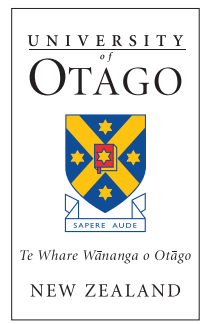
Cosc 241
Programming and Problem Solving
Lecture 12 (4/4/2019)
Random 2

Michael Albert
michael.albert@cs.otago.ac.nz

Keywords: odramn



Recap

- We can work with an instance `R` of `java.util.Random`.
- The main method we will use is `R.nextInt(k)` which returns a uniformly distributed (pseudo-)random integer greater than or equal to 0 and less than k (perfect as an array index into an array of length k).

Shuffling an array

One method:

- Exchange the last card with some earlier item or itself, then
- exchange the second last card with some earlier card or itself, then
- exchange the third last item with some earlier item or itself, then
- ...

```
public static void shuffle(Object[] cards) {
  for(int i = cards.length-1; i > 0; i--) {
    exchange(cards, i, R.nextInt(i+1));
  }
}
```

Dealing a hand

- This time we don't want to shuffle the whole array, but return k randomly chosen elements from it.
- No duplication (i.e. we want k distinct elements).
- One possible method - shuffle the array, and then use the first, or last, k (if we do this right, then we don't need to complete the shuffle - we can stop when the required elements are in place).
- Or generate the indices needed somehow.
- Or work with a form of sampling.

Generating indices

- We have some chosen set of $i < k$ indices, and need to generate the next one.
- There are $n - i$ candidates left, so we should use `R.nextInt(n-i)`.
- But now we need to convert that to an actual index by incrementing by one each time one of our previously chosen indices is less than or equal to it.
- To do this conveniently requires having the already chosen (and adjusted) indices in sorted order.

Choosing cards

- Look at the cards one at a time
- Each card is added to the hand with a probability of n/r where n is the number of cards still needed and r is the number of cards remaining in the deck (if we add a card, we update both n and r, if not we just update r).
- Stop when you have enough cards.

```
public static <T> T[] chooseHand(T[] deck, int handSize) {
  T[] result = (T[]) new Object[handSize];
  int stillNeeded = handSize; int handIndex = 0;
  for(int deckIndex = 0;  stillNeeded > 0; deckIndex++) {
    int i = R.nextInt(deck.length - deckIndex);
    if (i < stillNeeded) {
      result[handIndex] = deck[deckIndex];
      handIndex++; stillNeeded--;
    }
  }
  return result;
}
```

Complexity analysis

- Let n be the deck size and k the hand size.
- If we generate indices the complexity will be $O(k^2)$ (one loop to generate the initial index, and one loop to update it and insert it in the sorted list of indices generated so far).
- `chooseHand` has a single loop whose control is less clear.
- But, we might need to look at every card of the deck (since we might choose the last one) so its complexity is $O(n)$.
- Apparently there's a trade off!
- It sems like $O(k)$ should be possible.
- That's easy if we're allowed to shuffle the deck.

Slightly silly

- What if we are obligated to leave the deck in the same order that it started?
- Disclaimer: I have never actually needed this trick.
- Get the hand we want by shuffling with the first k exchanges only, and save it.
- Then undo the exchanges on the original deck by doing them again in reverse order.
- That's $O(k)$.

One more time

- Again we want to select k cards from a deck (short list k applicants, choose k prize winners etc.)
- This time we don't know how many cards there are.
- We receive cards one at a time and may choose to reject them permanently or accept (possibly rejecting a card we already hold).
- The process can stop at any time and at that point the hand we hold must be randomly chosen from all the cards we saw.
- Key idea – when card n arrives we know that we should hold it with probability k/n (for $n > k$).
- We do that in the usual way (see if `R.nextInt(n) < k`) and if we succeed we store it in the index returned in that call to `R`.

Managing a raffle

- In a raffle we have a number of entrants who may all have different numbers of tickets.
- How can we efficiently choose a winner?
- What are the efficiency concerns?