ĐỆ QUI QUAY LUI

- Phương pháp thử - sai
- Thuật toán quay lui
- Một số ví dụ
- Nhánh cận + ví dụ

Phương pháp thử - sai

- Dùng cho vấn đề không có quy tắc
- Quá trình thử - sai được xem xét trên các bài toán đơn giản hơn
- Giải quyết bài toán bằng cách thử tất cả các khả năng
  - Giả sử đã giải quyết đến bước i-1
  - Duyệt các khả năng từ 1 đến ni để tìm giải pháp cho bước thứ i
    - Với mỗi khả năng j chấp nhận được, thử i=n thì hoàn thành bài toán, ngược lại giải quyết tiếp bước i+1
    - Nếu không có khả năng nào chấp nhận được thì quay lại bước trước để thử khả năng khác cho bước i-1

Ba vấn đề quan trọng cho bài toán quay lui

- Tìm cách biểu diễn nghiệm của bài toán dưới dạng 1 dãy các đối tượng được chọn dần từng bước (x_1, x_2, ..., x_i, ...)
- Xác định tập S các ứng viên được chọn làm thành phần thứ i của nghiệm.
- Tìm các điều kiện để tập các thành phần đã chọn là nghiệm của bài toán

Cài đặt thuật toán

```
void Try(int i){
    int j
    for (j=1; j<n_i; j++){
         if Chấp nhận j{
              Ghi lại bước i theo j
              if (i==n)
                   Kết thúc thành công
              else Try(i+1)
         }
         Bỏ ghi nhận bước i
    }
}
```

Liệt kê các tổ hợp chập k của n

- Tập con k phần tử khác nhau không phân biệt thứ tự
- Để đơn giản ta xét các tập có thứ tự tăng dần
- Tại bước i, chọn 1 phần tử từ $x_{i-1}+1$ đến n-k+i
- Nếu đạt k phần tử, tiến hành ghi nhận nghiệm.
- Ngược lại gọi đệ quy tiến hành bước i+1.

Liệt kê các tổ hợp chập k của n

```
void Try(int i ) {
    for(int j = a[i-1]+1 ; j<=n-k+i ; j++ ) {
        a[i] = j ;
        if(i==k) {
            // in một cấu hình ra ngoài
            printResult() ;
        }
        else {
            Try(i+1) ;
        }
    }
}
```

Liệt kê các hoán vị của n

- Dùng mảng a[i] để lưu các hoán vị (i=1...n). Lưu ý a[i] != a[j] với mọi i !=j.
- Dùng mảng used[i] để đánh dấu phần tử i đã được dùng chưa.
- Ý tưởng:
  - Chọn một phần tử chưa được sử dụng
  - Lưu vào cấu hình tổ hợp và đánh dấu đã sử dụng
  - Lặp lại cho đến khi đủ cấu hình thì in kết quả
  - Quay trở lại bước trước để đánh dấu nó chưa sử dụng

Liệt kê các hoán vị của n

```
void Try(int k) {
      for (int i = 1; i <= n; i++) {
            if (!used[i]) {
                  a[k] = i;
                  used[i] = 1;
                  if (k == n)
                        printResults();
                  else
                        Try(k + 1);
                  used[i] = 0;
            }
      }
}
```

Rút tiền ATM

- Máy ATM có N tờ tiền, có mệnh giá t_1, t_2, ..., t_N (có thể bằng nhau)
- Đưa ra một cách trả tiền cho số tiền S
  - Sử dụng dãy nhị phân độ dài N để đánh dấu (x_i = 1 nếu tờ tiền được sử dụng và ngược lại)
  - X = (x_1, x_2, ..., x_N) là nghiệm nếu $x_1 \times t_1 + \ldots x_N \times t_N = S$
- Sử dụng biến *found* để kiểm tra việc tìm thấy nghiệm trong quá trình thử

Rút tiền ATM

```
void Try(int k) {
      for (int i = 0; i <= 1; i++) {
            x[k] = i;
            sum = sum + x[k]*t[k];
            if (k == n){
                  if (sum == S){
                        printResults();
                        found = 1;
                  }
            }else if sum<=s Try(k + 1);
            if found break;
            sum = sum - x[k]*t[k];
      }
}
```

Bài tập

- Viết chương trình bằng ngôn ngữ C/C++ giải quyết các bài toán bằng quay lui:
  - Tổ hợp chập k của n
  - Hoán vị của n
  - Rút tiền ATM

Nhánh cận

- Sử dụng quay lui để duyệt tất cả các phương án và chọn phương án tối ưu
  - Mỗi nghiệm $X = (x_1, x_2, ..., x_N)$ được xác định "độ tốt" bằng một hàm f(X).
  - Mục tiêu cần tìm nghiệm có giá trị f(X) min (max)
- Giả sử xây dựng được k thành phần của nghiệm $(x_1, x_2, ..., x_k)$, nếu mở rộng nghiệm đến k+1 đều không có kết quả tốt hơn nghiệm tốt nhất đã biết thì không mở rộng theo k nữa.
  - Cắt bỏ đi các "nhánh" không cần thiết.

Rút tiền ATM tối ưu

- Tìm phương án rút tiền với số tờ tiền ít nhất
  - Giả sử đã xây dựng cách trả tiền đến bước k (x_1, x_2, ..., x_k), trả được sum và dùng c tờ tiền.
  - Số tiền còn phải trả $S-sum$.
  - Gọi $tmax[k]$ là giá trị tờ tiền lớn nhất trong các tờ còn lại.
  - Ít nhất cần sử dụng $\frac{S-sum}{tmax[k]}$ tờ nữa
  - Nếu $c + \frac{S-sum}{tmax[k]} \geq$ cách trả tốt nhất hiện có thì dừng mở rộng.

Rút tiền ATM tối ưu

```
void Try(int k) {
      if (c+(s-sum))/tmax[k] >= cbest return;
      for (int i = 0; i <= 1; i++) {
            x[k] = i; sum = sum + x[k]*t[k]; c = c + i;
            if (k == n){
                  if (sum == S) && (c<cbest){
                        cbest = c;
                        xbest = x;
                  }
            }else if sum<=s Try(k + 1);
            sum = sum - x[k]*t[k]; c = c - i;
      }
}
```

Bài toán người du lịch

- N thành phố (1, 2, ..., N). Thành phố *i* nối với thành phố *j* bằng tuyến đường có khoảng cách *c[i,j] = c[j,i]*.
- Người du lịch xuất phát từ TP 1, muốn đi thăm tất cả TP (mỗi TP đúng 1 lần) và cuối cùng quay về TP 1.
- Tìm hành trình để người đó phải đi với tổng khoảng cách ít nhất.

Bài toán người du lịch

- Hành trình cần tìm có dạng (x_1=1, x_2, …, x_N, X_{N+1}=1): $x_i \neq x_j$ và (x_i, x_{i+1}) có đường đi trực tiếp
- Duyệt quay lui: Tại bước i, chọn x_i từ 1 trong các thành phố có thể tới từ x_{i-1}.
- Nhánh cận: Khởi tạo cấu hình best = ∞
  - Với mỗi bước chọn x_i, tính toán xem đường đi tới lúc đó có < best? Nếu không, dừng phương án.
  - Nếu tìm đạt đến x_n, thử xem (x_n,1) có kết nối không? Nếu có và tổng nhỏ hơn cấu hình best thì cập nhật cấu hình best bằng cách đi mới.

Bài toán người du lịch

```
void Try(int k) {
        if sum >= best return;
        for (int i = 0; i < n; i++) {
                if (!used[i]){
                        x[k] = i; used[i] = 1;
                        sum = sum + c[x[k-1],i];
                        if (k == n){
                                if (sum + c[x[n],1]<best){
                                        best = sum + c[x[n],1];
                                        xbest = x;
                                }
                        }else Try(k + 1);
                        sum = sum – c[x[k-1],i]; used[i]=0;
                }
        }
}
```