CE2101/ CZ2101: Algorithm Design and Analysis

Week 4: Review Lecture

Ke Yiping, Kelly

Content

- Comparison of Sorting Algorithms
- Greedy Strategy
- Dijkstra's Algorithm

Comparison of Sorting Algorithms

Time complexity comparison:

	Best	Average	Worst
Insertion Sort	n	n2	n2
Mergesort	n log n	n log n	n log n
Quicksort	n log n	n log n	n2
*Radix	n	n	n
Heapsort	n log n	n log n	n log n

* Radix sort is not required

Empirical Performance

Compared by time (in milliseconds)

Insertion Sort	0.1	168	342	23,382
Mergesort	2.0	2.3	2.2	30
Quicksort	0.7	0.9	0.7	12
*Radix	1.6	1.6	1.6	18
Heapsort	3.4	3.5	3.6	49

best performance

umbers

Reference: Shaffer, C. A. (2001). *A practical introduction to data structures and algorithm analysis*. Upper Saddle River, NJ: Prentice Hall.

'**UP**' and '**DOWN**' columns show the performance for inputs of size 10,000 where the numbers are in ascending (sorted) and descending (reversely sorted) order. Figures are timings obtained using workstation running UNIX.

Greedy Strategy

- To solve optimization problems: min or max
- Make a best choice at each step by using only knowledge available at the time.
- The choice is not expensive and cannot be undone.
- Global optimum may not be guaranteed.

Dijkstra's Algorithm

- Single source shortest path on a weighted, directed graph.
- All weights must be nonnegative.

Dijkstra's: Basic Steps

store direct parent

1. Initialise **d** and **pi**
2. Set **S** to empty
3. While there are still vertices in **V** − **S**
   i. step 1: Move ***u***, the vertex in **V** − **S** that has the shortest path estimate from source (smallest d), to **S**
   ii. step 2: For all the neighbors of ***u*** in **V** − **S**, update their estimates of shortest distances to the source (d)

Exercise

- Suppose our goal is to sort an array in ascending order, and in the input array A of size 600 there are no more than 10 inversions, i.e. pairs of array indices (i, j) such that $i < j$ and $A[i] > A[j]$. Which of the sorting algorithms learnt in the lectures would you use to sort array A? Briefly justify your answer. **[AY1718S1]**

Insertion. (1. Suitable for almost sorted array)
(2. 600 is considered small input size).

Set C

number in A and B are unique.

- The *union* of two sets *A* and *B* is a set *C* such that an element *x* is in *C* if and only if *x* is in *A* or *B*. Note that, if *x* is in both *A* and *B*, then *x* will appear in the union *C* exactly once. Suppose that sets *A* and *B* are represented by two arrays, and both arrays have been sorted in ascending order. The sum of their sizes $|A| + |B|$ is equal to *n*. Design an algorithm to compute the union of sets *A* and *B* in time $O(n)$. Briefly describe your algorithm and analyse its worst-case time complexity. **[AY1718S1]**

similar to merge function.

worst case: n elements → n-1 compares

* if have duplicates, can screen once first

Exercise

- Describe a linear-time algorithm to rearrange an array of *n* integers so that all the negative keys will be moved to the left side of all the nonnegative keys. You should not use any auxiliary array for temporary storage of keys. Show that the running time of your algorithm is $O(n)$. **[AY1617S1]**

辅助的 array

similar to partition function

① last_negative = -1 && use 0 as pivot value

最后不把0放进队列中

Exercise

- A weighted directed graph *G* is represented by the array of adjacency lists shown below. List the shortest path from vertex b to each vertex, obtained by Dijkstra's algorithm when running on G. **[AY1819S1]**

a: → **(d, 1)** → **(b, 2)**

b: → **(d, 3)** → **(e, 10)**

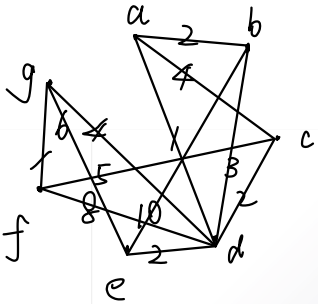
c: → **(a, 4)** → **(f, 5)**

d: → **(e, 2)** → **(c, 2)** → **(g, 4)** → **(f, 8)**

e: → **(g, 6)**

f:

g: → **(f, 1)**



a: → (d, 1) → (b, 2)

b: → (d, 3) → (e, 10)

c: → (a, 4) → (f, 5)

d: → (e, 2) → (c, 2) → (g, 4) → (f, 8)

e: → (g, 6)

f:

g: → (f, 1)

Dijastra's Algo

Step 1: Make the greedy choice

Step 2: Update its neighbours.

Initial State: (Source = b)

	a	b	c	d	e	f	g
visited	0	1	0	0	0	0	0
d	∞	0	∞	3	10	∞	∞
pi	–	b	–	b	b	–	–

Iteration 1: pick d.

	a	b	c	d	e	f	g
visited	0	1	0	1	0	0	0
d	∞	0	3+2	3	3+2	3+8	3+4
pi	–	b	d	b	d	d	d

update if distance become shorter

Iteration 2: c & e have same d.

arbitary choose 1. pick c

	a	b	c	d	e	f	g
visited	0	1	1	1	0	0	0
d	5+4	0	5	3	5	5+5	7
pi	c	b	d	b	d	c	d

Iter 3: pick e (smallest remaining). pick e

	a	b	c	d	e	f	g
visited	0	1	1	1	1	0	0
d	9	0	5	3	5	10	7
pi	c	b	d	b	d	c	d

e's neighbour. 7+6=11 > 7. ∴ no update.

Iter 4: pick g (smallest remaining). pick g

	a	b	c	d	e	f	g
visited	0	1	1	1	1	0	1
d	9	0	5	3	5	7+1	7
pi	c	b	d	b	d	g	d

update g's neighbour.

b → d → e: 5

b → d → g → f: 8

b → d → g: 7.

Iter 5: pick f (nothing to do)

Iter 6: pick a (no update).

Final table:

	a	b	c	d	e	f	g
visited	1	1	1	1	1	1	1
d	9	0	5	3	5	8	7
pi	c	b	d	b	d	g	d

From b to each vertex:

b → d → c → a: 9 → parent path.

b → d → c: 5 | b → d: 3 → child path.