Metody algorytmiczne

Wędruj i sprawdzaj

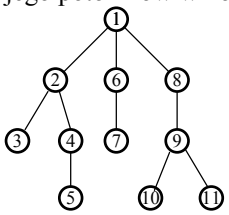
Prosty przegląd struktury danych np. w celu znalezienia największego elementu ze zbioru danych przechowywanych w tej strukturze:

iteracja	→	np. wektor, lista
iteracje zagnieżdżone	→	np. tablice wielowymiarowe, listy list itp.
rekurencja	→	np. drzewa

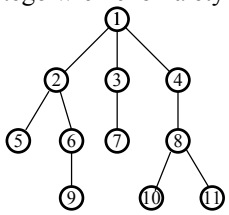
Przegląd struktury = budowanie ciągu zawierającego wszystkie obiekty w strukturze
Drzewa można przeglądać iteracyjnie:

Algorytm przeglądu drzewa w głąb (budowanie ciągu zawierającego wszystkie wierzchołki drzewa):
1. wstaw korzeń jako pierwszy element ciągu,
2. powtarzaj co następuje, aż do nadania korzeniowi etykiety „zamknięty”:
   2.1. wybierz z aktualnego ciągu ostatni wierzchołek, który nie ma etykiety „zamknięty”,
   2.2. jeśli wybrany wierzchołek nie ma potomstwa, które jeszcze nie zostało dopisane do ciągu, to nadaj mu etykietę „zamknięty”, w przeciwnym przypadku dopisz do ciągu pierwszego (licząc od lewej) jego potomka, który jeszcze nie występuje w ciągu.

Algorytm przeglądu drzewa wszerz (budowanie ciągu zawierającego wszystkie wierzchołki drzewa):
1. nadaj etykietę „nowy” wszystkim wierzchołkom drzewa,
2. wstaw korzeń jako pierwszy element ciągu,
3. dopóki w tworzonym ciągu występuje wierzchołek z etykietą „nowy”, powtarzaj co następuje:
   3.1. wybierz z aktualnego ciągu pierwszy wierzchołek, który ma etykietę „nowy”, dodaj do ciągu wszystkich jego potomków w kolejności od lewej i usuń dla tego wierzchołka etykietę „nowy”.

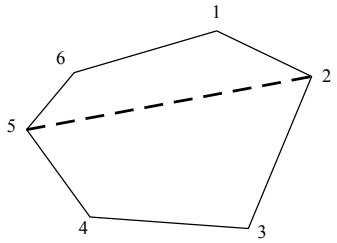


Przegląd drzewa w głąb



Przegląd drzewa wszerz

Przykład metody ”wędruj i sprawdzaj” - znajdowanie największej przekątnej w wielokącie wypukłym



N - liczba wierzchołków
Struktura danych dla opisu wielokąta - tablica dwuwymiarowa:

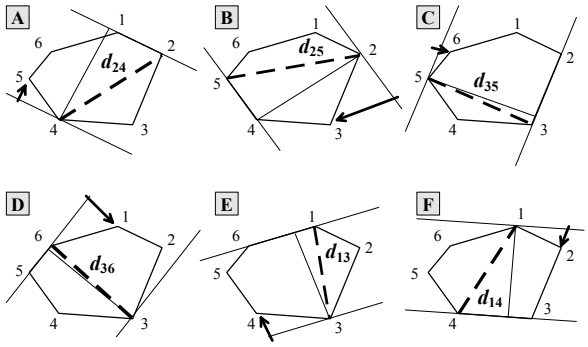
	1	2	3	...	N
X	x_1	x_2	x_3	...	x_N
Y	y_1	y_2	y_3	...	y_N

Co przeglądamy? Np. tablicę długości wszystkich odcinków pomiędzy wierzchołkami.

	1	2	3	...	N
1		d_{12}	d_{13}	...	d_{1N}
2	d_{21}		d_{23}		d_{2N}
3	d_{31}	d_{32}			d_{3N}
⋮	⋮	⋮	⋮		⋮
N	d_{N1}	d_{N2}	d_{N3}	...	

Przejrzenie górnej trójkątnej połowy tablicy (liczba elementów $N(N-1)/2$) pozwala znaleźć element o największej wartości.

Ale można inaczej:



Czyli dla 6 wierzchołków wystarczy 6 zamiast 15 kroków (15 = 6(6–1)/2)

Dziel i zwyciężaj

Jeśli nie możesz uporać się z rozwiązaniem zadania w całości, to spróbuj dzielić je na mniejsze o takiej samej strukturze i stosuj rekurencyjnie algorytm rozwiązywania. Uzyskane rozwiązania małych zadań łącz w rozwiązania tych zadań, które były wcześniej dzielone.

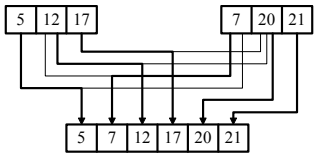
Przykład metody "dziel i zwyciężaj" - sortowanie przez scalanie

Dane: nieuporządkowana *N* elementowa lista

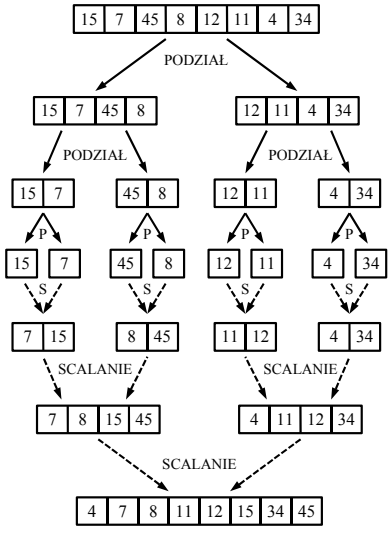
procedura ***sortuj-listę*** *L;*

1. jeśli *L* zawiera tylko jeden element, to jest posortowana;
2. w przeciwnym razie wykonaj co następuje:
   - 2.1. podziel listę *L* na dwie połowy L_1 i L_2;
   - 2.2. wywołaj ***sortuj-listę*** L_1;
   - 2.3. wywołaj ***sortuj-listę*** L_2;
   - 2.4. scal posortowane listy L_1 i L_2 w jedną posortowaną listę;
3. wróć do poziomu wywołania.

Przykładowy schemat scalania:



Przykładowy schemat metody:

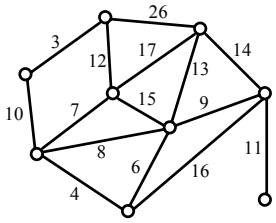


Metoda zachłanna

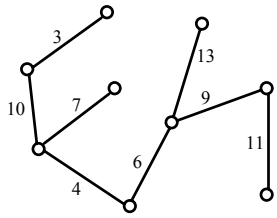
Istnieją zadania, których rozwiązanie może być budowane z elementów dobieranych kolejno według zasady „idź naprzód, łap najlepsze z tego co pod ręką i nigdy potem nie oddawaj tego co już masz".

Przykład metody „zachłannej" - wyznaczanie minimalnego drzewa rozpinającego w grafie

Problem polega na znalezieniu najtańszej sieci połączeń wiążącej wszystkie podane punkty.

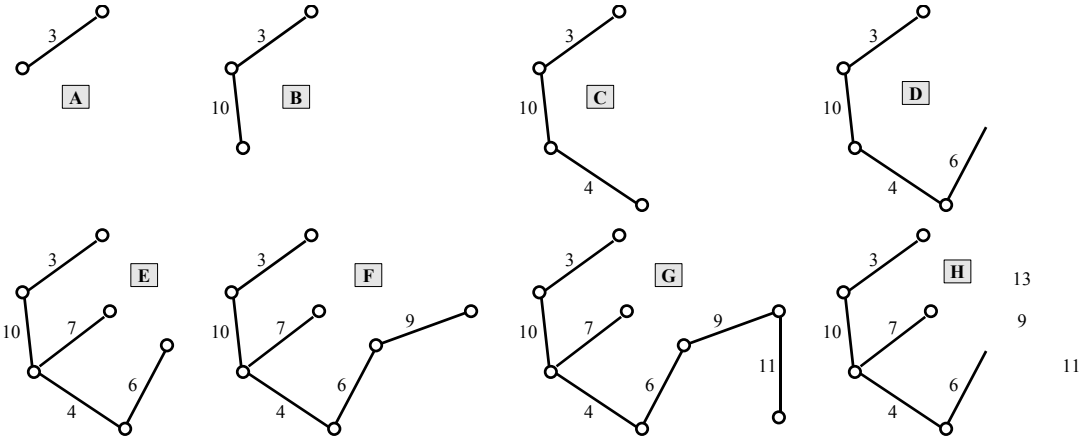


Spójny graf z wagami krawędzi



Minimalne drzewo rozpinające
(koszt: 63)

Przykładowa realizacja metody zachłannej:



Algorytm:

1. wybierz najkrótszy odcinek drogi
2. powtarzaj co następuje, aż do połączenia wszystkich punktów:
   2.1. wybierz najkrótszy odcinek spośród tych odcinków, które łączą jedno z już połączonych miast z jakimkolwiek miastem jeszcze nie połączonym

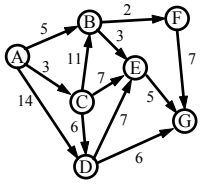
Programowanie dynamiczne

Zasada (optymalności Bellmana):

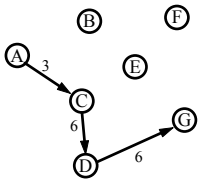
jeżeli znamy najlepszą drogę przejścia od punktu początkowego do punktu końcowego prowadzącą przez kolejne punkty, to każdy fragment tej drogi pomiędzy dowolnym punktem pośrednim a punktem końcowym jest najlepszą drogą przejścia od tego punktu do punktu końcowego.

Przykład metody "programowania dynamicznego" - znajdowanie najkrótszej ścieżki w grafie

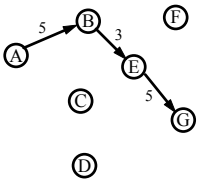
Problem polega na znalezieniu najkrótszej ścieżki, która wykorzystuje dostępne połączenia i łączy dwa wskazane punkty w podanej sieci połączeń jednokierunkowych.



Skierowany graf acykliczny
(spójny)



Ścieżka wyznaczona metodą
zachłanną (koszt: 15)



Ścieżka najkrótsza
(koszt: 13)

Realizacja metody:

L(X) – oznacza długość najkrótszej drogi z punktu X do punktu G ; chcemy wyznaczyć L(A).

Algorytm:
1 faza
Dla każdego punktu w sieci połączeń wyznacz wartość L(X), zaczynając od punktu docelowego i cofając się w każdym kroku o jedno połączenie dalej od punktu docelowego (zapamiętuj przy tym dla każdego punktu, jakie połączenie należy wybrać, aby przejść najkrótszą drogę od niego do punktu docelowego);
2 faza
Zacznij budować ścieżkę od punktu początkowego posługując się kolejno wskazaniami jakie połączenia tworzą najkrótszą ścieżkę od przechodzonych punktów do punktu docelowego.

Działanie algorytmu dla przykładowego zadania:

- L(G) = 0
- L(F) = 7,
- L(E) = 5,
- L(D) = min (**6**, 7 + L(E)) = min (**6**, 12) = **6**,
- L(B) = min (**3 + L(E)**, 2 + L(F)) = min (**8**, 9) = **8**,
- L(C) = min (11 + L(B), **6 + L(D)**, 7 + L(E)) = min (19, **12**, 12) = **12**,
- L(A) = min (14 + L(D), 3 + L(C), **5 + L(B)**) = min (20, 15, **13**) = **13**.