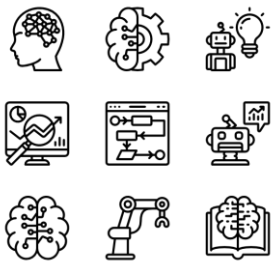
Computer Science for Practicing Engineers

Độ phức tạp của thuật toán



TS. Huỳnh Bá Diệu
Email: dieuhb@gmail.com
Phone: 0914146868

1

Độ phức tạp của thuật toán

Nội dung:

1. **Bài toán tìm mảng con liên tiếp có tổng lớn nhất**
2. **In danh sách theo thứ tự giảm dần**
3. **Tìm kiếm nhị phân và tìm kiếm nội suy**

2

Tìm mảng con liên tiếp có tổng lớn nhất

Bài toán 2:

Cho dãy A gồm n số nguyên. Hãy tìm dãy con liên tiếp có tổng lớn nhất.

Ví dụ A= { -1 4 3 9 -20 5 9 -6 3 4 -100 5 8}

Kết quả {4 3 9 }

Ví dụ A= { 2 7 -10 4 6 -5 4 2 -6 7 -8 1 2}

Kết quả {4 6 -5 4 2 -6 7}

3

Tìm mảng con liên tiếp có tổng lớn nhất

Bài toán 2:

Cho dãy A gồm n số nguyên. Hãy tìm dãy con liên tiếp có tổng lớn nhất.

Solution 1: Liệt kê tất cả các đoạn con và tính tổng từng đoạn, so sánh.

B1: Smax= a[0]; dau=0; cuoi=0;

B2:

Cho cận i từ 0 đến n-1

Cho cận j từ i đến n-1

+ Tính tổng s, đoạn từ i------>j

+ Nếu Smax< s thì { Smax =s; dau=i; cuoi=j;}

B3: In Smax và dãy từ a[dau] ……. a[cuoi];

4

Tìm mảng con liên tiếp có tổng lớn nhất

Bài toán 2:
Cho dãy A gồm n số nguyên. Hãy tìm dãy con liên tiếp có tổng lớn nhất.
Solution 1:
B1: Smax= a[0]; dau=0; cuoi=0;
B2:
```
for(int i=0; i<n; i++)
    for(int j=i; j<n; j++)
     {
        int s=0; for(int k=i; k<=j; k++) s=s+ a[k];
       if (Smax<s) { Smax =s; dau=i; cuoi=j;}
     }
```
B3: In Smax và dãy từ a[dau] ……. a[cuoi];

Độ phức tạp của thuật toán là $O(n^3)$

5

Tìm mảng con liên tiếp có tổng lớn nhất

```
void sol1(int *a, int n){
        int smax=a[0], d=0, c=0;
        for(int i=0; i<n; i++)
          for(int j=i; j<n; j++)
          {
                int s=0;
                for(int k=i; k<=j; k++) s=s +a[k];
                if(s>smax) { smax=s; d=i; c=j;}
          }
        cout<<"\n Doan con co tong lon nhat la "<<smax<<"\n";
        for(int k= d; k<=c; k++) cout<<" "<<a[k];
}
```

6

Tìm mảng con liên tiếp có tổng lớn nhất

```
#include<iostream>
using namespace std;
void in(int *a, int n) {
        cout<<"\n Noi dung mang:\n ";
        for(int i=0; i<n; i++) cout<<a[i]<<" ";
        cout<<"\n";
}
void sol1(int *a, int n){      }
int main() {
        int a[13]= { 2, 7, -10, 4, 6, -5, 4, 2, -6, 7, -8, 1, 2};
        int n= 13;
        in(a,n); sol1(a,n);
}
```

7

Tìm mảng con liên tiếp có tổng lớn nhất

Bài toán 2:

Cho dãy A gồm n số nguyên. Hãy tìm dãy con liên tiếp có tổng lớn nhất.

Solution 2: **Dựa vào cách tính tổng các dãy con.**

- **Khi đã tính tổng từ 2—8, muốn tính tổng từ 2→ 9 thì ta chỉ cần *lấy tổng từ 2 đến 8* cộng thêm với a[9].**
- **Nếu đã có tổng từ 0→ 9 và tổng từ 0—> 4 thì khi muốn tính tổng từ 5 đến 9 ta lấy $S_{0\to 9} - S_{0\to 4}$**

Tổng quát:

- **Chỉ cần tính tổng từ 0→ n-1** [S[i] là tổng từ 0→i]
- **Khi cần tính tổng trong đoạn từ i----->j thì ta lấy $S_j - S_{i-1}$, trừ trường hợp i=0**

8

Tìm mảng con liên tiếp có tổng lớn nhất

Bài toán 2:

Cho dãy A gồm n số nguyên. Hãy tìm dãy con liên tiếp có tổng lớn nhất.

Solution 2: Dựa vào cách tính tổng các dãy con.

Tính mảng S: S[0]=a[0]; S[i] = S[i-1] + a[i], với i>0

Ví dụ A= { 2 7 -10 4 6 -5 4 2 -6 7 -8 1 2}

Ta có mảng S như sau: S= {2 9 -1 3 9 4 8 10 4 11 3 4 6}

Khi cần tính tổng từ a[3] đến a[11] thì ta lấy: S[11] –S[2] = 4 – (-1) =5

9

Tìm mảng con liên tiếp có tổng lớn nhất

Bài toán 2: Cho dãy A gồm n số nguyên. Hãy tìm dãy con liên tiếp có tổng lớn nhất.

Solution 2:

B1: Smax= a[0]; dau=0; cuoi=0; SS[0]=a[0], t; **O(1)**

B2: for(int i=1; i<n; i++) SS[i] = SS[i-1] + a[i]; **O(n)**

B3: for(int i=0; i<n; i++) **$O(n^2)$**

```
for(int j=i; j<n; j++)
{
   if(i==0) s= SS[j]; else s= SS[j] – SS[i-1];        // O(1)
   if (Smax<s) { Smax =s; dau=i; cuoi=j;}     // O(1)
  }
```

B4: In Smax và dãy từ a[dau] ……. a[cuoi]; **O(n)**

Độ phức tạp của thuật toán là $O(n^2)$

10

Tìm mảng con liên tiếp có tổng lớn nhất

```
void sol2(int *a, int n){
        int smax=a[0], d=0, c=0;
        int *ss= new int[n];   ss[0] =a[0];
        for(int i=1; i<n; i++) ss[i]= ss[i-1] +a[i];
        for(int i=0; i<n; i++)
          for(int j=i; j<n; j++)
          {
                int s=0;
                if(i==0) s=ss[j]; else s= ss[j] -ss[i-1];
                if(s>smax) { smax=s; d=i; c=j;}
          }
        cout<<"\n Doan con co tong lon nhat la "<<smax<<"\n";
        for(int k= d; k<=c; k++) cout<<" "<<a[k];
}
```

11

Tìm mảng con liên tiếp có tổng lớn nhất

time.h
stdlib.h

```
void sinhmang(int *&a, int n){
        a= new int [n];
        for(int i=0; i<n; i++) { int x= rand()%1001; if(x%3==0) x=-2*x; a[i]=x;}
}
int main(){
        int *a, n; n=2000; sinhmang(a,n);
        in(a,n);
        long t1, t2;
        t1=clock(); sol2(a,n); t2=clock();
        cout<<"\n Thoi gian thuc hien thuat toan theo sol2 la "<<(t2-t1);
        t1=clock(); sol1(a,n); t2=clock();
        cout<<"\n Thoi gian thuc hien thuat toan theo sol1 la "<<(t2-t1);
}
```

12

Tìm mảng con liên tiếp có tổng lớn nhất Kadane's Algorithm

Bài toán 2:

Cho dãy A gồm n số nguyên. Hãy tìm dãy con liên tiếp có tổng lớn nhất.

Solution 3: Dựa vào các phần tử có giá trị âm mà trị tuyệt đối lớn hơn tổng các đoạn con trước đó.

Nếu tổng giá trị của đoạn trước số âm a[i] mà cộng với a[i] nhỏ hơn 0 thì ta xem như bắt đầu 1 đoạn mới.

Ví dụ A= { 2 7 -10 4 6 -5 4 2 -6 7 -18 1 2}

Ví dụ A= { -1 4 3 9 -20 5 9 -6 3 4 -100 5 8}

13

Tìm mảng con liên tiếp có tổng lớn nhất Kadane's Algorithm

Bài toán 2: Cho dãy A gồm n số nguyên. Hãy tìm dãy con liên tiếp có tổng lớn nhất.
Solution 3: Dựa vào các phần tử có giá trị âm mà trị tuyệt đối lớn hơn tổng các đoạn con trước đó.

```
B1: Smax= a[0]; dau=0, cuoi=0, S=0;
B2: for(int i=0; i<n; i++)
    {
        S= S + a[i];
        if (S> Smax) { Smax=S; cuoi=i; }
        if(S<0) S=0;        // bắt đầu đoạn mới
    }
B3: Từ Smax xác định vị trí đầu sau đó In Smax và dãy từ a[dau] ……. a[cuoi];
Độ phức tạp của thuật toán là O(n)
```

14

Tìm mảng con liên tiếp có tổng lớn nhất: Kadane's Algorithm

```
void sol3()
  {
   int Smax= a[0],dau=0, cuoi=0, S=0, d=0;  // d lưu vị trí đoạn mới
   for(int i=0; i<a.length; i++)
   {
         S= S + a[i];
         if (S> Smax) { Smax=S; dau= d; cuoi=i; }
         if(S<0) { S=0; d= i+1; };
   }
  cout<<"\n Mang lien tiep co tong max= " <<Smax;
  for(int i=dau; i<=cuoi; i++) cout<<" " << a[i];
  }
```

15

Tìm mảng con liên tiếp có tổng lớn nhất Kadane's Algorithm

```
int maxSubArraySum(int a[], int size)
{
int max_so_far = a[0];
int curr_max = a[0];
for (int i = 1; i < size; i++)
{
curr_max = max(a[i], curr_max+a[i]);
max_so_far = max(max_so_far, curr_max);
}
return max_so_far;
}
```

```
int main()
{
int a[] = {-2, -3, 4, -1, -2, 1, 5, -3};
int n = sizeof(a)/sizeof(a[0]);
int max_sum = maxSubArraySum(a, n);
cout << "Maximum contiguous sum is " <<
max_sum;
return 0;
}
```

16

Thuật toán (Algorithm)

2	**-5**	**-5**	**7**	**20**
4	7	3	6	-42
-30	-6	1	4	7
12	6	38	-1	-99
-9	1	-12	4	26
21	4	-45	-10	5
-40	1	21	-8	9

1. Cho bảng A có m hàng và n cột, chứa các giá trị nguyên.
Tìm bảng con trong A có tổng lớn nhất

2. Cho bảng A có m hàng và n cột, chứa các giá trị 0, 1.
Tìm ***bảng con lớn nhất trong A*** có chứa toàn số 1

-5	-6	3	1	0
9	7	8	3	7
-6	-2	-1	2	-4
-7	5	5	2	-6
3	2	-9	-5	1

Sum = 34

-5	-6	3	1	0
9	-7	8	3	7
-6	-2	-1	2	-4
-7	5	5	2	-6
3	2	-9	-5	1

Sum = 23

-5	-6	3	1	0
9	7	8	3	7
-6	-2	-1	2	-4
-7	5	5	2	-6
3	2	9	-5	1

Sum = 35

17

Độ phức tạp của Thuật toán (algorithm complexity)

Khi đánh giá độ phức tạp của thuật toán, ta chủ yếu quan tâm đến số câu lệnh phải thực hiện.

Các yếu tố khác như: phần cứng (tốc độ xử lý, tốc độ đọc ghi, dung lượng bộ nhớ), ngôn ngữ lập trình, phần mềm… có ảnh hưởng đến tốc độ thực hiện thuật toán nhưng không nhiều, vì vậy có thể bỏ qua.

18

Độ phức tạp của Thuật toán (algorithm complexity)

Một số dạng độ phức tạp

O(1) : Thời gian thực hiện nhanh, chỉ gồm một số lệnh đơn giản (lệnh cơ sở như gán, cộng, trừ ….)

O(lglgn)

O(lgn)

O(n), O(nlgn), $O(n^2)$, $O(n^3)$ ..

$O(2^n)$

O(n!)

Một số bài toán không có cách giải.

19

Ví dụ: In danh sách theo thứ tự giảm dần

Cho danh sách chứa n phần tử nguyên dương. Hãy in ra danh sách theo thứ tự giảm dần.

Ví dụ A= {3,2,7,6,9,8,5}

Kết quả: 9 8 7 6 5 3 2

Giải pháp 1:

Lặp lại các bước sau n lần:

- Tìm phần tử lớn nhất chưa xử lý trong dãy, in ra, sau đó đánh dấu không xử lý nữa.

20

Ví dụ: In danh sách theo thứ tự giảm dần

Giải pháp 1:

```
// mảng đánh dấu phần tử a[i] đã xét chưa
for(int i=0; i<n; i++) b[i]=0;  // chưa có phần tử nào xét
for(int i=0; i<n; i++)
{
        int vmax=-1, vt=-1;;
        for(int j=0; j<=n; j++)
                if(a[j]> vmax && b[j]==0) { vmax=a[j];  vt=j; }
        SOUT(vmax);  b[vt]=1;   // in giá trị max ra và đánh dấu đã xét
}
```

Độ phức tạp Max(O(n), O(n^2))= O(n^2)

Giải pháp khác: sắp xếp mảng rồi in (O(nlgn))!

21

Ví dụ: In danh sách theo thứ tự giảm dần

```
void solution1()
  {
    System.out.println("\n Noi dung day giam dan\n");
    int []b= new int [a.length];
    for(int i=0; i<b.length; i++) b[i]=0;
    for(int i=0; i<a.length; i++)
    {
       int vmax=-1, vt=-1;
       for(int j= 0; j<a.length; j++)
          if(a[j]>vmax && b[j]==0) {vmax= a[j]; vt=j;}
       System.out.print(vmax+ " "); b[vt]=1;
    }
  }
```

Mảng b dùng để đánh dấu: *b[i] =0 có nghĩa phần tử i chưa được xử lý*

Lưu giá trị max và ghi dấu vị trí max tại j

In giá trị max và đánh dấu vị trí vt đã được xử lý

22

Ví dụ: In danh sách theo thứ tự giảm dần

```
void solution2()
   {
      System.out.println("\n Noi dung day giam dan:\n");
      Arrays.sort(a);
     for(int i=a.length-1; i>=0; i--) sout(a[i] + " ");
}
```

import java.ulti.Arrays

23

Ví dụ: In danh sách theo thứ tự giảm dần

Viết chương trình

- Sinh dãy số nguyên
- In dãy số nguyên
- Thực hiện Sol1
- Thực hiện Sol2
- So sánh kết quả, thời gian thực thực hiện (n=1000, n=100000)

24

Ví dụ: In danh sách theo thứ tự giảm dần

```
int []a;
void gen(int n)
   {
      a= new int[n];
      for(int i=0; i<n; i++)    a[i] = (int) (Math.random()*10000)+1;
   }
   void in()
   {
      System.out.println("\n Mang A:---------- ");
      for(int i=0; i<a.length; i++) System.out.print(" " + a[i] );      System.out.println("\n -----");
   }
```

25

Tìm kiếm phần tử x trong danh sách có thứ tự

Cho dãy a= { 2 4 7 8 9 10 24 35 67 82}, cần tìm x=28.

Cần tìm x có trong dãy không?

Giải pháp 1: Tìm tuần tự trong dãy

Xét lần lượt từng phần tử a[i] rồi so sánh với x

```
for(int i=0; i<n; i++)
    if(a[i]==x) return true;
return false;
```

Độ phức tạp của thuật toán là O(n)

26

Tìm kiếm phần tử x trong danh sách có thứ tự

Giải pháp 2: Tìm nhị phân
Cần tìm x trong đoạn từ left đến right
1. Lặp cho đến khi tìm thấy hoặc left > right
{
mid= (left + right)/2;
nếu a[mid] ==x thì return true;
ngược lại nếu a[mid] >x thì right= mid-1;
ngược lại left= mid+1
}
2. return false;

27

Tìm kiếm phần tử x trong danh sách có thứ tự

Cho dãy a= { 2 4 7 8 9 10 **24** 35 67 82}, cần tìm x=28

Chạy lần thứ	Left	Right	Mid
0	0	9	// ko xác định
1	Left= mid+1=5	9	4
2	5	Right= mid-1=6	7
3	Left= mid+1=6	6	5
4	Left= mid+1=7	6	6
DO Left> Right nên kết thúc vòng lặp và thực hiện câu lệnh 2, trả về FALSE			

28

Tìm kiếm phần tử x trong danh sách có thứ tự

Cho dãy a= { 2 4 7 8 12 44 56 78 80 90 137 249 }, cần tìm x=168

Chạy lần thứ	Left	Right	Mid
0			
1			
2			
3			
4			

29

Tìm kiếm phần tử x trong danh sách có thứ tự

```
boolean bsearch(int x)   {
      int L=0, R=a.length-1;
      while(L<=R)
      {
         int m=(L+R)/2;
         if(a[m]==x) {  System.out.println(" vi tri =" +m); return true;}
         else
            if(a[m]>x) R= m-1;
            else L= m+1;
      }
      return false;
   }
```

30

Tìm kiếm phần tử x trong danh sách có thứ tự **TÌM NỘI SUY**

```
boolean Interpolation_Search(int x)
  {
    int L=0, R=a.length-1;
    while(L<=R)
    {
      if(x<a[L]|| x>a[R]) return false;
      int m= (int) (L + (x-a[L])*(R-L)/(a[R]-a[L]));
      if(a[m]==x) {  System.out.println(" vi tri =" +m); return true;}
        else   if(a[m]>x) R= m-1;     else L= m+1;
    }
    return false;
  }
```

int m=(L+R)/2;

31

Tìm kiếm phần tử x trong danh sách có thứ tự

```
int interpolation_search(int *arr, int size, int key)
{
   int low = 0,  high = size – 1, mid;
   while ((arr[high] != arr[low]) && (key >= arr[low]) && (key <= arr[high])) {
      mid = low + ((key - arr[low]) * (high - low) / (arr[high] - arr[low]));
      if (arr[mid] < key)          low = mid + 1;
      else if (key < arr[mid])   high = mid - 1;
      else return mid;
   }
   if (key == arr[low])  return low ;
   else  return -1;
}
```

32

Tìm kiếm phần tử x trong danh sách có thứ tự

Sinh dãy n phần tử tăng dần

```
void gen1(int n)
  {
     a= new int[n];
     a[0] = (int) (Math.random()*100);
     for(int i=1; i<n; i++)
        a[i] = a[i-1] + (int) (Math.random()*10);
  }
```

33

Bài tập về nhà

Tổ chức chương trình:

- Sinh dãy n phần tử tăng dần
- Tìm tuyến tính
- Tìm nhị phân
- Tìm nội suy
- So sánh thời gian chạy (n=10000000, x= 30)

34

Tài liệu đọc thêm về một số thuật toán tìm kiếm

Exponential Search :

https://www.geeksforgeeks.org/exponential-search/

Fibonacci Search:

https://en.wikipedia.org/wiki/Fibonacci_search_technique

35

Link YouTube

Exponential Search :

https://www.geeksforgeeks.org/exponential-search/

Fibonacci Search:

https://en.wikipedia.org/wiki/Fibonacci_search_technique

Mảng con có tổng max

https://www.youtube.com/watch?v=86CQq3pKSUw

36

Bài học tiếp theo: Các phương pháp thiết kế thuật toán

1. Phương pháp Vét cạn (Brute-force or exhaustive search)
2. Phương pháp quay lui (Backtracking)
3. Phương pháp Chia để trị (Divide and Conquer)
4. Phương pháp Qui hoạch động (Dynamic Programming)
5. Phương pháp tham lam (Greedy Algorithms)
6. Phương pháp nhánh cận (Branch and Bound Algorithm)
7. *Phương pháp ngẫu nhiên (Randomized Algorithm)*

37