CSE 250: Ordering, Priority Queues

Lecture 22

Oct 23, 2023

Reminders

- PA2: Implement **Map Routing**
  1. Create an adjacency list (discussed today)
  2. Find a path from A to B with the fewest intersections
  3. Find a path from A to B with the shortest distance
- PA2 implementation due Sun, Nov 5 at 11:59 PM

New ADT: Priority Queue

`PriorityQueue<E>` (E must be `Comparable``Comparable`)

- `public void add(E e)`: Add e to the queue.
- `public E peek()`: Return the *leastleast* element added.
- `public E remove()`: Remove and return the *leastleast* element added.

Examples

How might we order the following?

- "A+", "C", "B-"
- Taco Tuesday, Fish Friday, Meatless Monday
- Serenity, Gamer, Julie, Garfield
- Aardvark, Baboon, Capybara, Donkey, Echidna

Ordering

An **ordering on type** A $(A, \leq)$:

- A set of things of type A
- A "relation" or comparator $(\leq)$ that relates two things in the set.

- Numerical Order
  $5 \leq 30 \leq 999$
- Reverse-numerical order on the 2nd field
  $(E, 40) \leq (B, 10) \leq (D, 3)$
- Letter Grades
  C+ $\leq$ **B-** $\leq$ **B** $\leq$ **B+** $\leq$ **A-** $\leq$ **A**
- Compare first, then 2nd, 3rd, ... (Lexical Order)
  AA $\leq$ AM $\leq$ BZ $\leq$ CA $\leq$ CD

Ordering Properties

- **Team A $\leq$ Team B**
  Team B won its match against Team A
- **Team B $\leq$ Team C**
  Team C won its match against Team B
- **Team C $\leq$ Team A**
  Team A won its match against Team C

A

No Transitivity!

C $\geq$ B

Is this an ordering?

Ordering Properties

An ordering must be...

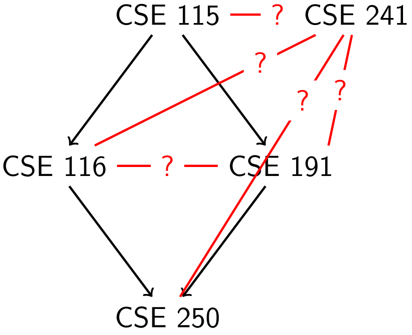
- **Reflexive** $x \leq x$
- **Antisymmetric** **if** $x \leq y$ **and** $y \leq x$ **then** $x = y$
- **Transitive** **if** $x \leq y$ **and** $y \leq z$ **then** $x \leq z$

Another Example

Define an ordering over CSE Courses
(Course 1 $\leq$ Course 2 iff Course 1 is a prereq of Course 2)

- CSE 115 $\leq$ CSE 116
- CSE 116 $\leq$ CSE 250
- CSE 115 $\leq$ CSE 191
- CSE 191 $\leq$ CSE 250

Is this a valid ordering?



(Partial) Ordering Properties

A partial ordering must be...

- **Reflexive** $x \leq x$
- **Antisymmetric** **if** $x \leq y$ **and** $y \leq x$ **then** $x = y$
- **Transitive** **if** $x \leq y$ **and** $y \leq z$ **then** $x \leq z$

(Total) Ordering Properties

A total ordering must be...

- **Reflexive** $x \leq x$
- **Antisymmetric** **if** $x \leq y$ **and** $y \leq x$ **then** $x = y$
- **Transitive** **if** $x \leq y$ **and** $y \leq z$ **then** $x \leq z$
- **Complete** **either** $x \leq y$ **or** $y \leq x$ **for any** $x, y \in A$

Some Other Definitions

For an ordering $(A, \leq)$

- The **greatest** element is some $x \in A$ such that there is **no** $y \in A$ where $x \leq y$
- The **least** element is some $x \in A$ such that there is **no** $y \in A$ where $x \geq y$

A partial ordering may not have a unique greatest or least element

Describing an Ordering

$\leq$ can be described **explicitly**, by a set of tuples:

$$\{\ (a,a), (a,b), (a,c), \ldots, (b,b), \ldots, (z,z)\ \}$$

If (x, y) is in the set, then $x \leq y$:

- (a, a): a is less than or equal to a
- (a, b): a is less than or equal to b
- (a, c): a is less than or equal to c
- ...
- (b, b): a is less than or equal to b
- ...
- (z, z): z is less than or equal to z

Describing an Ordering

$\leq$ can be described by a **mathematical rule**:

$$\{\ (x, y) \mid x, y \in \mathbb{Z}, \exists a \in \mathbb{Z}^+ \cup \{\ 0\ \} : x + a = y\ \}$$

$x \leq y$ iff x, y are integers and there is a non-negative integer a that you can add to x to get y.

Multiple Orderings

Multiple Orderings can be defined for the same set

- RottenTomatoes vs Metacritic vs Box Office Gross
- "Best Movie" first vs "Worst Movie" first
- Number of swear words

We use subscripts to separate orderings ($\leq_1$, $\leq_2$, $\leq_3$, . . .)

Transformations

We can transform orderings:

- **Reverse**:
  if $x \leq_1 y$ **then define** $y \leq_R x$

- **Lexical**: Given $\leq_1, \leq_2, \leq_3, \ldots$
  - If $x \leq_1 y$ then $x \leq_L y$
  - If $x =_1 y$ and $x \leq_2 y$ then $x \leq_L y$
  - If $x =_1 y$ and $x =_2 y$ and $x \leq_3 y$ then $x \leq_L y$
  - $\ldots$

Examples of Lexical Ordering

- **Names**: First letter, then second letter, then third...
- **Movies**: Average of reviews, then number of reviews
- **Records**: First field, then second field, then third...
- **Sports Teams**: Games won, points scored, speed of victory, ...

Ordering over Keys

$\leq$ can be described as a **ordering over a key derived from the element:**

$$x \leq_{edge} y \text{ iff } \texttt{weight}(x) \leq \texttt{weight}(y)$$
$$x \leq_{contact} y \text{ iff } \texttt{name}(x) \leq_{L} \texttt{name}(y)$$

Here, we say that weight/name are keys.

Topological Sort

A **topological sort** of a partial order $(A, \leq_1)$ is any total order $(A, \leq_2)$ that "agrees" with $(A, \leq_1)$.

For any two elements $x, y \in A$:

- If $x \leq_1 y$ then $x \leq_2 y$
- If $y \leq_1 x$ then $y \leq_2 x$
- Otherwise, either $x \leq_2 y$ or $y \leq_2 x$

Topological Sort

The following are all topological sorts over our partial order from earlier:

- CSE 115, CSE 116, CSE 191, CSE 241, CSE 250
- CSE 241, CSE 115, CSE 116, CSE 191, CSE 250
- CSE 115, CSE 191, CSE 116, CSE 250, CSE 241

(In this case, the partial ordering is a schedule requirement, and each topological sort is a possible schedule)

... back to our ordering-based ADT

New ADT: Priority Queue

`PriorityQueue<E>` (E must be `Comparable`)

- `public void add(E e)`: Add e to the queue.
- `public E peek()`: Return the *least* element added.
- `public E remove()`: Remove and return the *least* element added.

Priority Queues

- `add(5)`
- `add(9)`
- `add(2)`
- `add(7)`
- `peek()` $\rightarrow 2$
- `remove()` $\rightarrow 2$
- `size()` $\rightarrow 3$
- `peek()` $\rightarrow 5$
- `remove()` $\rightarrow 5$
- `remove()` $\rightarrow 7$
- `remove()` $\rightarrow 9$
- `size()` $\rightarrow 0$

How do we store this?

- Insertion Order? [5, 9, 2, 7]
- Sorted Order? [2, 5, 7, 9]
- Reverse Sorted Order? [9, 7, 5, 2]

Priority Queues

There are two mentalities...

- **Lazy**: Keep everything a mess. "Selection Sort"
- **Proactive**: Keep everything organized. "Insertion Sort"

Lazy Priority Queue

Base Data Structure: Linked List

- `public void add(T v)` $O(1)$
  Append `v` to the end of the linked list.
- `public T remove()` $O(N)$
  Traverse the list to find the least value and remove it.

Sorting with a Priority Queue

```
public List<T> prioritySort(List<T> items,
                            PriorityQueue<T> pqueue)
{
  T[] out = new T[items.size];
  for( item : items ){ pqueue.add(item) }   ← Add to pqueue
  for( int i = 0; i < items.size; i++ )
  {
    out[i] = items.remove()                 ← Remove from pqueue
  }
  return Arrays.asList(out)
}
```

Selection Sort (with a Lazy P.Queue)

	Input / Output	**P.Queue**
Input	(7, 4, 8, 2, 5, 3, 9)	()
Step 1	(4, 8, 2, 5, 3, 9)	(7)
Step 2	(8, 2, 5, 3, 9)	(7, 4)
...	...	...
Step n	()	(7, 4, 8, 2, 5, 3, 9)
Step n+1	[2, _, _, _, _, _, _]	(7, 4, 8, 5, 3, 9)
Step n+2	[2, 3, _, _, _, _, _]	(7, 4, 8, 5, 9)
Step n+3	[2, 3, 4, _, _, _, _]	(7, 8, 5, 9)
...	...	...
Step 2n	[2, 3, 4, 5, 7, 8, 9]	()

Sorting with a Priority Queue

```
public List<T> prioritySort(List<T> items,
                            PriorityQueue<T> pqueue)
{
  T[] out = new T[items.size];
  for( item : items ){ pqueue.add(item) }
  for( int i = 0; i < items.size; i++ )
  {
    out[i] = items.remove()
  }
  return Arrays.asList(out)
}
```

What is the complexity (with a lazy P.Queue)? $O(n^2)$

Proactive Priority Queue

Base Data Structure: Linked List

- `public void add(T v)` $O(N)$
  Traverse the list to insert `v` in sorted order.
- `public T remove()` $O(1)$
  Remove the head of the list.

Selection Sort (with a Proactive P.Queue)

	Input / Output	**P.Queue**
Input	(7, 4, 8, 2, 5, 3, 9)	()
Step 1	(4, 8, 2, 5, 3, 9)	(7)
Step 2	(8, 2, 5, 3, 9)	(4, 7)
Step 3	(2, 5, 3, 9)	(4, 7, 8)
...	...	...
Step n	()	(2, 3, 4, 5, 7, 8, 9)
Step n+1	[2, _, _, _, _, _, _]	(3, 4, 5, 7, 8, 9)
Step n+2	[2, 3, _, _, _, _, _]	(4, 5, 7, 8, 9)
...	...	...
Step 2n	[2, 3, 4, 5, 7, 8, 9]	()

Sorting with a Priority Queue

```
public List<T> prioritySort(List<T> items,
                            PriorityQueue<T> pqueue)
{
  T[] out = new T[items.size];
  for( item : items ){ pqueue.add(item) }
  for( int i = 0; i < items.size; i++ )
  {
    out[i] = items.remove()
  }
  return Arrays.asList(out)
}
```

What is the complexity (with a proactive P.Queue)? $O(n^2)$

Priority Queues

Operation	Lazy	Proactive
`add`	$O(1)$	$O(N)$
`remove`	$O(N)$	$O(1)$
`peek`	$O(N)$	$O(1)$

Can we do better?