Cấu trúc dữ liệu và giải thuật

TS. Phạm Tuấn Minh
Khoa Công nghệ Thông tin, Đại học Phenikaa
minh.phamtuan@phenikaa-uni.edu.vn
https://sites.google.com/site/phamtuanminh/

Chương 4: Sắp xếp

- **Bài toán sắp xếp**
- Một số phương pháp sắp xếp cơ bản
- Sắp xếp QuickSort
- Sắp xếp HeapSort

Bài toán sắp xếp

- Sắp xếp (Sorting) là quá trình bố trí lại các phần tử của một tập đối tượng nào đó, theo một thứ tự ấn định. Chẳng hạn thứ tự tăng dần (hay giảm dần) đối với một dãy số, thứ tự từ điển đối với một dãy chữ...
- Yêu cầu về sắp xếp thường xuyên xuất hiện trong các ứng dụng tin học, với những mục đích khác nhau: sắp xếp dữ liệu lưu trữ trong máy tính để tìm kiếm cho thuận lợi, sắp xếp các kết quả xử lý để in ra trên bảng biểu
- Trong chương này ta chỉ xét tới các phương pháp sắp xếp trong (intemal sorting), nghĩa là các phương pháp tác động trên một tập các bản ghi lưu trữ đồng thời ở bộ nhớ trong

Chương 4: Sắp xếp

- Bài toán sắp xếp
- **Một số phương pháp sắp xếp cơ bản**
- Sắp xếp QuickSort
- Sắp xếp HeapSort

Sắp xếp kiểu lựa chọn

- Sắp xếp kiểu lựa chọn (selection sort)
  - Nguyên tắc cơ bản: Ở lượt thứ i (i = 1, 2, n), chọn trong dãy khoá K(i), K(i+l), ..., K(n) khoá nhỏ nhất và đổi chỗ nó với K(i)
  - Sau j lượt, j khóa nhỏ hơn lần lượt ở vị trí thứ nhất, thứ hai,... thứ j theo đúng thứ tự sắp xếp

1-5

Ví dụ

i	K(i)	Lượt 1	2	3	4	...	9
1	42	**11**	**11**	**11**	**11**		11
2	23	23	**23**	**23**	**23**		23
3	74	74	74	**36**	**36**		36
4	11	42	42	42	**42**		42
5	65	65	65	65	65		58
6	58	58	58	58	58		65
7	94	94	94	94	94		74
8	36	36	36	74	74		87
9	99	99	99	99	99		94
10	87	87	87	87	87		99

1-6

Select-sort(K, n)

```
for (i = 1; i < n; i++) {
    m = i;
    for (j = i+1; j < n; j++)
        if (K[j] < K[m]) m = j;
    if (m != i) {
        X = K [i];
        K [i] = K [m];
        K[m] = x;
    }
}
```

1-7

Sắp xếp kiểu thêm dần

- Sắp xếp kiểu thêm dần (insertion sort)
  - Nguyên tắc cơ bản:
    - Khi có i-1 phần tử đã được sắp xếp, nay thêm phần tử thứ i nữa thì sắp xếp lại như thế nào?
    - Có thể so sánh phần tử mới lần lượt với phần tử thứ (i-1), thứ (i-2)... để tìm ra "chỗ" thích hợp và "chèn" nó vào chỗ đó

1-8

Ví dụ

Lượt	1	2	3	4	.	8	9	10
Khoá đưa vào	42	23	74	11	.	36	99	87
1	42	**23**	23	**11**	.	11	11	11
2	-	42	42	23	.	23	23	23
3	-	-	**74**	42	.	**36**	36	36
4	-	-	-	74	.	42	42	42
5	-	-	-	-	.	58	58	58
6	-	-	-	-	.	65	65	65
7	-	-	-	-	.	74	74	74
8	-	-	-	-	.	94	94	**87**
9	-	-	-	-	.	-	**99**	94
10	-	-	-	-	.	-	-	99

Insert-sort(K, n)

```
K[0] = vô cùng bé
for (i = 2; i < n; i++) {
    X = K[i] ; j = i - 1;
    // Xác định chỗ cho khoá mới được xét và dịch chuyển các khoá cần thiết
    while (X < K[j]) {
        K[j+1] = K[j];
        j = j - 1;
    }

    //Đưa X vào đúng chỗ
    K[j + 1 ] = X;
}
```

Sắp xếp kiểu đổi chỗ

- Sắp xếp kiểu đổi chỗ (exchange sort)
  - Nguyên tắc cơ bản:
    - Bảng các khoá sẽ được duyệt từ đáy lên đỉnh. Dọc đường, nếu gặp hai khoá kế cận ngược thứ tự thì đổi chỗ chúng cho nhau.
    - Như vậy trong lượt đầu khoá có giá trị nhỏ nhất sẽ chuyển dần lên đỉnh. Đến lượt thứ hai khoá có giá trị nhỏ thứ hai sẽ được chuyển lên vị trí thứ hai...
  - Phương pháp này còn gọi là sắp xếp kiểu nổi bọt (bubble sort)

1-11

Ví dụ

i	K(i)	Lượt 1	2	3	4	...	9
1	42	11	11	11	11		11
2	23	42	23	23	23		23
3	74	23	42	36	36		36
4	11	74	36	42	42		42
5	65	36	74	58	58		58
6	58	65	58	74	65		65
7	94	58	65	65	74		74
8	36	94	87	87	87		87
9	99	87	94	94	94		94
10	87	99	99	99	99		99

1-12

Bubble-sort(K, n)

```
for (i = 1; i < n; i++)
    for (j = n; j > i; j--)
        if (K[j] < K[j-1]) {
            x  = K[j];
            K[j] = K[j-1];
            K[j-1]  = x;.
        }
```

1-13

So sánh ba phương pháp

- Thời gian chủ yếu phụ thuộc vào việc thực hiện các phép so sánh giá trị khoá và các phép chuyển chỗ bản ghi, khi sắp xếp
- Với n khá lớn, chi phí về thời gian thực hiện được đánh giá qua cấp độ lớn, thì cả ba phương pháp đều có cấp O(n^2)

1-14

Chương 4: Sắp xếp

- Bài toán sắp xếp
- Một số phương pháp sắp xếp cơ bản
- **Sắp xếp QuickSort**
- Sắp xếp HeapSort

1-15

Sắp xếp QuickSort

- Sắp xếp QuickSort
  - Chọn một khoá ngẫu nhiên nào đó của dãy làm "chốt" (pivot). Mọi phần tử nhỏ hơn khoá "chốt" phải được xếp vào vị trí ở trước "chốt" (đầu dãy), mọi phần tử lớn hơn khoá "chốt" phải được xếp vào vị trí sau "chốt" (cuối dãy).
    - Các phần tử trong dãy sẽ được so sánh với khoá chốt và sẽ đổi vị trí cho nhau, hoặc cho chốt, nếu nó lớn hơn chốt mà lại nằm trước chốt hoặc nhỏ hơn chốt mà lại nằm sau chốt.
    - Khi việc đổi chỗ đã thực hiện xong thì dãy khoá lúc đó được phân làm hai đoạn: một đoạn gồm các khoá nhỏ hơn chốt, một đoạn gồm các khoá lớn hơn chốt còn khoá chốt thì ở giữa hai đoạn nói trên, đó cũng là vị trí thực của nó trong dãy khi đã được sắp xếp, tới đây coi như kết thúc một lượt sắp xếp.
  - Ở các lượt tiếp theo cũng áp dụng một kỹ thuật tương tự đối với các phân đoạn còn lại

1-16

QUICK-SORT(K, LB, UB)

- Quy ước chọn khóa "chốt" là khóa đầu tiên của dãy
- LB là chỉ số của phần tử đầu của dãy khoá đang xét (biên dưới)
- UB là chỉ số của phần tử cuối của dãy khoá đó (biên trên)
- Còn K là vectơ biểu diễn dãy khoá cho
- j là chỉ số ứng với khoá chốt sau khi đã tách dãy khoá đang xét thành 2 phân đoạn.
- PART(K, LB, UB,j): Thủ tục phân đoạn dãy khóa K thành một đoạn gồm các khoá nhỏ hơn chốt, một đoạn gồm các khoá lớn hơn chốt còn khoá chốt thì ở giữa hai đoạn nói trên

QUICK-SORT(K, LB, UB)

```
if (LB < UB) {
    call PART(K, LB, UB,j);
    call QUICK_SORT(K, LB, j - 1);
    call QUICK - SORT(K, j + 1 , UB);
}
```

1-17

Thủ tục phân đoạn

- Đưa thêm vào một giá trị khoá giả K[n+1], lớn hơn mọi giá trị khoá trong dãy khoá cho. Nó sẽ đóng vai trò như một lính gác (khoá gác biên) để khống chế biên trên, giúp cho việc xử lý được thuận lợi.
- Quá trình tìm số nhỏ hơn chốt để chuyển về phía trước chốt và khoá lớn hơn chốt để chuyển về phía sau chốt sẽ dựa vào hai biến chỉ số i và j để duyệt qua dãy khoá theo chiều ngược nhau.

PART(K, LB, UB,j)

```
i = LB + 1; j = UB;
while (i <= j) {
    while (K[i] < K[LB]) i = i + 1;
    while (K[j] > K[LB]) j = j - 1;
    if (i < j) {
        < Đổi chỗ K[i] <-> K[j] >
        i = i + 1;
        j = j - 1;
    }
}
if (K[LB] > K[j]) <Đổi chỗ K[LB] <-> K[j]>
```

1-18

Ví dụ: Lượt phân đoạn đầu tiên

42 23 74 11 65 58 94 36 99 87

i j

42 23 36 11 65 58 94 74 99 87

i

j

11 23 36 42 65 58 94 74 99 87

Phân đoạn 1 Phân đoạn 2

1-19

Ví dụ: Kết quả sau từng lượt

	K(i) 42	23	74	11	65	58	94	36	99	87
i = 1	(11	23	36)	42	(65	58	94	74	99	87)
2	11	(23	36)	42	(65	58	94	74	99	87)
3	11	23	(36)	42	(65	58	94	74	99	87)
4	11	23	36	42	(58)	65	(94	74	99	87)
5	11	23	36	42	58	65	(94	74	99	87)
6	11	23	36	42	58	65	(87	74)	94	(99)
7	11	23	36	42	58	65	(74)	87	94	(99)
8	11	23	36	42	58	65	74	87	94	(99)
9	11	23	36	42	58	65	74	87	94	99

1-20

Thảo luận

- Vấn đề chọn chốt
- Vấn đề phối hợp với cách sắp xếp khác
- Đánh giá giải thuật: $O(n \log_2 n)$

1-21

Thảo luận

- Gọi T(n) là thời gian thực hiện giải thuật ứng với một bảng n khoá, P(n) là thời gian để phân đoạn một bảng n khoá thành hai bảng con. Ta có thể viết: T(n) = P(n) + T(j - LB) + T(UB-j)
- Chú ý rằng P(n) = C*n với c là một hằng số.
- Trường hợp xấu nhất xảy ra khi bảng khoá vốn đã có thứ tự sắp xếp: sau khi phân đoạn một trong hai bảng con là rỗng (j = LB hoặc j = UB).
- Giả sử j = LB, ta có:

  Tx(n) = P(n) + Tx(0) + Tx(n-1)   (Tx(0) = 0)
  = C*n + Tx(n-1)
  = C*n + C*(n-1) + Tx(n-2)
  ...
  = sum(k=1..n) (C*k + Tx(0))
  = C*n(n+1)/2 = O(n^2)

1-22

Thảo luận

- Trường hợp tốt nhất xảy ra khi bảng luôn luôn được chia đôi, nghĩa là j = (LB + UB) / 2

  Tt = P(n) + 2*Tt(n/2)

  = C*n + 2*Tt(n/2)

  = C*n + 2*C*(n/2) + 4*Tt(n/4) = 2*C*n + 2^2*Tt(n/4)

  = C*n + 2*C*(n/2) + 4*C*(n/4) + 8*Tt(n/8) = 3*C*n + 2^3*T(n/8)

  ...

  = $(\log_2 n)$*C*n + 2^$\log_2 n$*Tt(1)

  = O($n\log_2 n$).
- Giá trị thời gian tính trung bình, chứng minh được là O($n\log_2 n$)

1-23

Chương 4: Sắp xếp

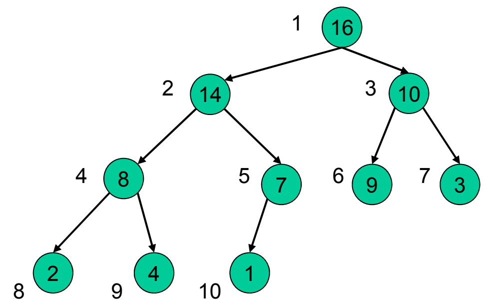
- Bài toán sắp xếp
- Một số phương pháp sắp xếp cơ bản
- Sắp xếp QuickSort
- **Sắp xếp HeapSort**

1-24

Định nghĩa "đống"

- Đống là một cây nhị phân hoàn chỉnh mà mỗi nút được gán một giá trị khoá sao cho khoá ở nút cha bao giờ cũng lớn hơn khoá ở nút con nó.
- Đống được lưu trữ trong máy bởi một vectơ K mà K[i] thì lưu trữ giá trị khoá ở nút thứ i trên cây nhị phân hoàn chỉnh, theo cách đánh số thứ tự khi lưu trữ kế tiếp.

- Lưu trữ kế tiếp:
  - Đánh số các nút trên cây theo thứ tự lần lượt từ mức 1 trở đi, hết mức này đến mức khác và từ trái sang phải đối với các nút ở mỗi mức
  - Quy luật: Con của nút thứ i là các nút thứ 2i và 2i+1, cha của nút j là nút floor(j/2)



K										
	16	14	10	8	7	9	3	2	4	1

1-25

Phép tạo đống

- Xét một cây nhị phân hoàn chỉnh có n nút, mà mỗi nút đã được gán một giá trị khoá. Cây này chưa phải là đống.
- Nhận xét:
  - Nếu một cây nhị phân hoàn chỉnh là đống: các cây con của các nút (nếu có) cũng là cây nhị phân hoàn chỉnh và cũng là đống.
  - Trên cây nhị phân hoàn chỉnh có n nút thì chỉ có $\lfloor n/2 \rfloor$ nút được là "cha"
  - Một nút lá bao giờ cũng có thể coi là đống
- Tạo đống từ đáy lên: Hãy tạo thành đống cho một cây nhị phân hoàn chỉnh có gốc được đánh số thứ tự là i, và gốc có 2 cây con đã là đống rồi

1-26

Phép tạo đống

- Vectơ K với n phần tử được coi như vectơ lưu trữ của một cây nhị phân hoàn chỉnh có n nút
- Thủ tục tạo đống ADJUST (i,n): Xét cây nút gốc là i

```
KEY = K[i]; //KEY nhận giá trị khoá ở nút gốc i
j = 2 * i;  //j ghi nhận số thứ tự nút con trái của nút i
while (j <= n) {
      if ( (j < n) && (K[j] <K [j +1]) ) j = j +1;
      //Nếu khoá con phải lớn hơn thì j ghi nhận số thứ tự của nó

      if (KEY > K[j]) { // TH KEY lớn hơn khoá con
            K[ floor(j/2) ] = KEY; // thì xác định được vị trí của KEY -> dừng lại
                                    // KEY sẽ ở vị trí nút cha của khóa con
            return;
      }
      // TH KEY nhỏ hơn khóa con -> Đưa khoá con lên
      K[ floor(j/2) ] = K[j];

      j = 2 * j; // Tiếp tục đi xuống nhánh này để tìm vị trí cho KEY
            // vì KEY có thể nhỏ hơn nút ở nhánh này
```

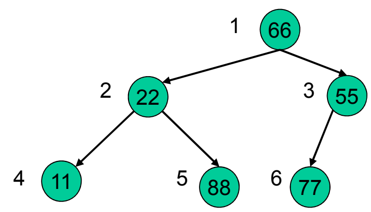

Bước tạo đống

- Tạo thành đống cho một cây nhị phân hoàn chỉnh có n nút: Thực hiện từ đáy lên

```
for (i = floor(n/2); i >=1; i--) ADJUST(i,n);
```

Ví dụ tạo đống

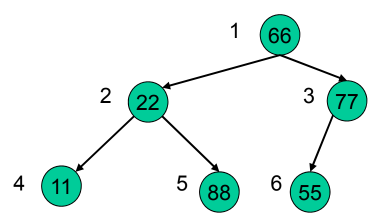
- Dữ liệu và cấu trúc lưu trữ ban đầu



K						
	66	22	55	11	88	77

Ví dụ tạo đống

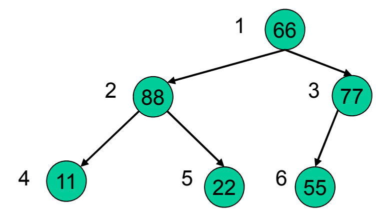
- Sau khi gọi ADJUST(3,6)



K						
	66	22	77	11	88	55

Ví dụ tạo đống

- Sau khi gọi ADJUST(2,6)

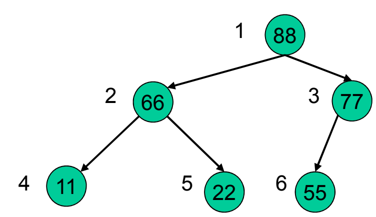


K						
	66	88	77	11	22	55

1-31

Ví dụ tạo đống

- Sau khi gọi ADJUST(1,6), cây đã là đống, khóa lớn nhất ở đỉnh đống



K						
	88	66	77	11	22	55

1-32

Sắp xếp HeapSort

- Sắp xếp kiểu vun đống bao gồm 2 giai đoạn:
  - Giai đoạn tạo đống ban đầu
  - Giai đoạn sắp xếp được thực hiện (n-1) lần, bao gồm 2 bước:
    - Vun đống
    - Đổi chỗ
- Khoá lớn nhất sẽ được xếp vào cuối dãy, nghĩa là nó được đổi chỗ với khoá đang ở "đáy đống", và sau phép đổi chỗ này một khoá trong dãy đã vào đúng vị trí của nó trong sắp xếp.
- Nếu không kể tới khoá này thì phần còn lại của dãy khoá ứng với một cây nhị phân hoàn chỉnh, với số lượng khoá nhỏ hơn 1, sẽ không còn là đống nữa, ta lại gặp bài toán tạo đống mới cho cây này (gọi là "vun đống") và lại thực hiện tiếp phép đổi chỗ giữa khoá ở đỉnh đống và khoá ở đáy đống tương tự như đã làm.v.v...
- Cho tới khi cây chỉ còn là 1 nút thì các khoá đã được xếp vào đúng vị trí của nó trong sắp xếp.

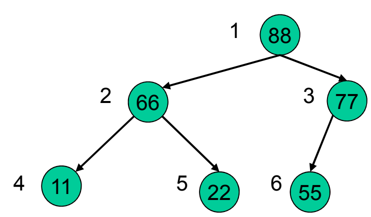
```
// Tạo đống
for (i = floor(n/2); i >= 1; i--)
    ADJUST(i,n);

// Sắp xếp
for (i = n - 1; i >= 1; i--) {
    < Đổi chỗ K[1] <-> K[i + 1] >
    ADJUST(1,i);
}
```

1-33

Ví dụ sắp xếp HeapSort

- Đống đã được tạo ra sau khi thực hiện giai đoạn 1



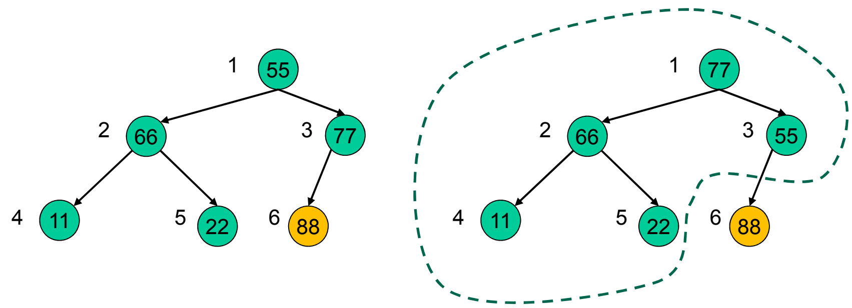
K

88	66	77	11	22	55

1-34

Ví dụ sắp xếp HeapSort

- Sau khi đổi chỗ và thực hiện ADJUST(1, 5)



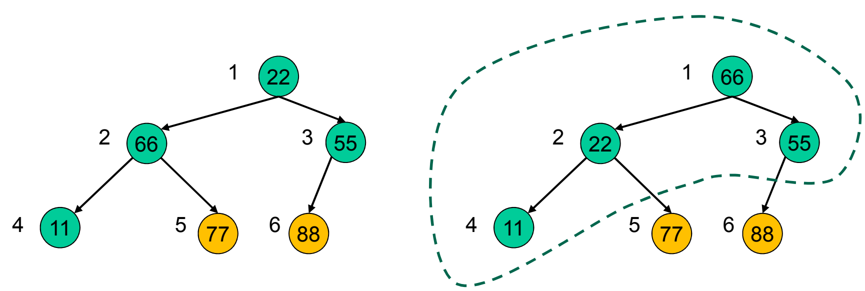
K

77	66	77	11	22	88

1-35

Ví dụ sắp xếp HeapSort

- Sau khi đổi chỗ và thực hiện ADJUST(1, 4)



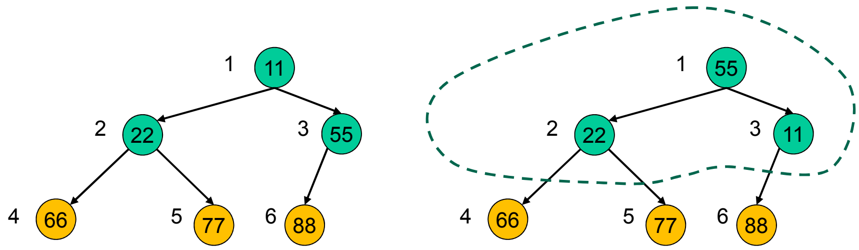
K

66	22	55	11	77	88

1-36

Ví dụ sắp xếp HeapSort

- Sau khi đổi chỗ và thực hiện ADJUST(1, 3)

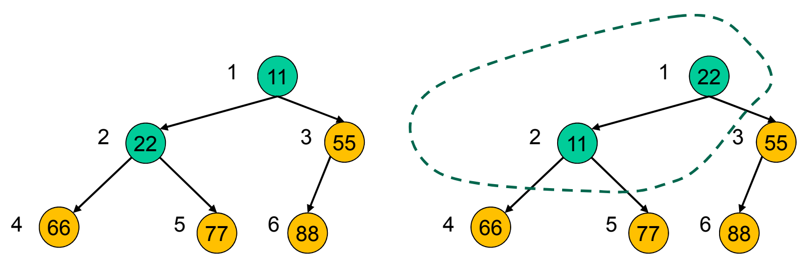


K						
K	55	22	11	66	77	88

1-37

Ví dụ sắp xếp HeapSort

- Sau khi đổi chỗ và thực hiện ADJUST(1, 2)

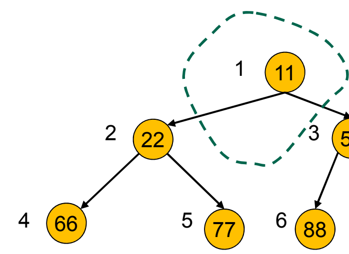


K						
K	22	11	55	66	77	88

1-38

Ví dụ sắp xếp HeapSort

- Sau khi đổi chỗ và thực hiện ADJUST(1, 1), lúc này các khóa đã được sắp xếp



K	11	22	55	66	77	88

1-39

Thảo luận

- Ở giai đoạn 1 (tạo đống ban đầu) có floor(n/2) lần gọi thực hiện ADJUST(i,n).
- Ở giai đoạn 2 (sắp xếp) thì phải gọi thực hiện (n-1) lần ADJUST(1,i).
- Như vậy có thể coi như phải gọi khoảng 3n/2 lần thực hiện giải thuật ADJUST mà cây được xét ứng với ADJUST thì nhiều nhất là n nút, nghĩa là chiều cao của cây lớn nhất cũng chỉ xấp xỉ $\log_2 n$.
- Số lượng phép so sánh giá trị khoá, khi thực hiện giải thuật ADJUST, cùng lắm cũng chỉ bằng chiều cao của cây tương ứng.
- Như vậy, trường hợp tồi nhất số lượng phép so sánh cũng chỉ xấp xỉ $(3/2)n\log_2 n$.

1-40

Thảo luận

- Thời gian thực hiện trung bình của HeapSort là $O(n \log_2 n)$
- Thời gian thực hiện trường hợp xấu nhất $O(n \log_2 n)$, là ưu điểm so với QuickSort

1-41

Cấu trúc dữ liệu và giải thuật

- Nội dung bài giảng được biên soạn bởi TS. Phạm Tuấn Minh.

1-42