Przeszukiwanie z nawrotami

- sformułowanie problemu
- przegląd drzewa poszukiwań
- przykłady problemów
- wybrane narzędzia programistyczne

Przeszukiwanie z nawrotami

Sformułowanie problemu

- Dany zbiór wartości D nazywany dziedziną.
- Niech $\Omega = D^n$.
- Na zbiorze Ω określona jest funkcja $C : \Omega \mapsto \{true, false\}$ wyznaczająca zbiór rozwiązań dopuszczalnych $S \subseteq \Omega$:

$$S = \{\langle x_1, x_2, \ldots, x_n\rangle \in \Omega : C(x_1, x_2, \ldots, x_n)\}.$$

- Warunek logiczny C, wyznaczający zbiór rozwiązań dopuszczalnych, nazywać będziemy *ograniczeniami*.

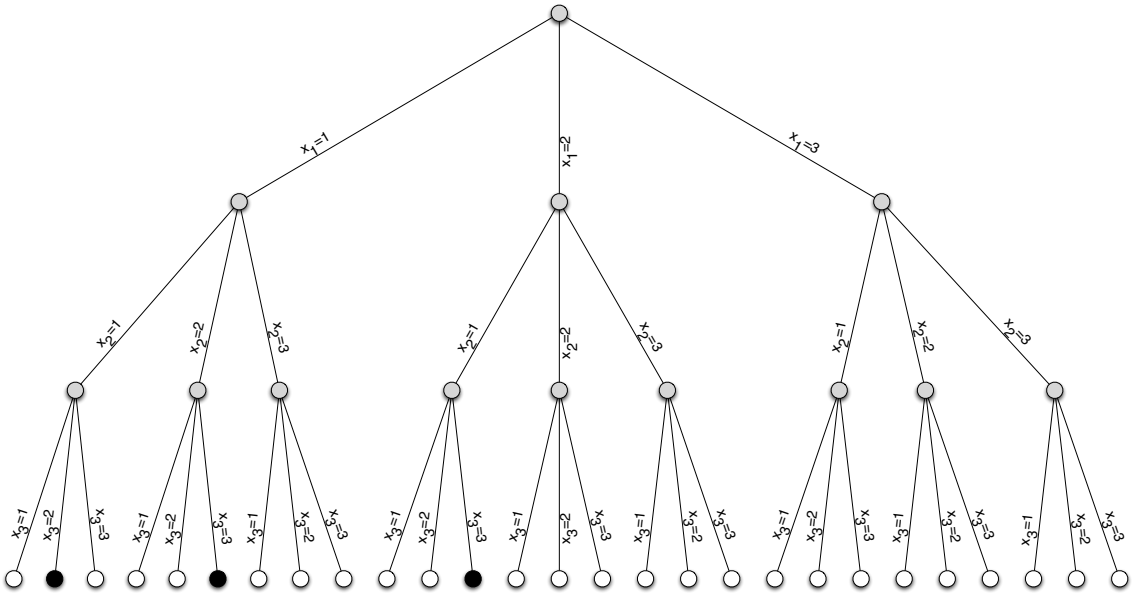
Problem (Problem spełnienia ograniczeń)

Interesuje nas znalezienie elementów zbioru S (wszystkich albo co najmniej jednego).

Przeszukiwanie z nawrotami

Przegląd drzewa poszukiwań

Przykład dla $n = 3$, $D = \{1, 2, 3\}$ i ograniczenia $x_1 + x_2 = x_3$:



Kolorem czarnym zaznaczono liście odpowiadające zbiorowi rozwiązań:

$$S = \{\langle x_1, x_2, x_3\rangle \in D^3 : x_1 + x_2 = x_3\}.$$

Przeszukiwanie z nawrotami

Przegląd drzewa poszukiwań

Poniższy program dokonuje przeglądu wszystkich dopuszczalnych rozwiązań dla przykładu z poprzedniego slajdu.

```
1: for X[1] ← 1, 3 do
2:     for X[2] ← 1, 3 do
3:         for X[3] ← 1, 3 do
4:             if X[1] + X[2] = X[3] then
5:                 write X[1], X[2], X[3]
6:             end if
7:         end for
8:     end for
9: end for
```

Przeszukiwanie z nawrotami

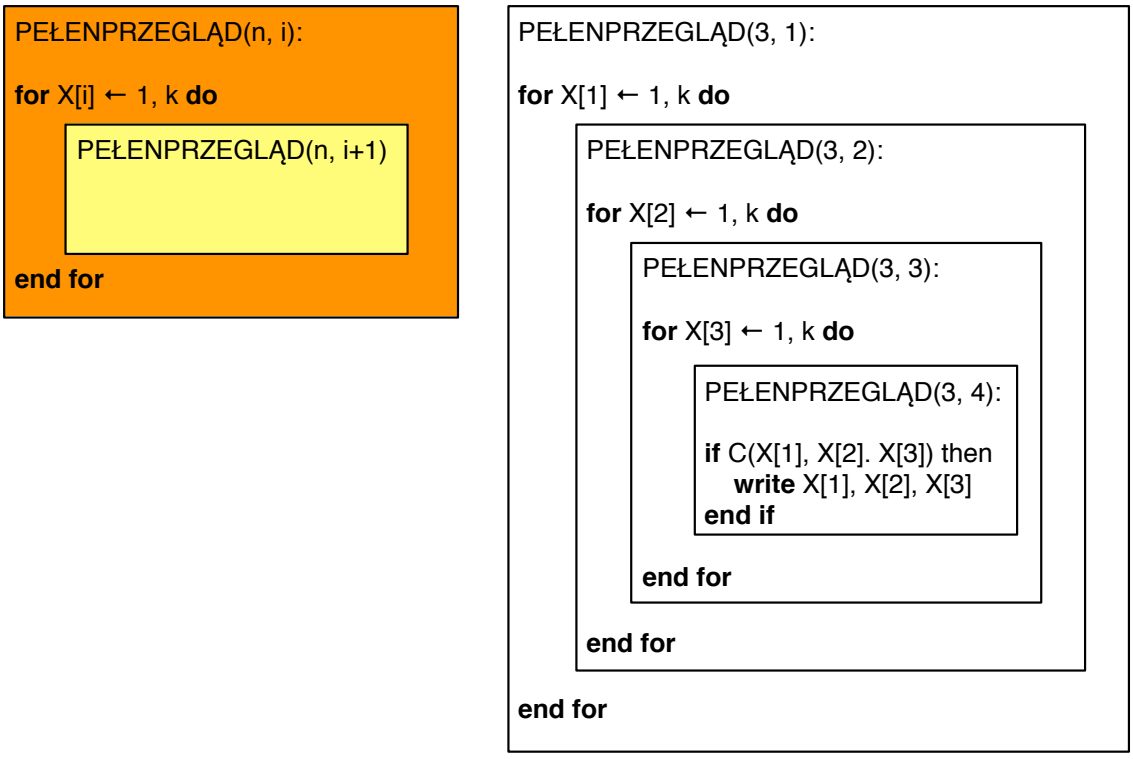
Przegląd drzewa poszukiwań

W przypadku dowolnej liczby zmiennych n i dziedziny k-elementowej $D = \{1, 2, \ldots, k\}$ potrzebna jest procedura dokonująca przeglądu z nawrotami (zmienna j jest lokalna w tej procedurze):

```
1:  procedure PEŁENPRZEGLĄD(n, i) ▷ ustalenie wartości X[i], ..., X[n]
2:      if i = n + 1 then
3:          if C(X[1], X[2], ..., X[n]) then
4:              drukuj wartości X[1], X[2], ..., X[n]
5:          end if
6:      else
7:          for X[i] ← 1, k do
8:              PEŁENPRZEGLĄD(n, i + 1)
9:          end for
10:     end if
11: end procedure
```


Przeszukiwanie z nawrotami

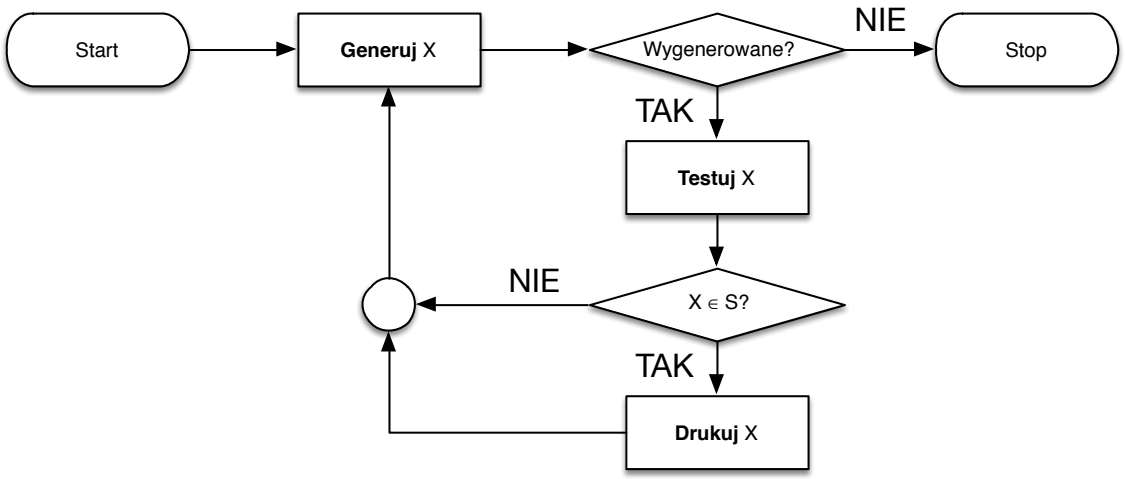
Przegląd drzewa poszukiwań



Przeszukiwanie z nawrotami

Przegląd drzewa poszukiwań

Procedura PEŁENPRZEGLĄD pracuje bardzo nieefektywnie bo najpierw generuje cały układ wartości $X[1], X[2], \ldots, X[n]$ i dopiero na koniec sprawdza czy spełnia on ograniczenia C (takie przeszukiwanie nazywa się *generowaniem i testowaniem*).



Przeszukiwanie z nawrotami

Przegląd drzewa poszukiwań

- Niech warunek MOŻLIWE$(X[1], X[2], \ldots, X[i])$, dla $i \leq n$, jest spełniony gdy istnieją takie wartości $X[i+1], X[i+2], \ldots, X[n]$, że $C(X[1], X[2], \ldots, X[i], \ldots, X[n])$.
- Warunek MOŻLIWE można zinterpretować w ten sposób, że ocenia on węzeł drzewa poszukiwań i przewiduje, czy w poddrzewie zakorzenionym w ocenianym węźle znajduje się liść odpowiadający rozwiązaniu dopuszczalnemu (elementowi ze zbioru S).
- Ponieważ najczęściej aby być pewnym odpowiedzi z warunku MOŻLIWE musiałby on przejrzeć całe rozpatrywane poddrzewo, więc będziemy zakładać, że warunek MOŻLIWE udziela bez pełnego przeglądu jedynie **niepewnej** odpowiedzi tj.
  1. jeśli warunek nie jest spełniony, to w analizowanym poddrzewie **na pewno** nie ma rozwiązania dopuszczalnego;
  2. jeśli warunek jest spełniony, to w analizowanym poddrzewie **może** jest rozwiązanie dopuszczalne.

Przeszukiwanie z nawrotami

Przegląd drzewa poszukiwań

Poniższa procedura dokonuje częściowego przeglądu, co może w wielu sytuacjach bardzo przyspieszyć poszukiwania:

```
1:  procedure CZĘŚCIOWYPRZEGLĄD(n, i)
2:      if i = n + 1 then
3:          drukuj wartości X[1], X[2], ..., X[n]
4:      else
5:          for X[i] ← 1, k do
6:              if MOŻLIWE(X[1], X[2], ..., X[i]) then
7:                  CZĘŚCIOWYPRZEGLĄD(n, i + 1)
8:              end if
9:          end for
10:     end if
11: end procedure
```


Przeszukiwanie z nawrotami

Przegląd drzewa poszukiwań

- Procedura CZĘŚCIOWYPRZEGLĄD działa przy dodatkowym założeniu, że warunek

  $$\text{MOŻLIWE}(X[1], X[2], \ldots, X[n])$$

  poprawnie sprawdza spełnienie ograniczeń C przy ustalonych wszystkich n wartościach.
- Często łatwiej rozstrzygnąć warunek $\text{NIEMOŻLIWE}(X[1], X[2], \ldots, X[i])$, który zachodzi gdy nie można rozszerzyć wartości $X[1], X[2], \ldots, X[i]$ do n wartości spełniających ograniczenia C.
- Oczywiście taka odpowiedź od warunku NIEMOLIWE również nie jest pewna tj.
  - jeśli warunek nie jest spełniony, to w analizowanym poddrzewie **może** jest rozwiązanie dopuszczalne;
  - jeśli warunek jest spełniony, to w analizowanym poddrzewie **na pewno** nie ma rozwiązania dopuszczalnego.

Przeszukiwanie z nawrotami

Przegląd drzewa poszukiwań

- Jeśli potrafimy sformułować warunek NIEMOŻLIWE, to warunek

$$\text{MOŻLIWE}(X[1], X[2], \ldots, X[i]),$$

z wiersza 6 w procedurze CZĘŚCIOWYPRZEGLĄD, można zastąpić warunkiem:

$$\neg\text{NIEMOŻLIWE}(X[1], X[2], \ldots, X[i]).$$

Przeszukiwanie z nawrotami

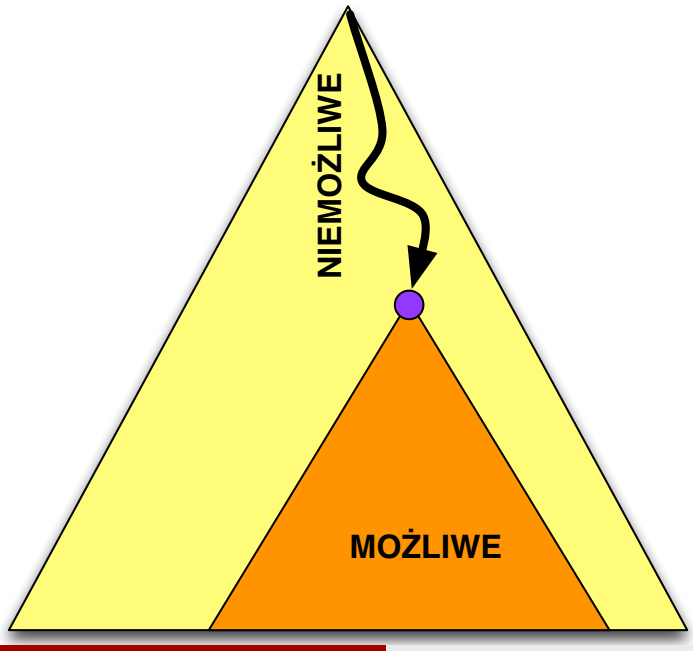
Przegląd drzewa poszukiwań

- MOŻLIWE – stara się przewidzieć czy w analizowanym poddrzewie jest jakieś rozwiązanie dopuszczalne;
- NIEMOŻLIWE – analizując dotychczas ustalone wartości $X[1], X[2], \ldots, X[i]$ sprawdza czy nie naruszają one ograniczeń i uniemożliwiają znalezienie w rozpatrywanym poddrzewie rozwiązania dopuszczalnego.

Przeszukiwanie z nawrotami

Przegląd drzewa poszukiwań

Warunek MOŻLIWE "przewiduje" przyszłość, natomiast warunek NIEMOŻLIWE analizuje przeszłość:



Przeszukiwanie z nawrotami

Przykłady problemów: ciąg bitów

Poniższa procedura drukuje wszystkie n-elementowe ciągi zero-jedynkowe, zatem nie ma potrzeby sprawdzania warunku MOŻLIWE ani NIEMOŻLIWE.

```
1:  procedure PODZBIÓR(n, i)                    ▷ n-elementowe ciągi bitów
2:      if i = n + 1 then
3:          drukuj wartości X[1], X[2], ..., X[n]
4:      else
5:          X[i] ← 0;                           ▷ gdy nie biorę i-tego elementu
6:          PODZBIÓR(n, i + 1);
7:          X[i] ← 1;                           ▷ gdy biorę i-ty element
8:          PODZBIÓR(n, i + 1)
9:      end if
10: end procedure
```

Przeszukiwanie z nawrotami

Przykłady problemów: ciąg bitów

Example

Po wywołaniu PODZBIÓR(3, 1) drukowane są następujące wartości:

000
001
010
011
100
101
110
111

Przeszukiwanie z nawrotami

Przykłady problemów: kombinacje bez powtórzeń

```
1:  procedure KOMBINACJE(n, k, i)                    ▷ wybór k spośród n
2:      if i = k + 1 then
3:          drukuj wartości X[1], X[2], ..., X[k]
4:      else
5:          if i = 1 then                  ▷ wyliczenie wartości początkowej dla X[i]
6:              start ← 1
7:          else
8:              start ← X[i − 1] + 1
9:          end if
10:         for X[i] ← start, n do
11:             if k − i ≤ n − X[i] then          ▷ MOŻLIWE ≡ k − i ≤ n − X[i]
12:                 KOMBINACJE(n, k, i + 1)
13:             end if
14:         end for
15:     end if
16: end procedure
```

Przeszukiwanie z nawrotami

Przykłady problemów: kombinacje bez powtórzeń

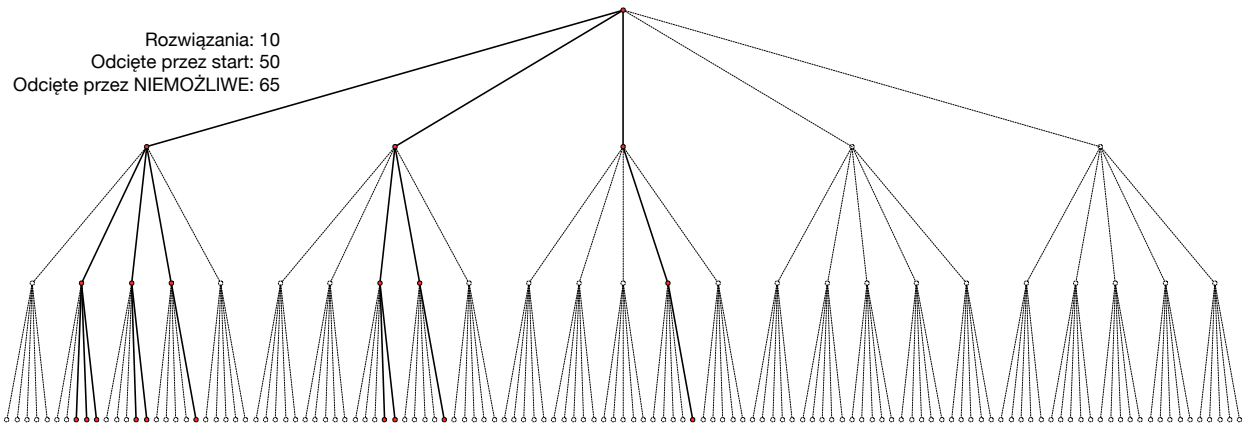
Example

Po wywołaniu KOMBINACJE(5, 3, 1) drukowane są następujące wartości:

1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5

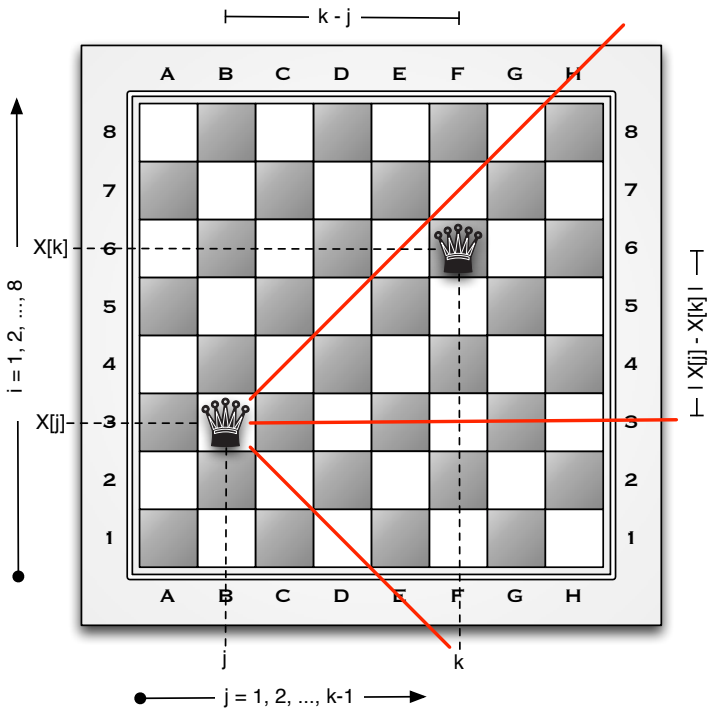
Przeszukiwanie z nawrotami

Przykłady problemów: kombinacje bez powtórzeń



Przeszukiwanie z nawrotami

Przykłady problemów: problem hetmanów



Przeszukiwanie z nawrotami

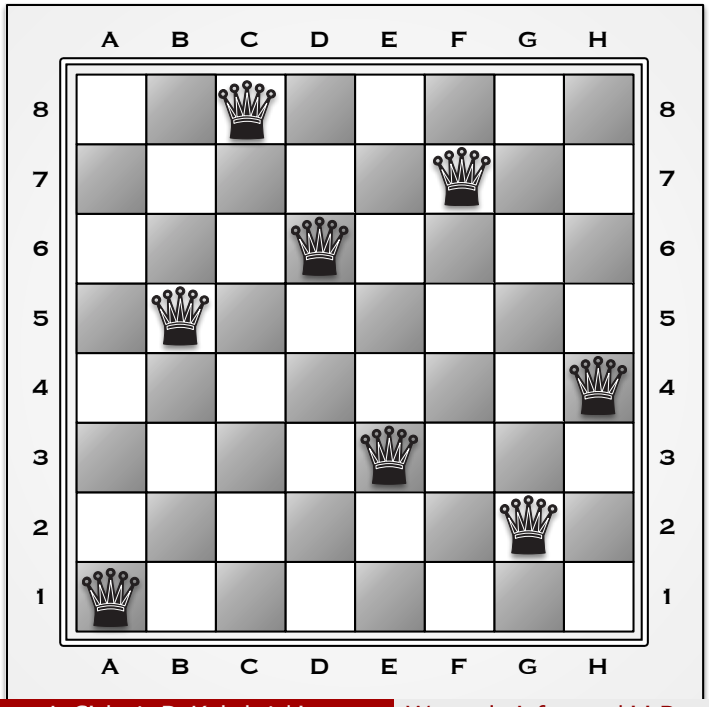
Przykłady problemów: problem hetmanów

```
1:  procedure HETMANY(k)          ▷ dostawienie hetmanów w kolumnach k..8
2:      if k = 9 then
3:          drukuj rozwiązanie X[1], X[2], ..., X[8]
4:      else
5:          for X[k] ← 1, 8 do                    ▷ wybierz kolejny wiersz
6:              OK ← true;
7:              for j ← 1, k − 1 do
8:                  OK ← OK ∧ X[j] ≠ X[k] ∧ |X[j] − X[k]| ≠ k − j
9:              end for
10:             if OK then
11:                 HETMANY(k + 1)
12:             end if
13:         end for
14:     end if
15: end procedure
```

Przeszukiwanie z nawrotami

Przykłady problemów: problem hetmanów

Pierwsze z 92 rozwiązań znalezionych procedurą HETMANY:



Przeszukiwanie z nawrotami

Przykłady problemów: problem hetmanów

Fragment kodu w wierszach 6-9 odpowiada sprawdzeniu warunku NIEMOŻLIWE. Jeśli po wyjściu z pętli **for-do** wartość zmiennej *OK* jest równa *false*, to znaczy, że hetman stawiany na przecięciu kolumny k-tej i wiersza i-tego jest bity przez któregoś z wcześniej postawionych hetmanów w kolumnach od 1 do $k-1$.

Ćwiczenie

Przekształć pętlę w wierszach 7-9 procedury HETMANY aby kończyła się zaraz po wykryciu hetmana bijącego pozycję na przecięciu k-tej kolumny z i-tym wierszem (zmienna *OK* powinna przyjąć wtedy wartość false).

Przeszukiwanie z nawrotami

Wybrane narzędzia programistyczne

- Dla efektywnego rozwiązywania problemów sformułowanych w postaci problemu spełnienia ograniczeń opracowano specjalną metodologię programowania nazywaną *technologią więzów*[3].
- Technologię więzów opracowano początkowo dla języka programowania Prolog[4] ale dostępna jest ona dla innych języków programowania jak np. Java czy C++.

[3] Na studiach drugiego stopnia jest możliwość wyboru kursu **Technologia więzów**.

[4] Na studiach pierwszego stopnia istnieje możliwość wyboru kursu **Programowanie w logice**, na którym wykładany jest język programowania Prolog, natomiast na studiach drugiego stopnia jest możliwość wyboru kursu **Metody programowania w logice**, na którym omawiane są teoretyczne podstawy programowania w logice.

Przeszukiwanie z nawrotami

Wybrane narzędzia programistyczne

Tabela: Wybrane narzędzia dostarczające technologię więzów

Narzędzie	Język programowania
IBM ILOG Solver	C++
JaCoP	Java
B-Prolog	Prolog
Ciao Prolog	Prolog
ECLiPSe	Prolog
GNU-Prolog	Prolog
IF/Prolog	Prolog
SICStus Prolog	Prolog
SWI-Prolog	Prolog
Yap	Prolog
Emma	Python
python-constraint	Python

Przeszukiwanie z nawrotami

Wybrane narzędzia programistyczne

Example

Prolog jest językiem deklaratywnym, zatem opisuje się w nim problem bez podawania sposobu (algorytmu) rozwiązywania. Rozpatrzmy problem generowania wszystkich permutacji liczb od 1 do n. Poniższy predykat `perm(N, X)` opisuje w GNU-Prologu ten problem:

```
perm(N, X) :- length(X, N),
              fd_domain(X, 1, N),
              fd_all_different(X),
              fd_labeling(X).
```


Przeszukiwanie z nawrotami

Wybrane narzędzia programistyczne

Example (cd.)

Predykat `perm(N, X)` wyraża warunki jakie powinny być spełnione aby `X` było permutacją liczb od 1 do `N`:

1. `X` powinno być listą długości `N`.
2. Elementy listy `X` powinny być liczbami z zakresu od 1 do `N`.
3. Elementy listy `X` powinny być parami różne.
4. Należy podstawić za elementy `X` takie wartości aby spełniały powyższe trzy warunki.

Przeszukiwanie z nawrotami

Wybrane narzędzia programistyczne

Example (cd.)

Dostępny w GNU-Prologu predykat `fd_labeling(X)` dokonuje przeglądu drzewa poszukiwań, przy czym jest on wykonywany w bardzo wyrafinowany sposób z uwzględnieniem warunku MOŻLIWE (na podstawie warunków 1, 2 i 3) oraz warunku NIEMOŻLIWE (na podstawie warunku 3). To jak system GNU-Prolog uwzględnia je podczas przeszukiwania stanowi kwintesencję technologii więzów. Przykład odpowiedzi udzielonych przez GNU-Prolog na pytanie `perm(3, X)`:

```
| ?- perm(3, X).
X = [1,2,3] ? ;
X = [1,3,2] ? ;
X = [2,1,3] ? ;
X = [2,3,1] ? ;
X = [3,1,2] ? ;
X = [3,2,1]
```