Thuật toán đệ quy

* Là một cách cài đặt thuật toán bên cạnh dùng vòng lặp
* Thuật toán biểu diễn bằng đệ quy thường gọn hơn dùng vòng lặp nhưng hiệu quả kém hơn do phải mất thêm thời gian và bộ nhớ phụ để xử lý các lời gọi hàm đệ quy lồng nhau
* Nếu gọi hàm đệ quy lồng nhau quá nhiều --> Stack Over Flow : hết bộ nhớ stack để lưu trữ biến phụ khi xử lý lời gọi đệ quy

Ví dụ. Tính xn nhanh dùng đệ quy

|  |
| --- |
| #include <stdio.h>    double fastPower\_v1(double x, int n)  {  if(n==0) return 1;  if(n%2==0) return fastPower\_v1(x,n/2)\*fastPower\_v1(x,n/2);  else return x\*fastPower\_v1(x,n/2)\*fastPower\_v1(x,n/2);  }    double fastPower\_v2(double x, int n)  {  if(n==0) return 1;  double y = fastPower\_v1(x,n/2);  if(n%2==0) return y\*y;  else return x\*y\*y;  }    int main()  {  double x = 1.02;  int n = 10000;  printf("V1: %lf\n",fastPower\_v1(x,n));  printf("V2: %lf\n",fastPower\_v2(x,n));  return 0;  } |
| #include <stdio.h>  #include <sys/time.h>  #include <time.h>    double fastPower\_v1(double x, int n)  {  if(n==0) return 1;  if(n%2==0) return fastPower\_v1(x,n/2)\*fastPower\_v1(x,n/2);  else return x\*fastPower\_v1(x,n/2)\*fastPower\_v1(x,n/2);  }    double fastPower\_v2(double x, int n)  {  if(n==0) return 1;  double y = fastPower\_v1(x,n/2);  if(n%2==0) return y\*y;  else return x\*y\*y;  }    int main()  {  double x = 1;  int n = 1000000000;  struct timeval stop, start;  gettimeofday(&start, NULL);  printf("V1: %lf\n",fastPower\_v1(x,n));  gettimeofday(&stop, NULL);  printf("took %lu us\n", (stop.tv\_sec - start.tv\_sec) \* 1000000 + stop.tv\_usec - start.tv\_usec);  gettimeofday(&start, NULL);  printf("V2: %lf\n",fastPower\_v2(x,n));  gettimeofday(&stop, NULL);  printf("took %lu us\n", (stop.tv\_sec - start.tv\_sec) \* 1000000 + stop.tv\_usec - start.tv\_usec);  return 0;  } |

So sánh tương quan thời gian gọi hàm và thời gian xử lý đệ quy

|  |
| --- |
| #include <stdio.h>  #include <sys/time.h>  #include <time.h>    double fastPower\_v1(double x, int n)  {  if(n==0) return 1;  if(n%2==0) return fastPower\_v1(x,n/2)\*fastPower\_v1(x,n/2);  else return x\*fastPower\_v1(x,n/2)\*fastPower\_v1(x,n/2);  }    double fastPower\_v2(double x, int n)  {  if(n==0) return 1;  double y = fastPower\_v1(x,n/2);  if(n%2==0) return y\*y;  else return x\*y\*y;  }  double fastPower\_v3(double x, int n)  {  if(n==0) return 1;  double y = 1;  for(int i=1; i<=n; i++)  y=y\*x;  return y;  }  int main()  {  double x = 1;  int n = 1000000000;  struct timeval stop, start;  gettimeofday(&start, NULL);  printf("V1: %lf\n",fastPower\_v1(x,n));  gettimeofday(&stop, NULL);  printf("took %lu us\n", (stop.tv\_sec - start.tv\_sec) \* 1000000 + stop.tv\_usec - start.tv\_usec);  gettimeofday(&start, NULL);  printf("V2: %lf\n",fastPower\_v2(x,n));  gettimeofday(&stop, NULL);  printf("took %lu us\n", (stop.tv\_sec - start.tv\_sec) \* 1000000 + stop.tv\_usec - start.tv\_usec);  gettimeofday(&start, NULL);  printf("V3: %lf\n",fastPower\_v3(x,n));  gettimeofday(&stop, NULL);  printf("took %lu us\n", (stop.tv\_sec - start.tv\_sec) \* 1000000 + stop.tv\_usec - start.tv\_usec);  return 0;  } |

Trong V3: KHÔNG mất thêm thời gian xử lý đệ quy  
V1 = V3 + thời gian xử lý đệ quy

**Ví dụ 1**. Tính tổ hợp chập k của n theo 3 cách

* Cách 1. đệ quy KHÔNG nhớ
* Cách 2. Đệ quy có nhớ (dùng mảng tra giá trị)
* Cách 3. Dùng vòng lặp dạng bottom-up

Ghi thời gian chạy của 3 phương pháp để so sánh

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  long long int c\_v3(int k, int n)  {  if (k==0 ||k==n) return 1;  long long int \*M1 = (long long int\*) calloc(sizeof(long long int),n+1);  long long int \*M2 = (long long int\*) calloc(sizeof(long long int),n+1);  long long int \*M3;  //memset(M,0,sizeof(int)\*(n+1));  M1[0]=1; M1[1]=1;  for(int i=2; i<=n ; i++)  {  M2[0]=1; M2[i]=1;  for(int j=1; j<i; j++)  M2[j]=M1[j] + M1[j-1];  M3=M1;  M1=M2;  M2=M3;  }  return M1[k];  }    int main()  {  printf("C100/50000=%lld\n",c\_v3(100,50000));  return 0;  } |

Back tracking

In ra các xâu nhị phân độ dài 5

|  |
| --- |
| #include <stdio.h>  void printSolution(int \*x, int n){  for(int k = 0; k < n; k++)  printf("%d",x[k]);  printf("\n");  }    int TRY(int k,int n, int \*x) {  for(int v = 0; v <= 1; v++){  x[k] = v;  if(k == n-1) printSolution(x, n);  else TRY(k+1,n,x);  }  }  const int MAX = 5;  int main() {  int x[MAX];  TRY(0,5,x);  } |

**Ví dụ 2.** In ra các xâu độ dài 5 nhưng ko có 2 bit 1 liên tiếp

**Bài tập 1**. In ra tên riêng trong văn bản và tần số xuất hiện

Yêu cầu

* Dùng mảng cấp phát động để lưu trữ các từ và tần số
* Nếu dùng mảng cấu trúc thì sẽ tiện hơn là 2 mảng riêng lẻ
* In ra tên riêng và tần số
* Nếu sắp xếp được thì sẽ tốt hơn (sắp xếp theo tần số giảm dần hoặc theo ABC)

VD. Đầu vào là 1 file văn bản

|  |
| --- |
| “Russia’s comments about suspending its participation in the Black Sea Grain Initiative are regrettable. This life-saving agreement between Russia, Ukraine, and Turkey, brokered in July by the United Nations, has allowed the export of more than nine million metric tons of grain and other food products to populations around the world in the midst of a devastating global food crisis,” USAID Administrator Samantha Power wrote in a statement Sunday. |

Các tên riêng trong văn bản sẽ là

|  |
| --- |
| Russia  Black  Sea  Grain  Initiative  This  Russia  Ukraine  Turkey  July  United  Nations  USAID  Administrator  Samantha  Power  Sunday |

Kết quả in ra sẽ là

|  |
| --- |
| Russia 2  Black 1  Sea 1  Grain 1  Initiative 1  This 1  Ukraine 1  Turkey 1  July 1  United 1  Nations 1  USAID 1  Administrator 1  Samantha 1  Power 1  Sunday 1 |

**Bài tập 2**. Cho đầu vào là 1 file văn bản chứa ma trận nhị phân dạng

|  |
| --- |
| 8 10  1 1 1 1 1 0 1 1 1 1  1 0 1 1 1 1 1 0 1 1  1 0 1 1 1 0 1 1 1 1  1 1 0 1 1 0 1 1 1 1  0 1 1 0 1 0 1 0 1 1  1 1 1 1 1 0 1 1 1 1  1 1 0 1 0 1 1 0 1 1  1 1 1 1 1 0 1 1 1 1  0 0  7 9 |

Trong đó dòng đầu là số hàng và cột của ma trận

Các dòng bit 0/1 tiếp theo sẽ là ma trận nhị phân với 1 là vị trí có thể đi và 0 là tường

2 dòng cuối là tọa độ điểm vào và điểm ra (đếm từ 0)

Cần viết hàm (dựa trên back tracking) kiểm tra xem có thể đi từ điểm vào tới điểm tra trong ma trận nhị phân đã cho hay không

Chỏ có thể đi theo 4 hướng: lên/trái/xuống/phải