Search

- The basic idea is:
  - look at all possible solutions
  - for each possible solution, check it
- For this to be feasible:
  - there cannot be too many possibilities to check
  - each possibility can be checked efficiently
- Sometimes we have no better ways of solving the problem.

Types of Problems

- Decision (yes/no), optimization, counting.
- Do we have to look through all possibilities, or can we quit at the first one that answers the question?
- Just the answer, or also the possibility leading to that answer?
- Search space:
  - permutatations
  - subsets/combinations
  - all factors
  - all possible moves (maybe better with BFS)
  - "possible answers" (Counterfeit Dollars 608)

Permutations

- Looking for a possible order.
- e.g. travelling salesman type problems.
- Assignment problems can also be modelled as permutations.
- Use `next_permutation` if there is no need to prune early. Each call can be $O(n)$ for a total of $O(n \cdot n!)$ over all permutations.
- Checking each permutation often takes $O(n)$ operations anyway.
- You can also write your own recursive routine to examine all permutations

216: Getting in Line

- Up to 8 computers (2D coordinates), find best way to connect them in a line.
- Try all 8! permutations and test each one.
- Feel free to use global variables to keep track of best distance and permutation.

102: Ecological Bin Packing

- 3 types of objects, 3 bins. Sort them so that each bin contains only one type of objects.
- Need to **assign** each type to a bin: permutation.
- Similar problems: assignment of tasks, n queens (permutation of column indices).
- Note: there are better algorithms for some assignment problems.

Subsets

- Sometimes the solution space is over a subset of items.
- If we need to look at all possible subsets, there are 2^n subsets.
- If we only need to look at subsets of size k, there are $\binom{n}{k}$ subsets.

Generating Subsets

- For larger n, we recursively select/skip elements to try all possible subsets:

```
void subset(bool selected[], int n, int index)
{
  if (index == n) {
    // selected[] contains the subset
    // do what is required
  } else {
    subset(selected, n, index+1);

    selected[index] = true;
    subset(selected, n, index+1);
    selected[index] = false;
  }
}
```

  Start with a selected array that has only false entries.

- For small n, we can loop from 0 to $2^n - 1$ and examine the bit patterns.

Generating Subsets of size k

```
void subset(bool selected[], int n, int k, int index)
{
  if (index == n) {
    // do what is required
  } else {
    if (k > 0) {
      selected[index] = true;
      subset(selected, n, k-1, index+1);
      selected[index] = false;
    }
    if (n - index > k) {
      subset(selected, n, k, index+1);
    }
  }
}
```

Pruning

- For some problems you may be able to prune as soon as possible.
- e.g. n Queens problem: if the first k queens conflict, there is no point searching.
- If the order to consider possibilities does not matter, try the one with the fewest possibilities first (so pruning is most effective).
- e.g. 229 Scanner

Binary Search

- Sometimes we can take an optimization problem and turn it into search.
- e.g. What is the smallest xxx such that yyy is possible?
- The search space is all possible "xxx".
- If the search space is reasonably small, we can search from smallest by linear search.
- We can also use binary search if all possibilities below the answer are "not possible" and all other ones are "possible."
- It must be possible to do the checks efficiently.

714 Copying Books

- Do binary search on the time allowed per scribe.
- For each time, check whether it is possible with a greedy algorithm.
- Also: see 11516 WiFi.