Giới thiệu

Các thuật toán sắp xếp

Nội dung trình bày

- Bài toán sắp xếp
- Tiếp cận sắp xếp đơn giản
  - Sắp xếp chọn
  - Sắp xếp chèn
  - Sắp xếp nổi bọt
- Tiếp cận sắp xếp độ phức tạp O(nlog(n))
  - Sắp xếp theo phân đoạn (Quick sort)
  - Sắp xếp hòa nhập
  - Sắp xếp vung đống
- Một số tiếp cận khác
  - Sắp xếp theo cơ số
  - Sắp xếp hòa nhập hai file lớn

Bài toán sắp xếp

- Cho một dãy dữ liệu có thể so sánh được (theo tiêu chí so sánh)
- Sắp xếp các phần tử mảng theo thứ tự (không giảm, không tăng)
- Ví dụ:
  - Cho danh sách các mức xám của các điểm ảnh: sắp xếp theo thứ tự không tăng của các mức xám
  - Cho danh sách sinh viên: sắp xếp sinh viên theo thứ tự không giảm theo ngày sinh

Đánh giá ứng dụng

- Tính ứng dụng:
  - Bài toán lựa chọn theo thứ tự nào đó là bài toán sắp xếp theo tiêu chí
  - Nhiều thuật toán thực hiện hiệu quả trên những bộ dữ liệu đã được sắp xếp theo xu hướng
- Đặc điểm
  - Phải có tiêu chí so sánh lớn hơn, bé hơn được
  - Có thể so sánh một số thành phần của đối tượng để xác định tiêu chí
  - Tính hiệu quả thuật toán phụ thuộc vào độ phức tạp của phép so sánh và hoán đổi vị trí
  - Một số thuật toán độ phức tạp về bộ nhớ cũng là vấn đề trong dữ liệu lớn

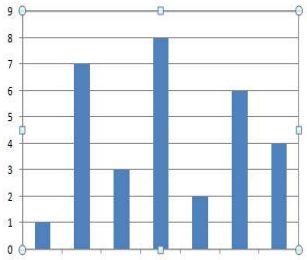
Phân loại theo độ phức tạp

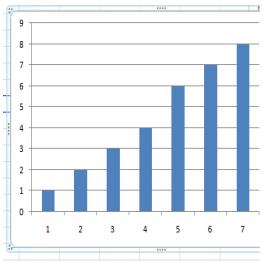
- Thuật toán đơn giản O(n2)
  - Sắp xếp chọn
  - Sắp xếp chèn
  - Sắp xếp nổi bọt
- Sắp xếp theo phương pháp chia để trị O(nlog(n))
  - Sắp xếp phân đoạn
  - Sắp xếp trộn
  - Sắp xếp vun đống
- Sắp xếp theo phương pháp O(n)
  - Sắp xếp theo cơ số

Sắp xếp chọn – selection sort

- Sắp xếp dãy không giảm
- 1 7 3 8 2 6 4

  1 2 3 4 6 7 8





Sắp xếp chọn (t)

- Ý tưởng
  - Chọn phần tử bé nhất đổi chỗ đưa lên đầu
  - Tiếp theo chọn phần tử bé thứ 2 đưa lên vị trí thứ hai
  - Chọn phần tử bé thứ k đưa đến vị trí thứ k
  - Lặp lại đến phần tử thứ N-1

Sắp xếp chọn (t)

- Phát biểu lại
  - Chọn phần tử bé nhất đổi chỗ đưa lên đầu
  - Giả sử có k phần tử ở đầu đã được sắp xếp
  - Tìm phần tử bé nhất từ k+1 đến n, đổi chỗ cho phần tử tại k+1.
  - Lặp tương tự cho đến phần tử N-1

Sắp xếp chọn (t)

- Thuật toán
  - Input: A[0..N-1]
  - Output: A[0..N-1] đã được sắp xếp không giảm

```
1. for i=0->N-2
    a. vt=i;
    b. for j=i+1->N-1
            if (a[j]<a[vt])
                vt=j;
    c. if(i!=vt)    swap(a[i],a[vt]);
```


Sắp xếp chọn (t)

- Thực hiện từng bước

i	vt	0	1	2	3	4	5	6
0	0	1	7	3	8	2	6	4
1	4	1	7	3	8	2	6	4
2	2	1	2	3	8	7	6	4
3	6	1	2	3	8	7	6	4
4	5	1	2	3	4	7	6	8
5	5	1	2	3	4	6	7	8

Sắp xếp chọn (t)

- Độ phức tạp
  - Số phép toán so sánh: N(N-1)/2 + N
  - Số phép toán gán chỉ số: N(N-1)/2 +N
  - Số phép gán giá trị phần tử: 3*(N-1)
  - Độ phức tạp là $O(n^2)$

Sắp xếp chèn - insertion sort

- Ý tưởng
  - Dựa trên ý tưởng việc sắp xếp quân bài
  - Chèn những quân bài đang cầm (xem xét) vào vị trí thích hợp
  - Ban đầu chỉ có một quân bài
  - Sau đó thêm các quân bài mới thì chèn quân bài đó vào vị trí thích hợp

Sắp xếp chèn (t)

- Ý tưởng
  - Xét mảng chỉ có phần tử - đã được sắp xếp
  - Mảng có k-1 phần tử đã được sắp xếp
  - Xét thêm phần tử thứ k (giá trị x) vào mảng trên
  - Xét từ vị trí k-1 đến đầu nếu các gặp phần tử lớn hơn x thì dịch phần tử đó về sau một ô
  - Gán giá trị x vào vị trí dịch chuyển tạo ra

Sắp xếp chèn (t)

- Thuật toán
  - Input: A[0..N-1] phần từ
  - Ouput: A[0..N-1] đã được sắp xếp không giảm

```
1. for i=1->N-1
    a. x=A[i];
    b. vt=i;
    c.  while (vt>0 && A[vt-1]>x)
            A[vt]=A[vt-1];
            vt--;
    d. A[vt]=x;
```

Sắp xếp chèn (t)

i	vt	0	1	2	3	4	5	6
1	1	1	7	3	8	2	6	4
2	1	1	7	3	8	2	6	4
3	3	1	3	7	8	2	6	4
4	1	1	3	7	8	2	6	4
5	3	1	2	3	7	8	6	4
6	3	1	2	3	6	7	8	4
		1	2	3	4	6	7	8

Sắp xếp chèn (t)

- Độ phức tạp thuật toán
  - Số phép so sánh N(N-1)/2
  - Số phép gán giá trị phần tử: N(N-1)+2N
  - Số phép gán chỉ số: N(N-1)
  - Độ phức tạp thuật toán $O(n^2)$

Sắp xếp nổi bọt - bubble sort

- Dựa trên ý tưởng về các bọt khí trong cốc bia
- Hai bọt khí cạnh nhau thì bọt lớn hơn sẽ nổi lên trên
- Đến khi không còn bọt khí nào trái quy luật đó thì các bọt khí đã được sắp xếp

Sắp xếp nổi bọt (t)

- Ý tưởng
  - Xét hai phần tử ở đầu tiên của dãy, nếu không đúng thứ tự đổi chỗ cho nhau
  - Tiếp tục xét các cặp đến cuối dãy
  - Lặp lại quá trình với cặp đầu dãy đến lúc không có cặp nào bị thay đổi

Sắp xếp nổi bọt (t)

- Thuật toán
- Input: A[0..N-1] phần tử
- Output: A[0..N-1] phần tử sắp xếp không giảm

```
1. for i=N-1 -> 1
       so_swap=0
           a. for j=0->i-1
                  if(a[j]>a[j+1])
                          swap(a[j],a[j+1]);
                   so_swap=so_swap+1;
              b. if(so_swap==0) break;
```


Sắp xếp nổi bọt (t)

i	j	0	1	2	3	4	5	6
6	1	1	7	3	8	2	6	4
6	3	1	3	7	8	2	6	4
6	4	1	3	7	2	8	6	4
6	5	1	3	7	2	6	8	4
5	2	1	3	7	2	6	4	8
5	3	1	3	2	7	6	4	8
5	4	1	3	2	6	7	4	8
4	1	1	3	2	6	4	7	8
4	3	1	2	3	6	4	7	8
		1	2	3	4	6	7	8

Sắp xếp nổi bọt (t)

- Độ phức tạp thuật toán
  - Số phép toán so sánh N(N-1)/2
  - Số phép toán gán 3(N)(N-1)/2
  - Độ phức tạp thuật toán $O(n^2)$

Nhận xét

- So sánh độ phức tạp thuật toán của ba thuật toán trên
  - Theo đánh giá chung
  - Đánh giá theo các tiêu chí
    - Số phép so sánh
    - Số phép gán dữ liệu
    - Số phép gán chỉ số

Nhận xét

- Nhìn chung ba thuật toán có độ phức tạp tương đương vì thế thời gian thực hiện là tương đương nhau
- Thực tế
  - Sự phức tạp của so sánh, phép gán, phép gán chỉ số là không giống nhau với các kiểu dữ liệu khác nhau (Ví dụ)

Nhận xét

- Thuật toán chọn, nổi bọt có thể chọn được k phần tử đứng đầu, hoặc cuối trong N phần tử mà không cần phải sắp xếp toàn bộ các phần tử (Ví dụ)
- Thuật toán sắp xếp chèn, nổi bọt phù hợp với hệ thống duy trì tính sắp xếp với dữ liệu thêm và bớt thường xuyên mà không phải sắp xếp lại toàn bộ (ví dụ)

Thảo luận

- Đánh trường hợp xấu nhất, tốt nhất
  - Chọn
  - Chèn
  - Nổi bọt

Thảo luận

- Trong trường hợp cho một dãy N phần tử đã sắp xếp, cần thêm M phần tử vào dãy

Thảo luận

- Thảo luận về tính ổn định thứ tự của các thuật toán
  - Các phần tử có cùng giá trị so sánh giữ nguyên thứ tự

Nội dung trình bày

- Bài toán sắp xếp
- Tiếp cận sắp xếp đơn giản
  - Sắp xếp chọn
  - Sắp xếp chèn
  - Sắp xếp nổi bọt

Bài tập trên lớp

- Sinh viên thực hiện các thuật toán với bộ dữ liệu
- 1 6 4 7 3 9 3

Bài tập

- Cài đặt thuật toán trên ngôn ngữ lập trình và chạy thử
- Thử nghiệm các thuật toán sắp xếp để đạt được dãy không tăng với các bộ dữ liệu sau
- 1 2 3 4 5 6
- 6 5 4 3 2 1
- 5 2 6 4 1 3
- Nhận xét trong trường hợp nào thuật toán nào thực hiện nhiều thao tác nhất (đâu là trường hợp tốt nhất, xấu nhất) và số thao tác trong trường hợp đó
- Trong trường hợp độ phức tạp của phép chuyển chỗ và so sánh là khác nhau thì sắp xếp nào tốt hơn, ngược lại.

Giới thiệu

Các thuật toán sắp xếp

Nội dung trình bày

- *Tiếp cận sắp xếp đơn giản*
  - *Sắp xếp chọn*
  - *Sắp xếp chèn*
  - *Sắp xếp nổi bọt*
- Tiếp cận sắp xếp độ phức tạp O(nlog(n))
  - Sắp xếp theo phân đoạn (Quick sort)
  - Sắp xếp hòa nhập
  - Sắp xếp vung đống
- *Một số tiếp cận khác*
  - *Sắp xếp theo cơ số*
  - *Sắp xếp hòa nhập hai file lớn*

Sắp xếp phân đoạn - quicksort

- Ý tưởng
  - Cho một dãy, chọn một phần tử ở giữa, chia đoạn thành 2 phần
  - Chuyển các phần tử nhỏ, hoặc bằng đến trước, các phần tử lớn hơn về sau
  - Sẽ được nửa đầu bé hơn nửa sau
  - Lặp lại việc chuyển đổi cho các phần tử nửa đầu, và nửa sau đến lúc số phần tử là 1

Sắp xếp phân đoạn – quicksort (t)

- Thuật toán ban đầu là chia: cố gắng chia thành hai đoạn khác nhau
- Trị: thực hiện các thuật toán sắp xếp trên các đoạn con
- Thực hiện kết hợp: thuật toán tự kết hợp kết quả

Sắp xếp phân đoạn – quicksort (t)

- Phân đoạn
  - Chọn một phần tử chốt x (đầu tiên)
  - Duyệt từ vị trí tiếp theo sang phải tìm vị trí phần tử đầu tiên >= x, i
  - Duyệt từ phải sang trái, tìm vị trí phần tử đầu tiên <x, j
  - Nếu i<j thì hoán đổi vị trí
  - Tiếp tục đến lúc j<i

Sắp xếp phân đoạn – quicksort (t)

- Thuật toán: partition
- Input: A[l..r], l,r: đoạn cần phân chia
- Ouput: A[l..r], i chi số phân chia

1. X=a[l]
2. i=l+1;
3. J=r;
4. While (i<j)
   a. While (i<j && a[i]<x) i++
   b. While (j>=i && a[j]>=x) j—
   c. If(i<j) swap(a[i],a[j])
5. Swap(a[l],a[j])
6. Return j;

Sắp xếp phân đoạn – quicksort (t)

- Partition

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

Sắp xếp phân đoạn – quicksort (t)

- Thuật toán: quicksort
- Input: A[l..r]: đoạn cần sắp xếp
- Ouput: A[l..r] đã sắp xếp

1. If(l>=r) return;
2. i=partition(A,l,r)
3. quicksort(A,l,i-1)
4. quicksort(A,i+1,r)

Sắp xếp phân đoạn – quicksort (t)

A	0	1	2	3	4	5	6
	3	1	7	8	2	6	9
Part	3	1	2	8	7	6	9
	2	1	3	8	7	6	9
Part	2	1		8	7	6	9
	1	2		6	7	8	9
Part	1			6	7		9
				6	7		
					7		
	1	2	3	6	7	8	9

Sắp xếp phân đoạn – quicksort (t)

- Đánh giá độ phức tạp
  - Số phép toán gán giá trị: 3 * n/2 * h
  - Số phép toán so sánh: n*h
  - Số phép toán gán chỉ số: n*h
- Trường hợp xấu nhất: h=n
- Trường hợp trung bình: h = log(n)
- Độ phức tạp trường hợp xấu nhất: $O(n^2)$
- Độ phức tạp trường hợp trung bình: O(nlog(n))

Sắp xếp trộn – mergesort

- Ý tưởng sắp xếp trộn
  - Nếu có hai dãy a và b đã được sắp xếp, tiến hành trộn hai dãy này thành dãy c đã được sắp xếp.
  - Nếu chia nhỏ mảng cần sắp xếp thành các đoạn 1 phần tử thì nó là đoạn được sắp xếp
  - Tiến hành ghép các đoạn nhỏ thành các đoạn lớn đã được sắp xếp

Sắp xếp trộn – mergesort

- Ý tưởng của thao tác trộn
  - Duyệt trên dãy a tại vị trí i
  - Duyệt trên dãy b tại vị trí j
  - Nếu a[i]>b[j] thì thêm b[j] và trong dãy c tăng biến j ngược lại thêm a[i] vào dãy và tăng biến i
  - Nếu một trong hai dãy hết trước tiến hành đưa toàn bộ dãy còn lại vào trong dãy c
  - Áp dụng trong trường hợp a, b là hai đoạn của mảng
    - a[l..t], a[t+1..r]
    - c[l..r]
  - Để thuận tiện trong xử lý tiến hành chuyển mảng đã sắp xếp về mảng a

Sắp xếp trộn – mergesort

- Thuật toán trộn – merge
  - Input: a[l..t], a[t+1..r] đã được sắp xếp không giảm
  - Ouput: a[l..r] được sắp xếp không giảm

1. i=l
2. j=t+1
3. p=l;

Sắp xếp trộn – mergesort

- Thuật toán trộn (t)

4. while (i<=t && j<=r)
   a. if(a[i]<a[j])
      c[p]=a[i]
      i++
   b. Else
      c[p]=a[j];
      j++
   c. p++

Sắp xếp trộn – mergesort

- Thuật toán trộn (t)

5. while (i<=t)

    c[p]=a[i]

    i++

    p++

6. while (j<=r)

    c[p]=a[j]

    j++

    p++

7. for (i=l; i<=r ;i++)

    a[i]=c[i];

Sắp xếp trộn – mergesort

	p	i	j	0	1	2	3	4	5	6
a		0	4	1	3	7	8	2	6	9
c	0			1						9
a		1	4		3	7	8	2	6	9
c	1			1	2					
a		1	5		3	7	8		6	9
c	2			1	2	3				
a		2	5			7	8		6	9
c	3			1	2	3	6			
a		2	6			7	8			9
c	4			1	2	3	6	7		
a		3	6				8			9
c	5			1	2	3	6	7	8	
a		4	6							9
c	6			1	2	3	6	7	8	9
a				1	2	3	6	7	8	9

Sắp xếp trộn – mergesort

- Thuật toán sắp xếp trộn mergesort
- Input: a[l..r]
- Ouput: a[l..r] đã được sắp xếp

1. if(l>=r) return ;
2. t=(l+r)/2
3. mergesort(l,t);
4. mergesort(t+1,r);
5. merge(a[l..t],a[t+1..r);

Sắp xếp trộn – mergesort

- Thuật toán sắp xếp trộn mergesort

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

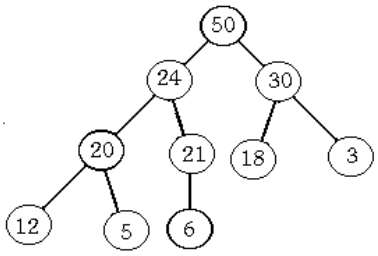
1	3	7	8	2	6	9
1	3	7	8	2	6	9
1	2	3	6	7	8	9

Sắp xếp trộn – mergesort

- Đánh giá độ phức tạp
  - Số phép so sánh: n*log(n)
  - Số phép gáp: 2*n*log(n)
  - Số phép gán chỉ số: 2*n
  - Độ phức tạp phép toán: O(nlog(n))

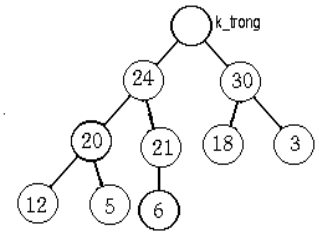
Sắp xếp vun đống – heapsort

- Dựa trên khái niệm cây nhị phân
  - Nếu xây dựng được cây nhị phân cực đại: phần tử trên cha lớn hơn hai con của nó
  - Xây dựng thuật toán duy trì đặc điểm này của cây

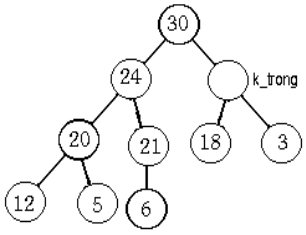


Sắp xếp vun đống – heapsort (t)

- Xây dựng thuật toán
  - Lấy phần tử lớn nhất khỏi cây, tiến hành xây dựng lại cây, với phần tử ở cuối được đưa lên đỉnh



Bước 1. Chọn max =50 là khoá ở gốc, k_trong ở gốc , key = 6



Bước 2. Dịch chuyển dần để tạo đống mới

Sắp xếp vun đống – heapsort (t)

- Biểu diễn dữ liệu
  - Mô tả cây có nhiều cấu trúc khác nhau nhưng trong bài toán này xét cấu trúc cây được lưu trên mảng
  - Nếu coi rằng tại vị trí a[0] là đỉnh của cây
  - a[1] sẽ là con trái, a[2] sẽ là con phải
  - A[3] con trái của a[1], a[4] con phải a[1]
  - a[5] con trái của a[2], a[6] con phải của a[2]
  - Con trái của phần tử a[i] là a[i*2+1]
  - Con phải của phần tử a[i] là a[i*2+2]

0	1	2	3	4	5	6	7	8	9
50	24	30	20	21	18	3	12	5	6

Sắp xếp vun đống – heapsort (t)

- Ý tưởng xây dựng đống
  - Một phần tử ban đầu là đống thỏa mãn điều kiện
  - Thêm một phần tử vào đống đã có
    - Thêm phần tử vào lá cuối của đống
    - Nếu phần tử đó lớn hơn cha của nó thì đảo vị trí
    - Lặp lại quá trình cho đến khi không còn cặp cha và con không đúng thứ tự

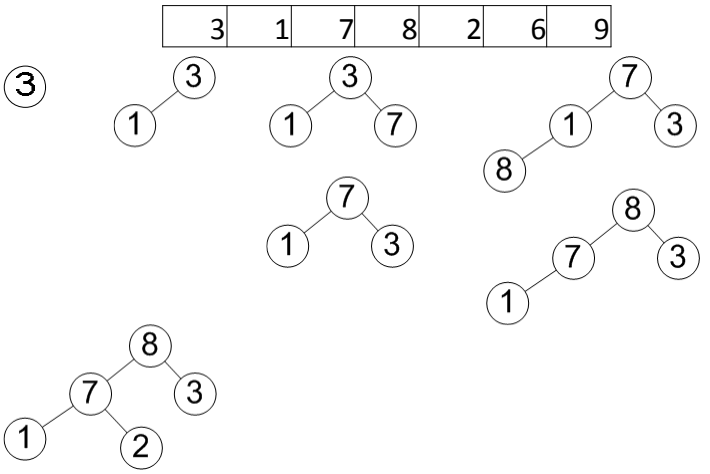
Sắp xếp vun đống – heapsort (t)

- Thuật toán buildheap
  - Input: a[0..N-1]
  - Ouput: a[0..N-1] tuân thủ qui tắc

1. for(i=1 -> N-1)
   a. x=a[i];
   b. r=i;
   c. while (r>0 && x>a[(r-1)/2])
      A[r]=a[(r-1)/2];
      R=(r-1)/2;
   d. a[r]=x;

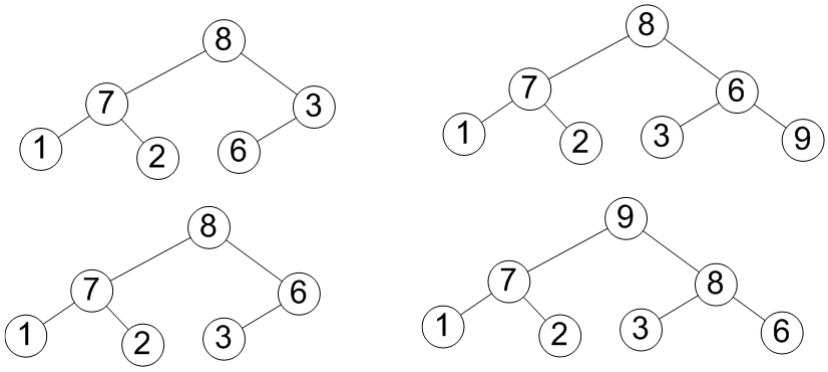
Sắp xếp vun đống – heapsort (t)

- Thử nghiệm



Sắp xếp vun đống – heapsort (t)

- Thử nghiệm



Sắp xếp vun đống – heapsort (t)

- Thử nghiệm

i	r	0	1	2	3	4	5	6
1		3	1	7	8	2	6	9
2	2	3	1	7	8	2	6	9
2		7	1	3	8	2	6	9
3	3	7	1	3	8	2	6	9
3	1	7	8	3	1	2	6	9
3		8	7	3	1	2	6	9
4		8	7	3	1	2	6	9
5	5	8	7	3	1	2	6	9
5		8	7	6	1	2	3	9
6	6	8	7	6	1	2	3	9
6	3	8	7	6	9	2	3	1
6	1	8	9	6	7	2	3	1
6		9	8	6	7	2	3	1

Sắp xếp vun đống – heapsort (t)

- Đánh giá phức tạp xây dựng đống
  - Số phép gán: N + Nlog(N)
  - Số phép so sánh: Nlog(N)
  - Số phép gán chỉ số: Nlog(N)
  - Độ phức tạp thuật toán: Nlog(N)

Sắp xếp vun đống – heapsort (t)

- Ý tưởng sắp xếp
  - Sau khi có đống thỏa mãn điều kiện
  - Lặp từ N-1 đến 1
    - Giá trị phần tử đang xét là x
    - Nhận giá trị từ đỉnh vào phần tử đang xét
    - Tiến hành chuyển xuống
      - Nếu x lớn hơn hai con của đỉnh thì gán x vào đỉnh, kết thúc
      - Nếu x bé hơn phần tử lớn nhất trong hai con của đỉnh thì đỉnh nhận giá trị đó và chuyển đến xét đỉnh con đó

Sắp xếp vun đống – heapsort (t)

- Thuật toán downheap
- Input: A[0..N-1] đống thỏa mãn điều kiện
- Output: A[0..N-1] đã sắp xếp

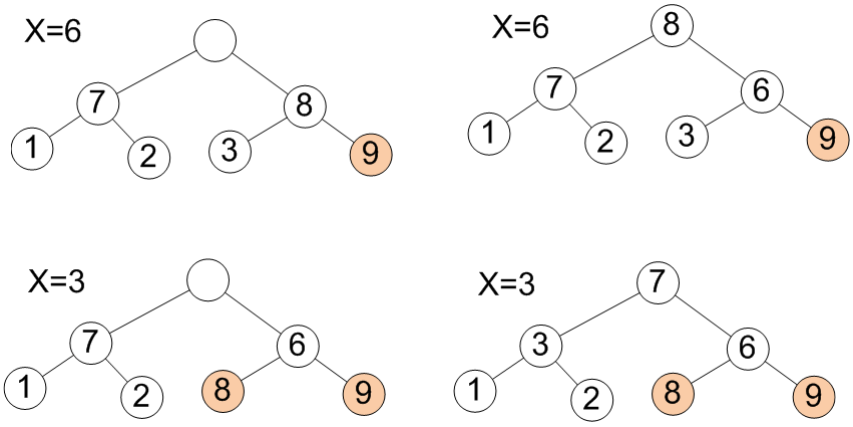
1. for(i=N-1->1)
   a. x=a[i]
   b. r=0
   c. j=r*2+1
   d. a[i]=a[0];

Sắp xếp vun đống – heapsort (t)

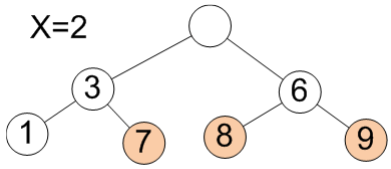
- Thuật toán downheap(t)
- Input: A[0..N-1] đống thỏa mãn điều kiện
- Output: A[0..N-1] đã sắp xếp

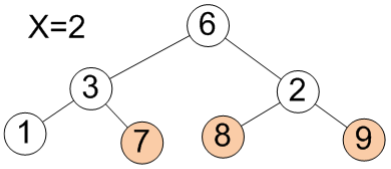
```
e. while (j<i)
       if(j+1<i && a[j+1]>a[j]) j++;
       if(x<a[j])
              a[r]=a[j]
              r=j;
              j=r*2+1
       else
       break;
 f. a[r]=x;
```

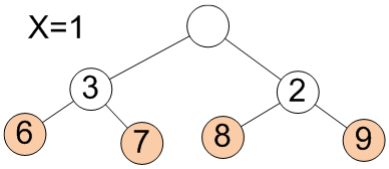

Sắp xếp vun đống – heapsort (t)

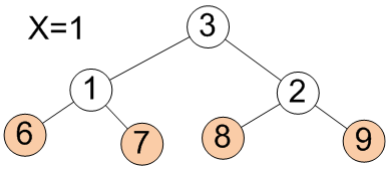


Sắp xếp vun đống – heapsort (t)

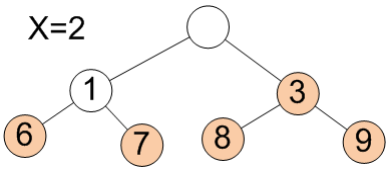


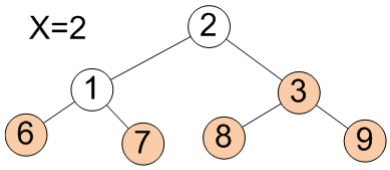


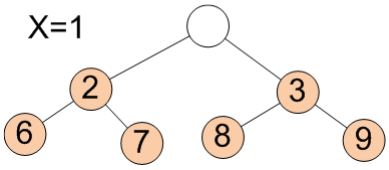


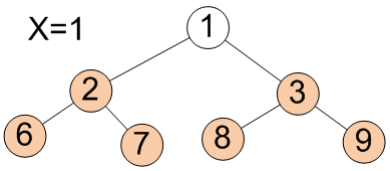


Sắp xếp vun đống – heapsort (t)









1	2	3	6	7	8	9

Sắp xếp vun đống – heapsort (t)

x	i	r	j	0	1	2	3	4	5	6
1	6	0	1		8	6	7	2	3	9
1	6	1	3	8		6	7	2	3	9
1	6	3		8	7	6	1	2	3	9
3	5	0	1		7	6	1	2	8	9
3	5	1		7	3	6	1	2	8	9
2	4	0	2		3	6	1	7	8	9
2	4	2		6	3	2	1	7	8	9
1	3	0	1		3	2	6	7	8	9
1	3	1		3	1	2	6	7	8	9
2	2	0		2	1	3	6	7	8	9
1	1	0		1	2	3	6	7	8	9

Sắp xếp vun đống – heapsort (t)

- Độ phức tạp thuật toán
- Số phép gán: nlog(n)
- Số phép so sánh: 2*nlog(n)
- Số phép toán chỉ số: 2*nlog(n)
- Độ phức tạp thuật toán: nlog(n)

Sắp xếp vun đống – heapsort (t)

- Thuật toán sắp xếp heapsort
- Input:a[0..N-1]
- Output: a[0..N-1] đã được sắp xếp

1. buildheap(a[0..N-1])
2. downheap(a[0..N-1])

Sắp xếp vun đống – heapsort (t)

- Đánh giá độ phức tạp
- Tổng cộng của hai thuật toán trên

Thảo luận

- Đánh giá trường hợp xấu nhất, tốt nhất
  - Quick sort
  - Heap sort
  - Merge sort

Thảo luận

- Xem xét tính ổn định thứ tự của các thuật toán

Thảo luận

- Mở rộng bài toán cho việc hòa nhập có nhiều hơn hai luồng dữ liệu

Bài tập trên lớp

- Sinh viên thực hiện với bộ dữ liệu
- 1 8 2 6 3 9 3

Nội dung

- Sắp xếp phân đoạn: quicksort
- Sắp xếp trộn: mergesort
- Sắp xếp vun đống: heapsort

Bài tập

- Cài đặt thuật toán trên ngôn ngữ lập trình và chạy thử
- Thử nghiệm các thuật toán sắp xếp để đạt được dãy không tăng với các bộ dữ liệu sau
- 1 2 3 4 5 6 7 8
- 8 7 6 5 4 3 2 1
- 5 2 6 4 8 3 7 1
- Nhận xét trong trường hợp nào thuật toán nào thực hiện nhiều thao tác nhất (đâu là trường hợp tốt nhất, xấu nhất) và số thao tác trong trường hợp đó

Giới thiệu

Các thuật toán tìm kiếm

Nội dung trình bày

- Bài toán tìm kiếm
- Tìm kiếm tuần tự, tìm kiếm nhị phân
  - Tìm kiếm tuần tự
  - Tìm kiếm nhị phân
- *Một số tiếp cận khác*
  - *Tìm kiếm dựa trên quy hoạch động*
  - *Tìm kiếm dựa trên đệ quy*
  - *Tìm kiếm dựa trên phân vùng*

Bài toán tìm kiếm

- Tìm kiếm một phương án, đáp án theo yêu cầu đầu vào
- Ví dụ:
  - Cho một danh sách xác định vị trí xuất hiện của một phần tử trong dãy
  - Tìm kiếm giải pháp lựa chọn để đạt giá trị cực đại trong bài toán cái túi
- Bài toán xảy ra trong hai tình huống
  - Dữ liệu xuất hiện được coi là ngẫu nhiên
  - Dữ liệu dữ liệu thỏa mãn một số ràng buộc nhất định

Tìm kiếm tuần tự

- Tìm kiếm với dữ liệu được không sắp xếp
- Ý tưởng
  - Bắt đầu duyệt từ phần tử 0 đến phần tử N-1
    - Nếu xuất hiện phần tử cần tìm kiếm ghi nhận vị trí xuất hiện
  - Thông tin trả về dựa theo tình trạng tìm thấy

Tìm kiếm tuần tự (t)

- Thuật toán
- Input: A[0..N-1], x cần tìm kiếm vị trí
- Output: Thông tin về vị trí

1. for(i=0->N-1)
    a. if(a[i]=x)
        break;
2. return i

Tìm kiếm tuần tự (t)

- Thực hiện

Tìm kiếm tuần tự (t)

- Đánh giá độ phức tạp
  - Số phép toán so sánh: n
  - Độ phức tạp thuật toán: O(n)

Tìm kiếm nhị phân

- Tìm kiếm trên dữ liệu đã được sắp xếp
- Ý tưởng
  - Thay vì tìm kiếm tuần tự sẽ tim kiếm dựa trên giá trị ở giữa dãy
    - Nếu chỉ số phải bé hơn chỉ số trái kết thúc
    - Nếu giá trị tìm kiếm trùng với giá trị ở giữa kết thúc
    - Nếu bé hơn thì tìm kiếm từ đầu đến phần tử trước phần tử hiển tại
    - Nếu lớn hơn phần tử ở giữa thì tìm kiếm từ phần tử sau phần tử ở giữa đến cuối dãy

Tìm kiếm nhị phân (t)

- Thuật toán
- Input: A[0..N-1], x cần tìm kiếm
- Output: thông tin về vị trí

```
1. vt=-1;
2. l=0;
3. r=N-1;
2. while(l<=r)
   a. k=(l+r)/2;
   b. if (a[k]=x)
        vt=k;
        break;
```


Tìm kiếm nhị phân (t)

```
   c. else
        if(a[k]<x)
              l=k+1;
        else
              r=k-1;
```

Tìm kiếm nhị phân (t)

- Thử nghiệm

	x=1									
vt	i	j	k	0	1	2	3	4	5	6
-1	0	6	3	1	2	3	6	7	8	9
-1	0	2	1	1	2	3				
0	0	0	0	1						
	x=4									
-1	0	6	3	1	2	3				
-1	0	2	1							
-1	2	2	2			3				
-1	3	2								

Tìm kiếm nhị phân (t)

- Đánh giá độ phức tạp
  - Số phép so sánh: 2*log(n)
  - Số phép gán chỉ số: 2*log(n)
  - Độ phức tạp thuật toán: O(log(n))
- Thuật toán này thực hiện dựa trên giả thiết là dữ liệu đã được sắp xếp
  - Nếu cần sắp xếp trước khi tìm kiếm thì độ phức tạp ít nhất là O(n+k), O(d*n) hoặc O(nlog(n)
  - Áp dụng trong tình huống dãy đã sắp xếp sẵn

Bài toán tìm kiếm mở rộng

- Tìm kiếm trên quy hoạch động
  - Bài toán cái túi cơ bản
- Tìm kiếm bằng đệ quy
  - Sử dụng thuật toán đệ quy cho bài toán cái túi
- Tìm kiếm phân vùng tìm kiếm
  - Phân tích quá trình chia vùng tìm kiếm với bài toán cái túi

Bài toán cái túi

- Tìm kiếm phương án lấy đồ cho cái túi
  - Một tên trộm mang túi có thể mang được trọng lượng là C
  - Đến một ngôi nhà có N vật, mỗi vật có trọng lượng là là w_i và có giá trị là p_i
  - Tìm các đồ vật mà tên trộm có thể lấy được mà có tổng giá trị lớn nhất

Bài toán cái túi

- Tiếp cận quy hoạch động
  - Dựa trên mô tả về U(k,i) = max(U(k-w_k)+p_k,U(k-1,i))
- Tiếp cận tổ hợp
  - Sử dụng các phương án có thể, kiểm tra lấy giá trị lớn nhất (sử dụng đệ quy)

Bài toán cái túi

- Thuật toán xây dựng phương án buildsolution
- Input: T, w[N], p[N]
- Output: Ma trận PA
- for(i=T->w[0])
  - PA[0,i]=p[0];
- For(i=0->w[0]-1)
  - PA[0,i]=0;

Bài toán cái túi

- Thuật toán xây dựng phương án buildsolution (t)
- For(i=1->N-1)
  - For(j=T->w[i])
    - PA[i,j]=max(PA[i-1,j], PA[i-1,j-w[i]]+p[i])
  - For(j=w[i]-1->0)
    - PA[i,j]=PA[i-1,j];

Bài toán cái túi

- Thuật toán xây dựng phương án getsolution
- Input: PA[N,T], w[N], p[N]
- Output: các vật cần lấy
- i=N
- j=T
- While(i>0 &&j>0)
  - If(PA[i,j]!PA[i-1,j])
    - Print(i)
    - J=j-w[i];
  - i=i-1;
- If(PA[i,j]!=0) print(i)

Bài toán cái túi

- Xây dựng bảng các phương án

T=19	wi	pi
1	3	7
2	4	10
3	5	20
4	7	19
5	6	13
6	9	40

Bài toán cái túi

- Xây dựng bảng các phương án

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	17	18	19
1	0	0	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
2	0	0	7	10	10	10	17	17	17	17	17	17	17	17	17	17	17	17
3	0	0	7	10	20	20	20	27	30	30	30	37	37	37	37	37	37	37
4	0	0	7	10	20	20	20	27	30	30	30	39	39	39	46	49	49	49
5	0	0	7	10	20	20	20	27	30	30	33	39	39	40	46	49	49	52
6	0	0	7	10	20	20	20	27	40	40	40	47	50	60	60	60	67	70

Bài toán cái túi

- Xây dựng bảng các phương án

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	17	18	19
1	0	0	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
2	0	0	7	10	10	10	17	17	17	17	17	17	17	17	17	17	17	17
3	0	0	7	10	20	20	20	27	30	30	30	37	37	37	37	37	37	37
4	0	0	7	10	20	20	20	27	30	30	30	39	39	39	46	49	49	49
5	0	0	7	10	20	20	20	27	30	30	33	39	39	40	46	49	49	52
6	0	0	7	10	20	20	20	27	40	40	40	47	50	60	60	60	67	70

Bài toán cái túi

- Tiếp cận đệ quy
  - Sinh tổ hợp để xét

Bài toán cái túi

- Phân tích xu hướng phân vùng để tìm kiếm với bài toán cái túi

Tìm kiếm đệ quy

- Dựa vào việc phân chia vùng tìm kiếm thành các vùng nhỏ
  - Chỉ giải quyết trong tình huống vùng tìm kiếm bé
  - Đệ quy cho các vùng tìm kiếm lớn
  - Sử dụng trạng thái, có thể lưu cả các giải pháp để xác định kết quả
- Ví dụ
  - Tìm kiếm nhị phân
  - Tìm kiếm tương tự nhi phân cho trường hợp không sắp xếp

Bài tập trên lớp

- Cho một mảng các phần tử
  - Tìm kiếm sự xuất hiện của phần tử x trong dãy

Bài tập trên lớp

- Cho một mảng tìm max bằng phương pháp đệ quy

Bài tập

- Cài đặt thuật toán trên ngôn ngữ lập trình và chạy thử
- Thử nghiệm các thuật toán sắp xếp để đạt được dãy không tăng với các bộ dữ liệu sau
- 5342 5435 7634 7632 3432 3232 3433 4534
- 5342 5342 5342 5342 5342 5342 5342 5342

Giới thiệu

Thao tác với danh sách

Nội dung trình bày

- Mô hình cấu trúc dữ liệu mảng
- Mô hình cấu trúc dữ liệu tự trỏ
  - Danh sách liên kết đơn
  - Danh sách liên kết vòng
  - Danh sách liên kết đôi
- Một số cấu trúc dữ liệu
  - Cấu trúc dữ liệu stack
  - Cấu trúc dữ liệu queue

Nội dung trình bày

- Mô hình cấu trúc dữ liệu mảng
- Mô hình cấu trúc dữ liệu tự trỏ
  - Danh sách liên kết đơn
  - Danh sách liên kết vòng
  - Danh sách liên kết đôi
- Một số cấu trúc dữ liệu
  - Cấu trúc dữ liệu stack
  - Cấu trúc dữ liệu queue

Cấu trúc dữ liệu mảng

- Là dãy các phần tử liên tiếp nhau trong bộ nhớ
  - Một mảng được trỏ bởi một con trỏ
  - Một mảng là mối khối nhớ liên tục
  - Truy xuất phần tử mảng là ngẫu nhiên (truy xuất đến phần tử theo chỉ số)
- Đặc trưng về quản lý
  - Mảng được cấp phát tại thời điểm khai báo
  - Không thay đổi được số lượng phần tử mảng tại thời điểm thực hiện
  - Cần khai báo lượng tối đa có thể cần phải lưu trữ

Cấu trúc dữ liệu mảng (t)

- Sử dụng con trỏ, và cấp phát động
  - Dữ liệu được cấp phát tại thời điểm hoạt động
  - Sự thay đổi về dung lượng bộ nhớ khó khăn

Cấu trúc dữ liệu mảng (t)

- Phù hợp
  - Không gian dữ liệu bé, ổn định
  - Cần phải tính toán với truy xuất phần tử là ngẫu nhiên
  - Ví dụ: sắp xếp đếm, sắp xếp nổi bọt, chọn, tìm kiếm nhị phân…
- Không phù hợp
  - Dữ liệu lớn, thay đổi thường xuyên về dung lượng
  - Xử lý theo phương thức tuần tự

Cấu trúc tự trỏ

- Cấu trúc tự trỏ đến chính bản thân nó

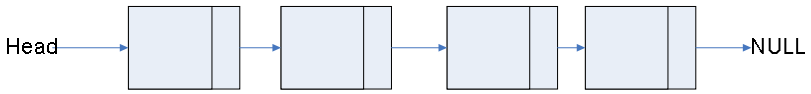
typedef struct {Tên_kiểu}
{
{Kiểu_1} {Tên_trường_1} ;
<Kiểu_2} {Tên trường_2} ;
……..
{Kiểu_n} {Tên_trường_n} ;
{ Tên_kiểu } *{Con_trỏ_tự_trỏ_1};
…
{ Tên_kiểu } *{Con_trỏ_tự_trỏ_n};
};

Cấu trúc tự trỏ (t)

```
typedef struct list{
   int data;
   list *next;
};
```

Danh sách liên kết đơn

- Mô hình



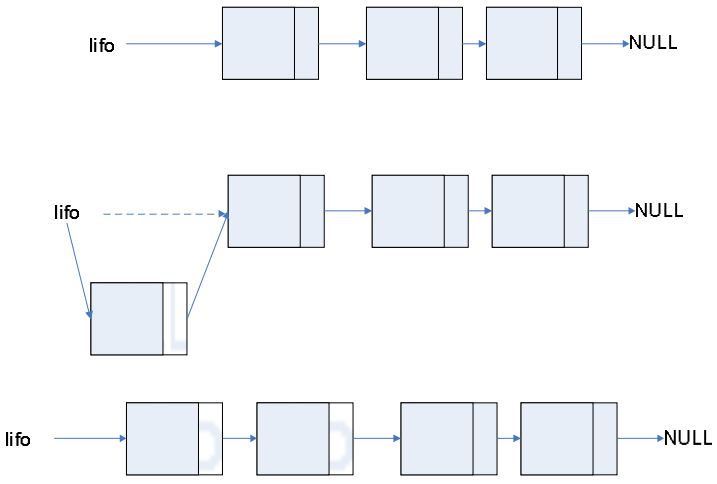
Danh sách liên kết đơn (t)

- Mô hình chức năng
  - Khởi tạo - init
  - Giải phóng danh sách - empty
  - Thêm phần tử (đầu, cuối) – addhead, addtail
  - Loại bỏ phần tử (đầu, cuối) – deletehead, deletetail
  - Tìm kiếm phần tử - search
  - Chèn phần tử ở sau - insert
  - Xóa phần tử -delete
  - Kiểm tra rỗng - isempty

Danh sách liên kết đơn (t)

- Void Init (list **head)
  - *head=null
- Int isempty(list *head)
  - If(head==null) return 0;
  - Return -1;
- list* search(list *head, int x)
  - t=head;
  - while(t!=null)
    - If(t.data==x) break;
    - T=t->next;
  - return t;

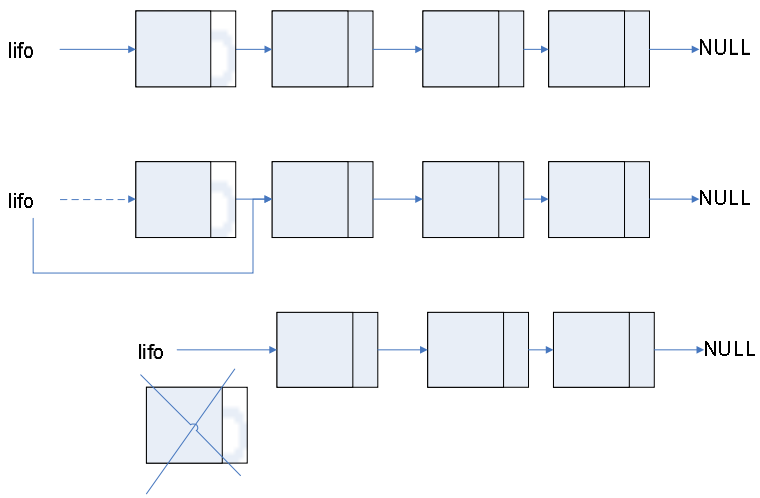
Danh sách liên kết đơn (t)



Danh sách liên kết đơn (t)

- Int addhead(list **head, int x)
  - list * T;
  - T=(list*) malloc(sizeof(list));
  - If(T==null)
    - Return -1;
  - T->data=x;
  - T->next=*head;
  - *Head=T;

Danh sách liên kết đơn (t)



Danh sách liên kết đơn (t)

- Int deletehead(list **head, int *x)
  - If(head==null)
    - Return -1;
  - T=*head;
  - *Head=t->next;
  - *X=t->data;
  - Free(t);
  - Return 0;

Danh sách liên kết vòng

- Thay vì phần tử ở đuôi chỉ đến null, danh sách liên kết vòng chỉ đến head;
  - Tạo vòng, mọi phần tử trong vòng có thể là đầu
  - Các thao tác cần kiểm tra với con trỏ head để biết kết thúc vòng
  - Phù hợp với dạng dữ liệu mô tả là vòng

Danh sách liên kết đôi

- Mỗi phần từ được định nghĩa có con trỏ left và right;
  - Con trỏ left chỉ về phần tử bên trái, right chỉ về phần tử phải
  - Mọi thao tác cần thực hiện với hai con trỏ
  - Cho phép duyệt theo chiều ngược và xuôi

Cấu trúc dữ liệu stack

- Khởi tạo – Init
- Đưa phần tử vào stack – push
- Lấy phần tử khỏi stack – pop
- Kiểm tra rỗng – isempty
- Lấy giá trị ở đỉnh stack - gettop

Cấu trúc dữ liệu stack (t)

- Triển khai trên mảng
  - Khai báo mảng đủ
  - Dùng chỉ số để quy định phần tử ở đỉnh
- Sử dụng danh sách liên kết
  - Init – init
  - Push-addhead
  - Pop-deletehead
  - Isempty – isempty

Cấu trúc dữ liệu stack (t)

- Int gettop(list *head, int *x)
  - If(head==null) return -1;
  - X=head->data;
  - Return 0;

Cấu trúc dữ liệu queue

- Khởi tạo – Init
- Đưa phần tử vào stack – put
- Lấy phần tử khỏi stack – get
- Kiểm tra rỗng – isempty

Cấu trúc dữ liệu queue (t)

- Triển khai dạng mảng
  - Sử dụng mảng với độ lớn chấp nhận được
  - Sử dụng hai con trỏ là đầu và đuôi để đưa vào và lấy ra
  - Do việc tăng liên tục nên cần kiểm tra tình huống là chỉ số đủ lớn thì quay lại bằng 0
- Triển khai dạng danh sách liên kết
  - Sử dụng hai con trỏ là head, tail để thêm vào lấy ra
  - Thêm bằng head, lấy ra bằng head

Cấu trúc dữ liệu queue (t)

- Init(list **head, **tail)
  - *Head=null
  - *Tail=null
- Isempty(list*head, *tail)
  - If(head==null)
    - Return 0
  - Return -1

Cấu trúc dữ liệu queue (t)

- Int Put(list **head, **tail, int x)
  - T=malloc(sizeof(list));
  - If(t==null)
    - Return -1;
  - T->data=x;
  - T->next=null;
  - If(*head==null)
    - *Head=t;
    - *Tail=t;
  - *Tail->next=t;
  - *Tail=t;
  - Return 0;

Cấu trúc dữ liệu queue (t)

- Int get(list **head, ** tail, int *x)
  - If(head==null)
    - Return -1;
  - T=*head;
  - *Head=*head->next;
  - If(*head==null) *tail=null;
  - *x=t->data;
  - Free(t);
  - Return 0;

Bài toán dò mìn

- Cho ma trận mxn ô, tại mỗi ô có giá trị 0 hoặc 1 – 0: không có mìn, 1: có mìn.
- Nếu ô (i,j) có mìn, bị kích nổ thì các ô (i-1,j), (i+1,j), (i,j-1), (i,j+1) nếu trong ma trận và có mìn sẽ bị kích nổ
- Cho ô bị kích nổ ban đầu là (i_0,j_0), tìm tất cả các ô bị kích nổ tiếp theo.

Bài toán đệ quy

- Bài toán dò min – đệ quy
- Depth=2;
- Min(A[n,m], i, j)
  - If(i<0 || i>=n || j<0 || j>=m)
    - Return ;
  - If(a[i,j]!=1)
    - Return
  - A[i,j]=depth; depth++;
  - Min(A[n,m],i-1,j)
  - Min(A[n,m],i+,j)
  - Min(A[n,m],i,j-1)
  - Min(A[n,m],i,j+1)

Dữ liệu thử nghiệm

- $(i_0,j_0)=(3,2)$

A	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	1	1	0	0	1
2	1	1	1	1	0	0
3	0	1	1	1	1	0
4	0	1	1	1	1	0
5	0	0	1	1	0	0
6	1	1	0	0	1	1
7	1	1	1	1	0	0

Bài toán dùng stack

- Depth=2;
- Min(A[n,m],l,k)
  - Push(l,k);
  - While (pop(i,j))
    - If(i<0 || i>=n || j<0 || j>=m)
      - continue;
    - If(a[i,j]!=1)
      - Continue;
    - A[i,j]=depth; depth++;
    - Push(i-1,j)
    - Push(i+,j)
    - Push(i,j-1)
    - Push(i,j+1)

Bài toán dùng queue

- Depth=2;
- Min(A[n,m],l,k)
  - put(l,k);
  - While (get(i,j))
    - If(i<0 || i>=n || j<0 || j>=m)
      - continue;
    - If(a[i,j]!=1)
      - Continue;
    - A[i,j]=depth; depth++;
    - put(i-1,j)
    - put(i+,j)
    - put(i,j-1)
    - put(i,j+1)

Bài toán nổ mìn

- So sánh kết quả
  - Thực hiện đệ quy – thực hiện stack là tương đương về thứ tự tiến hành
  - Thực hiện trên queue sẽ thấy sự khác biệt

Bài toán tính giá trị biểu thức

- Cho biểu thức dạng trung tố bao gồm số (toán hạng), các dấu +, -, *, / là toán tử, và dấu (,)
- Hãy tính giá trí của biểu thức đã cho
  - Ví dụ: 2+5*2-(3+2) sẽ tính được là 7

Bài toán tính giá trị biểu thức

- Chuyển biểu thức dạng trung tố sang dạng hậu tố
- Tính giá trị biểu thức từ biểu thức dạng hậu tố

Chuyển từ trung tố sang hậu tố

- Thuật toán chuyển từ trung tố sang hậu tố
  - Duyệt từ trái sang nếu gặp toán hạng đưa và chuỗi (mảng) đầu ra
  - Nếu gặp toán tử
    - Nếu lớn hơn toán tử trong stack thì đưa vào stack
    - Nếu bé hơn hoặc bằng thì lấy toán tử trong stack vào chuỗi đầu ra đến khi gặp toán tử lớn hơn; Đưa toán tử mới vào ngăn xếp
    - Hết chuỗi, lấy toàn bộ các toán tử đưa ra chuỗi đầu ra
- Độ ưu tiên của toán tử
  - *, / đến +,-
  - Nếu gặp (là lớn nhất trong toán tử)
  - Nếu gặp) sẽ lấy đến toán tử (đầu tiên

Không ngoặc

```
Trungto2hauto(A,n, B, m)
For(i=0->n-1)
    If(toanhang(A[i]))
        B=B+A[i]
    Else
        If(lonhon(A[i],top(s)))
            Push(s,A[i])
        Else
            While(!lonhon(A[i],top(s))
                Pop(s,v)
                B=B+v
            Push(s,A[i])
While not isempty(s) pop(s,v); B=B+v;
```


Có ngoặc

```
Trungto2hauto2(A,n, B, m)
For(i=0->n-1)
    If(toanhang(A[i] ))
        B=B+A[i]
    Else
      if(A[i]='(')
        push(s,A[i]); continue;
       if(A[i]=')')
        While(pop(s,v))
                if (v='(')
                break;
                B=B+v;
        Continue;
```

Có ngoặc (t)

```
Trungto2hauto2(A,n, B, m) – (t)
        If(lonhon(A[i],top(s)))
            Push(s,A[i])
        Else
            While(!lonhon(A[i],top(s))
                Pop(s,v)
                B=B+v
            Push(s,A[i])
    While not isempty(s) pop(s,v); B=B+v;
```


Tính giá trị biểu thức

- Thuật toán tính giá trị của hậu tố
  - Đọc từ trái sang phải
    - Nếu gặp toán hạng đưa vào stack
    - Nếu gặp toán tử (2 ngôi) lấy hai giá trị cuối cùng trên stack, thực hiện phép toán. Đưa giá trị tính được vào stack
  - Lặp đến hết chuỗi giá trị còn lại cuối cùng trong stack là giá trị của biểu thức

Tính giá trị biểu thức (t)

- Thuật toán tính biểu thức
- Tinh(B,m,V)

```
For(i=0->m-1)
    If(B[i] is toanhang)
        Push(s, B[i])
    Else
        Push(s,calc(pop(s),pop(s),b[i]));
V=pop(s);
```


Tính toán số lớn

- Sử dụng mảng để lưu các phần tử
- Mô hình chữ số khi dư sẽ tăng tiến từ phải sang trái
- Mô hình chỉ số mảng tăng từ trái sang phải
- Chuyển đổi cách lưu trữ để phù hợp mô hình thực tế

Phép cộng

- Cộng từ hàng đơn vị đến hàng chục hàng trăm…
- Đến khi gặp một số đã hết
- Cộng thêm số dư vào phần còn lại

Phép cộng (t)

- Int add(char *a, char *b, char *c, int na, int nb, int *nc)
  - D=0; nm=min(na,nb);
  - For(i=0;i<nm;i++)
    - C[i]=(a[i]+b[i]+d)%10;
    - D=(a[i]+b[i]+d)/10;
  - For(i=nm;i<na;i++)
    - C[i]=(a[i]+d)%10
    - D=(a[i]+d)/10
  - For(i=nm;i<nb;i++)
    - C[i]=(b[i]+d)%10
    - D=(b[i]+d)/10

Phép cộng (t)

- Int add(char *a, char *b, char *c, int na, int nb, int *nc) (t)
  - Nc=max(na,nb);
  - If(d>0)
    - nc++;
    - C[nc-1]=d;
  - Return nc;

Phép trừ

- Nếu số bị trừ bé hơn số trừ không thành công
- Nếu không trừ từ hàng đơn vị đến số lớn hơn,
- Nếu phép trừ không thành công thì nợ ở số trước

Phép trừ (t)

- Int sub(char *a, char *b, char *c, int na, int nb, int *nc)
- If(nb>na) return -1;
- D=0;
- For(i=0;i<nb;i++)
  - If(a[i]>=b[i]+d)
    - C[i]=a[i]-b[i]-d;
    - D=0
  - Else
    - C[i]=a[i]-b[i]-d+10;
    - D=1;

Phép trừ (t)

- Int sub(char *a, char *b, char *c, int na, int nb, int *nc) (t)
- If(d==1 && na=nb)
  - Return -1;
- While(i,na)
  - If(a[i]<d)
    - C[i]=a[i]+10-d;
    - D=1
  - Else
    - C[i]=a[i]-d;
    - D=0

Phép trừ (t)

- Int sub(char *a, char *b, char *c, int na, int nb, int *nc) (t)
- Nc=na;
- While(c[nc-1]==0 & nc>0)
  - Nc--;

Bài tập trên lớp

- Thực hiện triển khai stack bằng danh sách liên kết
- Thực hiện triển khai queue bằng danh sách liên kết

Nội dung trình bày

- Mô hình cấu trúc dữ liệu mảng
- Mô hình cấu trúc dữ liệu tự trỏ
  - Danh sách liên kết đơn
  - Danh sách liên kết vòng
  - Danh sách liên kết đôi
- Một số cấu trúc dữ liệu
  - Cấu trúc dữ liệu stack
  - Cấu trúc dữ liệu queue

Nội dung trình bày

- Mô hình cấu trúc dữ liệu mảng
- Mô hình cấu trúc dữ liệu tự trỏ
  - Danh sách liên kết đơn
  - Danh sách liên kết vòng
  - Danh sách liên kết đôi
- Một số cấu trúc dữ liệu
  - Cấu trúc dữ liệu stack
  - Cấu trúc dữ liệu queue

Bài tập

- Triển khai kiểu dữ liệu
  - Danh sách liên kết đơn
  - Danh sách liên kết vòng
  - Danh sách liên kết kép
- Stack
  - Mảng
  - Danh sách liên kết
- Queue
  - Mảng
  - Danh sách liên kết
- Triển khai chương trình cho các thuật toán đã đề cập
- Tìm hiểu cây