Cấu trúc dữ liệu và Thuật giải

Sắp xếp

Bin Sorts

Bài 2- Các điểm trọng yếu

- **Quicksort**
  - **Sử dụng tốt cho hầu hết các hệ thống, (kể cả trường hợp hệ thống có thời gian thực hiện không bị ràng buộc)**
- **Heap Sort**
  - **Chậm hơn quick sort, nhưng bảo đảm $O(n \log n)$**
  - **Dùng cho các hệ thống thời gian thực (những hệ thống bị phê phán về thời gian thực hiện)**

Sắp xếp (Sorting)

- **Bây giờ chúng ta đã biết một vài thuật toán sắp xếp sau:**
  - **Selection** $O(n^2)$
  - **Insertion** $O(n^2)$
  - **Bubble** $O(n^2)$
  - **Heap** $O(n \log n)$ *Bảo đảm*
  - **Quick** $O(n \log n)$ *Thời gian thực hiện nhanh nhất!*
- **Liệu chúng ta có thể làm tốt hơn?**

Sắp xếp – Tốt hơn $O(n \log n)$?

- **Nếu tất cả chúng ta biết về trật tự sắp xếp của các khóa?**
  - **Không!**
- **Tuy nhiên,**
  - ***Nếu chúng ta tính toán được mỗi địa chỉ của từng khóa (thời gian là một hằng số)*** **thì**
    Thuật toán bin sort sẽ cung cấp hiệu suất tốt hơn.

Sắp xếp - Bin Sort

- Giả thiết
  - Tất cả các khóa nằm trong một miền giá trị nhỏ và xác định
    - *Ví dụ*
      - Các số nguyên thuộc 0-99
      - Các ký tự thuộc ‘A’-’z’, ‘0’-’9’
  - Ít nhất, có một số (chữ số) ứng với mỗi giá trị của khóa
- Bin sort
  - ✶ Cấp một túi (bin) để chứa mỗi giá trị của khóa
    - Thường là từng phần tử trong dãy số
  - ✷ Với mỗi số,
    - Trích chọn khóa
    - Tính toán số thứ tự của túi tương ứng đựng nó
    - Đặt nó vào trong túi
  - ✹ Kết thúc!

Sắp xếp - Bin Sort: Phân tích

- Tất cả các khóa đều nằm trong một miền giá trị nhỏ và xác định.
  - Có m giá trị khóa tiềm năng
- Ít nhất, có một giá trị số ứng với mỗi khóa

- Bin sort
  - ✶ Cấp phát một túi (bin) cho mỗi giá trị của khóa $O(m)$
    - Thường là từng phần tử trong mảng
  - ✸ Với từng giá trị số, n lần
    - Trích chọn khóa $O(1)$
    - Tính toán số thứ tự của túi tương ứng $O(1)$
    - Gán nó vào trong túi $O(1) \times n \leftarrow O(n)$
  - ✹ Kết thúc! $O(n) + O(m) = O(n+m) = O(n)$ if $n >> m$

Trạng thái khóa

Sắp xếp - Bin Sort: Caveat

- **Miền xác định của khóa**
  - **Tất cả các khóa đều nằm trong một đoạn nhỏ và xác định**
    - **Có m giá trị khóa tiềm năng**
  - **Nếu điều kiện này không thích hợp, *VD:* $m >> n$, thì bin sort là $O(m)$**
- **Ví dụ**
  - **Khóa là một số nguyên 32-bit, $m = 2^{32}$**
  - **Rõ ràng, không có cách nào để sắp xếp ít nhất 1000 số nguyên.**
  - **Hơn nữa, chúng tôi không đủ không gian cho các túi (bin)*!***
- **Bin sort đánh đổi không gian cho tốc độ!**

Sorting - Bin Sort với các bản sao

Nới lỏng?

- Có ít nhất một miền nhỏ ứng với mỗi giá trị của khóa

- Bin sort
  - ✶ Cấp phát một bin cho mỗi giá trị của khóa $O(m)$
    - Thường là một phần tử trong một mảng
    - Mảng của các danh sách phần tử
  - ✸ Với mỗi số (chữ số - item), n lần
    - Trích chọn khóa $O(1)$
    - Tính toán số thứ tự túi (bin) tương ứng $O(1)$
    - Bổ xung nó vào danh sách $O(1) \times n \leftarrow O(n)$
    - **Ghép danh sách** $O(m)$
    - **Kết thúc!** $O(n) + O(m) = O(n+m) = \boxed{O(n)}$ *if* $n >> m$

Sorting – Tổng quát của Bin Sort

- **Radix sort**
  - **Bin sort trong các giai đoạn**
  - **Ví dụ** `36 9 0 25 1 49 64 16 81 4`

  - **Giai đoạn 1 – Sắp xếp dựa vào số có ý nghĩa tối thiểu**

0	1	2	3	4	5	6	7	8	9
0	1			64	25	36			9
	81			4		16			49

Sorting - Generalised Bin Sort

- **Radix sort - Bin sort trong các giai đoạn**
  - **Giai đoạn 1 – Sắp xếp dựa vào số có ý nghĩa tối thiểu**

0	1	2	3	4	5	6	7	8	9
0	1 81			64 4	25	36 16			9 49

  - **Giai đoạn 2 – Sắp xếp dựa vào phần lớn số có ý nghĩa**

0	1	2	3	4	5	6	7	8	9
0									

Sorting - Generalised Bin Sort

- **Radix sort - Bin sort trong các giai đoạn**
  - **Giai đoạn 1 – Sắp xếp dựa vào số có ý nghĩa tối thiểu**

0	1	2	3	4	5	6	7	8	9
0	1 81			64 4	25	36 16			9 49

  - **Giai đoạn 2 – Sắp xếp dựa vào phần lớn số có ý nghĩa**

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

Cẩn thận khi bổ Xung sau khi đã tồn tại các số trong Bin!

Sorting - Generalised Bin Sort

- **Radix sort - Bin sort trong các giai đoạn**
  - **Giai đoạn 1 – Sắp xếp dựa vào số có ý nghĩa tối thiểu**

0	1	2	3	4	5	6	7	8	9
0	1			64	25	36			9
	81			4		16			49

  - **Giai đoạn 2 – Sắp xếp dựa vào phần lớn số có ý nghĩa**

0	1	2	3	4	5	6	7	8	9
0								81	
1									

Sorting - Generalised Bin Sort

- **Radix sort - Bin sort trong các giai đoạn**
  - **Giai đoạn 1 – Sắp xếp dựa vào số có ý nghĩa tối thiểu**

0	1	2	3	4	5	6	7	8	9
0	1 81			64 4	25	36 16			9 49

  - **Giai đoạn 2 – Sắp xếp dựa vào phần lớn số có ý nghĩa**

0	1	2	3	4	5	6	7	8	9
0 1						64		81	

Sorting – Tổng quát của Bin Sort

- **Radix sort - Bin sort trong các giai đoạn**
  - **Giai đoạn 1 – Sắp xếp dựa vào số có ý nghĩa tối thiểu**

0	1	2	3	4	5	6	7	8	9
0	1 81			64 4	25	36 16			9 49

  - **Giai đoạn 2 – Sắp xếp dựa vào phần lớn số có ý nghĩa**

0	1	2	3	4	5	6	7	8	9
0 1 4						64		81	

Sorting - Generalised Bin Sort

- **Radix sort -Bin sort trong các giai đoạn**
  - **Giai đoạn 1 – Sắp xếp dựa vào số có ý nghĩa tối thiểu**

0	1	2	3	4	5	6	7	8	9
0	1			64	25	36			9
	81			4		16			49

  - **Giai đoạn 2 – Sắp xếp dựa vào phần lớn số có ý nghĩa**

0	1	2	3	4	5	6	7	8	9
0	16	25	36	49		64		81	
1									
4									
9									

Chú ý rằng: Bin 0 phải thực sự lớn

Sorting - Generalised Bin Sort

- **Radix sort - Bin sort trong các giai đoạn**
  - **Giai đoạn 1 – Sắp xếp dựa vào số có ý nghĩa tối thiểu**

0	1	2	3	4	5	6	7	8	9
0	1			64	25	36			9
	81			4		16			49

  - **Giai đoạn 2 – Sắp xếp dựa vào phần lớn số có ý nghĩa**

0	1	2	3	4	5	6	7	8	9
0	16	25	36	49		64		81	
1									
4									
9									

Không gian cần thiết cho mỗi giai đoạn là bao nhiêu?

n **items**

m **bins**

Sorting – Tổng quát của Bin Sort

- **Radix sort – Phân tích**
  - **Giai đoạn 1 – Sắp xếp dựa vào số có ý nghĩa tối thiểu**
    - **Tạo m bins** $O(m)$
    - **Cấp phát n items** $O(n)$
  - **Giai đoạn 2**
    - **Tạo m bins** $O(m)$
    - **Cấp phát n items** $O(n)$
  - **Cuối cùng**
    - **Liên kết m bins** $O(m)$
  - **Tất cả các bước thực hiện tuần tự, vì thế bổ xung**
  - **Tổng** $O(3m+2n)$ ➔ $O(m+n)$ ➔ $O(n)$ *cho* $m<<n$

Sorting - Radix Sort – Phân tích

- **Radix sort – Tổng quát**
  - **Về cơ bản: mỗi giai đoạn có thể phù hợp với các kiểu dữ liệu khác nhau.**
  - **Các số nguyên**
    - **Các giá trị cơ sở 10, 16, 100, …**
  - ***Các thành phần dữ liệu không cùng kiểu dữ liệu***

```
struct date {
    int day;   /* 1 .. 31 */
    int month; /* 1 .. 12 */
    int year;  /* 0 .. 99 */
    }
```

G đ 1 - s_1=31 bins
G đ 2 - s_2=12 bins
G đ 3 - s_3=100 bins

- **Vẫn là $O(n)$ nếu $n >> s_i$ với mọi i**

Radix Sort - Analysis

- **Generalised Radix Sort Algorithm**

`radixsort( A, n ) {`		
	`for(i=0;i<k;i++)` **For each of k radices** `for(j=0;j<s[i];j++) bin[j] = EMPTY;`	$O(s_i)$

	`for(j=0;j<n;j++) {`
	`move A[i]`
	`to the end of bin[A[i]->fi]`
	`}`
		$O(n)$

	`for(j=0;j<s[i];j++)`
	`concat bin[j] onto the end of A;`
	`}`
		$O(s_i)$

	`}`	

Radix Sort - Analysis

- **Generalised Radix Sort Algorithm**

`radixsort( A, n ) {`
`  for(i=0;i<k;i++) {`
`  for(j=0;j<s[i];j++) bin[j] = EMPTY;`
	$O(s_i)$

`  for(j=0;`
`  move A[j]`
`  to the end of bin[A[i]->fi]`
`  }`
	$O(n)$

`  for(j=0;j<s[i];j++)`
`  concat bin[j] onto the end of A;`
`  }`
`}`
	$O(s_i)$

Clear the s_i bins for the i^{th} radix

Radix Sort - Analysis

- **Generalised Radix Sort Algorithm**

`radixsort( A, n ) {`
`  for(i=0;i<k;i++) {`
`  for(j=0;j<s[i];j++) bin[j] = EMPTY;`
	$O(s_i)$

`  for(j=0;j<n;j++) {`
`  move A[i]`
`  to the end of bin[A[i]->fi]`
`  }`
	$O(n)$

`  for(j=0;`
`  conca`
`  }`
`}`
	$O(s_i)$

Move element `A[i]` to the end of the bin addressed by the i^{th} field of `A[i]`

Radix Sort - Analysis

- **Generalised Radix Sort Algorithm**

`radixsort( A, n ) {`
`  for(i=0;i<k;i++) {`
`  for(j=0;j<s[i];j++) bin[j] = EMPTY;`
	$O(s_i)$

`  for(j=0;j<n;j++) {`
`  move A[i]`
`  to the en`
`  }`
	$O(n)$

`  for(j=0;j<s[i];j++)`
`  concat bin[j] onto the end of A;`
`  }`
`  }`
	$O(s_i)$

Concatenate s_i bins into one list again

Radix Sort – Phân tích

- **Total**
  - ***k* vòng lặp, $2s_i + n$ cho mỗi vòng**

$$\sum_{i=1}^{k} O(s_i + n) = O(kn + \sum_{i=1}^{k} s_i)$$

$$= O(n + \sum_{i=1}^{k} s_i)$$

  - **Như vậy *k* là một hằng số**
  - ***Tổng quát,* Nếu keys thuộc $(0, b^k-1)$**
    - **Keys là những số *k*-digit base-*b***

    ←$s_i = b$ for all *k*

←Độ phức tạp $O(n+kb) = O(n)$

Radix Sort – Phân tích

? ***Bất cứ tập key nào cũng có thể ánh xạ về (0, b^k-1)***

! **Như vậy chúng ta thường xuyên đạt độ phức tạp sắp xếp $O(n)$?**

- **Nếu k là một hằng số, *Đúng như vậy.***

Radix Sort – Phân tích

- Nhưng, nếu k được phép tăng cùng với n

  Vd nó lấy $\log_b n$ các số có b chữ số cơ sở để biểu diễn n

- *Như vậy chúng ta có:*
- $k = \log n, \quad s_i = 2$ *(say)*

➔

$$\sum_{i=1}^{\log n} O(2 + n) = O(n \log n + \sum_{i=1}^{\log n} 2)$$

$$= O(n \log n + 2 \log n)$$
$$= O(n \log n)$$

➔Sắp xếp Radix không tốt hơn so với quicksort

Radix Sort – Phân tích

- **Radix sort không tốt hơn quicksort**
  - **Một cách nhìn nhận khác là:**
    - **Chúng tôi có thể giữ k constant như n *lần lặp if chúng tôi cho phép sao chép keys***
      - **keys nằm trong $(0, b^k)$, $b^k < n$**
  - ***nhưng* nếu các keys phải là duy nhất, thì k phải tăng theo n**
- ***Với hiệu suất $O(n)$, Các keys hải nằm trong một miền giới hạn xác định.***

Radix Sort - Realities

- **Radix sort sử dụng nhiều bộ nhớ**
  - $n\ s_i$ **vùng định vị cho mỗi giai đoạn**
  - **Trong tực tế, điều này sẽ rất khó để đạt được** $O(n)$
  - **Chi phí quản lý bộ nhớ ảnh hưởng nhiều đến lợi ích**
  - **Thách thức:**
    - **Thiết kế một radix sort tổng quát, nó chạy nhanh hơn so với `qsort` trên SGIs!**
    - **Chư ý: Bạn cần phải có những thuật toán hiệu quả về cấp phát, định vị bộ nhớ!**

BIN SORTS – Điểm chủ yếu

- **Bin Sorts**
  - ***If*** **tồn tại một hàm chuyển đổi một khóa về một địa chỉ tương ứng (*vd* một số nguyên nhỏ)**

    and **số lượng các địa chỉ (= sô lượng bins) là không lớn lắm**

    then **chúng ta đạt được độ phức tạp sắp xếp $O(n)$**

    … Nhưng nhớ rằng thực sự nó là $O(n + m)$
  - **Số lượng bins, m, phải là một hằng số và nhỏ (*constant and small).***