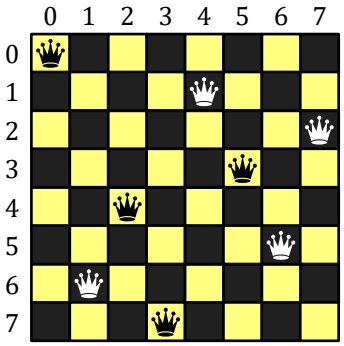
một cách xếp n quân hậu trên bàn cờ sao cho không quân nào ăn quân nào. Ví dụ một cách xếp với $n = 8$ được chỉ ra trong Hình 17-6.



Hình 17-6. Một cách xếp 8 quân hậu trên bàn cờ 8 × 8

❋ Đánh số hàng, cột và đường chéo

Nếu đánh số các hàng từ trên xuống dưới theo thứ tự từ 0 tới $n - 1$, các cột từ trái qua phải theo thứ tự từ 0 tới $n - 1$. Thì khi đặt n quân hậu lên bàn cờ, mỗi hàng phải có đúng 1 quân hậu (hậu ăn được ngang), ta gọi quân hậu sẽ đặt ở hàng 0 là quân hậu 0, quân hậu ở hàng 1 là quân hậu 1... quân hậu ở hàng $n - 1$ là quân hậu $n - 1$. Vậy một nghiệm của bài toán sẽ được biết khi ta tìm ra được vị trí cột của những quân hậu.

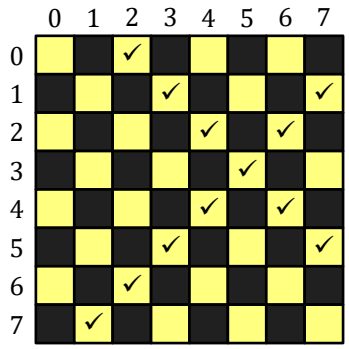
Quan sát trên bàn cờ, một quân hậu ở ô (x, y) (hàng x, cột y) sẽ khống chế (khóa):

- Toàn bộ hàng x
- Toàn bộ cột y
- Toàn bộ các ô (i, j) thỏa mãn đẳng thức $i + j = x + y$. Những ô này nằm trên một đường chéo mà ta gọi là đường chéo phụ.
- Toàn bộ các ô (i, j) thỏa mãn đẳng thức $i - j = x - y$. Những ô này nằm trên một đường chéo mà ta gọi là đường chéo chính.

Từ những nhận xét đó, ta có ý tưởng đánh số những hàng, cột, đường chéo mỗi khi đặt một quân hậu. Cụ thể là khi đặt quân hậu vào ô (x, y), hàng x, cột y, đường chéo phụ $x + y$, đường chéo chính $x - y$ sẽ bị đánh dấu cấm đặt bất kỳ một quân hậu nào khác.

Vấn đề hơi rắc rối đối với việc đánh số đường chéo chính, bởi C++ chỉ cho phép đánh số mảng từ chỉ số 0. Nếu ta gọi đường chéo chính qua ô (x, y) có chỉ số $x - y$ thì chỉ số này hoàn toàn có thể bị âm nếu $x < y$. Ta sẽ cộng chỉ số này thêm $n - 1$ để đảm bảo rằng nó là một số không âm. Như vậy theo cách đánh số mới, khi đặt quân hậu vào ô (x, y), nó sẽ khống chế hàng x, cột y, đường chéo phụ $x + y$ và đường chéo chính $x - y + n - 1$. Tức là:

- Các hàng được đánh số từ 0 tới $n-1$
- Các cột được đánh số từ 0 tới $n-1$
- Các đường chéo phụ được đánh số từ 0 tới $2n-2$
- Các đường chéo chính được đánh số từ 0 tới $2n-2$



Hình 17-7. Đường chéo chính và đường chéo phụ

Hình 17-7 là bàn cờ 8×8 trên đó thể hiện đường chéo phụ mang chỉ số 8 và đường chéo chính mang chỉ số 5. Hai đường chéo này có chung ô (3,5). Ta có thể kiểm tra lại:

Ô (3,5) không chế hàng 3, cột 5, đường chéo phụ $3 + 5 = 8$, đường chéo chính $3 - 5 + 8 - 1 = 5$.

❋ Cơ chế đánh số cột và đường chéo

Ba mảng được sử dụng để đánh dấu cột và đường chéo, ta không cần đánh dấu các hàng:

- `bool a[0 … n - 1]`:
  - `a[j] == true`, nếu như cột `j` còn tự do
  - `a[j] == false`, nếu cột `j` đã bị một quân hậu khóa
- `bool b[0 … 2 * n - 2]`
  - `b[p] == true`, nếu như đường chéo phụ thứ `p` còn tự do
  - `b[p] == false`, nếu như được chéo phụ thứ `p` đã bị một quân hậu khóa.
- `bool c[0 … 2 * n - 2]`
  - `c[q] == true`, nếu như đường chéo chính thứ `q` còn tự do
  - `c[q] == false`, nếu như đường chéo chính thứ `q` đã bị một quân hậu khóa.

Ban đầu chưa có quân hậu nào trên bàn cờ, các cột và đường chéo đều tự do nên cả ba mảng đánh dấu đều được khởi tạo giá trị `true`.

❋ Thuật toán quay lui

Xét tất cả các cột, thử đặt quân hậu 0 vào một cột, với mỗi cách đặt như vậy, xét tất cả các cách đặt quân hậu 1 không bị quân hậu 0 ăn, lại thử một cách đặt quân hậu 1 và xét tiếp các cách đặt quân hậu 2…Mỗi khi đặt được xong quân hậu cuối cùng, ta in ra cách xếp hậu và dừng chương trình.

- Khi chọn vị trí cột j cho quân hậu thứ i, ta phải chọn ô (i, j) không bị các quân hậu đặt trước đó ăn, tức là phải chọn j thỏa mãn: cột j, đường chéo phụ $i + j$, đường chéo chính $i - j + n - 1$ đều còn tự do. Có thể kiểm tra điều kiện này bằng:
  `a[j] && b[i + j] && c[i - j + n - 1]`
- Khi thử đặt được quân hậu vào ô (i, j), nếu đó là quân hậu cuối cùng ($i = n - 1$) thì ta có một nghiệm. Nếu không:
  - Trước khi gọi đệ quy tìm cách đặt quân hậu thứ $i + 1$, ta khóa cột và hai đường chéo đi qua ô (i, j), để quá trình chọn cách đặt các quân hậu kế tiếp sẽ không phạm vào cột và hai đường chéo này nữa.
    `a[j] = b[i + j] = c[i - j + n - 1] = false;`
  - Sau khi lời gọi đệ quy thoát, ta mở khóa cột và hai đường chéo đi qua ô (i, j) để cột và hai đường chéo đó lại thành tự do; bởi sắp tới ta sẽ thử một vị trí cột khác cho quân hậu i, nên cột j cũng như hai đường chéo chứa ô (i, j) hoàn toàn có thể đặt một quân hậu khác trong các bước kế tiếp.
    `a[i] = b[i + j] = c[i - j + n - 1] = true;`

Ta đã sử dụng kỹ thuật đánh dấu trong chương trình liệt kê chỉnh hợp không lặp/hoán vị. Tuy nhiên để liệt kê hoán vị chỉ cần một mảng đánh dấu xem giá trị có tự do không, còn bài toán xếp hậu thì cần phải đánh dấu cả 3 thành phần: Cột, đường chéo phụ, đường chéo chính. Trường hợp đơn giản hơn: Yêu cầu liệt kê các cách đặt n quân xe lên bàn cờ $n \times n$ sao cho không quân nào ăn quân nào chính là bài toán liệt kê hoán vị.

❋ Cài đặt

Input

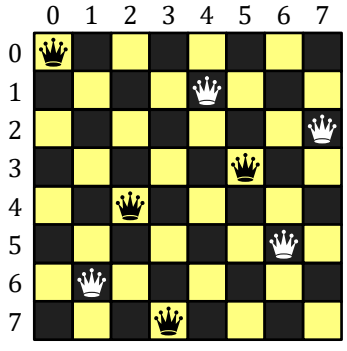
Số nguyên dương $n \leq 20$

Output

Một cách đặt các quân hậu lên bàn cờ $n \times n$

Sample Input	Sample Output
8	(0, 0) (1, 4) (2, 7) (3, 5) (4, 2) (5, 6) (6, 1) (7, 3)

Giải thích



NQUEENS_BT.cpp Thuật toán quay lui giải bài toán xếp hậu

```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4  const int maxN = 20;
5  int n;
6  bool a[maxN], b[2 * maxN - 1], c[2 * maxN - 1];
7  int column[maxN]; //Chỉ số cột của các quân hậu được đặt
8  bool found; //Cờ báo tìm ra nghiệm hay chưa
9
10 void Print() //In kết quả
11 {
12     for (int i = 0; i < n; ++i)
13         cout << "(" << i << ", " << column[i] << ")\n";
14 }
15
16 void Attempt(int i) //Thử các cách đặt quân hậu i
17 {
18     if (i == n) //Nếu đã đặt hết n quân hậu 0 … n – 1
19     {
20         found = true; //Đặt cờ báo đã tìm ra nghiệm
21         return; //Thoát
22     }
23     int& j = column[i]; //Đặt j tham chiếu tới cột của con hậu i cho tiện
24     for (j = 0; j < n; ++j) //Thử mọi cột
25         if (a[j] && b[i + j] && c[i - j + n - 1]) //Nếu có thể đặt hậu vào ô (i, j)
26         {
27             a[j] = b[i + j] = c[i - j + n - 1] = false; //Khóa cột và hai đường chéo
28             Attempt(i + 1); //Thử các cách đặt quân hậu i + 1
29             if (found) //Nếu đã tìm ra nghiệm
30                 return; //Thoát luôn cả dây chuyền đệ quy
31             a[j] = b[i + j] = c[i - j + n - 1] = true; //Mở khóa cột và hai đường chéo
32         }
33 }
34
35 int main()
36 {
37     cin >> n;
38     fill(a, a + n, true); //a[0 … n – 1] = true
39     fill(b, b + 2 * n - 1, true); //b[0 … 2n – 2] = true
40     fill(c, c + 2 * n - 1, true); //c[0 … 2n - 2] = true
41     found = false; //Cờ báo chưa tìm ra nghiệm
42     Attempt(0); //Khởi động thuật toán quay lui
43     if (!found) //Không tồn tại cách đặt
44         cout << "NO SOLUTION";
45     else //Có tồn tại cách đặt, in kết quả
46         Print();
47 }
```

Thuật toán dùng một biến `found` làm cờ báo xem đã tìm ra nghiệm hay chưa, nếu `found == true`, thuật toán quay lui sẽ ngưng ngay quá trình tìm kiếm.

Một sai lầm dễ mắc phải là chỉ đặt lệnh `return` khi `i == n` tức là chỉ để lệnh return ở dòng 21 mà quên mất lệnh `return` ở dòng 30. Nếu làm như vậy lệnh return chỉ có tác dụng thoát hàm `Attempt(n)`. Muốn ngưng cả một dây chuyền đệ quy, cần phải thoát liền một loạt các lời gọi hàm: `Attempt(n)`, `Attempt(n - 1)`, …, `Attempt(0)`, vì vậy khi cờ báo `found == true`, lệnh `return` đặt sau lời gọi đệ quy sẽ thoát luôn hàm gọi, điều đó sẽ khiến cả dây chuyền đệ quy thoát ra khi tìm được nghiệm.

```
30 | //Hàm cận: Ước lượng nếu chọn ∈ [From; n) thì cận trên đạt bao nhiêu người
31 | int UpperBound(int From)
32 | {
33 |     int res = x.size(); //Đếm những người đã quyết định chọn
34 |     for (int i = From; i < n; ++i)
35 |         if (c[i] == x.size() && a[i].size() >= best.size())
36 |             ++res; //Cộng tất cả những người có tiềm năng
37 |     return res;
38 | }
39 |
40 | void Attempt(int From) //Thuật toán quay lui, thử chọn thêm trong tập [From; n)
41 | {
42 |     if (x.size() > best.size()) //Mỗi khi tìm ra phương án tốt hơn best
43 |         best = x; //Cập nhật best theo phương án mới
44 |     //Xét tất cả mọi người trong tập [From; n)
45 |     for (int i = From; i < n; ++i)
46 |     {
47 |         if (UpperBound(i) <= best.size()) //Cận trên cũng không thể ra tốt hơn best
48 |             break; //Bỏ, không thử tiếp để lùi về
49 |         if (c[i] == x.size() && a[i].size() >= best.size()) //Chọn i có tiềm năng tốt hơn
50 |         {
51 |             Select(i, true); //Quyết định chọn i, cập nhật mảng c
52 |             x.push_back(i); //Thêm i vào cuối vector x
53 |             Attempt(i + 1); //Thử chọn thêm từ tập [i + 1; n)
54 |             x.pop_back();   //Bỏ i khỏi vector x, chuẩn bị thử người khác
55 |             Select(i, false); //Quyết định bỏ chọn i, cập nhật mảng c
56 |         }
57 |     }
58 | }
59 |
60 | void Print() //In ra phương án tối ưu best
61 | {
62 |     cout << "Number of guests: " << best.size() << '\n';
63 |     cout << "Guests to be invited: ";
64 |     for (int i: best)
65 |         cout << i << ' ';
66 | }
67 |
68 | int main()
69 | {
70 |     ReadInput();
71 |     Attempt(0);
72 |     Print();
73 | }
```

17.4.3. Bài toán xếp hậu

Ta quay trở lại với bài toán ở mục 17.3.6: Trên bàn cờ tổng quát kích thước $n \times n$, tìm một cách xếp n quân hậu sao cho không quân nào ăn quân nào. Bài toán trong mục này chỉ tăng giới hạn kích thước dữ liệu và tối ưu hóa cách cài đặt bằng phương pháp nhánh cận

❋ Mô hình duyệt

Trong mục 17.3.6 ta đã trình bày thuật toán quay lui, tuy nhiên nếu tăng kích thước dữ liệu, chỉ với khoảng $n = 30$ chương trình đã có dấu hiệu chậm, với $n = 80$ thì ít ai đủ đủ kiên nhẫn để đợi chương trình tìm ra một nghiệm.

Phương pháp trình bày trong mục này cho phép tìm giải pháp với n lớn trong thời gian nhanh hơn. Thực ra nó không hẳn là kỹ thuật nhánh cận, chỉ là một ví dụ để chúng ta thấy được rằng **thứ tự duyệt** rất quan trọng.

Trong chương trình trước, ta thử lần lượt cách đặt cho các quân hậu $0,1,2,\dots,n-1$. Nếu thử với giá trị n tương đối lớn (chẳng hạn $n = 30$) và in ra các kết quả trung gian trong quá trình duyệt (dãy x), ta sẽ thấy rằng thuật toán quay lui hay rơi vào một tiến trình duyệt vô vọng: khi xác định sai một vị trí đặt quân hậu, nó vẫn phải duyệt qua mọi cách đặt các quân hậu còn lại dù việc này rất tốn thời gian và không đi đến kết quả nào cả.

Để cải thiện tốc độ, ta vẫn giữ lại hầu hết các thiết kế trong thuật toán quay lui, tuy nhiên thứ tự các quân hậu được đặt sẽ khác đi. Thay vì thử đặt các quân hậu theo thứ tự từ 0 tới $n-1$, tại mỗi bước, quân hậu khó đặt nhất sẽ được ưu tiên đặt trước lên bàn cờ.

❋ Duyệt theo thứ tự độ khó của từng quân hậu

Ta đã quy ước gọi quân hậu i là quân hậu nằm trên hàng thứ i, độ khó của quân hậu i là số ô trên hàng i mà đã bị các quân hậu đã đặt không chế. Hay nói cách khác, số ô mà có thể đặt quân hậu i vào càng ít thì quân hậu đó càng khó đặt.

Thuật toán quay lui được viết bởi hàm `Attempt(level)`: Ở đây `level` là số quân hậu đã đặt và ta cần phải tìm cách đặt `n - level` quân hậu còn lại. Hàm `Attempt(level)` trước hết tìm quân hậu khó đặt nhất, thử mọi cách có thể đặt nó và gọi đệ quy `Attempt(level + 1)` để thử tiếp… Trong trường hợp `level == n`, ta đã tìm ra nghiệm, đặt cờ báo và ngưng thuật toán quay lui. Thuật toán quay lui được khởi động bằng lời gọi `Attempt(0)`.

Ta cũng dùng mảng `column[0 … n)` để lưu chỉ số cột của các quân hậu đã đặt, nếu một quân hậu `i` chưa được đặt thì `column[i] = -1`. Qua đó ta có thể tìm quân hậu khó đặt nhất bằng cách: Xét tất cả các quân hậu `i` có `column[i] == -1`, đếm số ô có thể đặt quân hậu `i` và từ đó xác định quân hậu mà số ô có thể đặt nó là ít nhất.

❋ Cài đặt

NQUEENS_BB.cpp 🗎 Bài toán xếp hậu

```
#include <iostream>
#include <algorithm>
using namespace std;
const int maxN = 80;
int n;
bool a[maxN], b[2 * maxN - 1], c[2 * maxN - 1];
int column[maxN]; //Chỉ số cột của các quân hậu được đặt
bool found; //Cờ báo tìm ra nghiệm hay chưa

```

```
10 | void Print() //In kết quả
11 | {
12 |     for (int i = 0; i < n; ++i)
13 |         cout << "(" << i << ", " << column[i] << ")\n";
14 | }
15 |
16 | int MostDifficultQueen() //Tìm quân hậu khó đặt nhất
17 | {
18 |     int res; //Chỉ số quân hậu khó đặt nhất
19 |     int MinDeg = n + 1; //Số ô quân hậu khó nhất có thể đặt vào
20 |     for (int i = 0; i < n; ++i) //Xét từng quân hậu i
21 |         if (column[i] == -1) //Nếu quân hậu i chưa được đặt
22 |         {
23 |             //Đếm deg là số ô có thể đặt quân hậu
24 |             int deg = 0;
25 |             for (int j = 0; j < n && deg < MinDeg; ++j)
26 |                 if (a[j] && b[i + j] && c[i - j + n - 1])
27 |                     ++deg;
28 |             if (deg < MinDeg) //Cập nhật quân hậu khó đặt nhất
29 |             {
30 |                 MinDeg = deg;
31 |                 res = i;
32 |             }
33 |         }
34 |     return res;
35 | }
36 |
37 | void Attempt(int level)
38 | {
39 |     if (level == n) //Nếu đã đặt hết n quân hậu
40 |     {
41 |         found = true; //Đặt cờ báo đã tìm ra nghiệm
42 |         return; //Thoát
43 |     }
44 |     int i = MostDifficultQueen(); //Tìm quân hậu khó đặt nhất
45 |     int& j = column[i]; //Đặt j tham chiếu tới cột của con hậu i cho tiện
46 |     for (j = 0; j < n; ++j)
47 |         if (a[j] && b[i + j] && c[i - j + n - 1])
48 |         {
49 |             a[j] = b[i + j] = c[i - j + n - 1] = false; //Khóa cột và hai đường chéo
50 |             Attempt(level + 1); //Thử các cách đặt quân hậu tiếp
51 |             if (found) return; //Nếu đã tìm ra nghiệm, thoát luôn cả dây chuyền đệ quy
52 |             a[j] = b[i + j] = c[i - j + n - 1] = true; //Mở khóa cột và hai đường chéo
53 |         }
54 |     j = -1; //Trước khi thoát, con hậu i trở về trạng thái chưa đặt
55 | }
56 |
57 | int main()
58 | {
59 |     cin >> n;
60 |     fill(a, a + n, true); //a[0 … n – 1] = true
61 |     fill(b, b + 2 * n - 1, true); //b[0 … 2n – 2] = true
62 |     fill(c, c + 2 * n - 1, true); //c[0 … 2n - 2] = true
63 |     found = false; //Cờ báo chưa tìm ra nghiệm
64 |     fill(column, column + n, -1); //Tất cả các quân hậu đều chưa đặt
65 |     Attempt(0); //Khởi động thuật toán quay lui
66 |     if (!found) //Không tồn tại cách đặt
67 |         cout << "NO SOLUTION\n";
68 |     else //Có tồn tại cách đặt, in kết quả
69 |         Print();
70 | }
```