n-1, k);
    return p[n][k];
}
```

- Liczba obliczeń rekurencyjnych dla każdego elementu tablicy wynosi 1.

Techniki konstruowania algorytmów

Strategia zachłanna

Strategia zachłanna

- Strategię zachłanną stosuje się głównie do problemów optymalizacyjnych.
- Aby rozwiązać problem techniką zachłanną musi on wykazywać własności:
  - podstruktury – rozwiązanie problemu można wyliczyć na podstawie rozwiązań podproblemów albo rozwiązanie problemu można skonstruować dodając albo nie dodając jakiegoś elementu do rozwiązania;
  - wyboru zachłannego – w każdym kroku algorytmu jesteśmy w stanie wybrać element należący do rozwiązania (albo wyeliminować element z rozwiązania).
- Idea: dla małych danych rozwiązanie wyliczamy ad hoc; dla dużego problemu wybieramy element należący do rozwiązania (albo stwierdzamy, że nie będzie on należał do rozwiązania optymalnego), redukując tym samym rozmiar zadania.
- Ważnym elementem strategii zachłannej jest uzasadnienie poprawności rozwiązania.

Przydział zajęć do sali

- Problem przydziału zajęć do sali:
  - dane są godziny, w których mogą się odbywać wykłady w pewnej sali;
  - należy wyznaczyć taki zbiór wykładów, aby odbyło się ich jak najwięcej.
- Rozwiązanie: sortujemy wykłady po ich godzinach zakończenia; wybieramy wykład, który kończy się najszybciej i nie zachodzi na poprzednie wykłady (aż do wyczerpania danych).
  - Czas (zdeterminowany procedurą sortowania): $O(n \cdot \log(n))$
- Poprawność rozwiązania: dowód nie wprost.

Szeregowanie zadań do jednego procesora

- Problem szeregowania zadań do procesora:
  - danych jest n zadań t_1,… t_n i procesor, na którym te zadania będą wykonane; każde z zadań ma określony czas realizacji d_1,… d_n;
  - należy tak poprzydzielać zadania do procesora, aby zminimalizować sumaryczny czas przebywania zadań w systemie.
- Rozwiązanie: sortujemy zadania po ich czasach wykonania; przydzielamy zadania do procesora w kolejności od najkrótszego do najdłuższego.
  - Czas (zdeterminowany procedurą sortowania): $O(n \cdot \log(n))$
- Poprawność rozwiązania: dowód nie wprost.

Szeregowanie zadań do wielu procesorów

- Problem szeregowania zadań do procesorów:
  - danych jest n zadań t_1, …t_n i m procesorów p_1, …p_m; każde z zadań ma określony czas realizacji d_1, …d_n;
  - należy tak poprzydzielać zadania do procesorów, aby zminimalizować sumaryczny czas przebywania zadań w systemie.
- Rozwiązanie: sortujemy zadania po ich czasach wykonania; sekwencyjnie przydzielamy zadania do kolejnych procesorów (do pierwszego procesora p_1 trafią zadania $t_{i[1]}$, $t_{i[m+1]}$, $t_{i[2m+1]}$, …).
  - czas: $O(n \cdot \log(n))$
- Poprawność rozwiązania: dowód nie wprost.
- Uwaga: zadania muszą być wykonane w kolejności od najkrótszego do najdłuższego na każdym procesorze.

Przykłady problemów rozwiązywanych techniką zachłanną

- Algorytm Huffmana wyznaczający kodowanie dla symboli, których prawdopodobieństwa wystąpienia w tekście są znane.
- Algorytm Kruskala wyznaczający minimalne drzewo rozpinające dla grafu nieskierowanego ważonego, o ile jest on spójny.
- Algorytm Dijkstry wyznaczający najkrótsze ścieżki z pojedynczego źródła w grafie o nieujemnych wagach krawędzi.