Backtracking and games

Algorithms - Tutorial #11

Today

Common Backtracking Problems

- Determine whether a solution exists
- Find the best solution

Backtracking

Decide if list is partitionable

Write function **`partitionable(numbers)`** which decided whether given list of numbers can be split into two partitions, partitions have equal sums

- [6, 2, 4, 1, 1]
  - [1, 2, 4] [1, 6]
- [1, 5, 3, 5]
  - Unable to partition

Backtracking

Decide if list is partitionable

```
def partitionable(numbers, index, sum1, sum2):
 v = numbers[index]
 p1 = partitionable(numbers, index + 1, sum1 + v, sum2)
 P2 = partitionable(numbers, index + 1, sum1, sum2 + v)
 return p1 or p2 # has solution if adding v to p1 or p2
```

- Enumerating over all combinations (recursively)
- Result of some recursive calls can be determined before calling
- `return false` if following condition is true

```
 sum1 > sum(numbers)/2 or sum2 > sum(numbers)/2
```

Backtracking

Knapsack

Given list items of **`(weight, value)`** and some **`capacity`**, write function **`knapsack(items, capacity)`**, finding subset of items with highest sum of values fitting into **`capacity`** of knapsack

- items = **`[(2,5),(3,10),(1,8)]`**
- capacity = **`5`**
- solution S **`[(3,10),(1,8)]`**

- Two recursive calls for each item: item in S / item not in S
- Omit combinations which are greater than capacity

Backtracking

Autocorrect

Given set validWords, dictionary of nearby keys `nearbyKeys`, user `input` and maximal number of typos `maxTypos`, find set of all potential intended words, considering only replacement typos

```
validWords = {’dal’,’nam’,’dan’,’san’,’den’}
maxTypos = 2
input = ‘dam’

solution = { ‘dal’,’dan’,’san’,’den’}
```

Autocorrect

Ideas

- Backtracking - generate replacements from nearby keys, omit generated prefixes not present in `validWords`
  - `validWords = {’dal’,’nam’,’dan’,’san’,’den’}`
  - `input = ‘dam’`
  - `generated (d -> r at 0) = ‘ram’`
    - `(prefix ‘r’ not as prefix of word in validWords)`