Budowa algorytmów - przegląd z nawrotami

Wstęp do Informatyki i Programowania

Maciek Gębala

19 grudnia 2024

Notatki

Sformułowanie problemu

Mamy dany zbiór wartości D nazywany dziedziną.

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

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

Problem spełnienia ograniczeń

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

Notatki

Przeszukiwanie z nawrotami

W przypadku dowolnej liczby zmiennych n i dziedziny k-elementowej $D = \{1, \ldots, k\}$ potrzebna jest procedura dokonująca rekurencyjnie przeglądu z nawrotami.

procedure PEŁENPRZEGLĄD(n, i)
 ▷ ustalenie wartości $X_i, \ldots, X_n$
  if $i = n + 1$ **then**
    if $C(X_1, \ldots, X_n)$ **then**
      drukuj rozwiązanie $X_1, \ldots, X_n$
    end if
  else
    for $X_i \in \{1, \ldots, k\}$ **do**
      PEŁENPRZEGLĄD(n, $i + 1$)
    end for
  end if
end procedure
PEŁENPRZEGLĄD(n,1) ▷ początkowe wywołanie

Notatki

Przeszukiwanie z nawrotami

Procedura PEŁENPRZEGLĄD pracuje bardzo nieefektywnie, bo najpierw generuje cały układ wartości $X_1, \ldots, X_n$ a dopiero na koniec sprawdza, czy spełnia on ograniczenia C (takie przeszukiwanie nazywa się generowaniem i testowaniem).

W takim przeszukiwaniu można zmienić sposób generowania np. na nierekurencyjny.

Notatki

Ulepszenia

Niech warunek MOŻLIWE$(X_1, \ldots, X_i)$, dla $i \leqslant n$, jest spełniony, gdy istnieją takie wartości $X_{i+1}, \ldots, X_n$, że zachodzi $C(X_1, \ldots, X_i, \ldots, X_n)$.

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.

Notatki

Ulepszenia

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:          if C(X_1, ..., X_n) then
4:              drukuj rozwiązanie X_1, ..., X_n
5:          end if
6:      else
7:          for X_i ∈ {1, ..., k} do
8:              if Możliwe(X_1, ..., X_i) then
9:                  CzęściowyPrzegląd(n, i + 1)
10:             end if
11:         end for
12:     end if
13: end procedure
14: CzęściowyPrzegląd(n,1)                ▷ początkowe wywołanie
```


Notatki

Ulepszenia

Często łatwiej rozstrzygnąć warunek NIEMOŻLIWE$(X_1, \ldots, X_i)$, który zachodzi gdy nie można rozszerzyć wartości pierwszych i wartości do n wartości spełniających ograniczenia C. Wtedy w linii 8 użyjemy warunku $\neg$NIEMOŻLIWE.

Gdy warunek MOŻLIWE zawsze prowadzi do rozwiązania dopuszczalnego, lub warunek NIEMOŻLIWE zawsze uniemożliwia doprowadzenie do rozwiązania dopuszczalnego, to możemy zrezygnować ze sprawdzania warunku C.

Notatki

Przykład: Problem skoczka szachowego

Ścieżka

Na szachownicy $m \times n$ chcemy znaleźć taką sekwencję ruchów skoczka, aby odwiedził każde pole szachownicy dokładnie raz.

Cykl

Dodajemy warunek, że z ostatniego pola które odwiedził może przeskoczyć na pierwsze.

Notatki

Istnienie rozwiązania

Twierdzenie

Dla każdej planszy $m \times nm$, gdzie $\min(m, n) \geqslant 5$ istnieje ścieżka konika szachowego.

Twierdzenie Schwenk'a

Dla każdej planszy $m \times n$ (gdzie $m \leqslant n$) istnieje cykl konika szachowego chyba, że jeden z następujących warunków jest prawdziwy:

1. m i n są nieparzyste,
2. $m = 1, 2$ lub 4, oraz $n > 1$,
3. $m = 3$ i $n = 4, 6$ lub 8.

Istnieje liniowy algorytm wyznaczania ścieżki lub cyklu skoczka, ale przedstawimy metodę częściowego przeglądu z nawrotami.

Notatki

Implementacja ścieżki w Adzie

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

procedure Konik2 is
   procedure Sciezka (n, m : Integer) is
      plansza : array (1 .. m, 1 .. n) of Integer :=
         (others => (others => 0));
      dx : array (1 .. 8) of Integer :=
         (1, 2, 2, 1, -1, -2, -2, -1);
      dy : array (1 .. 8) of Integer :=
         (2, 1, -1, -2, -2, -1, 1, 2);
      licznik : Integer := 0;

      procedure Wypisz is
      begin
         for i in 1 .. m loop
            for j in 1 .. n loop
               Put (plansza (i, j));
            end loop;
            New_Line;
         end loop;
         New_Line;
      end Wypisz;
```


Notatki

Implementacja ścieżki w Adzie

```
      procedure NastRuch (x, y, l : Integer) is
         nx, ny : Integer;
      begin
         plansza (x, y) := l;
         if l = m * n then
            Wypisz;
            licznik := licznik + 1;
         else
            for i in 1 .. 8 loop
               nx := x + dx (i);
               ny := y + dy (i);
               if (1 <= nx and nx <= m and 1 <= ny and ny <= n)
                  and then plansza (nx, ny) = 0
               then
                  NastRuch (nx, ny, l + 1);
               end if;
            end loop;
         end if;
         plansza (x, y) := 0;
      end NastRuch;
```


Notatki

Implementacja ścieżki w Adzie

```
   begin
      for i in 1 .. m loop
         for j in 1 .. n loop
            NastRuch (i, j, 1);
         end loop;
      end loop;
      Put (licznik);
      New_Line;
   end Sciezka;
begin
   Sciezka (5, 5);
end Konik2;
```


Notatki

Implementacja cyklu w Adzie

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

procedure Konik3 is
   procedure Cykl (n, m : Integer) is
      plansza : array (1 .. m, 1 .. n) of Integer :=
         (others => (others => 0));
      dx : array (1 .. 8) of Integer :=
         (1, 2, 2, 1, -1, -2, -2, -1);
      dy : array (1 .. 8) of Integer :=
         (2, 1, -1, -2, -2, -1, 1, 2);
      licznik : Integer := 0;

      procedure Wypisz is
      begin
         for i in 1 .. m loop
            for j in 1 .. n loop
               Put (plansza (i, j));
            end loop;
            New_Line;
         end loop;
         New_Line;
      end Wypisz;
```


Notatki

Implementacja cyklu w Adzie

```
      procedure NastRuch (x, y, l : Integer) is
         nx, ny : Integer;
      begin
         plansza (x, y) := l;
         if l = m * n then
            for i in 1 .. 8 loop
               nx := x + dx (i);
               ny := y + dy (i);
               if (1 <= nx and nx <= m and 1 <= ny and ny <= n)
                  and then plansza (nx, ny) = 1
               then
                  Wypisz;
                  licznik := licznik + 1;
               end if;
            end loop;
```


Notatki

Implementacja cyklu w Adzie

```
         else
            for i in 1 .. 8 loop
               nx := x + dx (i);
               ny := y + dy (i);
               if (1 <= nx and nx <= m and 1 <= ny and ny <= n)
                  and then plansza (nx, ny) = 0
               then
                  NastRuch (nx, ny, l + 1);
               end if;
            end loop;
         end if;
         plansza (x, y) := 0;
      end NastRuch;
   begin
      NastRuch (1, 1, 1);
      Put (licznik);
      New_Line;
   end Cykl;
begin
   Cykl (5, 6);
end Konik3;
```


Notatki

Dla efektywnego rozwiązywania problemów sformułowanych w postaci problemu spełnienia ograniczeń opracowano specjalną metodologię programowania nazywaną *Constraint programming*.

Constraint programming opracowano początkowo dla języka programowania logicznego Prolog, który w naturalny sposób wykonuje algorytmy z nawrotami, ale dostępna jest ona także dla innych języków programowania jak np. C++.

Notatki