Informatyka 1

Wykład VI

Metody algorytmiczne, procedury i funkcje cd., iteracje i rekurencja

Robert Muszyński
ZPCiR IIAiR PWr

Zagadnienia: konstruowanie algorytmów, metoda „dziel i zwyciężaj”, metoda przyrostowa, programowanie dynamiczne, systematyka metod konstruowania algorytmów, specyfikacja procedur i funkcji, warunki PRE i POST, asercje, struktury sterujące, iteracje a rekurencja, niebezpieczeństwa rekurencji.

– Skład FoilTEX –

Metoda „dziel i zwyciężaj”

Klasy metod algorytmicznych:

- metody zstępujące (analityczne),

Metoda „dziel i zwyciężaj”

Klasy metod algorytmicznych:

- metody zstępujące (analityczne),
- metody wstępujące (syntetyczne).

Metoda „dziel i zwyciężaj”

Klasy metod algorytmicznych:

- metody zstępujące (analityczne),
- metody wstępujące (syntetyczne).

Etapy metody „dziel i zywciężaj”:

Dziel: podziel problem na podproblemy,

Metoda „dziel i zwyciężaj”

Klasy metod algorytmicznych:

- metody zstępujące (analityczne),
- metody wstępujące (syntetyczne).

Etapy metody „dziel i zywciężaj”:

Dziel: podziel problem na podproblemy,

Zwyciężaj: rozwiąż podproblemy, być może również stosując metodę „dziel i zywciężaj”,

Metoda „dziel i zwyciężaj”

Klasy metod algorytmicznych:

- metody zstępujące (analityczne),
- metody wstępujące (syntetyczne).

Etapy metody „dziel i zywciężaj”:

Dziel: podziel problem na podproblemy,

Zwyciężaj: rozwiąż podproblemy, być może również stosując metodę „dziel i zywciężaj”,

Łącz: połącz rozwiązania podproblemów, aby otrzymać rozwiązanie problemu wyjściowego.

Metoda „dziel i zwyciężaj”

Klasy metod algorytmicznych:

- metody zstępujące (analityczne),
- metody wstępujące (syntetyczne).

Etapy metody „dziel i zywciężaj”:

Dziel: podziel problem na podproblemy,

Zwyciężaj: rozwiąż podproblemy, być może również stosując metodę „dziel i zywciężaj”,

Łącz: połącz rozwiązania podproblemów, aby otrzymać rozwiązanie problemu wyjściowego.

Przykład

Posortuj n-elementowy ciąg liczb.

Metoda „dziel i zwyciężaj”

Klasy metod algorytmicznych:

- metody zstępujące (analityczne),
- metody wstępujące (syntetyczne).

Etapy metody „dziel i zywciężaj”:

Dziel: podziel problem na podproblemy,

Zwyciężaj: rozwiąż podproblemy, być może również stosując metodę „dziel i zywciężaj”,

Łącz: połącz rozwiązania podproblemów, aby otrzymać rozwiązanie problemu wyjściowego.

Przykład

Posortuj n-elementowy ciąg liczb.

Dziel: dzielimy n-elementowy ciąg na dwa podciągi po $n/2$ elementów każdy,

Metoda „dziel i zwyciężaj”

Klasy metod algorytmicznych:

- metody zstępujące (analityczne),
- metody wstępujące (syntetyczne).

Etapy metody „dziel i zywciężaj”:

Dziel: podziel problem na podproblemy,

Zwyciężaj: rozwiąż podproblemy, być może również stosując metodę „dziel i zywciężaj”,

Łącz: połącz rozwiązania podproblemów, aby otrzymać rozwiązanie problemu wyjściowego.

Przykład

Posortuj n-elementowy ciąg liczb.

Dziel: dzielimy n-elementowy ciąg na dwa podciągi po $n/2$ elementów każdy,

Zwyciężaj: sortujemy otrzymane podciągi,

Metoda „dziel i zwyciężaj”

Klasy metod algorytmicznych:

- metody zstępujące (analityczne),
- metody wstępujące (syntetyczne).

Etapy metody „dziel i zywciężaj”:

Dziel: podziel problem na podproblemy,

Zwyciężaj: rozwiąż podproblemy, być może również stosując metodę „dziel i zywciężaj”,

Łącz: połącz rozwiązania podproblemów, aby otrzymać rozwiązanie problemu wyjściowego.

Przykład

Posortuj n-elementowy ciąg liczb.

Dziel: dzielimy n-elementowy ciąg na dwa podciągi po $n/2$ elementów każdy,

Zwyciężaj: sortujemy otrzymane podciągi,

Łącz: łączymy posortowane podciągi w jeden posortowany ciąg.

Metoda przyrostowa

Mając rozwiązany problem dla $n-1$ elementów znajdź rozwiązanie dla n elementów.

Metoda przyrostowa

Mając rozwiązany problem dla $n-1$ elementów znajdź rozwiązanie dla n elementów.

Przykład

Wylicz silnię liczby.

Metoda przyrostowa

Mając rozwiązany problem dla $n-1$ elementów znajdź rozwiązanie dla n elementów.

Przykład

Wylicz silnię liczby.

```
FUNCTION Silnia(n: INTEGER): INTEGER;
{ Funkcja wylicza wartosc silni danej }
{ liczby.  UWAGA: dla n <= 0 zwraca 1 }
BEGIN
  IF n <= 1 THEN
    Silnia := 1
  ELSE
    Silnia := n * Silnia(n-1)
END;
```

Formułowanie warunków logicznych

```
VAR     a : REAL;
...
   IF a = 0 THEN ...
```

Formułowanie warunków logicznych

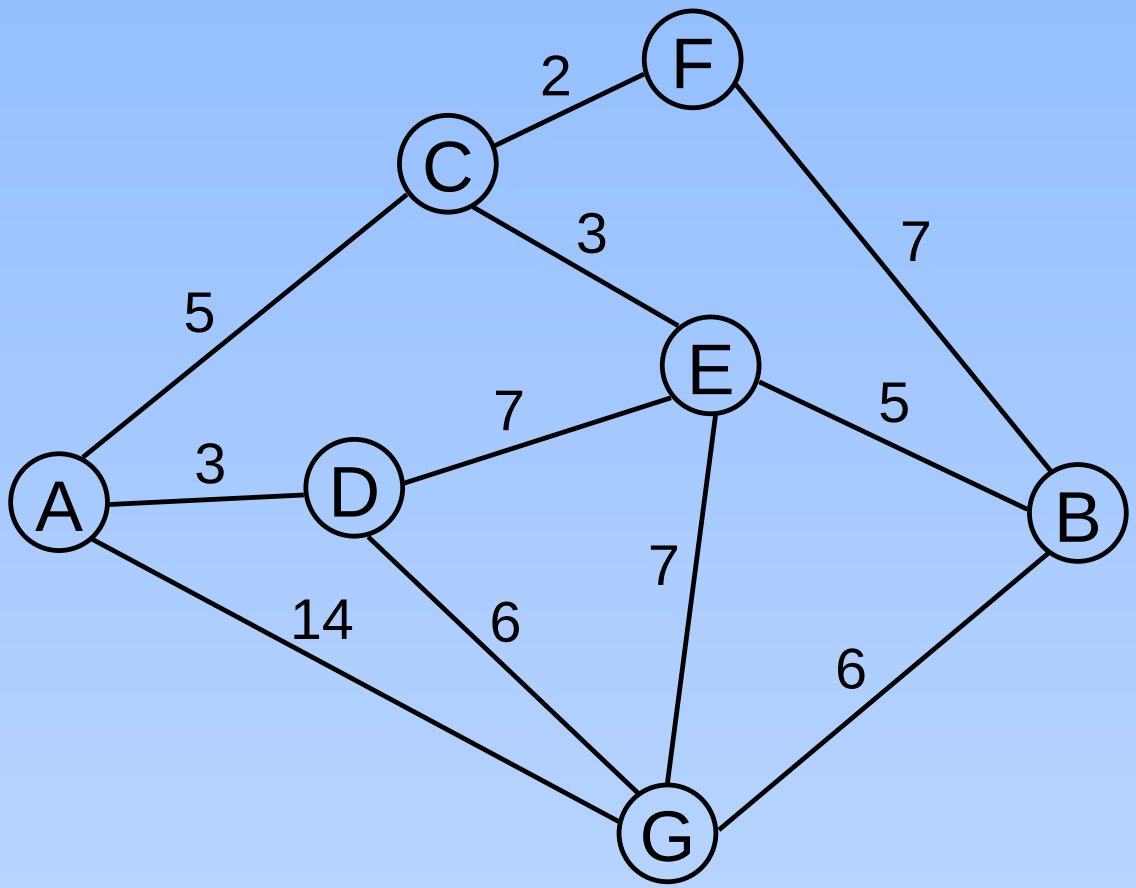
```
VAR     a : REAL;
...
  IF a = 0 THEN ...
```

```
CONST  epsilon = 0.0001;
VAR     a : REAL;
...
  IF ABS(a) < epsilon THEN ...
```

Algorytmy zachłanne

Przykład

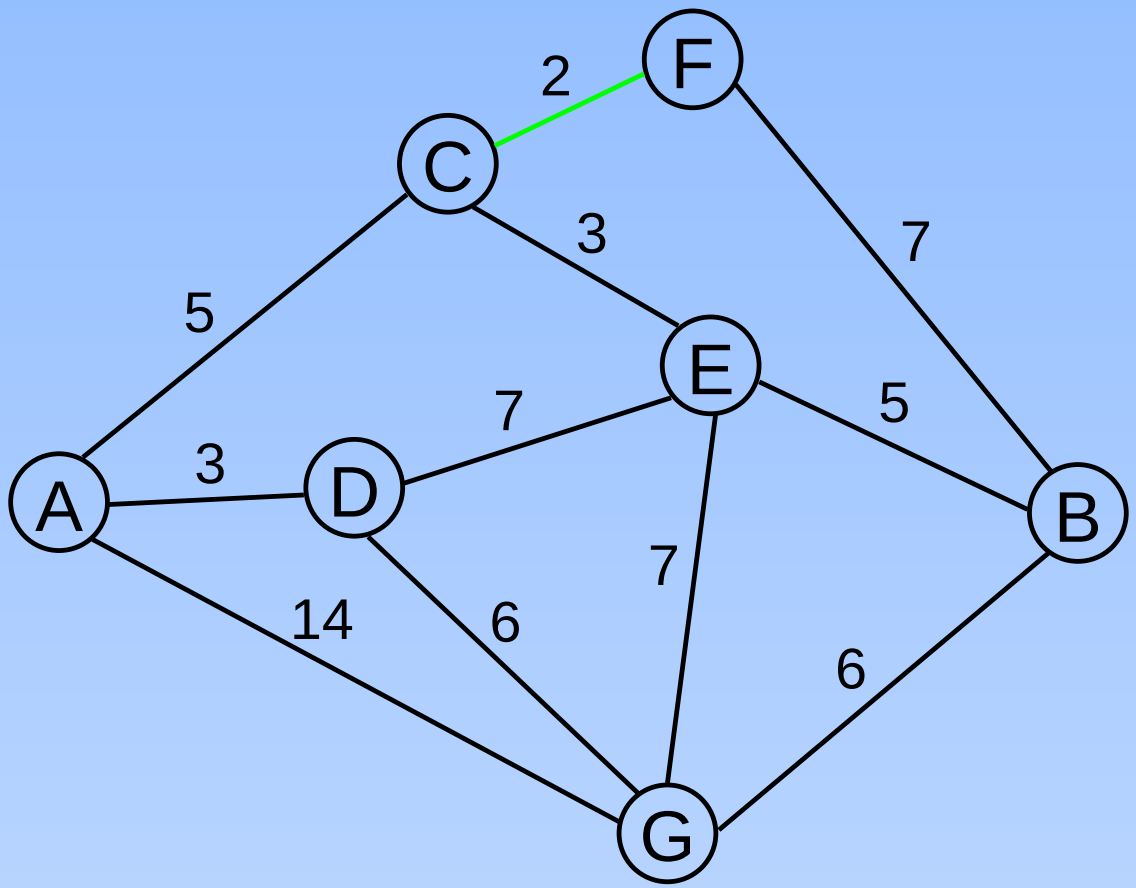
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

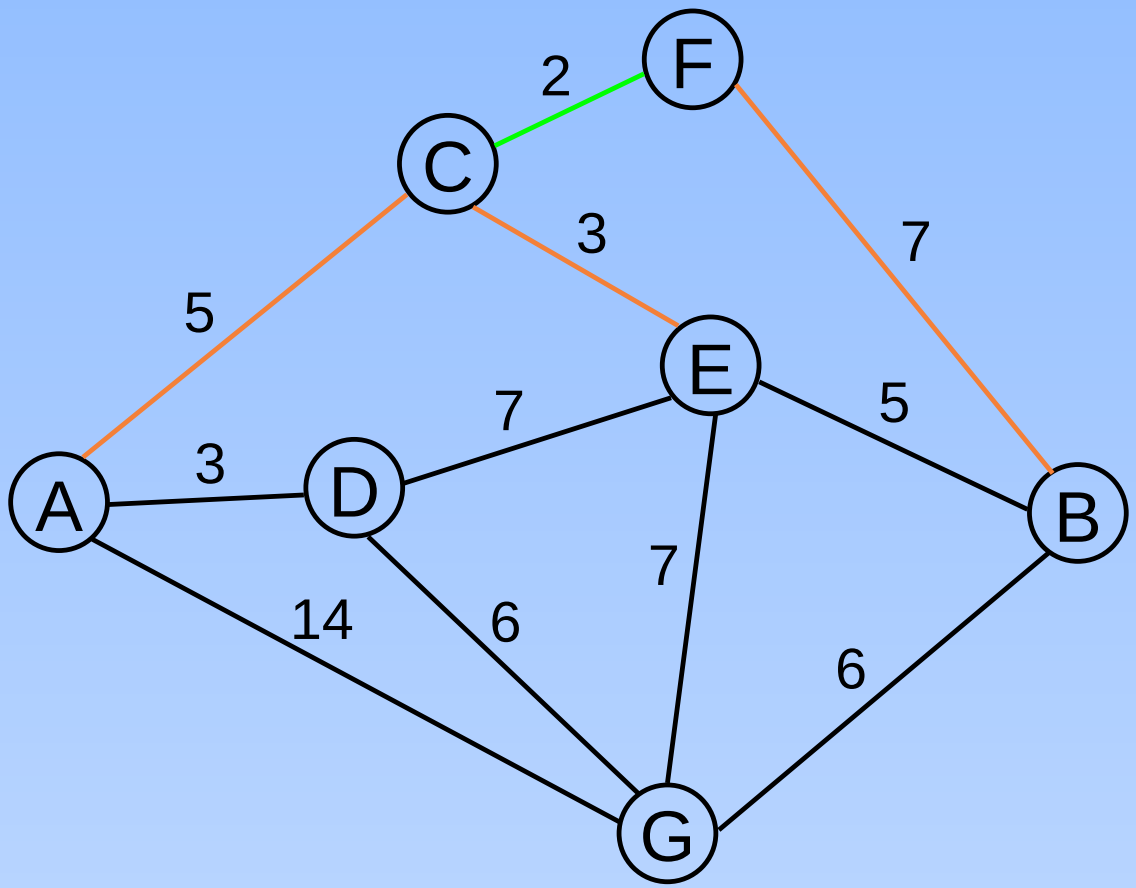
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

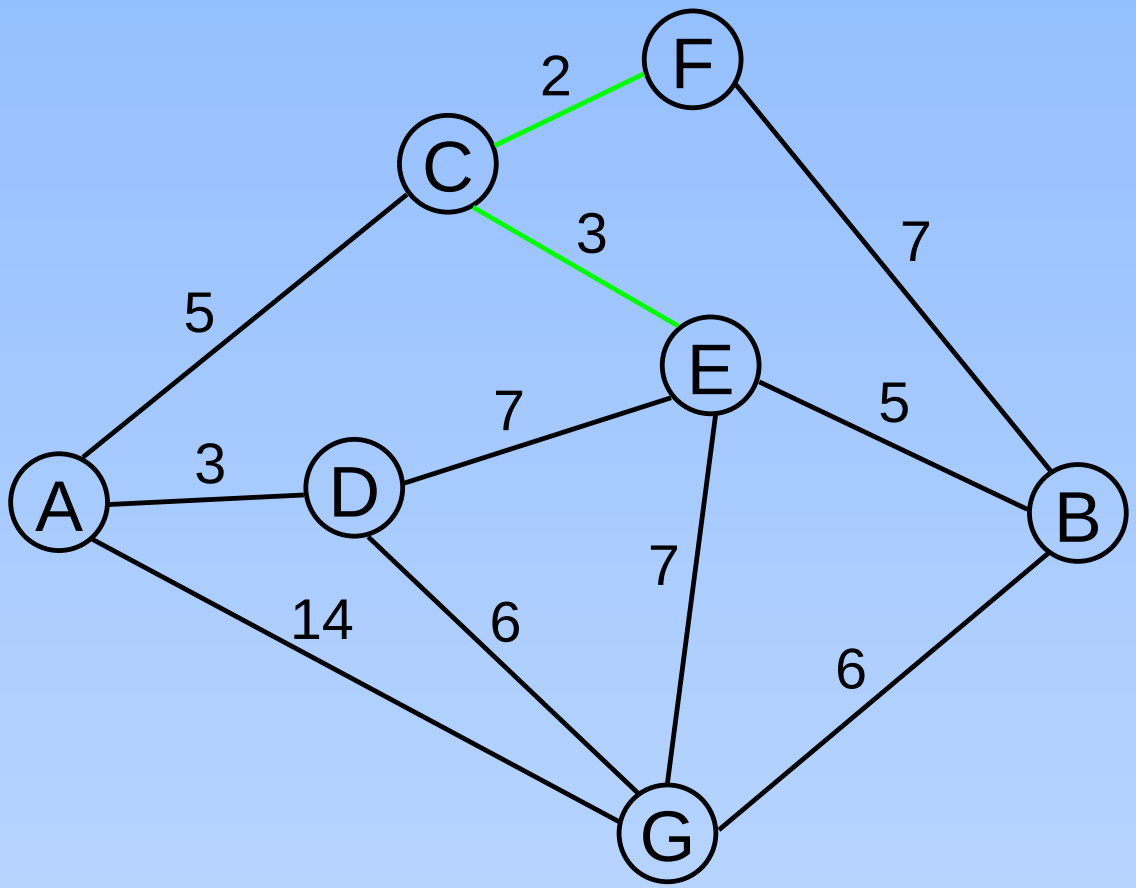
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

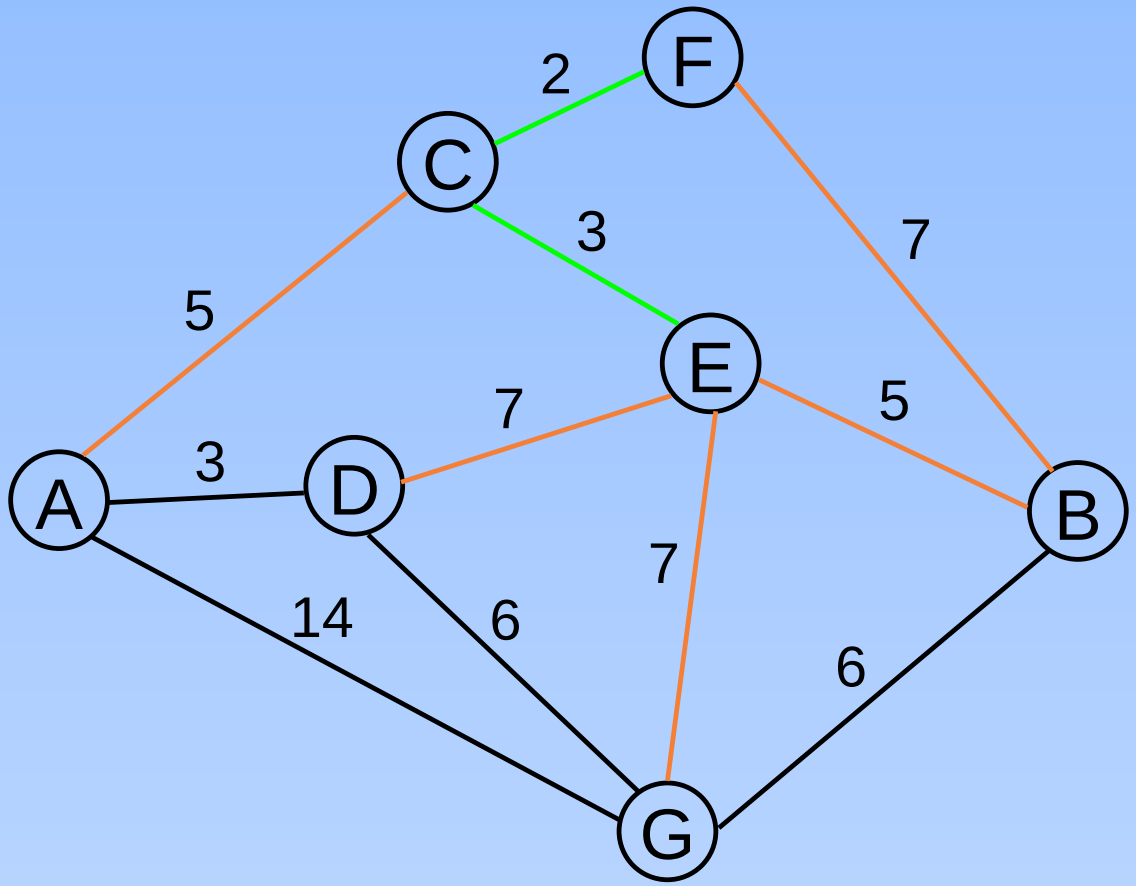
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

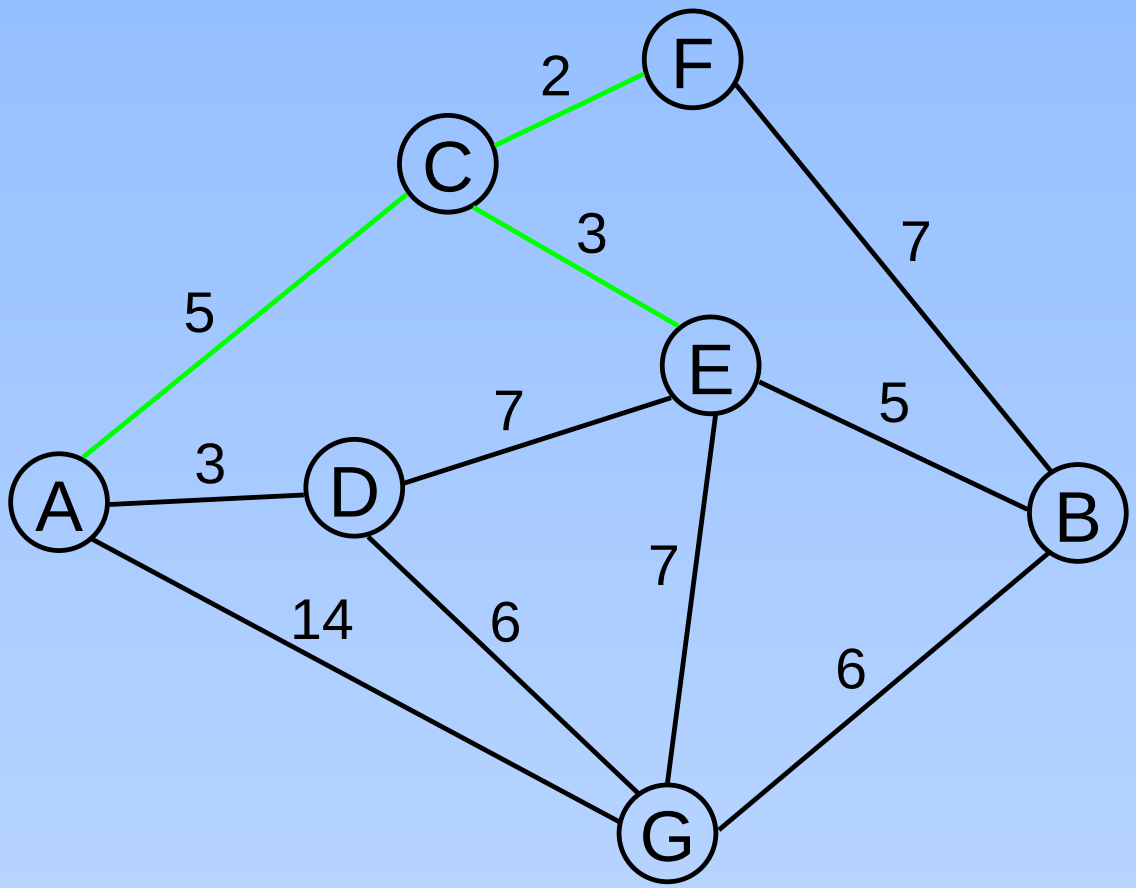
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

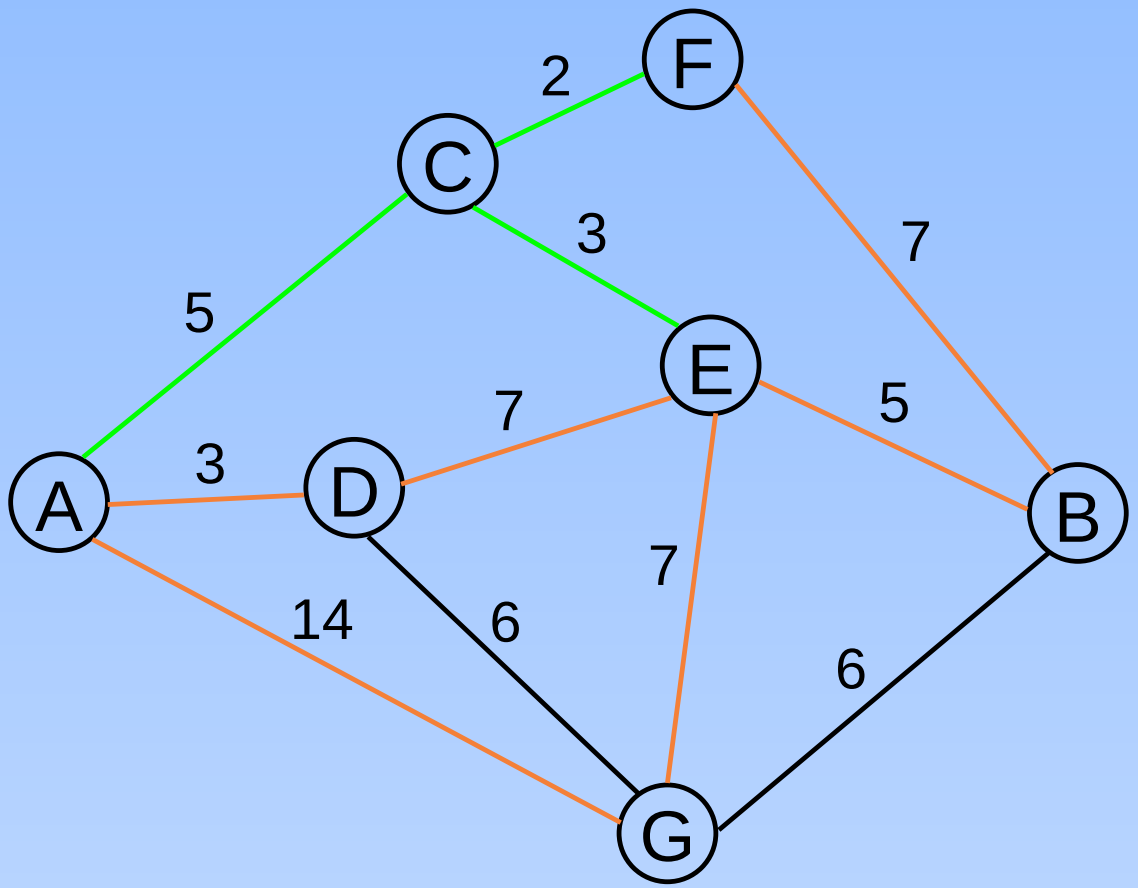
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

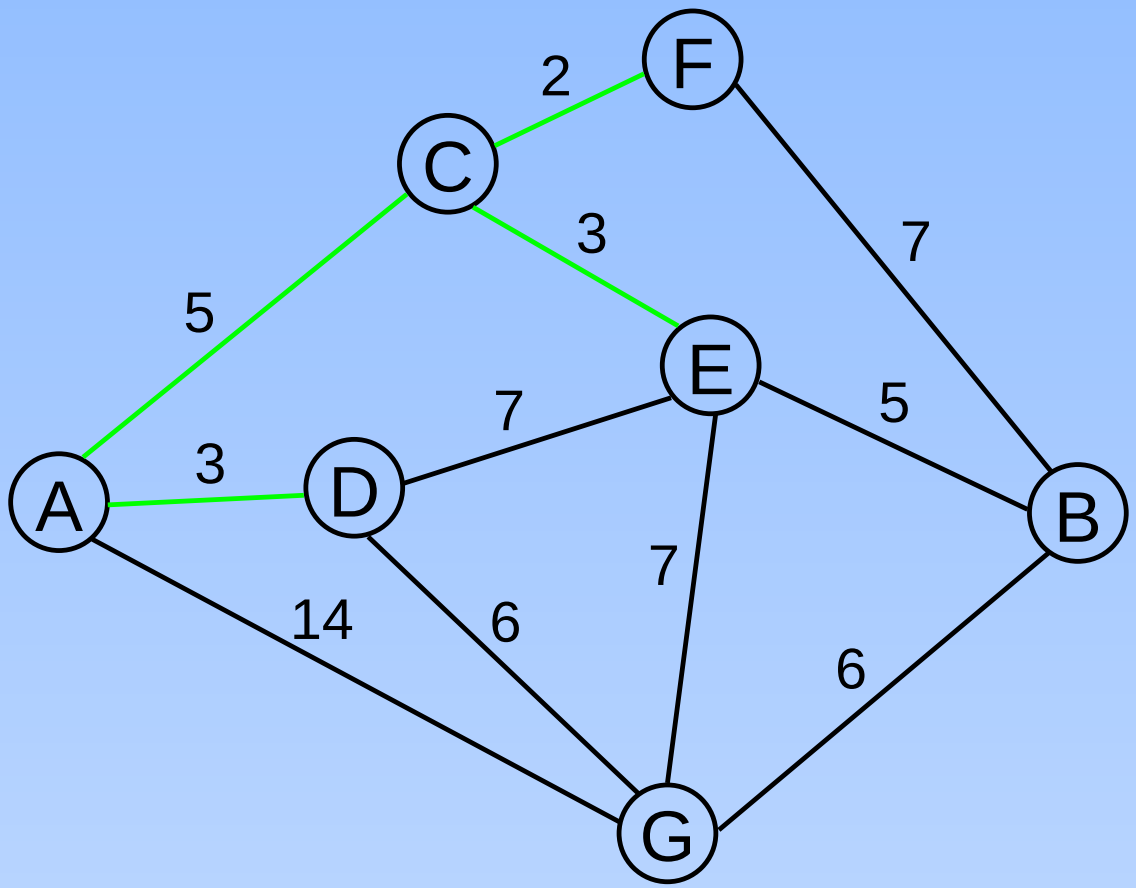
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

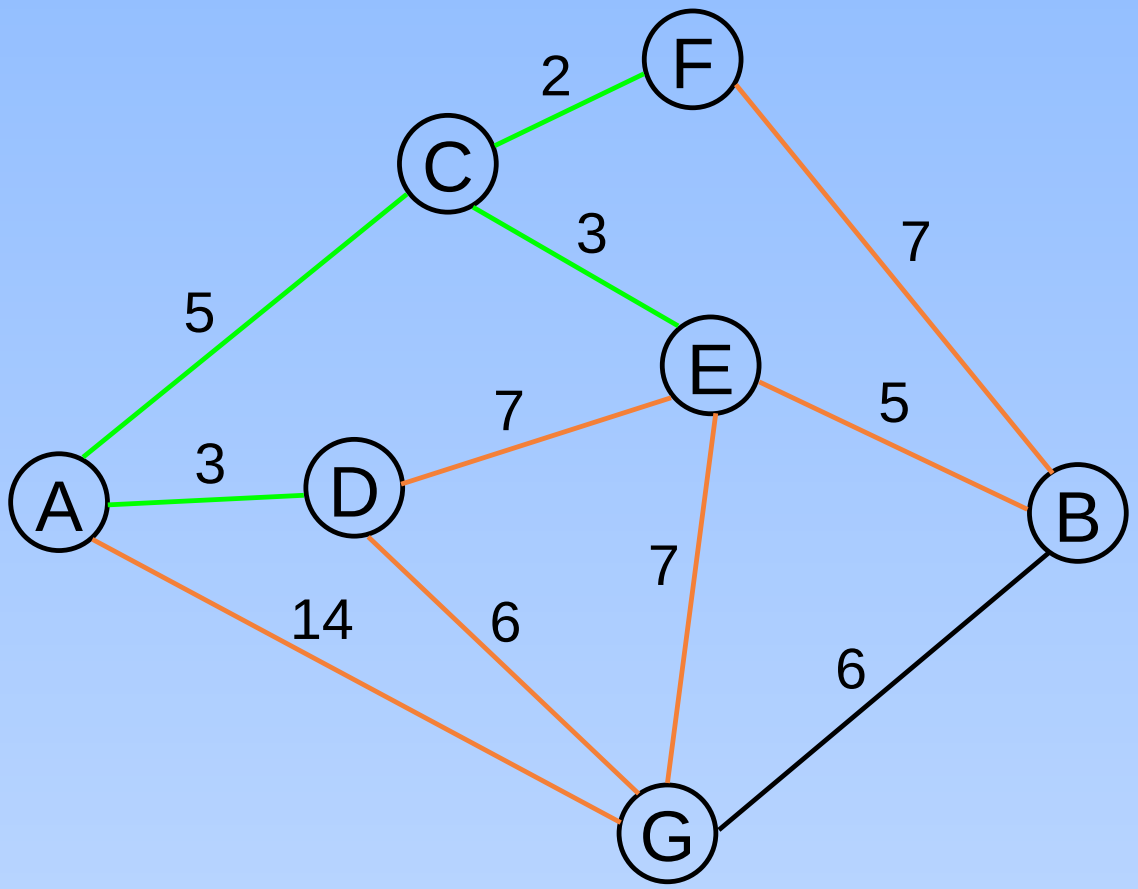
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

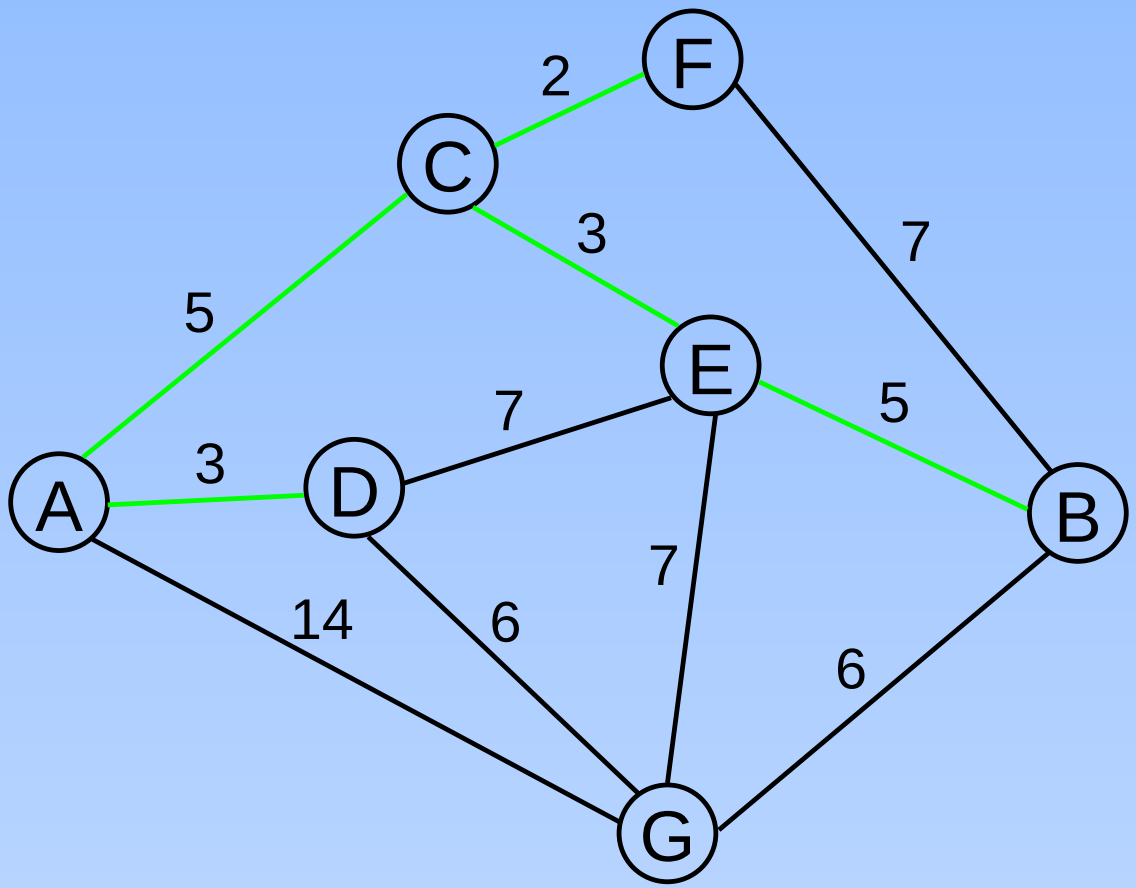
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

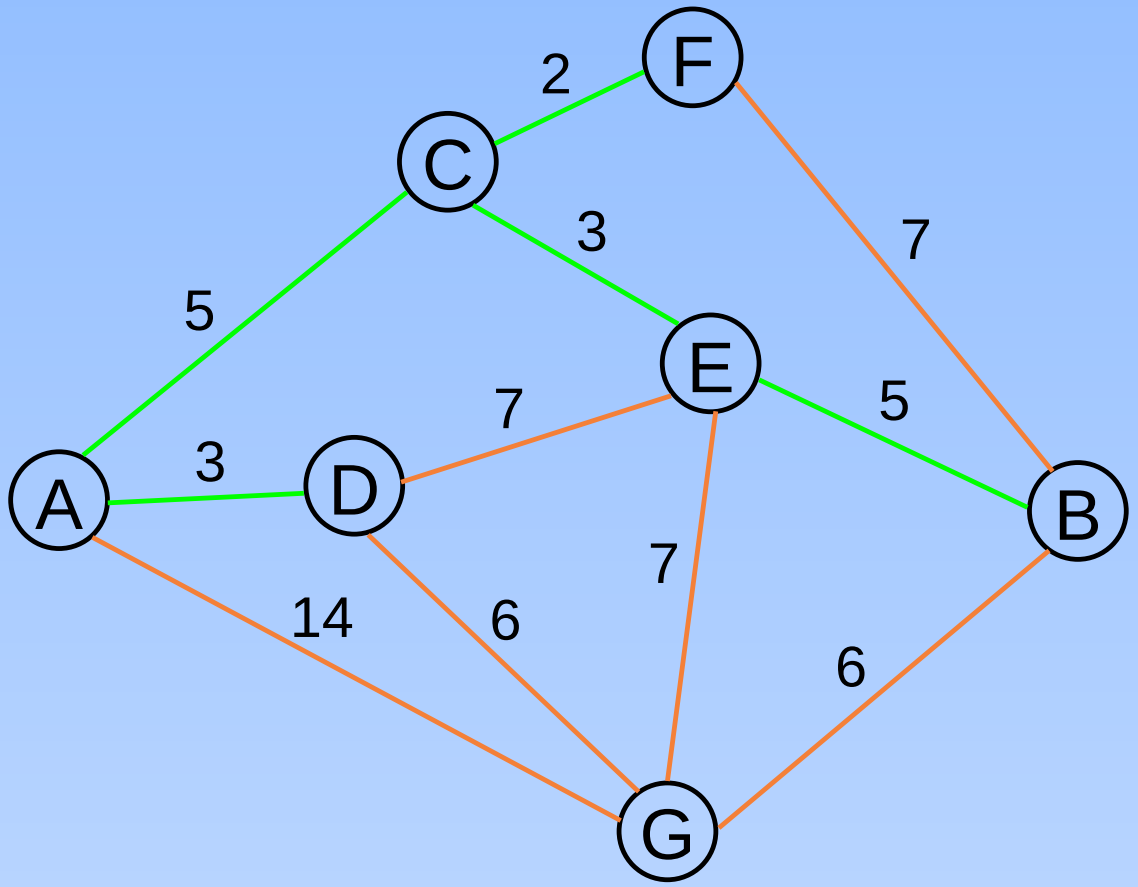
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

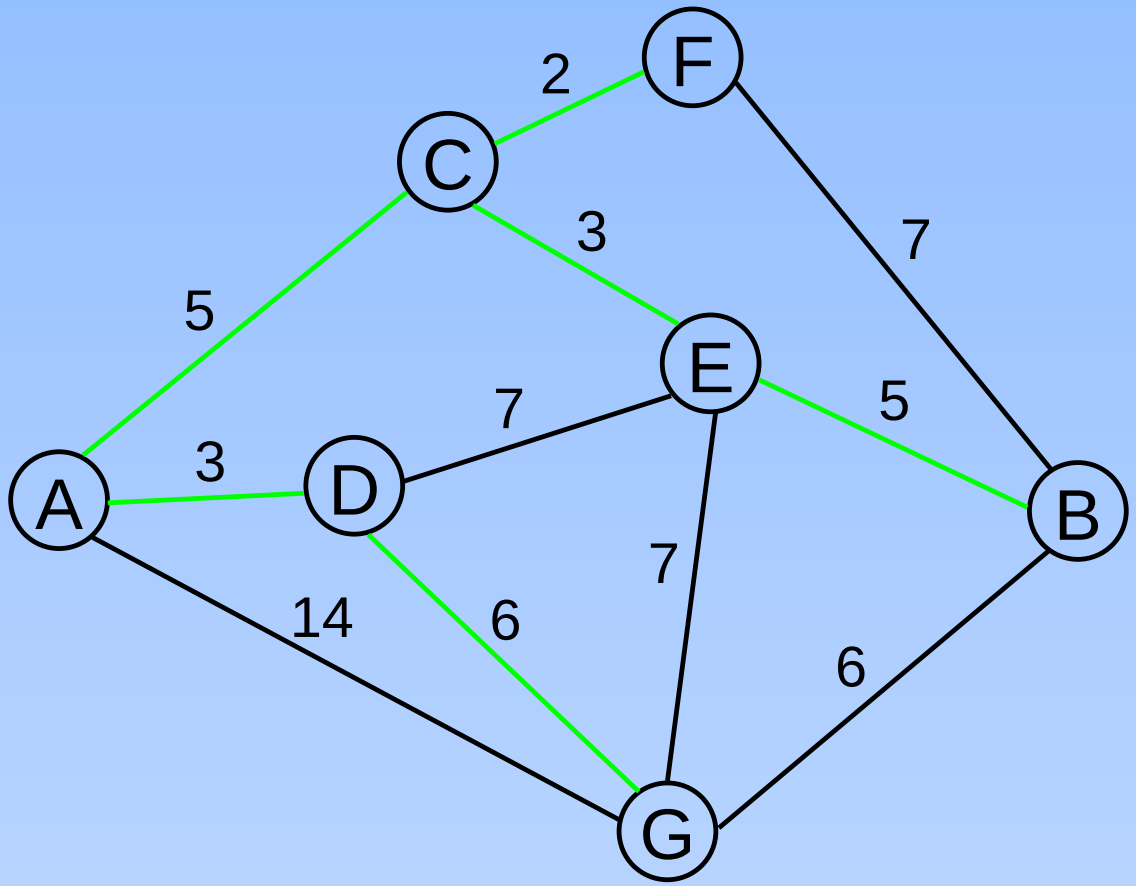
Problem leniwego przedsiębiorcy budującego autostrady.



Algorytmy zachłanne

Przykład

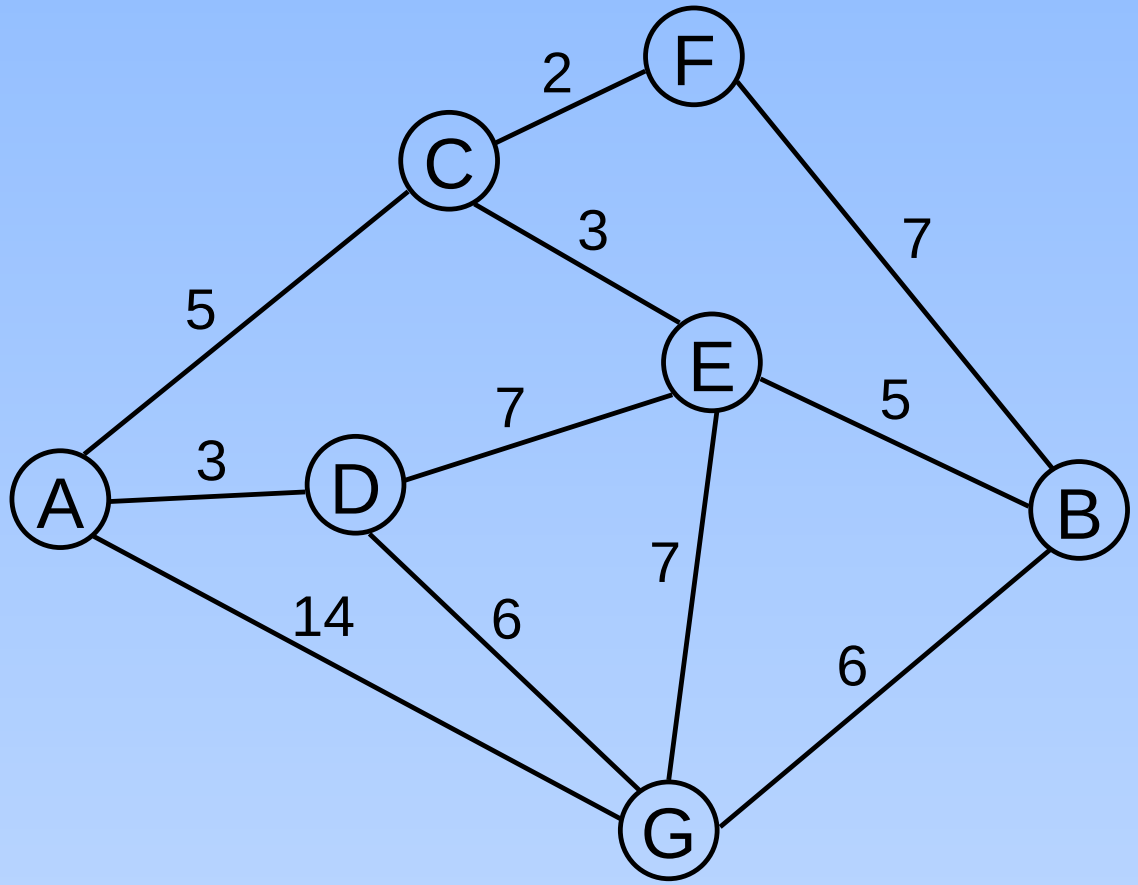
Problem leniwego przedsiębiorcy budującego autostrady.



Programowanie dynamiczne

Przykład

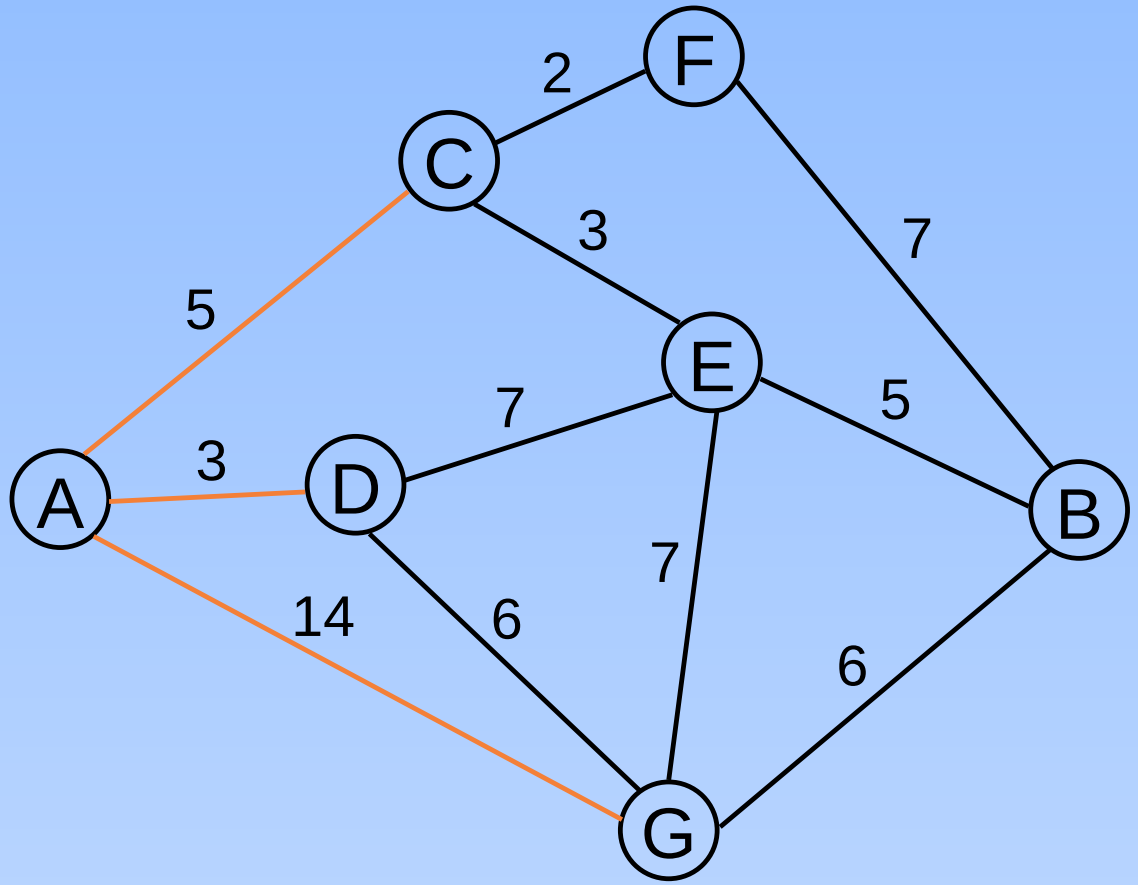
Problem znużonego wędrowcy



Programowanie dynamiczne

Przykład

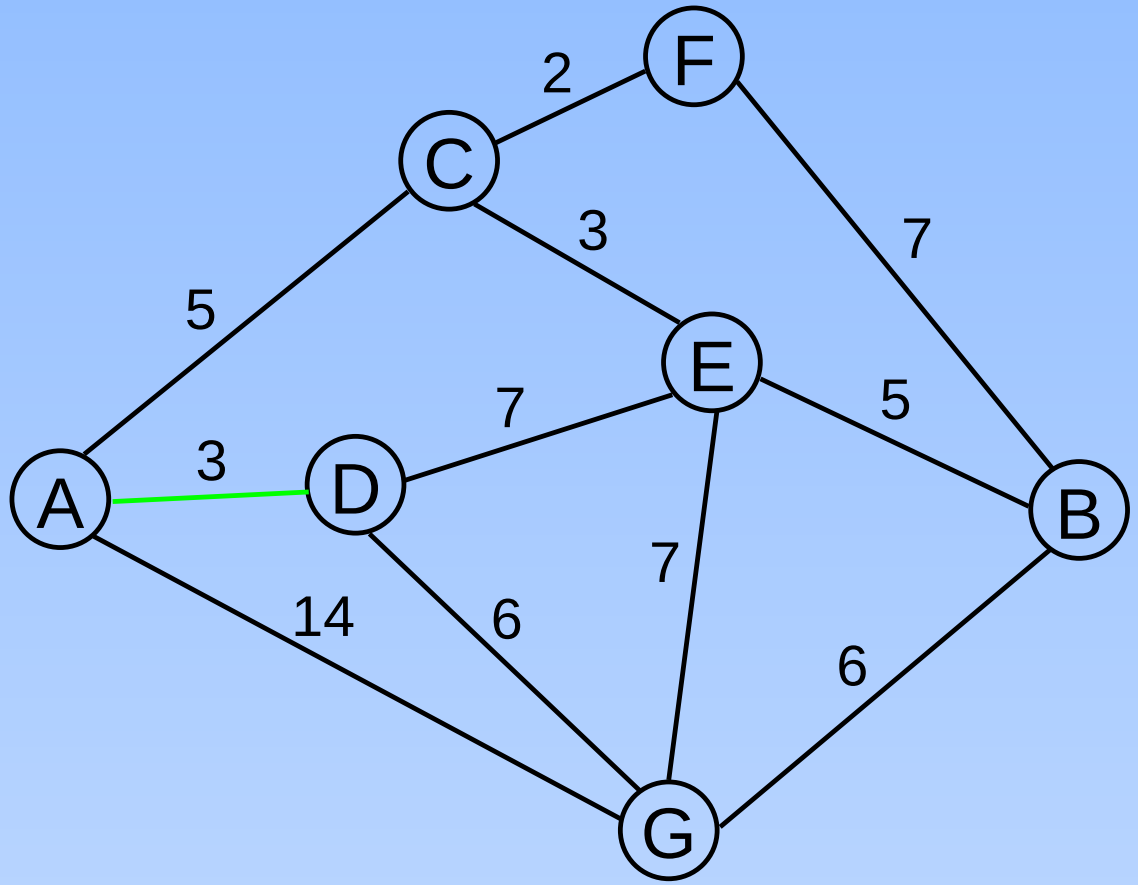
Problem znużonego wędrowcy



Programowanie dynamiczne

Przykład

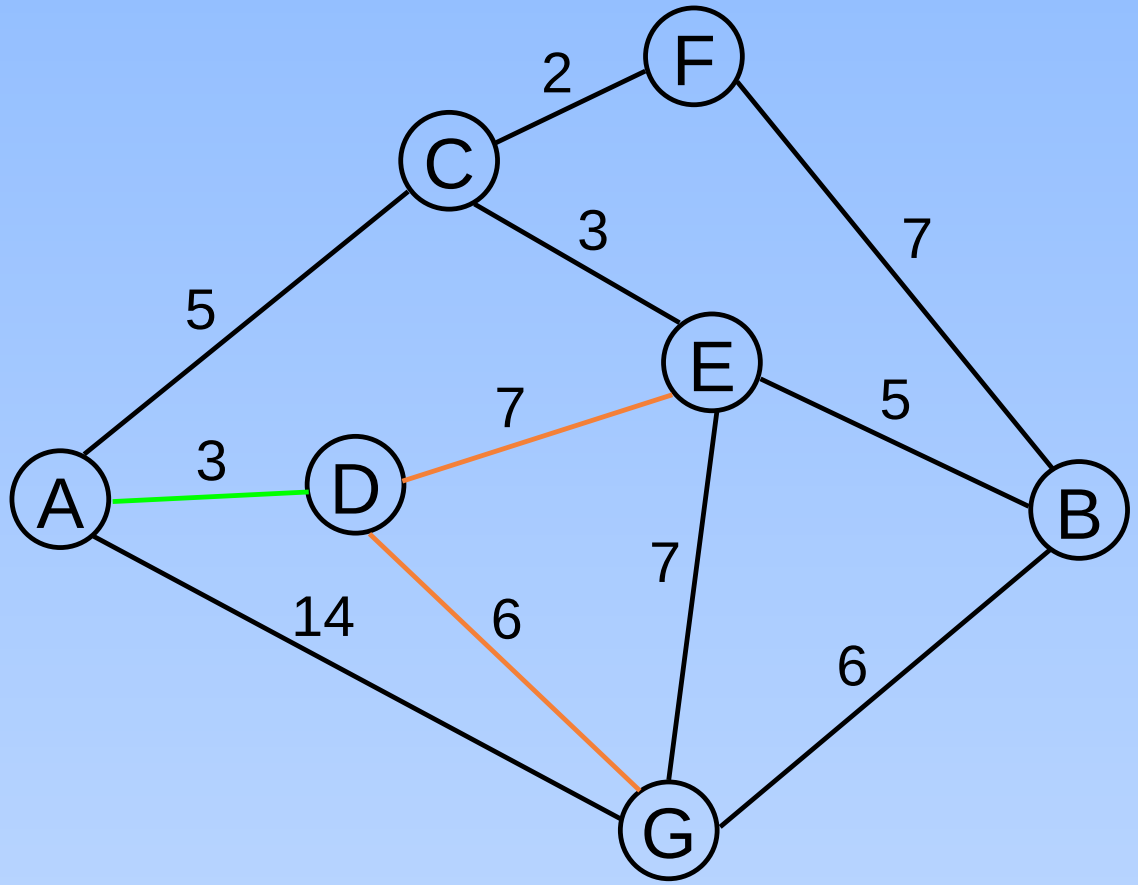
Problem znużonego wędrowcy



Programowanie dynamiczne

Przykład

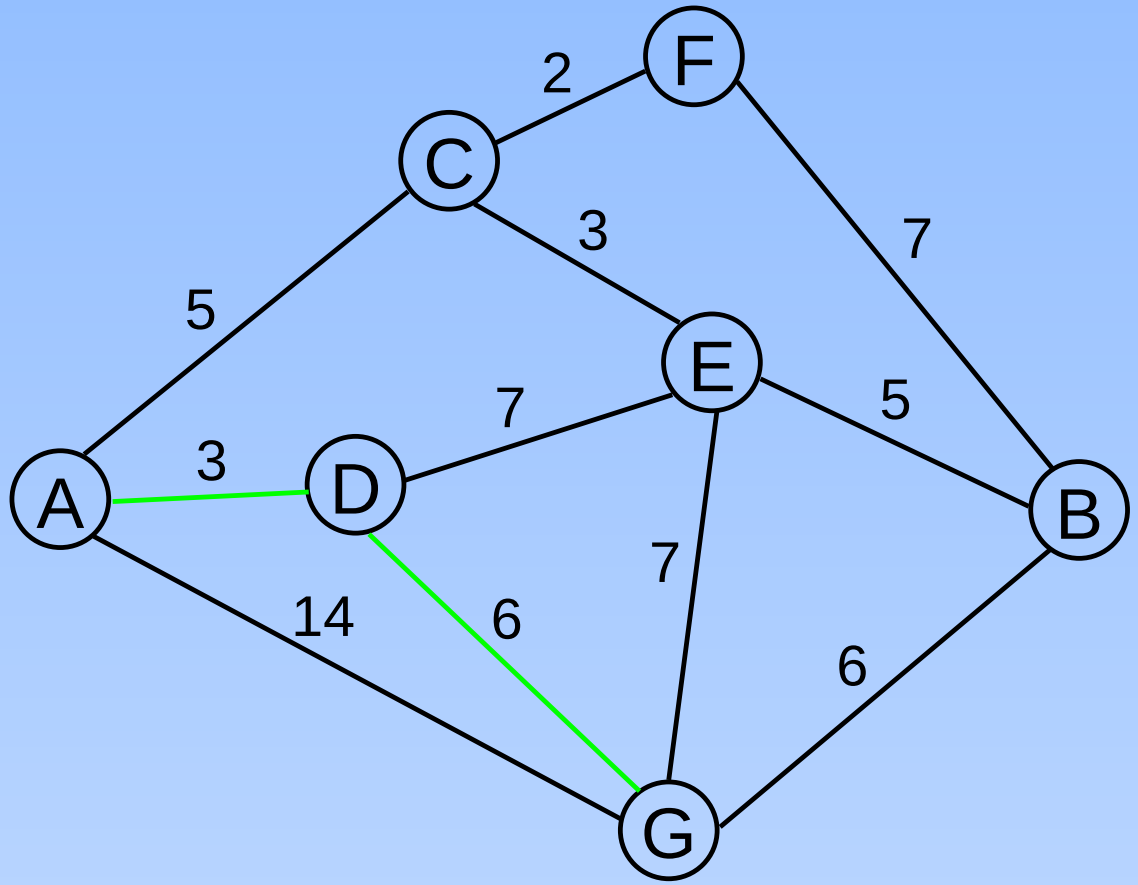
Problem znużonego wędrowcy



Programowanie dynamiczne

Przykład

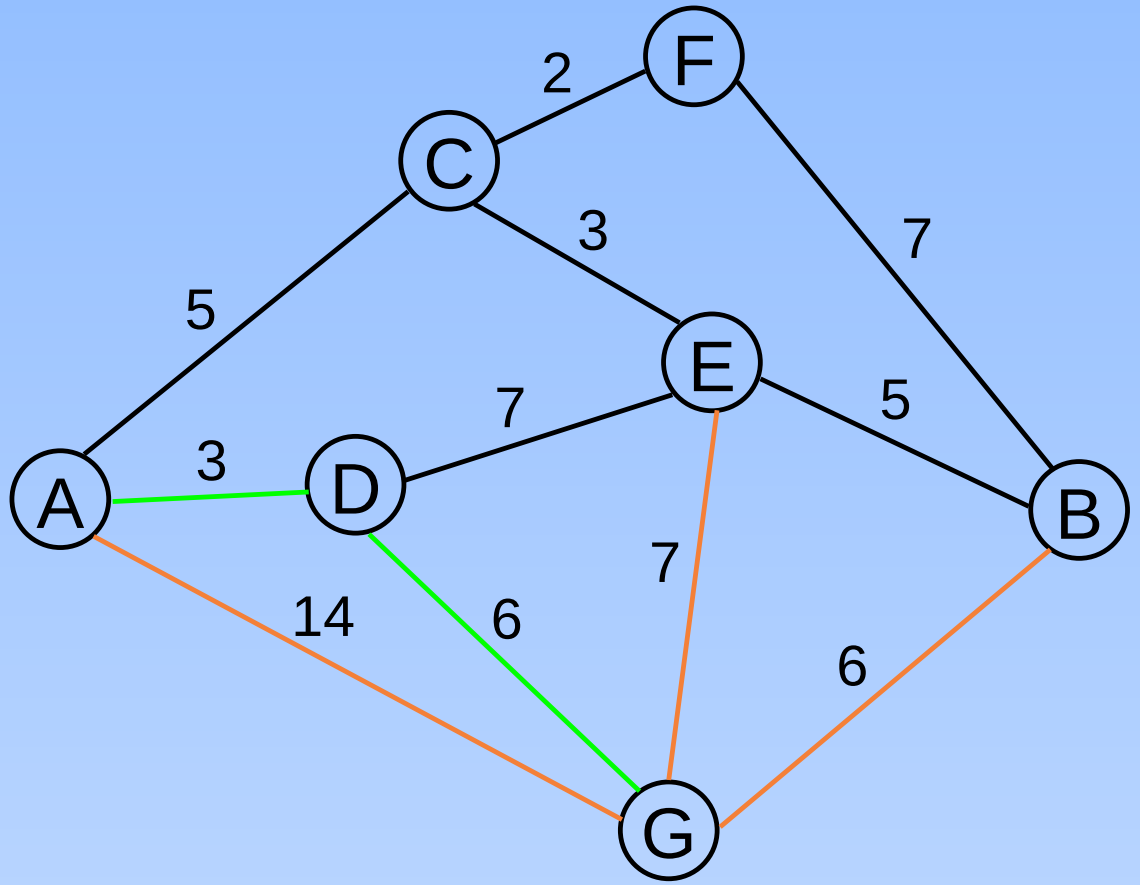
Problem znużonego wędrowcy



Programowanie dynamiczne

Przykład

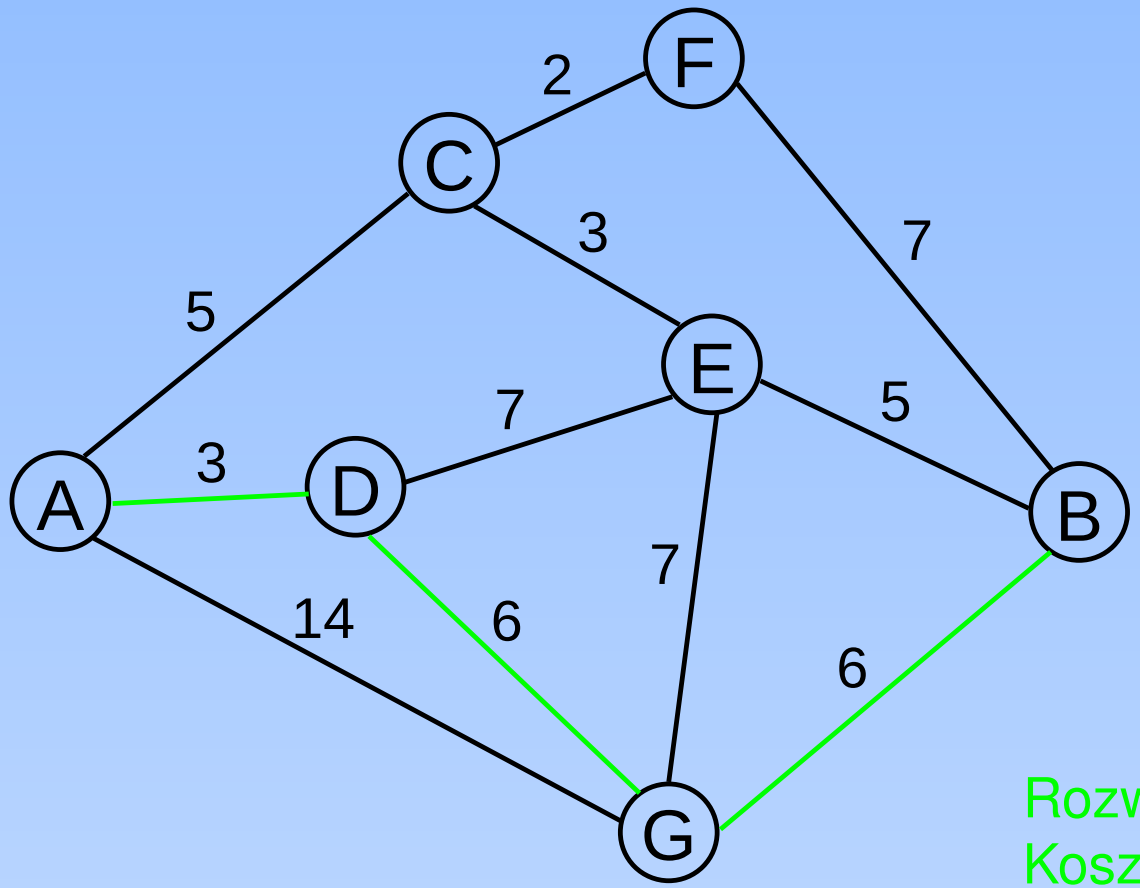
Problem znużonego wędrowcy



Programowanie dynamiczne

Przykład

Problem znużonego wędrowcy

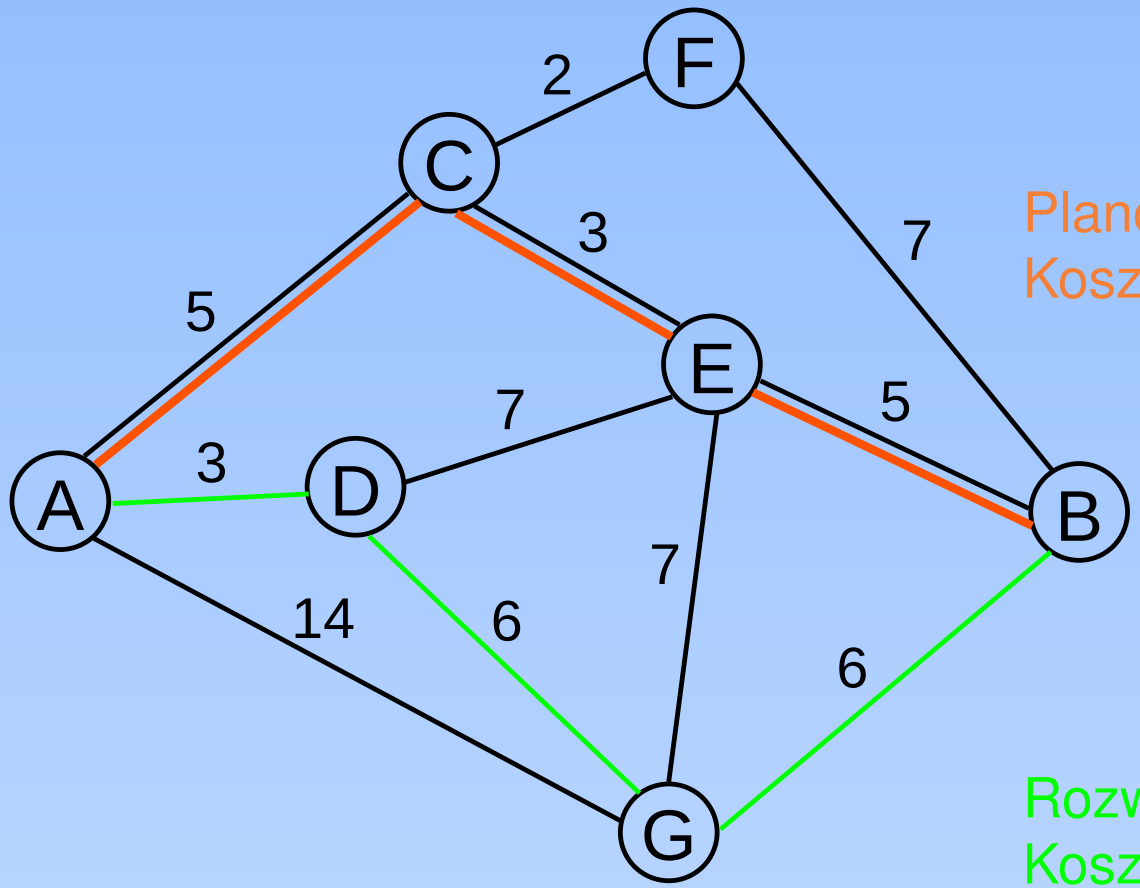


Rozwiązanie zachłanne.
Koszt całkowity: 15.

Programowanie dynamiczne

Przykład

Problem znużonego wędrowcy

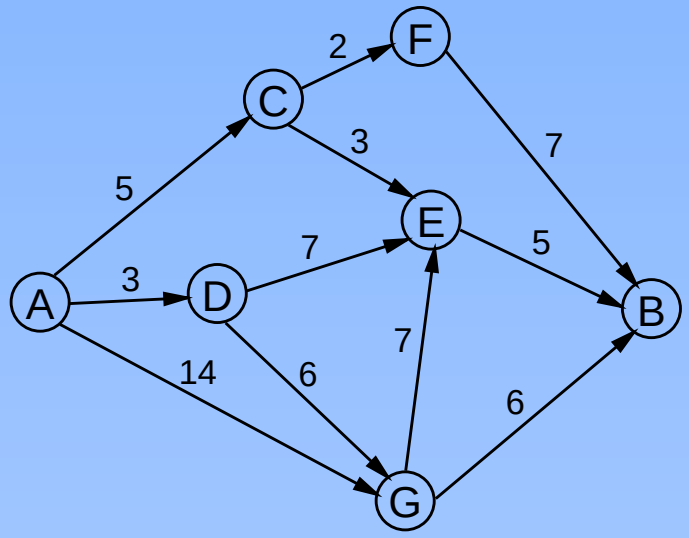


Planowanie dynamiczne.
Koszt całkowity: 13.

Rozwiązanie zachłanne.
Koszt całkowity: 15.

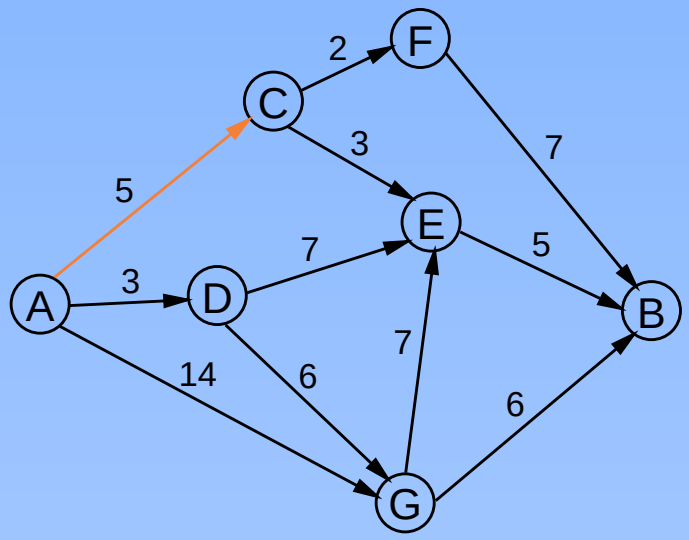
Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy



Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

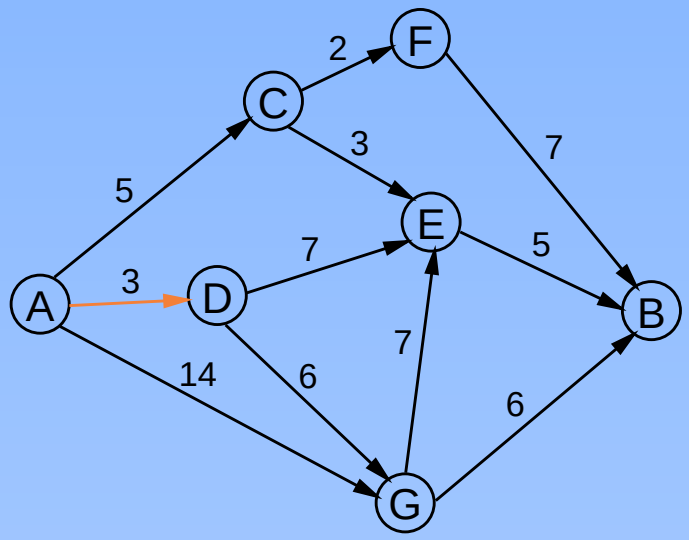


L(X) — odległość z X do B

5+L(C)

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

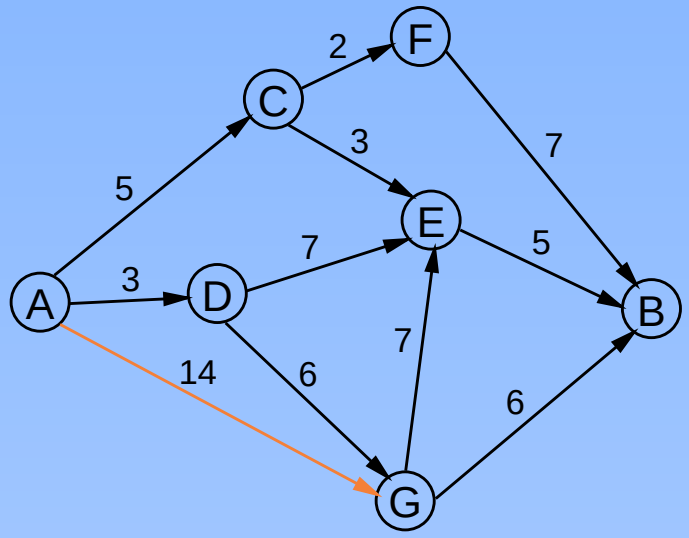


L(X) — odległość z X do B

5+L(C)	3+L(D)	

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

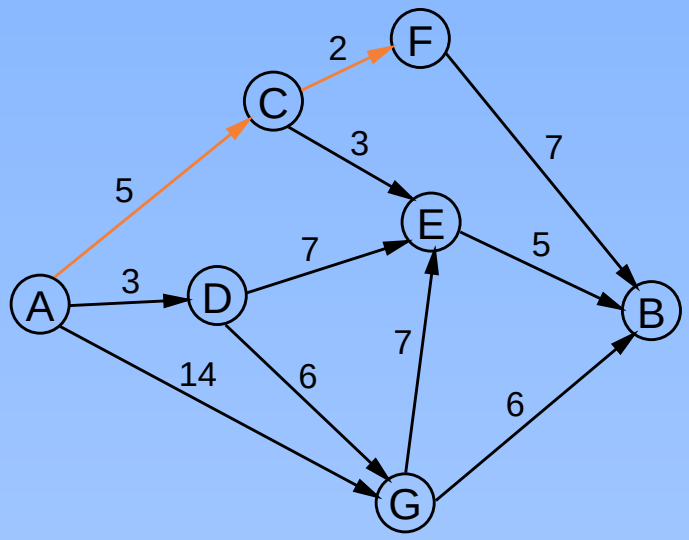


L(X) — odległość z X do B

5+L(C)	3+L(D)	14+L(G)

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

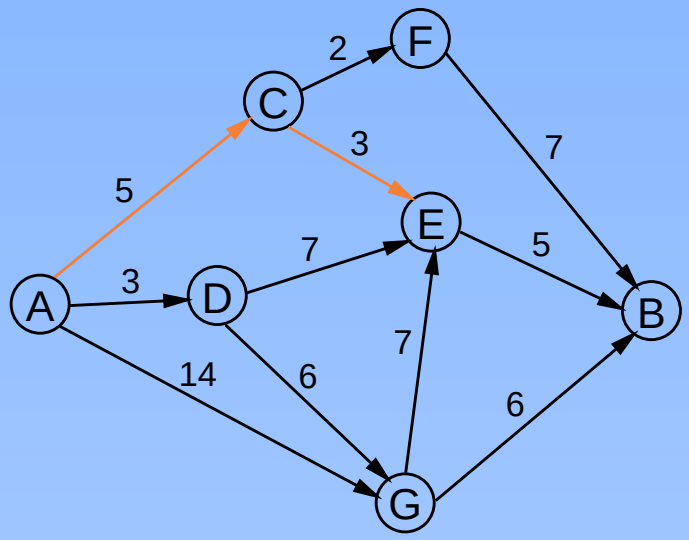


L(X) — odległość z X do B

5+L(C)	3+L(D)	14+L(G)
7+L(F)		

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

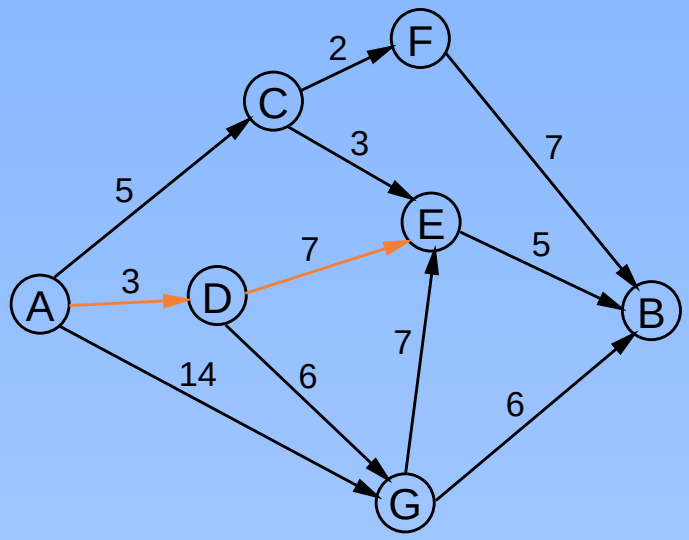


L(X) — odległość z X do B

5+L(C)	3+L(D)	14+L(G)
7+L(F) | 8+L(E)		

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

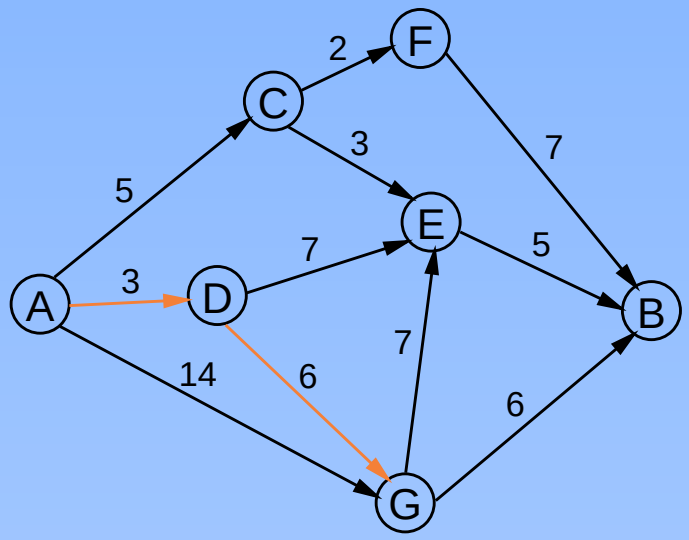


L(X) — odległość z X do B

5+L(C)		3+L(D)		14+L(G)
7+L(F)	8+L(E)	10+L(E)		

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

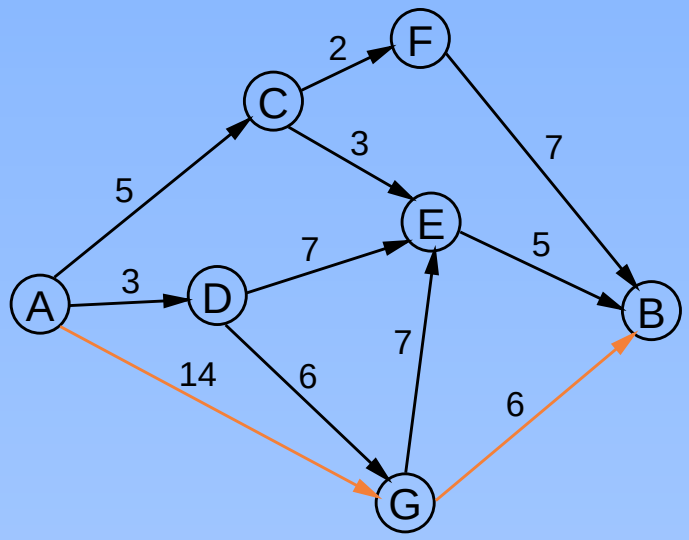


L(X) — odległość z X do B

5+L(C)		3+L(D)		14+L(G)
7+L(F)	8+L(E)	10+L(E)	9+L(G)	

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

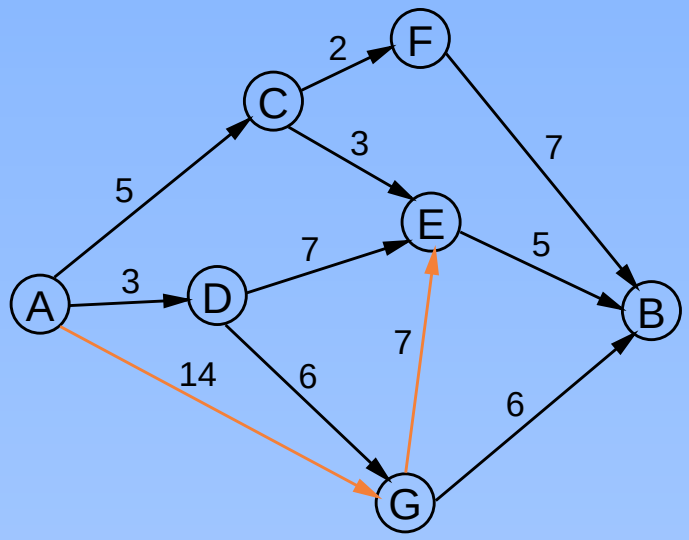


L(X) — odległość z X do B

5+L(C)		3+L(D)		14+L(G)
7+L(F)	8+L(E)	10+L(E)	9+L(G)	20+L(B)

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

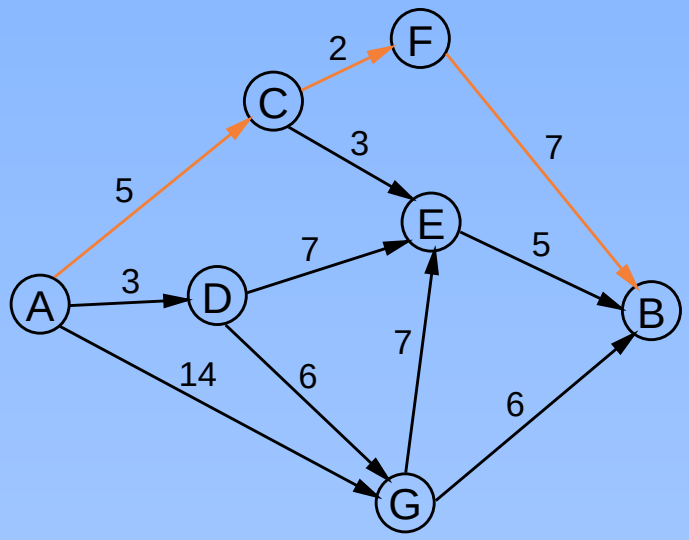


L(X) — odległość z X do B

5+L(C)		3+L(D)		14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)	20+L(B)	21+L(E)

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

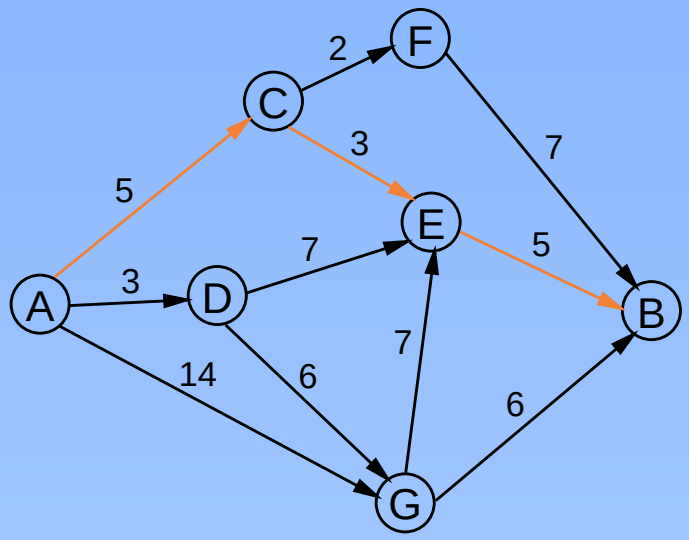


L(X) — odległość z X do B

5+L(C)		3+L(D)		14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)	20+L(B)	21+L(E)
14+L(B)					

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

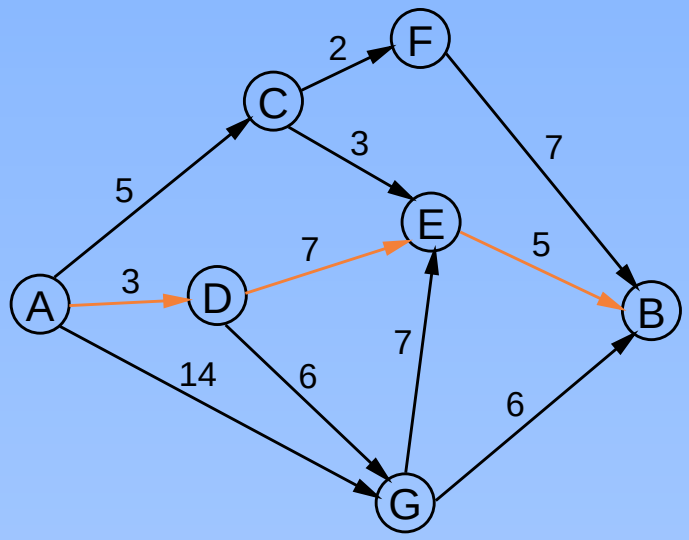


L(X) — odległość z X do B

5+L(C)		3+L(D)		14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)	20+L(B)	21+L(E)
14+L(B)	13+L(B)				

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

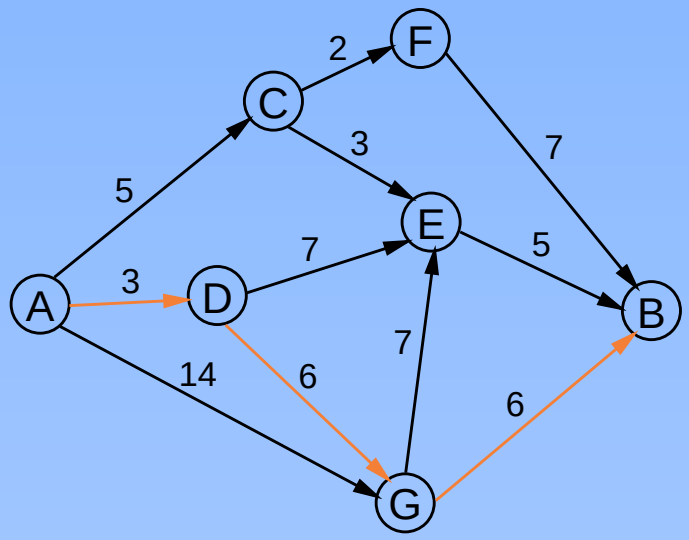


L(X) — odległość z X do B

5+L(C)		3+L(D)		14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)	20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)			

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

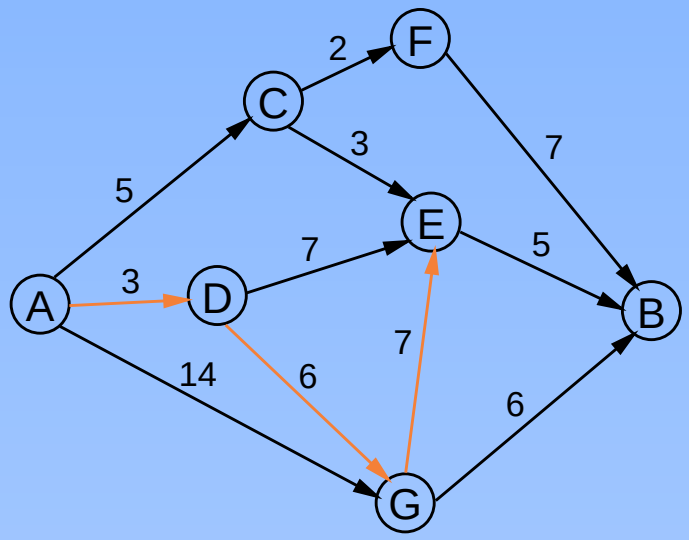


L(X) — odległość z X do B

5+L(C)		3+L(D)		14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)	20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)		

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

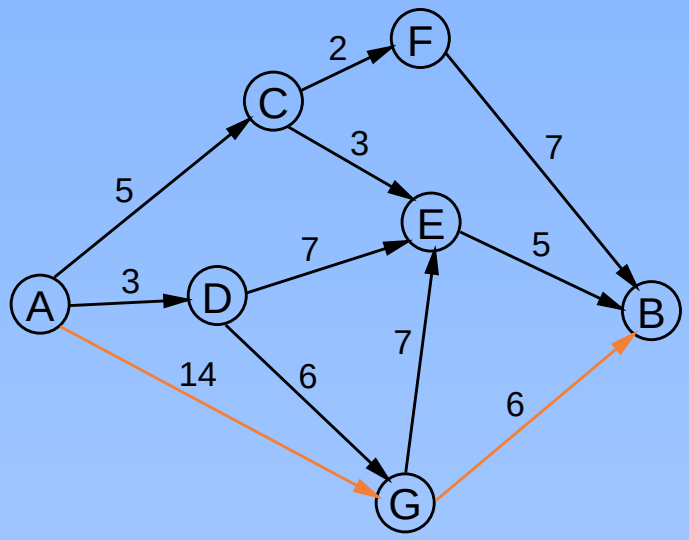


L(X) — odległość z X do B

5+L(C)		3+L(D)		14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)	20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

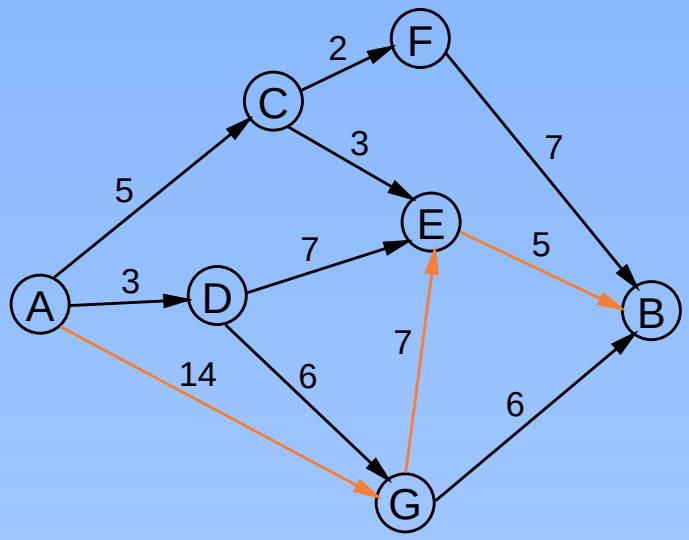


L(X) — odległość z X do B

5+L(C)		3+L(D)			14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)		20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	20+L(B)	

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

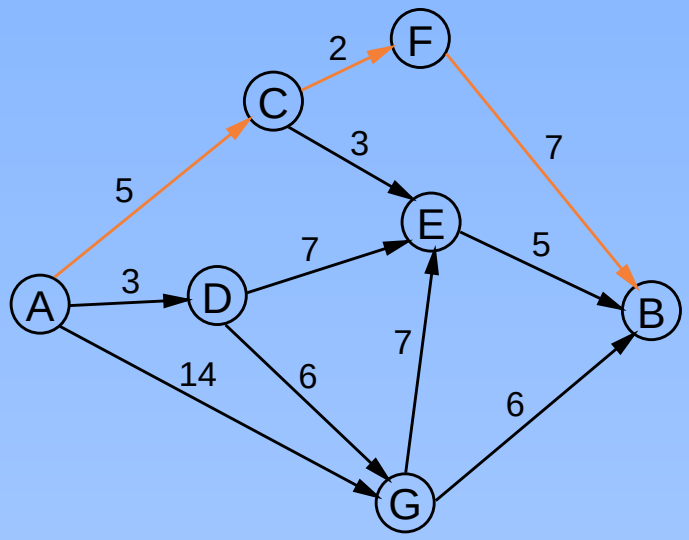


L(X) — odległość z X do B

5+L(C)		3+L(D)			14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)		20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	20+L(B)	26+L(B)

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

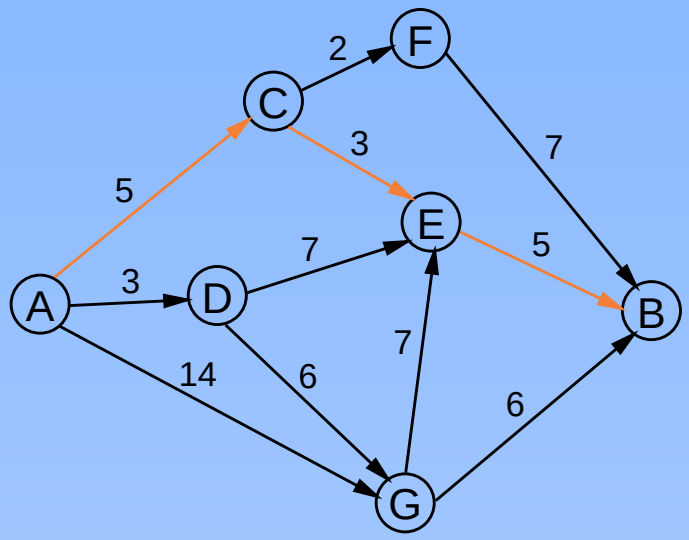


L(X) — odległość z X do B

5+L(C)		3+L(D)			14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)		20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	20+L(B)	26+L(B)
14						

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

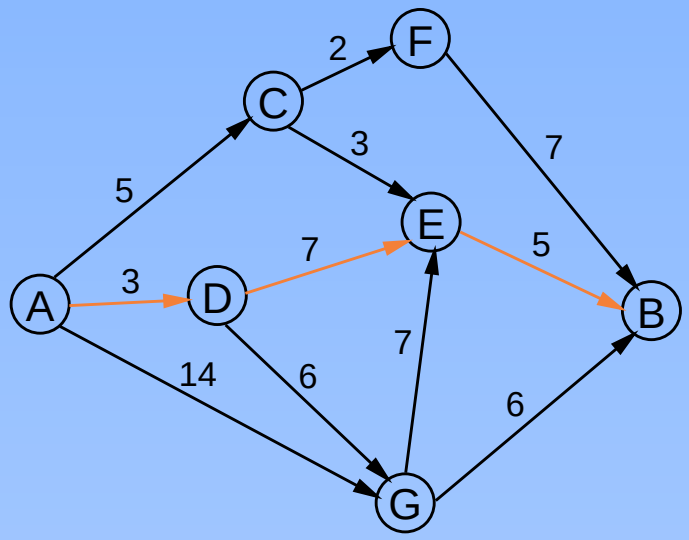


L(X) — odległość z X do B

5+L(C)		3+L(D)			14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)		20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	20+L(B)	26+L(B)
14	13					

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

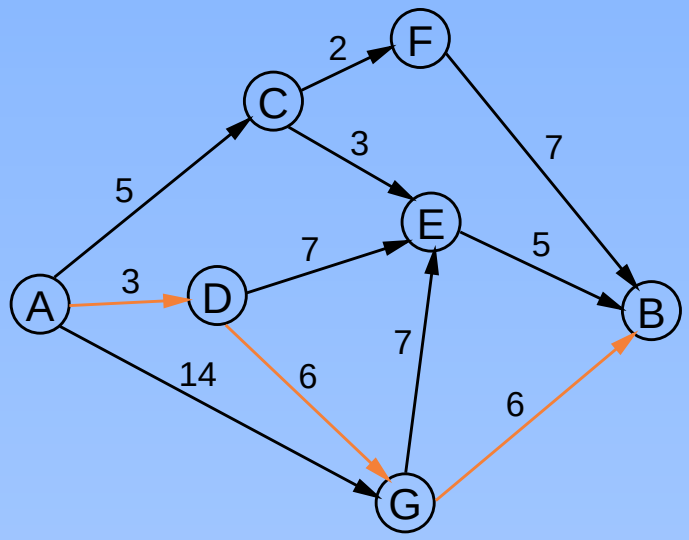


L(X) — odległość z X do B

5+L(C)		3+L(D)			14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)		20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	20+L(B)	26+L(B)
14	13	15				

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

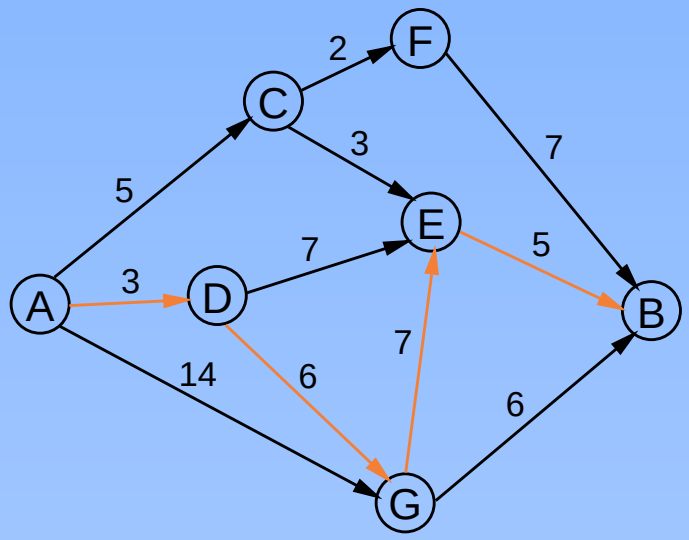


L(X) — odległość z X do B

5+L(C)		3+L(D)			14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)		20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	20+L(B)	26+L(B)
14	13	15	15			

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

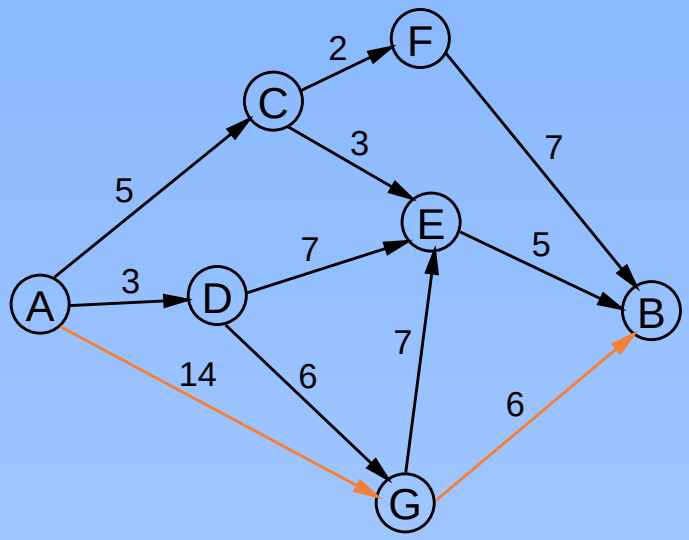


L(X) — odległość z X do B

5+L(C)		3+L(D)			14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)		20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	20+L(B)	26+L(B)
14	13	15	15	27		

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy

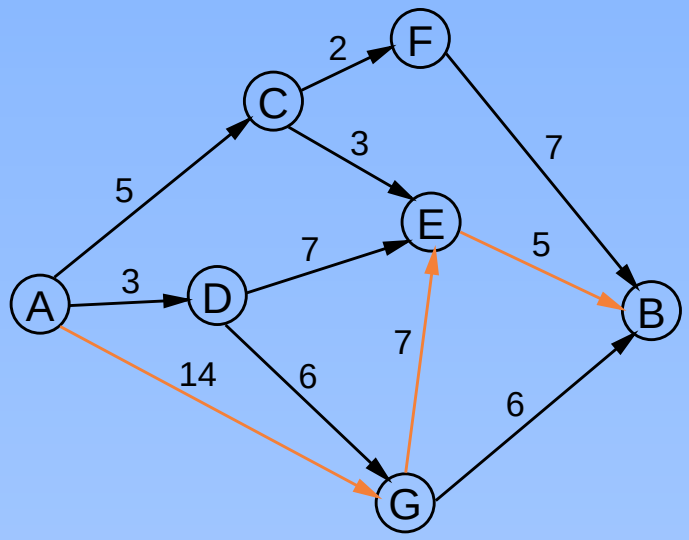


L(X) — odległość z X do B

5+L(C)		3+L(D)			14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)		20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	20+L(B)	26+L(B)
14	13	15	15	27	20	

Programowanie dynamiczne

Przykład
Problem znużonego wędrowcy



L(X) — odległość z X do B

5+L(C)		3+L(D)			14+L(G)	
7+L(F)	8+L(E)	10+L(E)	9+L(G)		20+L(B)	21+L(E)
14+L(B)	13+L(B)	15+L(B)	15+L(B)	22+L(E)	20+L(B)	26+L(B)
14	13	15	15	27	20	26

Metody konstruowania algorytmów

- bezpośrednie

Metody konstruowania algorytmów

- bezpośrednie
- pośrednie

Metody konstruowania algorytmów

- bezpośrednie
- pośrednie
  - ⋆ dekompozycji

Metody konstruowania algorytmów

- bezpośrednie
- pośrednie
  - ⋆ dekompozycji
    - ∗ przyrostowa

Metody konstruowania algorytmów

- bezpośrednie
- pośrednie
  - ⋆ dekompozycji
    - ∗ przyrostowa
    - ∗ dziel i zwyciężaj

Metody konstruowania algorytmów

- bezpośrednie
- pośrednie
  - ⋆ dekompozycji
    - ∗ przyrostowa
    - ∗ dziel i zwyciężaj
    - ∗ zachłanna

Metody konstruowania algorytmów

- bezpośrednie
- pośrednie
  - ⋆ dekompozycji
    - ∗ przyrostowa
    - ∗ dziel i zwyciężaj
    - ∗ zachłanna
    - ∗ programowanie dynamiczne

Metody konstruowania algorytmów

- bezpośrednie
- pośrednie
  - ⋆ dekompozycji
    - ∗ przyrostowa
    - ∗ dziel i zwyciężaj
    - ∗ zachłanna
    - ∗ programowanie dynamiczne

  - ⋆ szeregowa
  - ⋆ równoległa
  - ⋆ iteracyjna

Metody konstruowania algorytmów

- bezpośrednie
- pośrednie
  - ⋆ dekompozycji
    - ∗ przyrostowa
    - ∗ dziel i zwyciężaj
    - ∗ zachłanna
    - ∗ programowanie dynamiczne

    - ∗ szeregowa
    - ∗ równoległa
    - ∗ iteracyjna
  - ⋆ transformacji

Specyfikacja procedur i funkcji

Specyfikacja powinna zawierać komplet informacji, niezbędnych do napisania poprawnego wywołania procedury lub funkcji.

Specyfikacja procedur i funkcji

Specyfikacja powinna zawierać komplet informacji, niezbędnych do napisania poprawnego wywołania procedury lub funkcji. A wśród nich:

(a) co dana procedura lub funkcja robi, w tym dla funkcji również określenie wartości, którą wylicza,

Specyfikacja procedur i funkcji

Specyfikacja powinna zawierać komplet informacji, niezbędnych do napisania poprawnego wywołania procedury lub funkcji. A wśród nich:

(a) co dana procedura lub funkcja robi, w tym dla funkcji również określenie wartości, którą wylicza,

(b) znaczenie wszystkich parametrów,

Specyfikacja procedur i funkcji

Specyfikacja powinna zawierać komplet informacji, niezbędnych do napisania poprawnego wywołania procedury lub funkcji. A wśród nich:

(a) co dana procedura lub funkcja robi, w tym dla funkcji również określenie wartości, którą wylicza,

(b) znaczenie wszystkich parametrów,

(c) warunki, które muszą być spełnione, aby procedura/funkcja działała poprawnie, np. przedział dopuszczalnych wartości parametru,

Specyfikacja procedur i funkcji

Specyfikacja powinna zawierać komplet informacji, niezbędnych do napisania poprawnego wywołania procedury lub funkcji. A wśród nich:

(a) co dana procedura lub funkcja robi, w tym dla funkcji również określenie wartości, którą wylicza,

(b) znaczenie wszystkich parametrów,

(c) warunki, które muszą być spełnione, aby procedura/funkcja działała poprawnie, np. przedział dopuszczalnych wartości parametru, oraz

(d) opis trwałych zmian dokonywanych przez procedurę/funkcję, takich jak: zmiana wartości zmiennych globalnych(!), stanu plików zewnętrznych, informacje wyświetlane na ekranie, itp.

Zapis specyfikacji — warunki PRE i POST

PRE — określają wymagania, które muszą być spełnione, aby procedura lub funkcja działała poprawnie.

Zapis specyfikacji — warunki PRE i POST

PRE — określają wymagania, które muszą być spełnione, aby procedura lub funkcja działała poprawnie.

POST — opisują zmiany, które nastąpiły w wyniku zadziałania procedury lub funkcji.

Zapis specyfikacji — warunki PRE i POST

PRE — określają wymagania, które muszą być spełnione, aby procedura lub funkcja działała poprawnie.

POST — opisują zmiany, które nastąpiły w wyniku zadziałania procedury lub funkcji.

Przykładowo:

- PRE: program wymaga podania wartości w przedziale [0, 21]

Zapis specyfikacji — warunki PRE i POST

PRE — określają wymagania, które muszą być spełnione, aby procedura lub funkcja działała poprawnie.

POST — opisują zmiany, które nastąpiły w wyniku zadziałania procedury lub funkcji.

Przykładowo:

- PRE: program wymaga podania wartości w przedziale [0, 21]
- POST: program zwraca liczbę szczęśliwych dni do końca roku

Warunki PRE i POST

- PRE: wartość parametru N musi być zawarta w przedziale [0,100]

Warunki PRE i POST

- PRE: wartość parametru N musi być zawarta w przedziale [0,100]
- PRE: parametry A, B, C muszą spełniać warunek istnienia rozwiązań rzeczywistych równania kwadratowego: B B - 4 A C >= 0 oraz A <> 0

Warunki PRE i POST

- PRE: wartość parametru N musi być zawarta w przedziale [0,100]
- PRE: parametry A, B, C muszą spełniać warunek istnienia rozwiązań rzeczywistych równania kwadratowego: B B - 4 A C >= 0 oraz A <> 0
- PRE: parametr Oper musi mieć jedną z wartości: ”+”, ”-”, ”*”, ”/”

Warunki PRE i POST

- PRE: wartość parametru N musi być zawarta w przedziale [0,100]
- PRE: parametry A, B, C muszą spełniać warunek istnienia rozwiązań rzeczywistych równania kwadratowego: B B - 4 A C >= 0 oraz A <> 0
- PRE: parametr Oper musi mieć jedną z wartości: ”+”, ”-”, ”*”, ”/”
- PRE: stałe globalne MinWart i MaxWart muszą spełniać zależność MinWart <= MaxWart

Warunki PRE i POST

- PRE: wartość parametru N musi być zawarta w przedziale [0,100]
- PRE: parametry A, B, C muszą spełniać warunek istnienia rozwiązań rzeczywistych równania kwadratowego: B B - 4 A C >= 0 oraz A <> 0
- PRE: parametr Oper musi mieć jedną z wartości: ”+”, ”-”, ”*”, ”/”
- PRE: stałe globalne MinWart i MaxWart muszą spełniać zależność MinWart <= MaxWart
- PRE: zmienna plikowa f1 musi być związana z przygotowanym do zapisu plikiem zewnętrznym (wykonane REWRITE)

Warunki PRE i POST

- PRE: wartość parametru N musi być zawarta w przedziale [0,100]
- PRE: parametry A, B, C muszą spełniać warunek istnienia rozwiązań rzeczywistych równania kwadratowego: B B - 4 A C >= 0 oraz A <> 0
- PRE: parametr Oper musi mieć jedną z wartości: ”+”, ”-”, ”*”, ”/”
- PRE: stałe globalne MinWart i MaxWart muszą spełniać zależność MinWart <= MaxWart
- PRE: zmienna plikowa f1 musi być związana z przygotowanym do zapisu plikiem zewnętrznym (wykonane REWRITE)
- POST: parametry X1, X2 przyjmują wartości pierwiastków równania kwadratowego zadanego parametrami A, B, C

Warunki PRE i POST

- PRE: wartość parametru N musi być zawarta w przedziale [0,100]
- PRE: parametry A, B, C muszą spełniać warunek istnienia rozwiązań rzeczywistych równania kwadratowego: B B - 4 A C >= 0 oraz A <> 0
- PRE: parametr Oper musi mieć jedną z wartości: "+", "-", "*", "/"
- PRE: stałe globalne MinWart i MaxWart muszą spełniać zależność MinWart <= MaxWart
- PRE: zmienna plikowa f1 musi być związana z przygotowanym do zapisu plikiem zewnętrznym (wykonane REWRITE)
- POST: parametry X1, X2 przyjmują wartości pierwiastków równania kwadratowego zadanego parametrami A, B, C
- POST: wartością funkcji jest wpisana przez użytkownika z klawiatury liczba z przedziału [0,N] lub liczba -1, jeśli użytkownik wpisał na klawiaturze liczbę spoza tego przedziału

Warunki PRE i POST

- PRE: wartość parametru N musi być zawarta w przedziale [0,100]
- PRE: parametry A, B, C muszą spełniać warunek istnienia rozwiązań rzeczywistych równania kwadratowego: B B - 4 A C >= 0 oraz A <> 0
- PRE: parametr Oper musi mieć jedną z wartości: "+", "-", "*", "/"
- PRE: stałe globalne MinWart i MaxWart muszą spełniać zależność MinWart <= MaxWart
- PRE: zmienna plikowa f1 musi być związana z przygotowanym do zapisu plikiem zewnętrznym (wykonane REWRITE)
- POST: parametry X1, X2 przyjmują wartości pierwiastków równania kwadratowego zadanego parametrami A, B, C
- POST: wartością funkcji jest wpisana przez użytkownika z klawiatury liczba z przedziału [0,N] lub liczba -1, jeśli użytkownik wpisał na klawiaturze liczbę spoza tego przedziału
- POST: na pliku zewnętrznym związanym ze zmienną plikową PLIK została zapisana linia zawierająca wartości parametrów X1, X2, X3

Asercje

Dodatkowe instrukcje, sprawdzające spełnienie warunków PRE

Asercje

Dodatkowe instrukcje, sprawdzające spełnienie warunków PRE

```
PROCEDURE Pierwiastki(A,B,C: REAL; VAR X1,X2: REAL);
{ PRE: parametry A,B,C musza spelniac warunki
       istnienia rozwiazan rzeczywistych rownania
       kwadratowego: B*B-4*A*C>=0.0 oraz A<>0.0
 POST: parametry X1,X2 przyjmuja wartosci pierwiastkow
       rownania kwadratowego o wspolczynnikach A,B,C  }
VAR PierwDelta: REAL;
BEGIN
  IF (B*B-4.0*A*C<0.0) OR (A=0.0) THEN
    WRITELN('Blad asercji: procedura Pierwiastki')
  ELSE
  BEGIN
    PierwDelta := SQRT(B*B-4.0*A*C);
    X1 := (-B - PierwDelta) / (2.0*A);
    X2 := (-B + PierwDelta) / (2.0*A);
  END
END; (*Pierwiastki*)
```

Asercje

Dodatkowe instrukcje, sprawdzające spełnienie warunków PRE

```
PROCEDURE Pierwiastki(A,B,C: REAL; VAR X1,X2: REAL);
{ PRE: parametry A,B,C musza spelniac warunki
       istnienia rozwiazan rzeczywistych rownania
       kwadratowego: B*B-4*A*C>=0.0 oraz A<>0.0
 POST: parametry X1,X2 przyjmuja wartosci pierwiastkow
       rownania kwadratowego o wspolczynnikach A,B,C  }
VAR PierwDelta: REAL;
BEGIN
  IF (B*B-4.0*A*C<0.0) OR (A=0.0) THEN
    WRITELN('Blad asercji: procedura Pierwiastki')
  ELSE
  BEGIN
    PierwDelta := SQRT(B*B-4.0*A*C);
    X1 := (-B - PierwDelta) / (2.0*A);
    X2 := (-B + PierwDelta) / (2.0*A);
  END
END; (*Pierwiastki*)
```

Asercje

Dodatkowe instrukcje, sprawdzające spełnienie warunków PRE

```
PROCEDURE Pierwiastki(A,B,C: REAL; VAR X1,X2: REAL);
{ PRE: parametry A,B,C musza spelniac warunki
       istnienia rozwiazan rzeczywistych rownania
       kwadratowego: B*B-4*A*C>=0.0 oraz A<>0.0
 POST: parametry X1,X2 przyjmuja wartosci pierwiastkow
       rownania kwadratowego o wspolczynnikach A,B,C  }
VAR PierwDelta: REAL;
BEGIN
  IF (B*B-4.0*A*C<0.0) OR (A=0.0) THEN
    WRITELN('Blad asercji: procedura Pierwiastki')
  ELSE
  BEGIN
    PierwDelta := SQRT(B*B-4.0*A*C);
    X1 := (-B - PierwDelta) / (2.0*A);
    X2 := (-B + PierwDelta) / (2.0*A);
  END
END; (*Pierwiastki*)
```

Asercje — dalsze przykłady

```
PROCEDURE Assert(a: BOOLEAN; komunikat: String);
{ PRE: zadne
 POST: w przypadku niespelnionego warunku a wyswietla
       na ekranie komunikat i zatrzymuje program     }
BEGIN
  IF NOT a THEN
  BEGIN
    WRITELN('Blad asercji: ',komunikat);
    WRITELN('Program nie moze kontynuowac!');
    HALT; {procedura z biblioteki Unixowej}
  END
END; (*Assert*)
```

Asercje — dalsze przykłady

```
PROCEDURE Assert(a: BOOLEAN; komunikat: String);
{ PRE: zadne
 POST: w przypadku niespelnionego warunku a wyswietla
       na ekranie komunikat i zatrzymuje program     }
BEGIN
  IF NOT a THEN
  BEGIN
    WRITELN('Blad asercji: ',komunikat);
    WRITELN('Program nie moze kontynuowac!');
    HALT; {procedura z biblioteki Unixowej}
  END
END; (*Assert*)
```

Asercje — dalsze przykłady

```
PROCEDURE Assert(a: BOOLEAN; komunikat: String);
{ PRE: zadne
 POST: w przypadku niespelnionego warunku a wyswietla
       na ekranie komunikat i zatrzymuje program     }
BEGIN
  IF NOT a THEN
  BEGIN
    WRITELN('Blad asercji: ',komunikat);
    WRITELN('Program nie moze kontynuowac!');
    HALT; {procedura z biblioteki Unixowej}
  END
END; (*Assert*)
```

```
PROCEDURE Pierwiastki(A,B,C: REAL; VAR X1,X2: REAL);
{ PRE: parametry A,B,C musza spelniac warunki
       istnienia rozwiazan rzeczywistych rownania
       kwadratowego: B*B-4*A*C>=0.0 oraz A<>0.0
 POST: parametry X1,X2 przyjmuja wartosci pierwiastkow
       rownania kwadratowego o wspolczynnikach A,B,C  }
VAR PierwDelta: REAL;
BEGIN
  Assert((B*B-4.0*A*C>=0.0) AND (A<>0.0),
         'Pierwiastki: niepopr.wspolcz.rown.kwadrat.');
  PierwDelta := SQRT(B*B-4.0*A*C);
  X1 := (-B - PierwDelta) / (2.0*A);
  X2 := (-B + PierwDelta) / (2.0*A);
END; (*Pierwiastki*)
```

Iteracje a rekurencja

Struktury sterujące:

- bezpośrednie następstwo,

Iteracje a rekurencja

Struktury sterujące:

- bezpośrednie następstwo,
- wybór warunkowy,

Iteracje a rekurencja

Struktury sterujące:

- bezpośrednie następstwo,
- wybór warunkowy,
- iteracja ograniczona,

Iteracje a rekurencja

Struktury sterujące:

- bezpośrednie następstwo,
- wybór warunkowy,
- iteracja ograniczona,
- iteracja nieograniczona (warunkowa).

Iteracje a rekurencja

Struktury sterujące:

- bezpośrednie następstwo,
- wybór warunkowy,
- iteracja ograniczona,
- iteracja nieograniczona (warunkowa).

Inna metoda tworzenia powtórzeń — **rekurencja**.

Iteracje a rekurencja

Struktury sterujące:

- bezpośrednie następstwo,
- wybór warunkowy,
- iteracja ograniczona,
- iteracja nieograniczona (warunkowa).

Inna metoda tworzenia powtórzeń — **rekurencja**.

Rodzaje rekurencji (rekursji):

- Rekurencja bezpośrednia — zdolność podprogramu do wywołania samego siebie.

Iteracje a rekurencja

Struktury sterujące:

- bezpośrednie następstwo,
- wybór warunkowy,
- iteracja ograniczona,
- iteracja nieograniczona (warunkowa).

Inna metoda tworzenia powtórzeń — **rekurencja**.

Rodzaje rekurencji (rekursji):

- Rekurencja bezpośrednia — zdolność podprogramu do wywołania samego siebie.
- Rekurencja pośrednia — wzajemne wywoływanie się dwóch lub większej liczby podprogramów.

Rekurencja a iteracje

```
FUNCTION Silnia(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie }
{ wartosc silni.                }
{ UWAGA: dla n <= 0 zwraca 1    }
BEGIN
  IF n <= 1 THEN
    Silnia := 1
  ELSE
    Silnia := n * Silnia(n-1)
END;
```

Rekurencja a iteracje

```
FUNCTION Silnia(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie }
{ wartosc silni.                }
{ UWAGA: dla n <= 0 zwraca 1    }
BEGIN
  IF n <= 1 THEN
    Silnia := 1
  ELSE
    Silnia := n * Silnia(n-1)
END;
```

Rekurencja a iteracje

```
FUNCTION Silnia(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie }
{ wartosc silni.                }
{ UWAGA: dla n <= 0 zwraca 1    }
BEGIN
  IF n <= 1 THEN
    Silnia := 1
  ELSE
    Silnia := n * Silnia(n-1)
END;
```

```
FUNCTION Silnia(n: INTEGER): INTEGER;
{ Funkcja wylicza iteracyjnie }
{ wartosc silni.              }
{ UWAGA: dla n <= 0 zwraca 1  }
VAR wynik: INTEGER;
BEGIN
  wynik := 1;
  WHILE n > 1 DO
  BEGIN
    wynik := n * wynik;
    n := n - 1;
  END;
  Silnia := wynik
END;
```

Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

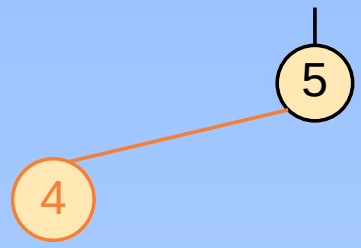
Diagram wywołań dla `n=5`:

Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

Diagram wywołań dla n=5:

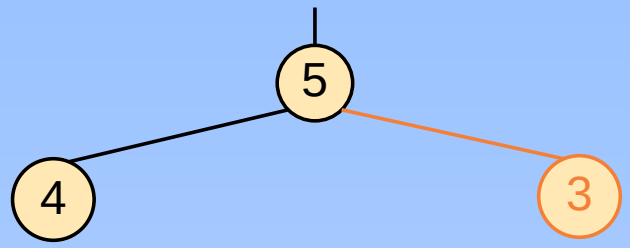


Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

Diagram wywołań dla `n=5`:

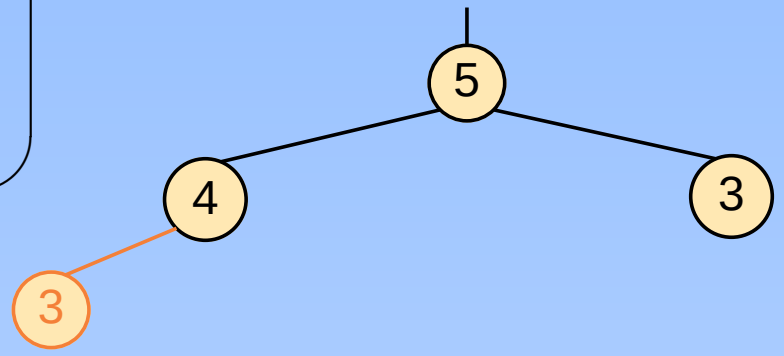


Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

Diagram wywołań dla n=5:

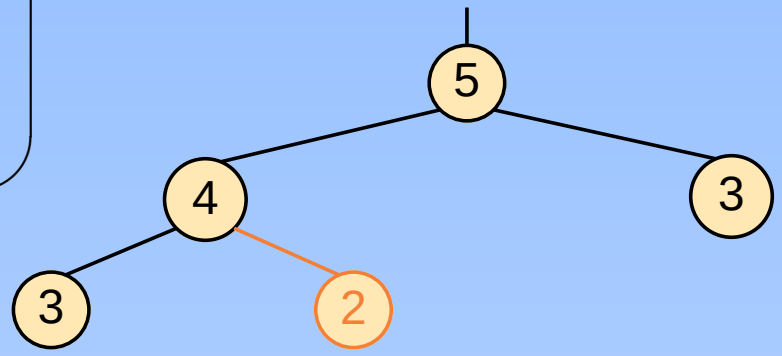


Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

Diagram wywołań dla n=5:

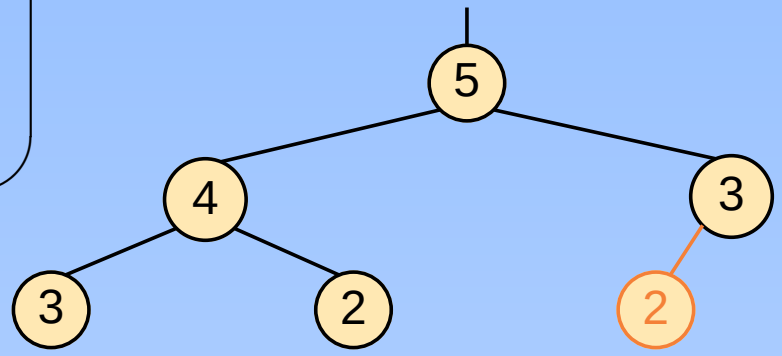


Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

Diagram wywołań dla n=5:

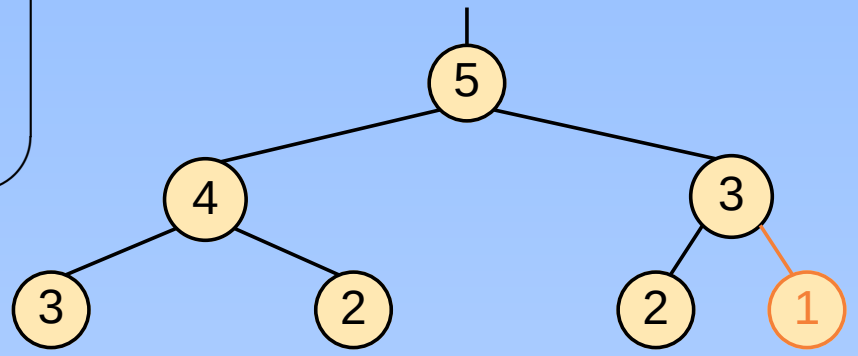


Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

Diagram wywołań dla n=5:

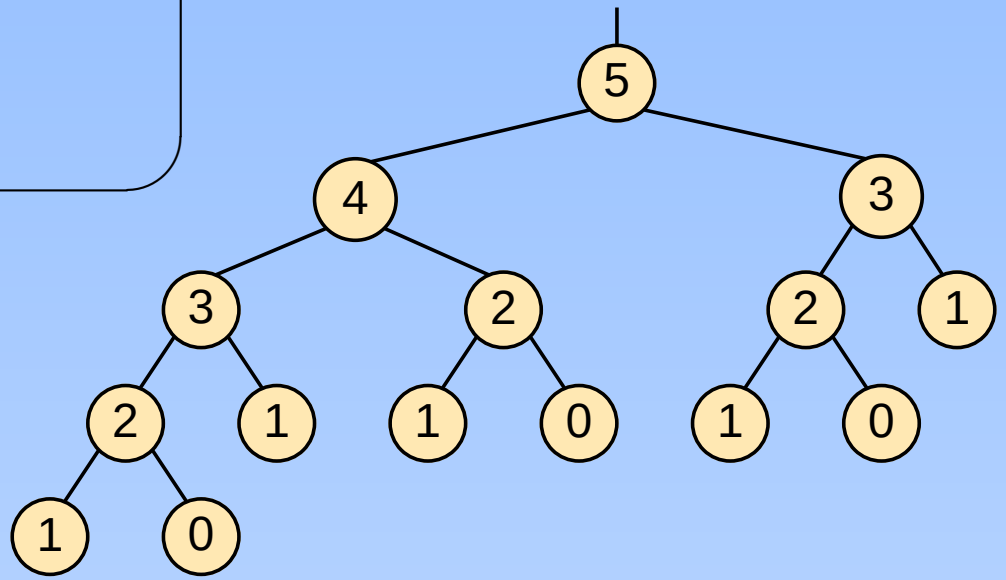


Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

Diagram wywołań dla `n=5`:

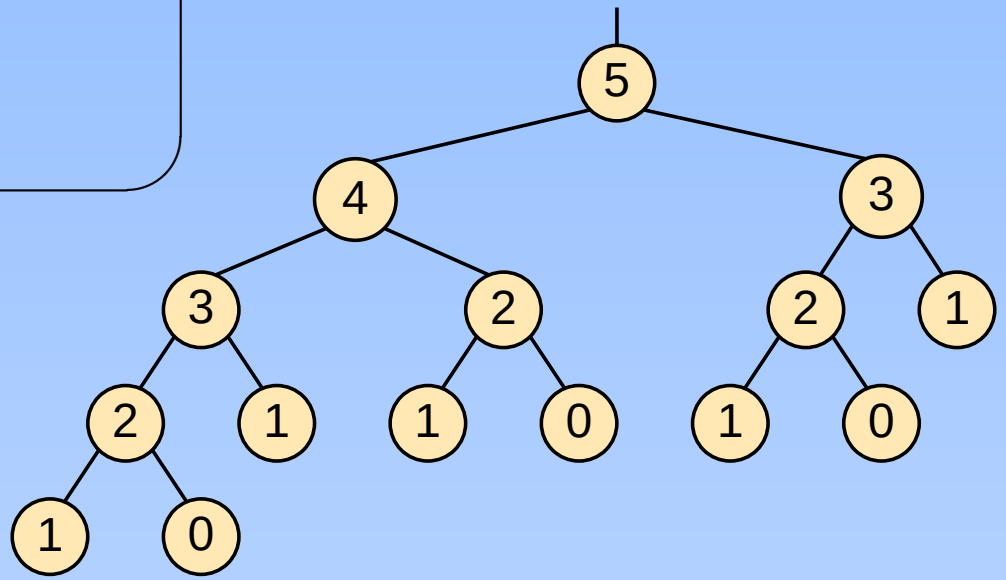


Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
FUNCTION Fib(n: INTEGER): INTEGER;
{ Funkcja wylicza rekurencyjnie liczby  }
{ Fibonacciego.  UWAGA: n musi byc >= 0 }
BEGIN
  IF n = 0 THEN Fib := 0 ELSE
  IF n = 1 THEN Fib := 1 ELSE
  Fib := Fib(n-1) + Fib(n-2)
END;
```

Diagram wywołań dla `n=5`:



- problem stopu

Indeks