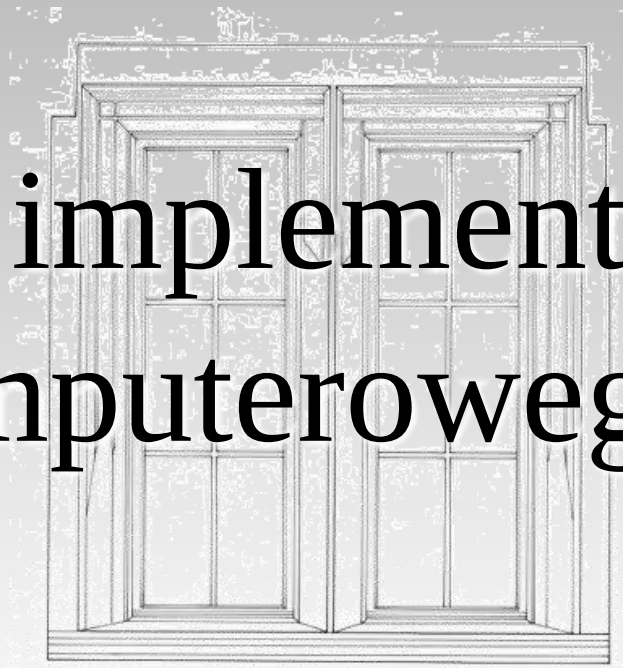


Projektowanie i implementacja programu komputerowego

ETAPY tworzenia programu komputerowego

1. Sformułowanie problemu i specyfikacja

   Poszukiwanie modelu, w którego terminach można formułować problem (graf, równanie różniczkowe).

2. Projekt rozwiązania

   Stworzenie algorytmu.

3. Implementacja

   Napisanie i uruchamianie programu na konkretnym sprzęcie i pod konkretnym systemem operacyjnym.

ETAPY tworzenia programu komputerowego c.d.

4. Testowanie i dokumentacja

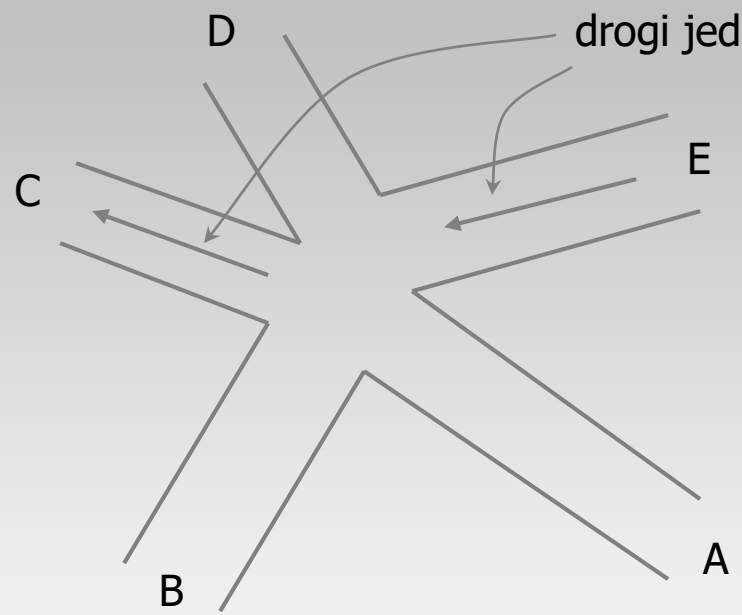
   Uruchamianie programu na różnych zestawach danych testowych i poprawianie błędów.

   Dokumentacje: projektowa, implementacyjna i użytkowa.

5. Ocena rozwiązania

   Ocena pod względem czasu wykonania i niezbędnych zasobów pamięciowych.

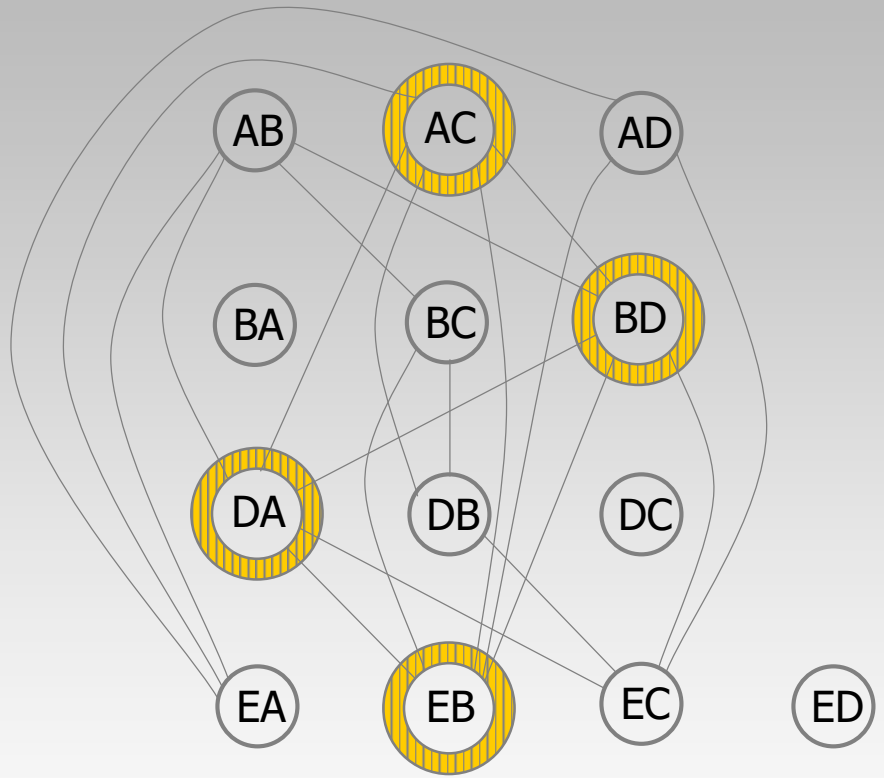
Problem świateł na skrzyżowaniu



Np. przejazdy AB i ED są dopuszczalne równocześnie natomiast AD i EC - nie

Należy zbiór przejazdów podzielić na klasy, wewnątrz których znajdują się przejazdy nie kolidujące ze sobą tak, żeby tych klas było jak najmniej.

Model problemu - graf



Łączymy krawędziami te przejazdy, które nie mogą się odbywać równocześnie

węzły tworzące 4-clique

Problem świateł na skrzyżowaniu sprowadza się do problemu kolorowania grafu.

Kolorowanie grafu polega na takim przyporządkowaniu kolorów wierzchołkom, aby każde dwa wierzchołki połączone krawędzią miały różne kolory

Problem świateł polega zatem na takim pokolorowaniu grafu przejazdów, aby liczba kolorów była najmniejsza z możliwych.

Niestety, tak postawiony problem jest w ogólnym przypadku NP – zupełny.

Zastosujemy zatem rozwiązanie, które nie gwarantuje optymalności, ale też jej nie wyklucza i daje „przyzwoite” rezultaty.

Algorytm „zachłanny” kolorowania grafu (greedy)

Kolorujemy jednym kolorem tyle wierzchołków grafu, ile się tylko da, potem drugim – ile się da pozostałych, etc.

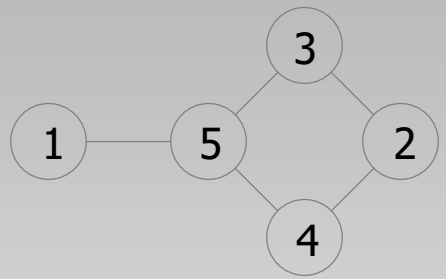
Aby pokolorować wierzchołki nowym kolorem postępujemy tak:

1. Wybierz jakiś nie pokolorowany wierzchołek i pokoloruj go nowym kolorem.

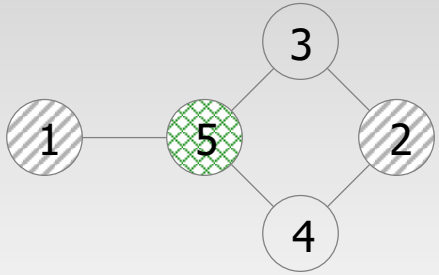
2. Przejdź przez listę nie pokolorowanych wierzchołków. Dla każdego nie pokolorowanego wierzchołka sprawdź, czy ma krawędź łączącą z wierzchołkiem już pokolorowanym nowym kolorem. Jeśli nie ma, to go pokoloruj nowym kolorem.

Algorytm zachłanny kolorowania grafu nie musi dać optymalnego rozwiązania.

Np. graf

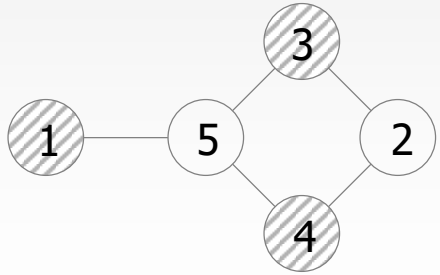


zostanie pokolorowany następująco



trzema kolorami

a mógłby być pokolorowany



dwoma kolorami

Rezultat działania algorytmu kolorowania na grafie przejazdów

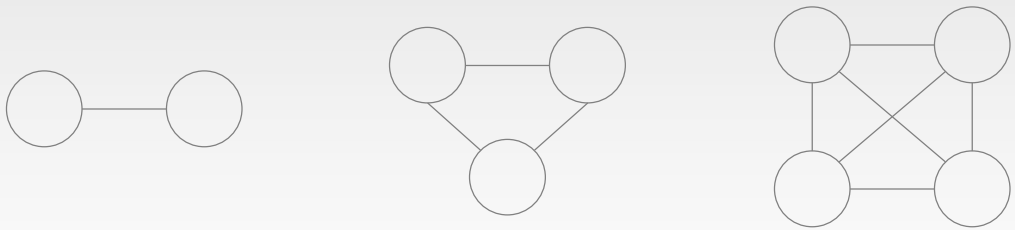
kolor	przejazdy	dodatkowo
niebieski	AB AC AD BA DC ED	
czerwony	BC BD EA	BA DC ED
zielony	DA DB	AD BA DC ED
żółty	EB EC	BA DC EA ED

Zatem, system sygnalizacji świetlnej powinien działać w czterech fazach.

Czy mógłby w mniejszej liczbie faz?

Ocena optymalności

K - klika (k – clique) w grafie, jest to zbiór złożony z k wierzchołków, które są połączone każdy z każdym krawędziami:



2 – clique 3 – clique 4 - clique

Jest oczywistym, że do pokolorowania grafu, który zawiera k – klikę potrzeba co najmniej k kolorów.

W grafie przejazdów AC, DA, BC, EB stanowią 4 – klikę, zatem otrzymane rozwiązanie jest optymalne.

Od projektu rozwiązania do implementacji

STEPWISE REFINEMENT – metoda kolejnych uściśleń

Stosując tę metodę pokażemy (w przybliżeniu) przejście od ogólnego zarysu algorytmu do jego konkretnej implementacji.

Oznaczmy „newclr” – zbiór wierzchołków, które mogą być pokolorowane nowym kolorem

Procedury i funkcje

Deklaracja procedury

```
const  n=100;
type wektor = array [1..n] of integer;
var   w,v: wektor;  il_skalarny: integer;
procedure  Iloczyn_skalarny (a:wektor,  b:wektor; var  is: integer);
var i: integer;
begin
   is:=0;
   for i:=1 to n do is:= is + a[i]*b[i]
end; {Iloczyn skalarny}
```

Parametry przekazywane przez wartość

Parametr przekazywany przez zmienną

Parametry formalne procedury

Deklaracja procedury

Procedury i funkcje

Wywołanie procedury

```
Begin {main}
  il_skalarny:=0;
  for i:=1 to n do read(w[i]);
  for i:=1 to n do read(v[i]);
  Iloczyn_skalarny(w, v, il_skalarny);
  writeln (' Iloczyn skalarny wektorów w i v = ' , il_skalarny)
end. {main}
```

⟵ wywołanie procedury

parametry aktualne (parametry wywołania procedury)

Procedury i funkcje

Deklaracja funkcji

```
const  n=100;
type wektor = array [1..n] of integer;
var   w,v: wektor;  il_skalarny: integer;
function  Iloczyn_skalarny (a:wektor,  b:wektor): integer;
var i: integer;
begin
  is:=0;
  for i:=1 to n do is:= is + a[i]*b[i];
  Iloczyn_skalarny:= is
end; {Iloczyn skalarny}
```

Parametry przekazywane przez wartość

Typ wartości funkcji

Parametry formalne funkcji

Nazwa funkcji, która przekazuje wartość funkcji

Deklaracja funkcji

Procedury i funkcje

Wywołanie funkcji

```
Begin {main}
  il_skalarny:=0;
  for i:=1 to n do read(w[i]);
  for i:=1 to n do read(v[i]);
  il_skalarny:=Iloczyn_skalarny(w, v);
  writeln (' Iloczyn skalarny wektorów w i v = ' , il_skalarny)
end. {main}
```

⟵ wywołanie funkcji

parametry aktualne (parametry wywołania funkcji)

Poniższa procedura będzie wykonywana, aż do momentu, gdy wszystkie wierzchołki grafu zostaną pokolorowane:

```
procedure greedy (var G: GRAPH; var newclr: SET);
{„greedy” umieszcza w newclr te wierzchołki z G, które
mogą mieć ten sam kolor }
begin
(1) newclr := 0;
(2) for każdy nie pokolorowany, wierchołek v w G do
(3)    if v nie jest związany w G krawędzią z żadnym
        wierzchołkiem z newclr then
                        begin
(4)                       zaznacz, że v jest pokolorowany;
(5)                       umieść v w newclr
                        end
end; {greedy}
```

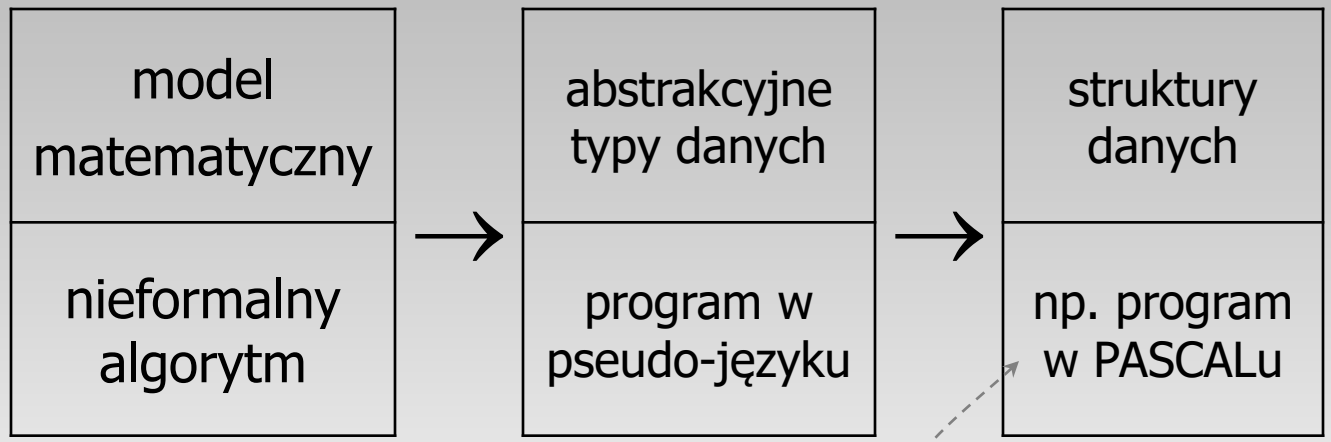
Następne uściślenie:

```
procedure greedy (var G : GRAPH, var newclr : SET);
begin
(1)   newclr := 0;
(2)   for każdy nie pokolorowany wierzchołek v w G do
      begin
(3.1)   found := false;
(3.2)   for każdy wierzchołek w w newclr do
(3.3)     if jest krawędź między w a v w G then
(3.4)       found := true;
(3.5)   if found = false then
        begin{v nie jest związany z żadnym wierzchołkiem w newclr}
(4)       zaznacz, że v jest pokolorowany;
(5)       umieść v w newclr
        end
      end
end; {greedy}
```

Następne uściślenie

```
procedure greedy (var G : GRAPH; var newclr : LIST);
var found : boolean; v, w : integer;
begin
  newclr := 0;  v := pierwszy nie pokolorowany wierzchołek w G;
  while v <> null do
  begin
    found := false;    w := pierwszy wierzchołek w newclr;
    while w <> null do
    begin
      if istnieje krawędź między v a w w G then found := true;
      w := następny wierzchołek w newclr;
    end;
    if found = false then
    begin
      zaznacz, że v jest pokolorowany; umieść v w newclr;
    end;
    v := następny nie pokolorowany wierzchołek w G
  end;
end; {greedy}
```

Proces rozwiązywania problemu



program w konkretnym języku programowania uruchomiony na konkretnym komputerze (implementacja rozwiązania)

Praktyczne uwagi o dobrym programowaniu

1. Przy tworzeniu programu postępuj planowo – przedstawiona wcześniej metoda polegająca na pierwotnym stworzeniu zarysu, szkicu algorytmu, utworzeniu na jego podstawie pseudo - programu, a następnie uściślanie pseudo - programu aż stanie się wykonywalnym na komputerze kodem.

2. Zamykaj kod obsługujący każdą podstawową operację i typy danych, na których jest ona wykonywana, w jednym miejscu programu – używając procedur i abstrakcyjnych typów danych. Wtedy wszelkie niezbędne zmiany będą dokonywane tylko w jednym miejscu.

Praktyczne uwagi o dobrym programowaniu

3. Używaj istniejących programów i modyfikuj je. Główną nieefektywnością wielu projektów jest zachowywanie się jakby nigdy dotąd żaden program nie był napisany. Najpierw należy rozejrzeć się za programem, który rozwiązałby całość lub część naszego problemu.

Praktyczne uwagi o dobrym programowaniu.

4. Twórz narzędzia. W języku programistów „narzędzie” jest programem o różnorodnych zastosowaniach. Podczas pisania programu warto się zastanowić, czy nie mógłby on być napisany w nieco ogólniejszy sposób nieco większym wysiłkiem. Na przykład, ktoś dostał za zadanie napisanie programu do obsługi egzaminów końcowych, to powinien napisać program kolorujący wierzchołki grafu możliwie najmniejszą liczbą kolorów. W kontekście egzaminów wierzchołki będą grupami studentów, kolory terminami egzaminów a krawędzie będą oznaczać, że grupy mają wspólnych studentów. Program kolorujący, włącznie z procedurami tłumaczącymi listy grup na grafy a kolory na odpowiednie dni i godziny da nam właściwy harmonogram egzaminów. Ponadto można go użyć do rozwiązania znanego problemu świateł na skrzyżowaniu i innych pokrewnych.

Praktyczne uwagi o dobrym programowaniu

5. Wykorzystuj możliwości (komendy) systemu operacyjnego.
   W połączeniu z istniejącymi programami można (używając makrodefinicji) napisać program, który nie zawiera żadnych komend języka programowania.