TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐỒ ÁN I
LẬP TRÌNH

Tóm tắt nội dung

Báo cáo gồm ba phần trình bày các bài toán sắp xếp, giải thuật quay lui, đánh chỉ mục từ khóa trong văn bản.

Phần thứ nhất trình bày về ý tưởng, cài đặt và độ phức tạp của các thuật toán sắp xếp cơ bản bao gồm `buble sort, selection sort, insertion sort, shell sort, quick sort, heap sort`.

Phần thứ hai trình bày một số bài toán về giải thuật quay lui trong bài toán *tám hậu*, *mã đi tuần* và kĩ thuật nhánh cận trong bài toán *người du lịch*.

Phần thứ ba trình bày về các cách cài đặt khác nhau của bài toán đánh chỉ mục từ khóa trong văn bản trên các cấu trúc dữ liệu khác nhau bao gồm *danh sách liên kết*, *bảng băm*, *cây nhị phân tìm kiếm*

Các chương trình trong báo cáo được trình bày bằng ngôn ngữ lập trình C++

1 Các thuật toán sắp xếp

1.1 Bài toán sắp xếp

Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng nào đó theo một thứ tự nhất định. Chẳng hạn như thử tự tăng dần (hay giảm dần) đối với một dãy số, thứ tự từ điển đối với các từ v.v... Yêu cầu về sắp xếp thường xuyên xuất hiện trong các ứng dụng Tin học với các mục đích khác nhau: sắp xếp dữ liệu trong máy tính để tìm kiếm cho thuận lợi, sắp xếp các kết quả xử lý để in ra trên bảng biểu v.v...

Nói chung dữ liệu có thể xuất hiện dưới nhiều dạng khác nhau. Một tập các đối tượng cần được sắp xếp là tập các bản ghi (records), mỗi bản ghi bao gồm một số trường (fields) khác nhau. Nhưng không phải toàn bộ mà chỉ là một trương nào đó (hay một vài trường nào đó được chú ý tới thôi. Trường như vậy chúng ta gọi là là khóa (textsfkey). Sắp xếp sẽ được tiến hành dựa vào giá trị của khóa này.

Khi sắp xếp, các bản ghi trong bảng sẽ được đặt vào các vị trí sao cho các giá trị khóa tương ứng của chúng có đúng thứ tự đã ấn định. thì kích thước của toàn bản ghi có thể rất lớn, nên nếu việc sắp xếp thực hiện trực tiếp trên các bản ghi sẽ đòi hỏi sự chuyển đổi vị trí của các bản ghi, kéo theo việc thường xuyên phải di chuyển, copy những vùng nhớ lớn, gây ra những tổn phí thời gian khá nhiều. Thường người ta khắc phục tình trạng này bằng cách xây dựng một bảng khoá: Mỗi bản ghi trong bảng ban đầu sẽ tương ứng với một bản ghi trong **bảng khoá**. Bảng khoá cũng gồm các bản ghi nhưng mỗi bản ghi chỉ gồm có hai trường:

- Trường thứ nhất chứa khóa.
- Trường thứ hai chứa liên kết tới một bản ghi trong bảng ban đầu, tức là một thông tin đủ để biết bản ghi tương ứng với nó trong bảng ban đầu là bản ghi nào.

Có thể coi *khóa như là đại diện cho các bản ghi* và để đơn giản, ta chỉ nói tới giá trị khóa mà thôi. Các thao tác trong kĩ thuật sắp xếp lẽ ra là tác động lên toàn bản ghi giờ đây chỉ làm việc trên khóa.

1.2 Một số quy ước

Mục này trình bày các cài đặt một số giải thuật sắp xếp phổ biến. Để thuận tiện hơn trong việc theo dõi chương trình sau này, ta đưa vào một số quy ước như sau:

- Dãy khóa cần được sắp xếp được lưu trong mảng `arr` gồm các phần tử từ $0...len-1$, trong đó, `len` là số phần tử của mảng.
- hàm `swap(a, b)` có tác dụng đổi chỗ hai phần tử `a` và `b`
- Kí hiệu arr[i...j] được hiểu là các phần tử từ arr[i] đến arr[j] trong mảng arr.

1.3 Thuật toán sắp xếp nổi bọt

Trong thuật toán săp xếp nổi bọt, các dãy khóa sẽ được duyệt từ cuối dãy lên đầu dãy (từ `arr[len-1]` về `arr[0]`), nếu gặp hai khóa kề nhau bị ngược thứ tự thì đổi chỗ của chúng cho nhau, sau lần duyệt như vậy, khóa nhỏ nhất sẽ trở về vị trí đầu tiên, quá trình lại tiếp tục với các khóa từ dãy `arr[1]` tới `arr[len-1]`:

Cài đặt thuật toán trong C++ như sau:

```
1  void bublesort()
2  {
3    int i,j;
4    for (i = 1; i < len; i++)
5      for (j = len-1; j >= i; j--)
6        //Neu arr[j] va arr[j-1] nguoc thu tu
7        if (arr[j] < arr[j-1])
8      {
9        //Doi cho ve dung vi tri
10       swap(arr[j], arr[j-1]);
11       s++;
```

```
    }
}
```

1.4 Thuật tóan sắp xếp chọn

Một trong những thuật toán sắp xếp đơn giản nhất là phương pháp sắp xếp chọn. Ý tưởng của thuật toán:

- Trong lần duyệt đầu tiên, duyệt dãy từ [0...len-1] tìm ra phần tử có giá trị nhỏ nhất đổi vị trí về đầu dãy.
- Trong lần duyệt thứ k duyệt dãy từ [k-1...len-1] tìm ra phần tử tử nhỏ nhất trong dãy và đổi về vị trí k-1.
- ...
- Trong lần duyệtthứ `len`-1 chọn trong hai khóa `arr[len-2]` và `arr[len-1]` đổi với vị trí `len-2`

Kết thúc quá trình duyệt thì dãy còn lại đã được sắp xếp.
Sau đây là cài đặt của thuật toán trên C++:

```
void selectionsort()
{
  int i,j,min;
  for (i=0;i<len-1;i++)
    {
      //Dat arr[i] lam phan tu be nhat
      min=i;
      //Tim phan tu be nhat trong day arr[i]-->arr[len-1]
      for (j=i+1;j<len;j++)
    if (arr[j]<arr[min])
      min=j;
      if (i!=min)
    //Doi cho phan tu be nhat ve vi tri
    //Day arr[0]-->arr[i] la day tang dan
    swap(arr[i],arr[min]);
    }
}
```

1.5 Thuật toán sắp xếp kiểu chèn

Xét dãy khóa arr[0...len-1], ta thấy chỉ gồm một khóa arr[0] chỉ gồm 1 khóa và có thể coi là đã sắp xếp rồi. Xét thêm arr[1], ta so sánh nó với arr[0], nếu thấy arr[1]<arr[0] thì ta chèn nó vào trước arr[1],... Một cách tổng quát, ta sẽ sắp xếp dãy arr[0...i-1] trong điều kiện dãy khóa đã được sắp xếp rồi chèn arr[i] vào dãy đó tại đúng vị trí để được dãy arr[0...i] đã được sắp xếp.

```
void insertionsort()
{
  int i,j,tmp;

  for (i = 1; i < len; i++)
    {
      tmp = arr[i];
      //Chen phan tu arr[i] vao day da co thu tu
      j = i - 1;
      //Tim tu ben phai sang phan tu be hon arr[i]
      while ((j >= 0) && (arr[j] > tmp))
    {
      //Don cac phan tu ve ben phai tao 1 cho chua cho arr[i]
      arr[j+1] = arr[j];
      j = j - 1;
    }
      //Chen phan tu vao dung vi tri,
      //Day arr[0]-->arr[i] la day tang dan
      arr[j+1] = tmp;
    }
}
```

1.6 Shell sort

Nhược điểm của thuật toán sắp xếp chèn thể hiện khi mà ta luôn phải chèn một khá vào vị trí gần đàu dãy. Trong trường hợp đó, người ta sử dụng phương pháp *Shell Sort*. Xét dãy khóa arr[0...len-1]. Với một số nguyên dương $0 <= h <= len - 1$, ta có thể chia dãy đó thành dãy con:

- Dãy con 1: arr[0], arr[h], arr[2h], ...
- Dãy con 2: arr[1], arr[h+1], arr[2h+1], ...
- Dãy con 3: arr[2], arr[h+2], arr[2h+2], ...
- ...
- Dãy con h-1: arr[h-1], arr[2h-1], arr[3h-1], ...

Những dãy con như vậy được gọi là dãy con sắp xếp theo độ dài h. Tư tưởng của thuật toán ShellSort là : Với một bước h, áp dụng thuật toán sắp xếp kiểu chèn từng dãy con độc lập để làm mịn dần dãy khóa chính. Rồi lại làm tương tự đối với bước $h\%2$ ... cho tới khi $h = 1$ thì ta được dãy khóa đã sắp xếp. Đây chính là nguyên nhân ShellSort hiệu quả hơn thuật toán sắp xếp chèn: khóa nhỏ được nhanh chóng đưa về *gần* vị trí đúng của nó.

1.7 Thuật toán sắp xếp Quick Sort

Thuật toán Quick Sort được đề xuất bởi C.A.R.Hoare là một phương pháp sắp xếp tốt nhất, nghĩa là dù dãy khóa thuộc kiểu dữ liệu *có thứ tự nào*, Quick Sort cũng có thể sắp xếp được và chauw có một thuật toán sắp xếp tổng quát nào nhanh hơn Quick Sort về mặt tốc dộ trung bình.

Ý tưởng chủ đọa của phương pháp có thể tóm tắt như sau:

Sắp xếp dãy khóa arr[0...len-1] thì có thể coi là sắp xếp từ chỉ số 0 tới chỉ số len-1 trong dãy khóa đó. Để sắp xếp một đoạn trong dãy khóa, nếu đoạn đó có ít hơn 1 khóa thì không cần phải làm gì cả, còn nếu đoạn đó có ít nhất 2 khóa thì ta chọn ngẫu nhiên khóa nào đó của đoạn làm "chốt" (*pivot*). Mọt khóa nhỏ hơn chốt đươc xếp vào vị trí đứng sau chốt. Sau phép hoán chuyển như vậy thì đoạn đang xét được chia làm hai đoạn khá rỗng mà mọi khóa trong đoạn đầu đều $\leq$ chốt và một khóa trong đoạn sau đều $\geq$ chốt. Và vấn đề trở thành sắp xếp hai đoạn mới tạo ra (có độ dài ngắn hoăn đoạn ban đầu) bằng phương pháp tương tự.

Để cài đặt thuật toán này trong C++, ta sử dụng hai hàm:

- Hàm `void partition(int L, int H` chọn một chốt pivot bất kì trong dãy từ arr[L...H], chuyển các phần tử trong dãy có giá trị bé hơn pivot về trước pivot và các phần tử có giá trị lớn pivot về sau pivot. Sau đó gọi đệ quy tiếp tục với hai dãy con nhỏ hơn.
- Hàm `void quicksort()` gọi hàm `partition(0, len-1)` thực hiện sắp xếp toàn bộ mảng. Thực ra ta có thể không cần tới hàm này.

Cài đặt `quicksort`:

```
1  void partition(int L, int H)
2  {
3      int i,j,pivot;
4      //Neu chi co duoi 1 phan tu thi khong phai lam di
5      if (L>=H)
6          return;
7      i=L;j=H;
8      srand(time(NULL));
9      //Chon chot ngau nhien, tranh cac truong hop dac biet
10     pivot=arr[L+rand()%(H-L+1)];
11
12     while (i<=j)
13     {
14         //Tim phan tu lon hon hoac bang pivot tu trai sang
15         while (arr[i]<pivot)
16             i++;
17         //Tim phan tu lon hon hoac bang pivot tu phai sang
18         while (arr[j]>pivot)
19             j--;
20
21         if (i<=j)
22         {
23             //Neu tim thay va hai phan tu khac nhau thi doi cho
24             if (i<j)
25                 swap(arr[i],arr[j]);
26             i++; j--;
27         }
28     }
```

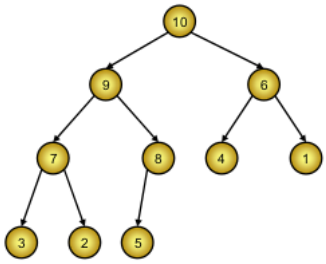
```
29      //Tiep tuc voi day con trai
30      partition(L,j);
31      //Tiep tuc voi day con phai
32      partition(i,H);
33  }
34
35
36  void quickSort()
37  {
38      partition(0,len-1);
39  }
```

1.8 Thuật toán sắp xếp vung đống - Heap Sort

Heap Sort được đề xuất bởi J.W.J.Williams năm 1981, thuật toán không những đóng góp một phương pháp sắp xếp quan trọng để biểu diễn một cấu trúc dữ liệu quan trọng

1.8.1 Đống - Heap

Đống là một dạng *cây nhị phân hoàn chỉnh đặc biệt* mà giá trị tại mọi nút có độ ưu tiên cao hơn hay bằng giá trị lưu trong hai nút con của nó. Trong thuật toán sắp xếp kiểu vun đống, ta coi quan hệ "ưu tiên hơn hay bằng" là quan hệ "lớn hơn hay bằng": $\geq$



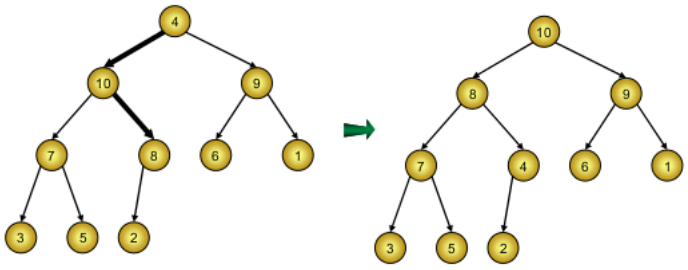
Hình 1: Heap

1.8.2 Vun đống

Để biểu diễn dãy khóa arr[0...len-1] thành một cây nhị phân hoàn chỉnh. Giá trị arr[i] lưu trong nút thứ i (i bắt đầu từ 0). Nút con trái của nút thứ i là arr[2i+1] và con phải của nút thứ i là arr[2i+2]. Nút cha của nút thứ j là $(j-1)\%2$.

Vì cây nhị phân gồm một nút thì hiển nhiên là đống. nên **để vun một nhánh cây gốc r thành đống, ta có thể coi hai nhánh con của nó là đống rồi và thực hiện thuật toán vun đống từ dưới lên đối với cây**. Gọi h là chiều cao của cây, nút ở mức h (nút lá) đã là một đống, ta vun lên những nút ở mức $h-1$ cũng là gốc của đống,... cứ như vậy cho tới mức 1 cũng là gốc của đống.

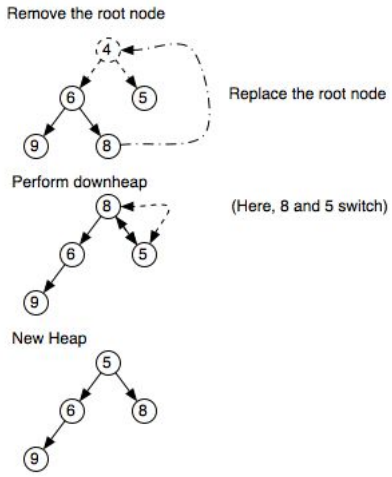
Thuật toán vun đống đối với gốc r khi hai nhánh con đã là đống Giả sử nút r chứa giá trị V. Tử r, ta cứ đi tới nút con chứa giá trị lớn nhất trong hai nút con, cho tới khi gặp phải một nút c mà mọi nút con của c đều có giá trị $\leq$ V (nút lá cũng là trường hợp riêng của điều kiện này). Dọc đường đi từ r tới c, ta đẩy giá trị chứa ở nút con lên nút cha và đặt giá trị V vào nút c.



Hình 2: Vun đống

1.8.3 Tư tưởng của Heap Sort

Đầu tiên với dãy khóa arr[0...len-1] được vun từ dưới lên để nó biểu diễn một đống, khi đó khóa arr[0] tương ứng với nút gốc của đống là khóa lớn nhất, ta đảo giá trị khóa đó cho arr[len-1] và không tính tới arr[len-1] nữa. Còn lại dãy khóa arr[0...len-2] tuy không là đống nữa nhưng là biểu diễn của cây nhị phân hoàn chỉnh mà hai nhánh cây đã là đống rồi. Vậy chỉ cần vun một lần, ta lại dược một đống, đảo giá trị arr[0] với arr[len-2] rồi tiếp tục tới khi đống chỉ còn lại một nút.



Hình 3: Đảo giá trị của arr[0] và arr[len-1] rồi xét lại

Thuật toán Heap Sort có hai hàm chính:

- Hàm `maxHeapify` làm nhiệm vụ vun cây gốc root thành đống trong điều kiện hai gốc con $2 * root + 1$ và $2 * root + 2$ đã là đống rồi. Các nút từ endnode+1 tới len-1 đã nằm đúng vị trí, không được tính tới nữa.
- Hàm `heapSort` mô tả lại quá trình vun đống theo ý tưởng trên

```
void maxHeapify(int root, int endnode)
{
  int c, key;
  key = arr[root];
  //Khi nao root chua phai la la
  while ((root*2 + 1) <= endnode)
    {
      c=root * 2 + 1;
      //Neu con trai be hon con phai thi c=chi so con phai
      if ((c < endnode-1) && (arr[c] < arr[c+1]))
    c++;
      //Neu 2 con be hon root thi cay da la maxheap
      if (arr[c] <= key)
    break;
      //Doi cho cho con lon hon
      arr[root] = arr[c];
      // Tiep tuc voi root tai vi tri moi
      root = c;
    }
  //root ban dau nam dung vi tri
  arr[root] = key;
}

void heapSort()
{
  int r, i;
  // Xay dung dong tren toan bo cay
  for (r = (len/2 - 1); r >= 0; r--)
    maxHeapify(r, len-1);
  for (i = len-1; i>0; i--)
    {
      //Doi cho nut lon nhat ve cuoi day
```

```
33          swap(arr[0], arr[i]);
34          //Xay lai dong, ko kem nut vua hoan doi
35          maxHeapify(0, i-1);
36      }
37  }
```

2 Các bài toán về thuật toán quay lui và kĩ thuật nhánh cận

2.1 Thuật toán quay lui

Thuật toán quay lui dùng để giải bài toán *liệt kê* các cấu hình. Mỗi cấu hình được xây dựng bằng cách xây dựng từng phần tử, mỗi phần tử được chọn bằng cách thử tất cả các khả năng. Giả sử cấu hình liệt kê có dạng X[1...n], khi đó thuật toán quay lui thực hiện qua các bước:

1. Xét tất cả các giá trị X[1] có thể nhận, thử cho X[1] nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho X[1] ta sẽ:
2. Xét tất cả các giá trị X[2] có thể nhận, thử cho X[2] nhận lần lượt các giá trị đó. Với mỗi giá trị thử gán cho X[2] lại xét tiếp khả năng chọn X[3]... cứ tiếp tục như vậy đến bước:
3. Xét tất cả các giá trị X[n] có thể nhận, thử cho X[n] nhận lần lượt các giá trị đó, thông báo cầu hình tìm được (X[1], X[2], ... , X[n]).

Trên phương diện quy nạp, có thể nói rằng thuật toán quay lui liệt kê các cấu hình n phần tử dạng X[1...n] bằng cách thử cho X[1] nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho X[1] bài toán trở thành liệt kê cấu hình n-1 phần tử sau X[2...n].

Mô hình của thuật toán quay lui có thể mô tả như sau:

```
void Try(int i)
{
    for (mọi giá trị V có thể gán cho X[i])
    {
        (Thử cho X[i] = V);
        if (X[i] là phần tử cuối cùng trong cấu hình)
        (thông báo cấu hình tìm được);
        else
        {
            (Ghi nhận việc X[i] đã được gán giá trị V);
            Try(i+1);
            (Nếu cần, bỏ việc ghi nhận X[i] = V để thử giá trị khác);
        }
    }
}
```

Thuật toán quay lui sẽ bắt đầu bằng lời gọi Try(1) [1]

2.2 Bài toán mã đi tuần

2.2.1 Bài toán

Xét bàn cờ kích thước kích thước $N * N$. Một quân mã xuất phát tại vị trí (firstRow, firstCol). Quân mã phải đi theo đúng quy tắc bàn cờ vua và đi hết các ô trên bàn cờ sao cho mỗi ô đều được đi qua 1 lần.

2.2.2 Cài đặt thuật toán

Với bài toán này, ta sẽ giải quyết theo thuật toán quay lui. Bài toán sử dụng các biến và hàm sau:

Hằng N Kích thước của bàn cờ.

Biến firstRow, firstCol Vị trí ban đầu của quân mã.

Mảng A[N [N]] Lưu thứ tự các nước đi của quân mã, đồng thời cũng là mảng đóng vai trò đánh dầu khi ô tương ứng đã được đi qua. Vì vậy cần bỏ việc ghi nhận sau lời gọi đệ quy Try(i+1) để thử giá trị khác theo sơ đồ trên.

Hàm printBoard() In ra bàn cờ và thứ tự vị trí quân mã đã đi qua.

Hàm Patrol Hàm thực hiện quay lui để thử tất cả các cách đi quân mã trên bàn cờ có thể được và xuất kết quả ra màn hình nếu đi hết các ô.

[1] Theo mô tả thuật toán quay lui ở đây thì sẽ bắt đầu với Try(1). Không phải các bài toán sử dụng thuật toán quay lui phảiluôn luôn bắt đầu bằng lời gọi Try(1), tùy theo việc cài đặt thuật toán.

Cài đặt thuật toán trên C++

```cpp
#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;
#define N 5 // Kich thuoc ban co

/*
  CLASS BAN CO
  Thuoc tinh:
  +) A[N][N]: Trang thai ban co
  +) firstRow, firstCol: Vi tri xuat phat cua quan ma
  Phuong thuc:
  +) chessBoard(m,n): Khoi tao ban co
  +) printBoard(): Hien thi ban co
  +) Partol(m,n): Di tuan tu vi tri m*n
*/
class chessboard
{
private:
    int A[N][N];
    int firstRow, firstCol;
public:
    chessboard(int m=0,int n=0)
        {
            int i,j;
            for (i = 0; i < N; i++)
                for (j = 0; j < N; j++)
                    A[i][j] = 0;
            firstRow = m;
            firstCol = n;
            A[m][n] = 1;
        }

    void printBoard() //Hien thi ban co
        {
            int i,j;
            static int n = 1; // Bien phu, khong can de y
            cout<<"\nCach "<<n<<":\n"<<endl;
            for (i = 0; i < N; i++)
            {
                for (j = 0; j < N; j++)
                    cout<<setw(3)<<A[i][j];
                cout<<endl;
            }
            n++;
            cout<<endl;
        }

    // Tim nuoc di, nuoc di dau tien da duoc di tai firstRow, firstCol
    void Patrol(int m, int n, int k=2)
        {
            int i,j;
            for (i = 0; i < N; i++)
                for (j = 0; j < N; j++)
                    // Neu co the di duoc toi o i,j va  o do chua duoc di toi
                    if ((fabs(i-m)+fabs(j-n) == 3) && i!=m && j!=n && (A[i][j]==0))
                    {
                        // Thu di toi A[i][j]
                        A[i][j] = k;
                        // Neu da la nuoc di cuoi
                        if (k == N*N)
                        {
                            // Hien thi ban co
                            printBoard();
                            // Buoc dau tien thi khong the thay doi,neu khong co dieu
```

```
                            // kien nay se hien thi toan bo moi cach di tu bat ki nao
                            if ((i != firstRow) && (j != firstCol))
                                //Bo danh dau
                                A[i][j] = 0;
                        }
                        else
                        {
                            // Thu di nuoc tiep theo
                            Patrol(i,j, k+1);
                            // Bo danh dau
                            A[i][j] = 0;
                        }
                    }
        }



};


int main()
{
    int m,n;
    cout<<"BAI TOAN MA DI TUAN\nKich thuoc: "<<N<<'x'<<N<<endl;
    cout<<"Nhap vi tri xuat phat: ";
    cin>>m>>n;
    chessboard a(m,n);
    a.Patrol(m,n);
    cout<<endl;
    return 0;
}

```

2.3 Bài toán tám quân hậu

2.3.1 Bài toán

Bài toán tám quân hậu là bài toán đặt tám *quân hậu* trên bàn cờ vua kích thước 8x8 sao cho không có quân hậu nào có thể "ăn" được quân hậu khác. Như vậy, lời giải của bài toán là một cách xếp tám quân hậu trên bàn cờ sao cho không có hai quân nào đứng trên cùng hàng, hoặc cùng cột hoặc cùng đường chéo. Bài toán tám quân hậu có thể được tổng quát hóa thành bài toán N quân hậu.

2.3.2 Cài đặt thuật toán

Bài toán này cũng sẽ được giải quyết theo thuật toán quay lui.

Phân tích Rõ ràng n quân hậu, mỗi con sẽ được đặt ở một hàng vì quân hậu được ăn ngang, ta gọi quân hậu sẽ đặt ở hàng 0 là quân hậu 0, quân hậu được đặt ở hàng 1 là quân hậu 1, quân hậu được đặt ở hàng n là quân hậu n. Như vậy, một nghiệm của bài toán sẽ được biết khi ta tìm ra được **vị trí cột của những quân hậu**.

Nếu ta định hướng Đông - Phải, Tây - Trái, Nam - Dưới, Bắc - Trên thì ta sẽ nhận thấy rằng:

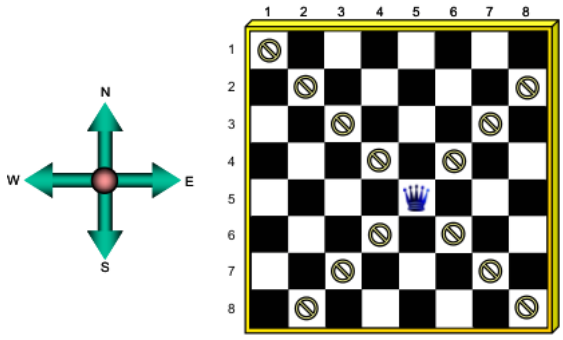
- Một đường chéo theo hướng ĐB-TN bất kì sẽ đi qua một số ô, các ô đó có tính chất Hàng + Cột = C (hằng số). Với mỗi đường chéo ĐB-TN ta có một hằng số C và với một hằng số C: $0 \leq C \leq 2n-2$ xác định duy nhất một đường chéo ĐB-TN từ 0 đến 2n-2.
- Một đường chéo hướng ĐN-TB bất kì đi qua một số ô, các ô có tính chất: Hàng - Cột = C. Với mỗi đường chéo ĐN-TB ta có một hằng số C và với một hằng số C: $-N+1 \leq C \leq N-1$ xác định duy nhất một đường chéo ĐN-TB.

Cài đặt

- Sử dụng 3 mảng để đánh dấu:

  C[N] $C[i] = 0$ nếu cột i còn tự do, $C[i] \neq 0$ nếu cột i đã bị một quân hậu khống chế.

  D1[2*N] $D1[i] = 0$ nếu đường chéo DB-TN thứ i [$0 \leq i = row + col \leq 2N$] còn tự do, $D1[i] \neq 0$ nếu đường chéo đó đã bị khống chế.



Hình 4: Đường chéo ĐB-TN mang chỉ số 10 và đường chéo ĐN-TB mang chỉ số 0

D2[2*N] $D2[i] = 0$ nếu đường chéo DN-TB thứ i [$0 \leq i = row - col + N - 1 \leq 2N$] còn tự do, $D2[i] \neq 0$ nếu đã bị khống chế.

- Sử dụng mảng A[N][N] để lưu kết quả.
- Hàm `printBoard` dùng để xuất kết quả ra màn hình.
- Hàm `Set` dùng để để đánh dấu và hủy đánh dấu các mảng đánh dấu khi thử đặt quân hậu vào vị trí hàng m, cột n. Tham số `unset` mặc định là 0 - bật đánh dấu. Nếu đặt bằng 1 sẽ là hủy đánh dấu.
- Hàm checkBoard dùng để kiểm tra xem nước đi vào vị trí hàng i, cột j có đi được không bằng cách kiểm tra các mảng đánh dấu.
- Hàm setBoard dùng giải thuật quay lui để thử tất cả các cách đặt quân hậu có thể được. Đây là chính trong bài toán này.

C ài đặt bài toán tám hậu trên C++

```
#include <iostream>
#include <iomanip>
using namespace std;
#define N 5

/*
  CLASS BAN CO
  -Thuoc tinh:
  +)C[N]: Mang cang cot da bi kiem soat
  +)D1[2*N]: Duong cheo i+j=const da bi kiem soat
  +)D2[2*N]: Duong cheo i-j=const da bi kiem soat
  +)A[N][N]: Mang bieu dien cach dat cac quan co
  -Phuong thuc
  +)printBoard: Hien thi ban co
  +)Set(m,n,unset): Dat quan co vao vi tri A[m][n] va cai
  dat cac thuoc tinh, neu unset!=1 thi khoi phuc lai
  cac thuoc tinh
  +)checkBoard(m,n): Kiem tra xem co the dat duoc quan hau
  tai vi tri A[m][n] khong
  +setBoard(k): Dat co theo phuong phap quay lui
*/
class chessboard
{
private:
    int C[N]; //Cot da bi kiem soat
    int D1[2*N]; // Duong cheo / (i+j=const, 0 <= i+j <= 2*N-2) da bi kiem soat
    int D2[2*N]; // Duong cheo \ (i-j=const, -N+1 <= i+j <= N-1)da bi kiem soat
    int A[N][N]; //Ban co, neu Board[i][j]!=0 la da duoc dat

public:
    chessboard() //Ham tao cho ban co
        {
            int i,j;
            for (i=0;i<N;i++)
            {
                C[i]=0;
```

```
                D1[i]=0;
                D1[N+i]=0;
                D2[i]=0;
                D2[N+i]=0;
            }
            for (i=0;i<N;i++)
                for (j=0;j<N;j++)
                    A[i][j]=0;
        }

    void printBoard() //In ban co tu ma tran A[N][N]
        {
            int i,j;
            static int n=1;
            cout<<"\n\nCach "<<n<<":\n"<<endl;
            for (i=0;i<N;i++)
            {
                for (j=0;j<N;j++)
                    cout<<setw(3)<<A[i][j];
                cout<<endl;
            }
            cout<<endl;
            n++;
        }

    void Set(int m,int n,int unset=1) //Thu di quan co vao A[m][n]
        {
            if (unset==1) //Danh dau la da di
            {
                C[n]=1;
                D1[m+n]=1;
                D2[m-n+N-1]=1;
            }
            else //Quay tro lai, khong di buoc do nua
            {
                C[n]=0;
                D1[m+n]=0;
                D2[m-n+N-1]=0;
                A[m][n]=0; //Van de do bieu dien du lieu dau ra
            }
        }

    int checkBoard(int i, int j) //Kiem tra xem nuoc di co duoc khong
        {
            if (C[j]==0 && D1[i+j]==0 && D2[i-j+N-1]==0)
                return 1;
            return 0;
        }

    void setBoard(int k=0)
        {
            int i;

            for (i=0;i<N;i++) //Voi moi cot
                if (checkBoard(k,i)) //Kiem tra xem co the dat quan hau o hang
                    //thu k vao cot thu i hay ko
                {
                    A[k][i]=k+1; //Luu thong tin vao ban co
                    if (k==N-1) //Neu dat cau hinh cuoi
                    {
                        printBoard(); //Hien thi ban co
                        A[k][i]=0; //Van de ve cach bieu dien du lieu dau ra
                    }
                    else
                    {
                        Set(k,i,1); //Danh dau la da di
```

```
                        setBoard(k+1); //Di quan co tiep theo
                        Set(k,i,0); //Quay tro lai vi tri truoc, bo danh dau
                        // coi nhu chua di quan nay
                    }
                }
            }
};


int main()
{
    chessboard a;
    a.setBoard();
    cout<<endl;
    return 0;
}
```