Реферат на тему:

**Задача про розміщення ферзів. Дерево пошуку та його обхід**

Розглянемо шахівницю, що має розміри не 8 8, а *n n*, де *n*>0. Як відомо, шаховий ферзь атакує всі клітини та фігури на одній з ним вертикалі, горизонталі та діагоналі. Будь-яке розташування кількох ферзів на шахівниці будемо називати їх ***розміщенням***. Розміщення називається ***допустимим***, якщо ферзі не атакують одне одного. Розміщення *n* ферзів на шахівниці *n n* називається ***повним***. Допустимі повні розміщення існують не при кожному значенні *n*. Наприклад, при *n*=2 або 3 їх немає. За *n*=4 їх лише 2 (рис.19.1), причому вони дзеркально відбивають одне одного.

**Задача.** Написати програму побудови всіх повних допустимих розміщень *n* ферзів, де 4 *n* 20.

Для початку з'ясуємо деякі властивості допустимих розміщень. Очевидно, що в них кожний ферзь займає окрему вертикаль і горизонталь. Занумеруємо вертикалі й горизонталі номерами 1, … , *n* та позначимо через <*H*1, *H*2,  , *Hi*> послідовність номерів горизонталей, зайнятих ферзями, що стоять у вертикалях 1, 2,  , *i*, де 0 *i n*. Випадок *i*=0 відповідає порожньому розміщенню <>.

Існує *n* способів розмістити ферзя в першій вертикалі, тобто перейти від порожнього розміщення до непорожнього. Цей перехід позначимо стрілкою (рис. 19.2(а)). За кожного з розміщень ферзя в першій вертикалі є *n* варіантів розміщення ферзя в другій вертикалі, але з них слід відкинути недопустимі. Відмітимо їх знаком '\*' (рис.19.2(б)).

Узагалі, нехай зафіксовано розміщення ферзів у перших *i*-1вертикалях:

*S*(*i*-1)=<*H*1, ,*Hi*-1>.

*Для побудови всіх допустимих розміщень із початком S(i-1) треба перебрати всі допустимі розміщення S(i)з ферзем у i-й вертикалі та для кожного побудувати всі допустимі розміщення з початком S(i)*.

Отже, маємо рекурсивний алгоритм побудови всіх допустимих розміщень, за яким пошук усіх допустимих заповнень ферзями останніх *n*-*i*+1вертикалей зводиться до пошуку заповнень *n*-*i* вертикалей.

Уточнимо цей алгоритм рекурсивною процедурою deps. Нехай розмір шахівниці не більше nm=20. Номери вертикалей та діагоналей містяться в діапазоні nums=1..nm, а розміщення зображається станом масиву *H* типу

arh = **array**[ nums ] **of** nums.

Процедура deps задає побудову розміщення, починаючи з *i*-ї вертикалі за фіксованих *H*[1],  , *H*[*i*-1]. Підпрограми test та writs задають відповідно перевірку допустимості розміщення <*H*[1], … , *H*[*i*-1], *H*[*i*]> та друкування повного розміщення. Вони викликаються у процедурі deps:

**procedure** deps ( var H : arh; n, i : nums);

**var** j, k : nums;

**begin**

**for** k := 1 **to** n **do**

**begin**

H[i] := k;

**if** test ( H, i) **then**

**if** i = n **then** writs ( H, n) {друкування повного розміщення }

**else** deps ( H, n, i+1 ) {рекурсивний виклик}

**end**

**end**

Функція test задає перевірку допустимості розміщення <*H*[1],  , *H*[*i*-1], *H*[*i*]> за умови, що <*H*[1],  , *H*[*i*-1]> є допустимим:

**function** test ( **var** H : arh; i : nums ) : boolean;

**var** j : nums; flag : boolean;

**begin**

j := 1; flag := **true**;

{перевірка, чи займається нова горизонталь і діагональ}

**while** ( j < i ) **and** flag **do**

**begin**

flag := ( H[i] <> H[j] ) **and** ( abs ( H[i]-H[j] ) <> i-j ); j := j+1

**end**;

test := flag

**end**

Розробка процедури writs друкування повного розміщення залишається вправою.

Програма розв'язання задачі має такий вигляд:

**program** Queens ( input, output );

**const** nm = 20;

**type** nums = 1..nm;

arh = **array**[ nums ] of nums;

**var** H : arh; n : nums;

**procedure** writs  **end**;

**function** test  **end**;

**procedure** deps  **end**;

**begin**

writeln ('задайте розмір дошки: 4..20>'); readln ( n );

deps ( H, n, 1)

**end**.

**2. Дерево пошуку та його обхід**

Розміщення ферзів на шахівниці, що будуються в процесі виконання програми Queens, можна подати вузлами ***кореневого орієнтованого дерева***(рис.19.3).

У цьому дереві кожний вузол <*H*[1],  , *H*[*i*]>, де 0 *i*<*n*, має ***синів***

<*H*[1],  , *H*[*i*], 1>, <*H*[1],  , *H*[*i*], 2>,  , <*H*[1],  , *H*[*i*], *n*>.

Відповідно цей вузол називається їхнім ***батьком***. Сини вузла, сини його синів тощо називаються його ***нащадками***, а він – їхнім ***попередником***. Порожнє розміщення <> є ***коренем дерева***, повні чи недопустимі розміщення – його ***листками***, а допустимі неповні – ***проміжними вузлами***. Кожний вузол дерева має певну ***глибину***, або ***рівень*** у дереві. Глибиною кореня є 0, його синів – 1 тощо. Повним розміщенням відповідають листки дерева, які в даному разі мають глибину *n*. Зазначимо, що в даному разі глибина вузлів дерева збігається з довжиною їх як розміщень.

Це дерево відбиває пошук повних допустимих розміщень, тому називається ***деревом пошуку***. Пересування по вузлах дерева у визначеному порядку називається ***обходом дерева***. Отже, пошук розміщень у дереві є результатом його обходу.

Задамо алгоритм, реалізований процедурою deps із програми Queens, в узагальненому вигляді. Нехай *A* позначає вузол дерева, *ОБХІД( A )* – обхід дерева з коренем *А*, а синами вузла *A* є *A*(1), *A*(2),  , *A*(*n*). Тоді процедура deps із програми Queens має таку схему:

**for** k := 1 **to** n **do**

**begin**

*перехід до вузла A(k)*;

**if** *A(k) є допустимим* **then**

**if** *A(k) є листком* **then** *обробка листка A(k)*

**else** *ОБХІД( A(k) )*

**end**

Як бачимо, процедура deps задає обхід дерева пошуку з вузлів-розміщень ферзів. Цей обхід називається ***обходом*** дерева ***у глибину***. Ця назва зумовлена тим, що *обхід дерева з довільним коренем закінчується лише після того, як закінчено обхід усіх його нащадків*. Тобто від вузла ми переходимо до його нащадків, заглиблюючися в дерево.

Обхід дерева в глибину відтворюється за допомогою магазина (стека), до якого додаються та з якого вилучаються вузли дерева.

З кожним вузлом дерева пов'яжемо інформацію, яка додається при переході до цього вузла. В задачі про розміщення ферзів кореневий вузол відповідає порожньому розміщенню, тому з ним ніяка інформація не пов'язана. При переході від вузла, що подає розміщення <*H*[1],  , *H*[*i*]>, до вузла, відповідного розміщенню <*H*[1],  , *H*[*i*], *k*>, збільшується номер останньої вертикалі *i*, в *k*-у клітину якої ставиться ферзь. Отже, з вузлом зв'язується пара чисел (*i*, *k*), що є номерами вертикалі й горизонталі. Саме такі пари додаються до магазина вузлів.

У задачі про ферзі роль магазина відіграє масив H. Збільшення номера вертикалі *i*, тобто перехід до наступного компонента масиву, разом із присвоюванням H[i]:=*k* відтворюють додавання до магазина нового елемента – пари (*i*, *k*). Цикл із заголовком

**for** k := 1 **to** n **do**

у процедурі deps задає перебирання вузлів-"братів"

<*H*[1], , *H*[*i*-1], 1>, <*H*[1], , *H*[*i*-1], 2>,  , <*H*[1], , *H*[*i*-1], *n*>,

що рівносильно послідовному вилученню з магазина попереднього брата з додаванням наступного.

Опишемо обхід дерева пошуку розміщень без застосування рекурсії. Розглянемо пересування, пов'язані з вузлами дерева. З допустимого вузла-листка ми одразу рухаємося до його батька, з недопустимого – до його брата. Пересування, пов'язані з кожним його проміжним вузлом, можна подати, як на рис.19.4.

Як бачимо, відвідувати проміжний вузол доводиться лише двічі – на початку та в кінці обходу дерева, коренем якого він є. Для того, щоб відрізнити ці два випадки, потрібні додаткові змінні. У разі розміщень ферзів перехід від вузла до його правого брата задається збільшенням H[i] на 1. Це рівносильне одночасному виштовхуванню вузла з магазина та додаванню його правого брата. Звідси випливає, що коли обробляється вузол глибини *i*, в магазині є лише по одному вузлу кожної глибини *m*, *m i*. Тому достатньо однієї додаткової змінної для кожної можливої глибини. Отже, означимо додатковий масив D того ж самого типу, що й масив H. Значенням D[i] стає 0, коли до вузла глибини *i* ми приходимо згори або зліва, та 1 – коли знизу.

Перехід до вузла знизу – це повернення до батька, і його умовою в задачі про ферзі є H[i]=n.

Повернення до кореня дерева означає кінець його обходу. Тому використаємо умову i=0 як умову закінчення пошуку. Отже, пошук повних допустимих розміщень ферзів має таке описання, яке по суті є тілом процедури пошуку:

i:=1; H[i]:=1; D[i]:=0;

**while** (i<>0) **do**

**begin**

**if** i=n **then** {обробка вузла-листка}

**if** test(H, i) **then** {друкування повного допустимого розміщення}

{ та повернення до батька незалежно від наявності братів}

**begin** writs(H, n); i:=i-1; {i>0!} D[i]:=1 **end**

**else**

**if** H[i]<n **then** H[i]:=H[i]+1 {перехід до правого брата}

**else** {повернення до батька – }

{піддерево, в якому він є коренем, вже обійшли}

**begin** i:=i-1; {i>0!} D[i]:=1 **end**

**else** {обробка проміжного вузла}

**if** (D[i]=0) **and** test(H, i) **then** {рух у глибину}

**begin** i:=i+1; H[i]:=1; D[i]:=0 **end**

**else** {рух праворуч або нагору}

**if** H[i]<n **then** {рух праворуч}

**begin** H[i]:=H[i]+1; D[i]:=0 **end**

**else** {рух нагору}

**begin** i:=i-1; **if** i>0 **then** D[i]:=1 **end**

**end**

Оформлення програми з необхідними означеннями, ініціалізаціями та нерекурсивною процедурою пошуку залишаємо як вправу.

Узагальнимо наведений алгоритм, вважаючи, що, на відміну від задачі про розміщення ферзів, кореневий вузол дерева також містить деяку відповідну інформацію:

*заштовхнути кореневий вузол у магазин*;

**while** *магазин не порожній* **do**

**begin**

*нехай A – вузол на верхівці магазина*;

**if** *A є листком* **then**

**begin**

*обробити листок A*;

*виштовхнути A з магазина*;

**if** *A не є правим сином свого батька* **then**

*заштовхнути в магазин правого брата A*;

**end**

**else** {*A – проміжний вузол*}

**if** *A* *є допустимим і дерево з коренем A ще не оброблено* **then**

*заштовхнути в магазин лівого сина A*

**else** {*дерево з коренем A вже оброблено або A не є допустимим*}

**begin**

*виштовхнути A з магазина*;

**if** *A не є правим сином свого батька і не є коренем* **then**

*заштовхнути правого брата A в магазин*;

**end**

**end**.

Наведений опис задає так званий ***вичерпний пошук*** у дереві пошуку варіантів, оскільки рано чи пізно ми дістаємося кожного допустимого вузла дерева. Зазначимо, що цей опис є схемою багатьох алгоритмів розв'язання різноманітних задач, пов'язаних із перебиранням варіантів.