PROGRAMOWANIE DYNAMICZNE

Metody algorytmiczne c.d.

Dwuetapowa metoda polegająca najpierw na stopniowym przyrostowym gromadzeniu dodatkowej wiedzy o cząstkowych rozwiązaniach zadania, a potem na wykorzystaniu tej wiedzy do wybrania najlepszego rozwiązania.

Przykładem zastosowania metody jest **algorytm wyznaczania najkrótszej drogi w grafie skierowanym**

Dane wejściowe:
skończona liczba punktów (*N*), odcinki skierowane łączące wybrane pary punktów (*M*), liczby przypisane podanym odcinkom (ich długość, koszt, waga itp.) oraz wskazany punkt początkowy i końcowy drogi.

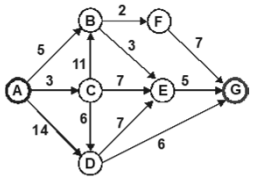
Wynik: taki zbiór odcinków, który tworzy najkrótszą drogę przejścia od punktu początkowego do punktu końcowego.

Algorytm programowania dynamicznego do wyznaczania najkrótszej drogi przejścia z punktu początkowego do końcowego

Postępowanie w tym algorytmie ma związek z tzw. *zasadą 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 pośredniego do punktu końcowego.

Przykład dla N = 7 i M = 12:



L(X) – długość najkrótszej drogi przejścia od punktu X do G

***L*(A) = ?**

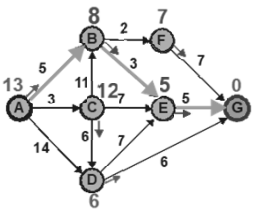
Algorytm

Faza 1:

1. **Przypisz punktowi docelowemu G wartość *L*(G) = 0,**
2. **Powtarzaj co następuje, aż do wyznaczenia *L*(A):**
   - **2.1. Wybierz taki punkt Y, dla którego wszystkie odcinki z niego wychodzące prowadzą do punktów o już wyznaczonych wartościach *L*(X),**
   - **2.2. Wyznacz dla wybranego punktu wartość *L*(Y) wybierając najmniejszą sumę długości odcinka z niego wychodzącego i wartości *L*(X) dla punktu, do którego ten odcinek prowadzi**
   - **2.3. Zapamiętaj oprócz wartości *L*(Y) dla jakiego punktu ta suma była najmniejsza**

Faza 2:

1. **Wyznacz najkrótszą drogę przejścia zaczynając od punktu A i odczytując kolejno, które kolejne punkty związane były z najmniejszymi sumami wybieranymi w 1. fazie**



Faza 1:

***L*(G) = 0**

***L*(F) = 7**

***L*(E) = 5**

***L*(D) = min(7+*L*(E), 6+*L*(G)) = min(7+5, 6+0) = 6**

***L*(B) = min(3+*L*(E), 2+*L*(F)) = min(3+5, 2+7) = 8**

***L*(C) = min(11+*L*(B), 7+*L*(E), 6+*L*(D)) = min(11+8, 7+5, 6+6) = 12**

***L*(A) = min(5+*L*(B), 3+*L*(C), 14+*L*(D)) = min(5+8, 3+12, 14+6) = 13**

Faza 2:

A
⇓
B
⇓
E
⇓
G

Elementy zaznaczone na czerwono są zapamiętywane w trakcie 1. fazy

POPRAWNOŚĆ PROGRAMÓW

Na poprawność programu komputerowego składa się:

- poprawność algorytmu,
- poprawność zapisu w języku programowania.

Jeśli uważasz, że jakiś program komputerowy jest bezbłędny, to się mylisz – po prostu nie zauważyłeś jeszcze skutków błędu, który jest w nim zawarty. ☺ 😐 ☹

Na poprawność algorytmu składa się:

- poprawność logiczna (założenia, rozumowanie, metoda, ...)
- poprawność algorytmiczna (struktura, sterowanie procesem, ...)

Gdzie można popełnić błąd?

1. **Błędy logiczne**

   Mogą pochodzić z przyjęcia fałszywych założeń, wad rozumowania lub źle dobranej metody wyznaczania wyniku.

   Np. w algorytmie zliczania zdań zawierających słowo „algorytm" nie zauważyliśmy, że sekwencja dwóch znaków „**.** " może występować także wewnątrz zdania:
   „Na Rys. 2 pokazano schemat...",
   a używaliśmy jej do wyszukiwania jego końca.

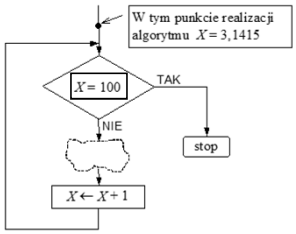
Możliwe skutki i znaczenie:

- algorytm przestaje być poprawnym rozwiązaniem zadania algorytmicznego,
- dla pewnych zestawów danych wejściowych algorytm podaje wyniki niezgodne z oczekiwanymi,
- procesor może nie być w stanie wykonać pewnych instrukcji (np. żądamy dzielenia przez 0),
- błędy bardzo groźne – mogą być trudne do znalezienia i pozostawać długo w ukryciu, nawet w trakcie wielokrotnego używania programu w postaci kodu.

2. **Błędy algorytmiczne**

   Wynikają z wadliwie skonstruowanych struktur sterujących np. niewłaściwych zakresów iteracji, niewłaściwych warunków użytych do zatrzymywania iteracji warunkowych lub przeniesienia sterowania w niewłaściwe miejsce procesu w wyniku zastosowania wyboru warunkowego (lub instrukcji skoku).

   Np. pętla nieskończona:



Możliwe skutki i znaczenie:

- algorytm dla pewnych dopuszczalnych danych wejściowych daje niepoprawny wynik,
- wykonanie programu realizującego algorytm jest przerywane w trybie awaryjnym,
- program realizujący algorytm nie kończy w normalnym trybie swego działania,
- błędy groźne, ale możliwe do uniknięcia poprzez staranne opisanie struktury algorytmu przed jego zapisaniem w jakimkolwiek języku oraz zachowanie zasad „dobrej praktyki" programowania.

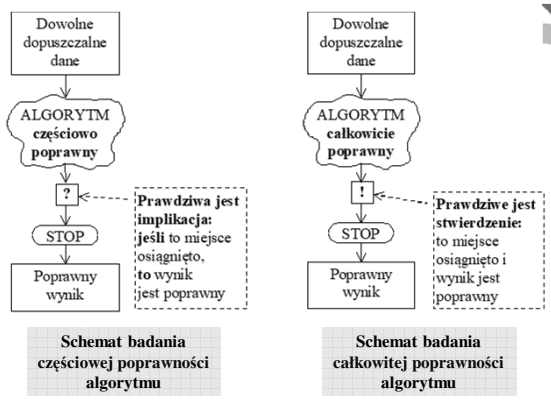
Algorytmem **całkowicie poprawnym** nazywamy algorytm, dla którego **udowodniono**, że nie zawiera on ani błędów logicznych, ani algorytmicznych.

Twierdzenie, które należy udowodnić brzmi:

„Dany algorytm dla każdego zestawu dopuszczalnych danych wejściowych samoistnie przestaje działać po wykonaniu skończonej liczby operacji elementarnych, dając poprawny (spełniający założone warunki) wynik końcowy"

Etapem pośrednim dla wykazania całkowitej poprawności może być **udowodnienie częściowej poprawności** algorytmu.

„Dla każdego zestawu dopuszczalnych danych wejściowych z faktu, że dany algorytm samoistnie przestał działać po wykonaniu skończonej liczby operacji elementarnych wynika, że dał poprawny (spełniający założone warunki) wynik końcowy"



Schemat badania częściowej poprawności algorytmu

Schemat badania całkowitej poprawności algorytmu

Złożony program komputerowy może zawierać sporą liczbę błędów różnych typów, popełnionych na różnych etapach jego powstawania.

Dla złożonego algorytmu badanie jego całkowitej poprawności może być trudne i czasochłonne

W praktyce opracowywania programów komputerowych pomija się (niestety) etap badania całkowitej poprawności algorytmu i bada się „produkt końcowy", czyli sam program:

- testowanie na licznych zestawach danych wejściowych (dla takiego zestawu powinien być znany wynik końcowy)
- uruchamianie w obecności narzędzi diagnostycznych (badanie stanów pośrednich i końcowych)

Podstawową metodą badania **częściowej poprawności** algorytmu jest **metoda niezmienników**, polegająca na:

- wybraniu w schemacie algorytmu **punktów kontrolnych**,
- związaniu z każdym punktem kontrolnym tzw. **asercji**, czyli warunku logicznego zależnego od stanu realizacji procesu wyznaczania założonego wyniku końcowego,
- ustaleniu w obrębie każdej z iteracji takiej asercji, której prawdziwość będzie można wykazać po dowolnej liczbie powtórzeń tej iteracji (tzw. **niezmiennik** iteracji),
- wykazaniu, że z prawdziwość jednej asercji wynika prawdziwość następnej, że niezmienniki pozostają prawdziwe po kolejnych iteracjach i pociągają za sobą prawdziwość ostatniej asercji, która oznacza osiągnięcie założonego wyniku.

Po wykazaniu częściowej poprawności algorytmu podstawową metodą badania jego **całkowitej poprawności** jest **metoda zbieżników**, polegająca na:

- ustaleniu dla każdej iteracji **zbieżnika**, czyli takiej zmiennej, której wartości zależą od stanu realizacji algorytmu po wykonaniu kolejnych powtórzeń tej iteracji i tworzą ograniczony ciąg monotoniczny,
- wykazaniu, że każdy ze zbieżników nie może zmieniać się nieskończoną liczbę razy i algorytm po wykonaniu skończonej liczby powtórzeń w każdej z iteracji zatrzyma się w ostatnim punkcie kontrolnym.