Sum of Subsets and Graph Coloring

COMP 215 Lecture 10
Sum-of-Subsets

- Similar to the knapsack problem, except we don't worry about cost, and we want to find all subsets with total weights that equal the weight limit.
- Before we get started... How many subsets are there of a set containing $N$ items?
- Let's design a brute force solution...
Backtracking for Sum-of-Subsets

• First sort the items so that weight is non-decreasing.

• Two conditions that allow backtracking:
  1) At level $i$, if the total weight is not $W$, and adding $w_{i+1}$ would bring the total weight above $W$.
     • (because all weights after $w_{i+1}$ are $\geq w_{i+1}$)

  2) At level $i$, if the total weight is not $W$, and all following weights can't bring it to $W$. 
//i = index of current item.
//weight = summed weight of items included so far.
//total = total weight of not-yet-considered items.

void sum_of_subsets (index i, int weight, int total){
  if (promising(i, weight, total)) {
    if (weight == W) {
      cout << include[1] through include[i];
    } else {
      include[i+1] = true;
      sum_of_subsets(i+1, weight + w[i+1], total - w[i+1]);
      include[i+1] = false;
      sum_of_subsets(i+1, weight, total - w[i+1]);
    }
  }
}
Sum-of-Subset Pseudocode

```java
void bool promising (index i, int weight, int total){
    return ((weight + total >= W) &&
        (weight == W || weight + w[i+1] <= W));
}
```

- Before this code is called we need to do some prep work.
  - Sort the items by weight.
  - Compute the total weight of all items.
- What is the maximum size of the search tree?
- Can we guarantee that the portion searched with backtracking will be much smaller?
Analysis

• What is the maximum size of the search tree?
  \[ 1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1 \]

• Can we guarantee that the nodes visited with backtracking will be much smaller?
  \[ \text{No. Consider a set of items with weights such that} \]
  \[ \sum_{i=1}^{n-1} w_i < W \quad w_n = W. \]

• Whether or not a solution can be found efficiently depends both on \( n \) and the specific weights.
Map Coloring

• Color a map such that no two countries that share a border have the same color.
  – Easy if we have as many colors as countries.

• We may want to know the minimum number of colors required for a given map.

• Or, for a given map we may want to know if it is 2-colorable, 3-colorable, or \( m \)-colorable.

• The problem is easier to work with if we think in terms of graphs...
  – Maps lead to planar graphs.

• There are many applications...
The $m$-Coloring Problem

- Find all ways to color an undirected graph using at most $m$ colors.
- Let's think through the brute force algorithm...
- How can this be improved with backtracking?
//i = index of current vertex.

void m_coloring (index i){
    if (promising(i)) {
        if (i == n) {
            cout << vcolor[1] through vcolor[n];
        } else {
            for (int color = 1; color <= m; color++) {
                vcolor[i+1] = color;
                m_coloring(i+1);
            }
        }
    }
}

$m$-Coloring With Backtracking

```c
void bool promising (index i){
    for (int j = 1; j < i; j++) {
        if (W[i][j] && vcolor[i] == vcolor[j])
            return false;
    }
    return true;
}
```

- Size of the complete search space:
  
  \[ 1 + m + m^2 + \ldots + m^n = \frac{m^{n+1} - 1}{m - 1} \]

- Backtracking might not save us much.
0-1 Knapsack

• We construct the search tree as we did for the sum of subsets problem.

• Two things to notice:
  – This is an optimization problem, so in some sense, we need to search the whole tree.
  – Every node represents a possible solution.

• Any ideas for backtracking?
0-1 Knapsack

- Again we have two possibilities for backtracking:
  1) We do not need to explore a node's children if we have hit the weight limit.
  2) We do not need to explore a node's children if there is no possibility that the profit will exceed the best profit found so far.

- In order to determine two we first sort items by price/weight.

- We then use the greedy (fractional) approach at each node to compute an upper bound on profit.

- Once again, we can find a worst case scenario that requires us to explore almost all of the tree.