Lecture 17: Chapter 5 Part 2

Backtracking

CS3310

Backtracking

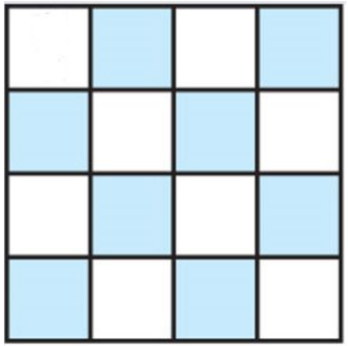
- Recall that backtracking algorithms solve problems in which a **sequence** of objects is to be chosen from a specified **set** of objects.
- We want this sequence to satisfy some **criterion**.
- The n-Queens problem (How can we position n queens on an $n \times n$ chessboard so no queens threaten each other?) has the following criterion, set, and sequence:

Criterion: No two queens can threaten each other.

Set: The n^2 positions on the board in which a queen can be placed:

{ <1, 1>, <1, 2>, <1, 3>, <1, 4>, <2, 1>, <2, 2>, …<4, 4> }

Sequence: n positions from the above set. If queens are placed in each of these n positions, no two should threaten each other.



Sum-of-Subsets

- Suppose we have *n* objects, each with a specific weight.
- We want to determine the different ways we can select some (or all) of the objects so that their total weight equals exactly W

Criterion: ??

Set: ??

Sequence: ??

Sum-of-Subsets

- Suppose we have *n* objects, each with a specific weight.
- We want to determine the different ways we can select some (or all) of the objects so that their total weight equals exactly W

Criterion: The total weight of the selected objects is exactly W.

Set: The *n* objects we can choose from.

Sequence: The objects chosen from the set of *n* objects.

Sum-of-Subsets

- Suppose we have n objects, each with a specific weight.
- We want to determine the different ways we can select some (or all) of the objects so that their total weight equals exactly W

Example:

- $n = 5$, $W = 21$
- $w_1 = 5$, $w_2 = 6$, $w_3 = 10$, $w_4 = 11$, $w_5 = 16$
- How many solutions does this instance have?

Sum-of-Subsets

- Suppose we have n objects, each with a specific weight.
- We want to determine the different ways we can select some (or all) of the objects so that their total weight equals exactly W

Example:

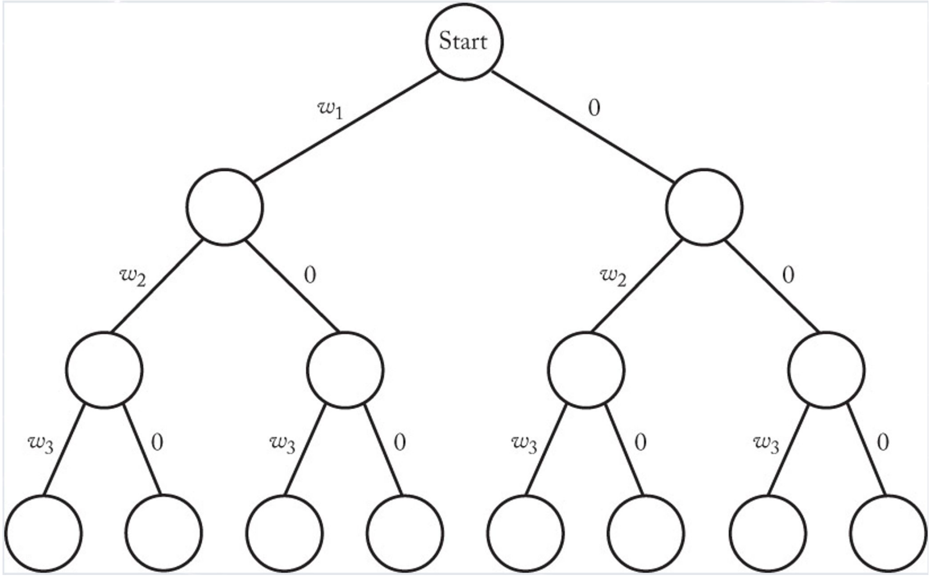
- $n = 5$, $W = 21$
- $w_1 = 5$, $w_2 = 6$, $w_3 = 10$, $w_4 = 11$, $w_5 = 16$
- How many solutions does this instance have? **Three**:
  - $\{ w_1, w_2, w_3 \}$
  - $\{ w_1, w_5 \}$
  - $\{ w_3, w_4 \}$

Sum-of-Subsets

- How could we draw a state space tree for this problem?

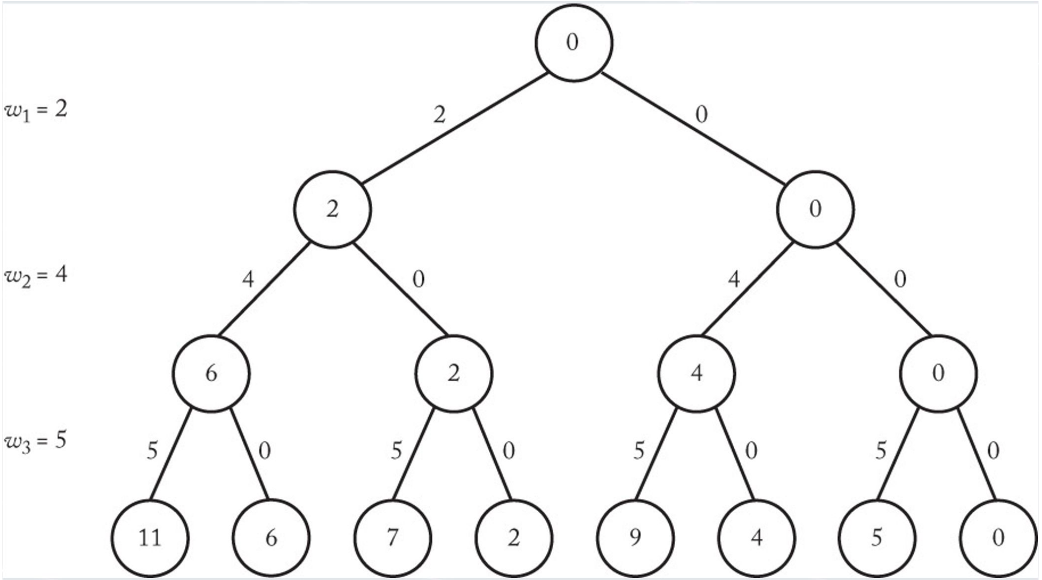
Sum-of-Subsets

- A state space tree for $n = 3$.
- We start at a dummy node, which indicates we have not yet taken or rejected any items.
- Moving left indicates we take item 1. Moving right indicates we do not.



Sum-of-Subsets

- **Note**: If we take an item, we add its weight to a total. Otherwise, we add nothing.
- Each node holds the total weight of the items taken up to that point.
- For the instance W = 6, the only solution is { w_1, w_2 }



Sum-of-Subsets Steps

A backtracking algorithm for larger values of n:

- Sort the items in nondecreasing order by weight.
- Start at the root (dummy node): move left to take item 1, right to reject it.
- When we move left, we add the weight of the chosen item to `weight`.

- Suppose we are at the i th level in our state space tree and we notice that adding w_{i+1} would bring `weight` above W
  - In this scenario, how can we determine if w_i is promising?

Sum-of-Subsets Steps

A backtracking algorithm for larger values of *n*:

- Sort the items in nondecreasing order by weight.
- Start at the root (dummy node): move left to take item 1, right to reject it.
- When we move left, we add the weight of the chosen item to `weight`.

- Suppose we are at the i^{th} level in our state space tree and we notice that adding w_{i+1} would bring `weight` above W
  - In this scenario, how can we determine if w_i is promising?
    - w_{i+1} is the lightest item remaining (since we sorted them), so taking any item after it would *also* bring `weight` above W.
    - Therefore, `weight` <u>must</u> equal W for w_i to be promising

- w_i is *nonpromising* if: `weight + w`$_{i+1}$ `> W && weight != W`

Sum-of-Subsets Steps

- w_i is *nonpromising* if: `weight +` $\texttt{w}_{\texttt{i+1}}$ `> W && weight != W`

- Is there another way to determine if w_i is nonpromising?

Sum-of-Subsets Steps

- w_i is *nonpromising* if: `weight + w`$_{i+1}$ `> W && weight != W`

- Is there another way to determine if w_i is nonpromising?
  - Let `total` be the total weight of the remaining objects to choose from
    - `weight` is the weight of the items we have chosen up to this point.
  - If adding `total` to `weight` results in a value less than W, we can backtrack. Even if we take w_i and every item after it, we will not have enough weight.
    - i.e. if `weight + total < W`, we can also backtrack

We backtrack if:

`(weight + total < W) || (weight + w`$_{i+1}$ `> W && weight != W)`

Sum-of-Subsets

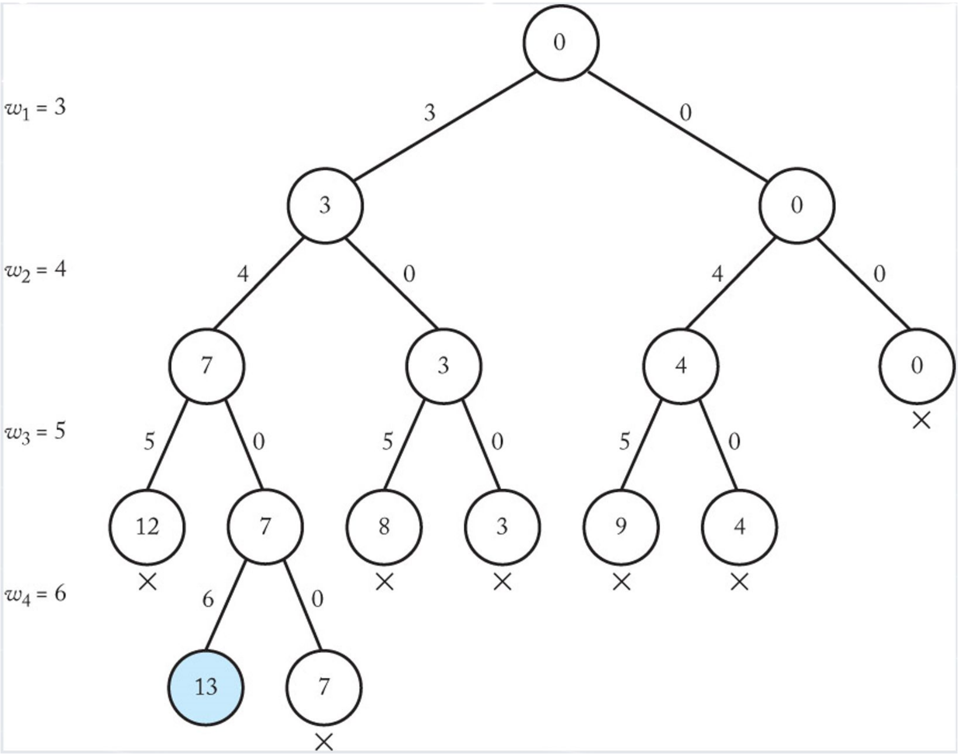
The entire pruned state space tree for:

$n = 4 \qquad W = 13$

$w_1 = 3 \qquad w_2 = 4 \qquad w_3 = 5 \qquad w_4 = 6$

Note: in the path where we reject items 1 and 2, we backtrack because even if we take items 3 and 4, `weight` will be less than W:

- $w_3 + w_4 < 13$



Sum-of-Subsets

- When a node is reached and `weight` = W, we can also backtrack. Why?

Sum-of-Subsets

- When a node is reached and `weight` = W, we can also backtrack. Why?
  - Adding more weight will clearly not obtain another solution.
  - If `weight` = W, we print that solution and backtrack.
  - Our backtracking procedure, `checknode`, will not expand beyond a promising node if a solution has been found at that node.

- In code, a one-dimensional array `include` will implicitly build the state space tree.
  - We set `include[i]` to "yes" if `w[i]` is included and "no" if it is not.

Sum-of-Subsets Example

$n = 4$ $\quad\quad$ $\mathrm{W} = 15$ $\quad\quad$ $w = \{\, 2, 4, 6, 9 \,\}$

```
include = { }
weight = 0
total = 2 + 4 + 6 + 9 = 21
```

Sum-of-Subsets Example

$n = 4$ $W = 15$ $w = \{ \mathbf{2}, 4, 6, 9 \}$

```
include = { "yes" }
weight = 2
total = 4 + 6 + 9 = 19
```

Is taking w_1 promising? (we want both tests listed below to pass)

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ \mathbf{2}, 4, 6, 9 \}$

```
include = { "yes" }
weight = 2
total = 4 + 6 + 9 = 19
```

Is taking w_1 promising? (we want both tests listed below to pass)

Yes!

`2 + 19 > 15` and `2 + 4 < 15`

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\qquad$ $W = 15$ $\qquad$ $w = \{ 2, \mathbf{4}, 6, 9 \}$

```
include = { "yes", "yes" }
weight = 6
total = 6 + 9 = 15
```

Is taking w_2 promising?

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ 2, \mathbf{4}, 6, 9 \}$

```
include = { "yes", "yes" }
weight = 6
total = 6 + 9 = 15
```

Is taking w_2 promising?

Yes!

`6 + 15 > 15` and `6 + 6 < 15`

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\qquad$ $W = 15$ $\qquad$ $w = \{ 2, 4, \mathbf{6}, 9 \}$

```
include = { "yes", "yes", "yes" }
weight = 12
total = 9
```

Is taking w_3 promising?

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ 2, 4, \mathbf{6}, 9 \}$

```
include = { "yes", "yes", "yes" }
weight = 12
total = 9
```

Is taking w_3 promising?

No!

`12 + 9 > 15` and `12 != 15`

➢ We backtrack and do not take w_3

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ 2, 4, \mathbf{6}, 9 \}$

```
include = { "yes", "yes", "no" }
weight = 6 + 0 = 6
total = 9
```

Is rejecting w_3 promising?

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $W = 15$ $w = \{ 2, 4, \mathbf{6}, 9 \}$

```
include = { "yes", "yes", "no" }
weight = 6 + 0 = 6
total = 9
```

Is rejecting w_3 promising?

Yes!

`6 + 9 = 15` and `6 + 9 = 15`

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\quad$ $W = 15$ $\quad$ $w = \{\, 2, 4, 6, \mathbf{9} \,\}$

```
include = { "yes", "yes", "no", "yes" }
weight = 6 + 9 = 15
total = 0
```

Is taking w_4 promising?

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $W = 15$ $w = \{ 2, 4, 6, \mathbf{9} \}$

```
include = { "yes", "yes", "no", "yes" }
weight = 6 + 9 = 15
total = 0
```

Is taking w_4 promising?

Yes!

`15 + 0 = 15` and `15 = 15`

We have found a solution: $\{ w_1, w_2, w_4 \}$ print it and backtrack, this time rejecting w_4

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ 2, 4, 6, \mathbf{9} \}$

```
include = { "yes", "yes", "no", "no" }
weight = 6 + 0 = 6
total = 0
```

Is rejecting w_4 promising?

No!

`6 + 0 < 15`

Where do we backtrack to?

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\qquad$ W = 15 $\qquad$ $w = \{ 2, 4, 6, \mathbf{9} \}$

```
include = { "yes", "yes", "no", "no" }
weight = 6 + 0 = 6
total = 0
```

Is rejecting w_4 promising?

No!

`6 + 0 < 15`

Where do we backtrack to? The last “yes” and reject it.

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\quad$ $W = 15$ $\quad$ $w = \{ 2, \mathbf{4}, 6, 9 \}$

```
include = { "yes", "no" }
weight = 2 + 0 = 2
total = 15
```

Is rejecting w_2 promising?

Yes!

`2 + 15 > 15` and `2 + 6 < 15`

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\qquad$ $W = 15$ $\qquad$ $w = \{ 2, 4, \mathbf{6}, 9 \}$

```
include = { "yes", "no", "yes" }
weight = 2 + 6 = 8
total = 9
```

Is taking w_3 promising?

No!

`8 + 9 > 15` but `8 + 9 > 15`

- ➢ Backtrack to first "yes" and reject it.

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\qquad$ $W = 15$ $\qquad$ $w = \{ \mathbf{2}, 4, 6, 9 \}$

```
include = { "no" }
weight = 0
total = 19
```

Is rejecting w_1 promising?

Yes!

`0 + 19 > 15` and `0 + 4 < 15`

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $W = 15$ $w = \{ 2, \mathbf{4}, 6, 9 \}$

```
include = { "no", "yes" }
weight = 0 + 4 = 4
total = 15
```

Is including w_2 promising?

Yes!

`4 + 15 > 15` and `4 + 6 < 15`

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\qquad$ $W = 15$ $\qquad$ $w = \{ 2, 4, \mathbf{6}, 9 \}$

```
include = { "no", "yes", "yes" }
weight = 4 + 6 = 10
total = 9
```

Is including w_3 promising?

No!

`10 + 9 > 15` but `10 + 9 < 15`

➢ Backtrack and do not take w_3

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ 2, 4, \mathbf{6}, 9 \}$

```
include = { "no", "yes", "no" }
weight = 4 + 0 = 4
total = 9
```

Is rejecting w_3 promising?

No!

```
4 + 9 < 15
```

➢ Backtrack to last "yes" and reject it

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\quad\quad$ $W = 15$ $\quad\quad$ $w = \{ 2, \mathbf{4}, 6, 9 \}$

```
include = { "no", "no" }
weight = 0
total = 15
```

Is rejecting w_2 promising?

Yes!

`0 + 15 = 15 and 0 + 6 < 15`

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ 2, 4, \mathbf{6}, 9 \}$

```
include = { "no", "no", "yes" }
weight = 0 + 6 = 6
total = 9
```

Is including w_3 promising?

Yes!

```
6 + 9 = 15 and 6 + 9 = 15
```

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ 2, 4, 6, \mathbf{9} \}$

```
include = { "no", "no", "yes", "yes" }
weight = 6 + 9 = 15
total = 0
```

Is including w_4 promising?

Yes!

`15 + 0 = 15 and 15 = 15`

We have found the second solution: $\{ w_3, w_4 \}$ Print it and backtrack, rejecting w_4

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ $\quad$ $W = 15$ $\quad$ $w = \{ 2, 4, 6, \mathbf{9} \}$

```
include = { "no", "no", "yes", "no" }
weight = 6 + 0 = 6
total = 0
```

Is rejecting w_4 promising?

No!

```
6 + 0 < 15
```

Backtrack to last “yes” and reject it

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ 2, 4, \mathbf{6}, 9 \}$

```
include = { "no", "no", "no" }
weight = 0
total = 9
```

Is rejecting w_3 promising?

No!

`0 + 9 < 15`

Nowhere else to backtrack to, so we're done!

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

Sum-of-Subsets Example

$n = 4$ W = 15 $w = \{ 2, 4, 6, 9 \}$

Using the backtracking method, we found the following two solutions for this problem:

1. $\{ w_1, w_2, w_4 \}$
2. $\{ w_3, w_4 \}$

Sum-of-Subsets Code

```
void sum_of_subsets (index i, int weight, int total)
{
     if (promising(i))
          if (weight == W) // we have reached a solution. Print it and backtrack
               cout << include[1] through include[i];
          else
               // try taking the next item, then try rejecting it
               include[i + 1] = "yes";
               sum_of_subsets(i + 1, weight + w[i + 1], total - w[i + 1]);
               include[i + 1] = "no";
               sum_of_subsets(i + 1, weight, total - w[i + 1]);
}

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

A top-level call: ???

Sum-of-Subsets Code

```
void sum_of_subsets (index i, int weight, int total)
{
     if (promising(i))
          if (weight == W) // we have reached a solution. Print it and backtrack
               cout << include[1] through include[i];
          else
               // try taking the next item, then try rejecting it
               include[i + 1] = "yes";
               sum_of_subsets(i + 1, weight + w[i + 1], total - w[i + 1]);
               include[i + 1] = "no";
               sum_of_subsets(i + 1, weight, total - w[i + 1]);
}

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

A top-level call: `sum_of_subsets(0, 0, WeightOfAllItems)`

In-Class Exercise

1. Use the Backtracking algorithm for the Sum-of-Subsets problem to find the first two combinations of the following numbers that sum to W = 32;

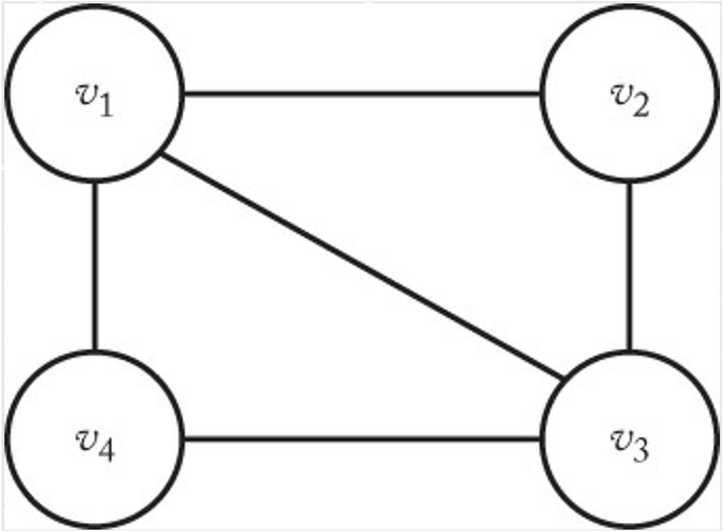
   $w_1 = 2 \quad w_2 = 10 \quad w_3 = 13 \quad w_4 = 17 \quad w_5 = 22$

Show “include”, “total”, and “weight” in each step

```
(weight + total >= W) && (weight == W || weight + w[i + 1] <= W)
```

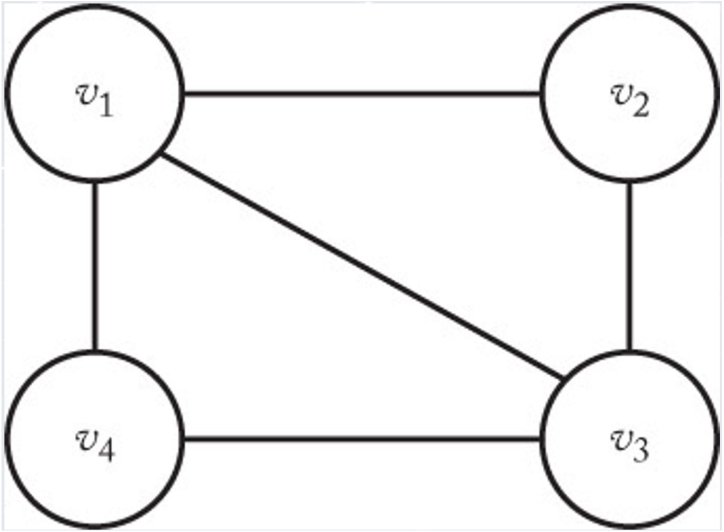
Graph Coloring

- Suppose we want to find all the ways to color the vertices in an undirected graph using at most m colors, so that no two adjacent vertices are the same color.
  - This is called the m-Coloring problem.
- How many ways can we color the following graph for $m = 2$?



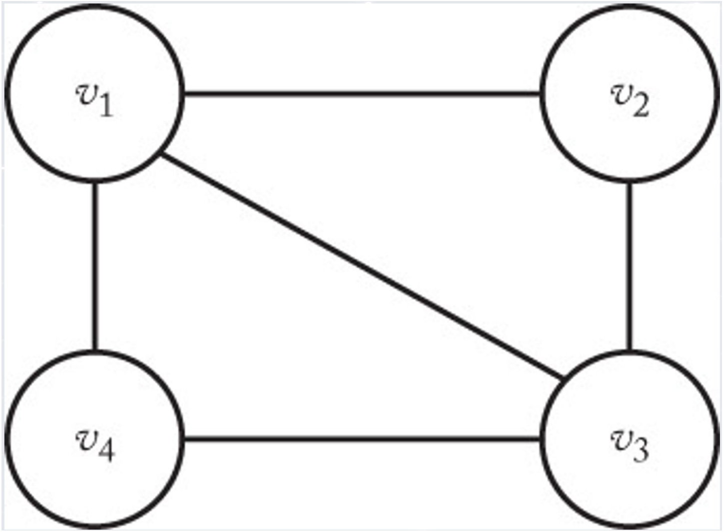
Graph Coloring

- Suppose we want to find all the ways to color the vertices in an undirected graph using at most m colors, so that no two adjacent vertices are the same color.
  - This is called the m-Coloring problem.
- How many ways can we color the following graph for $m = 2$? **None!**



Graph Coloring

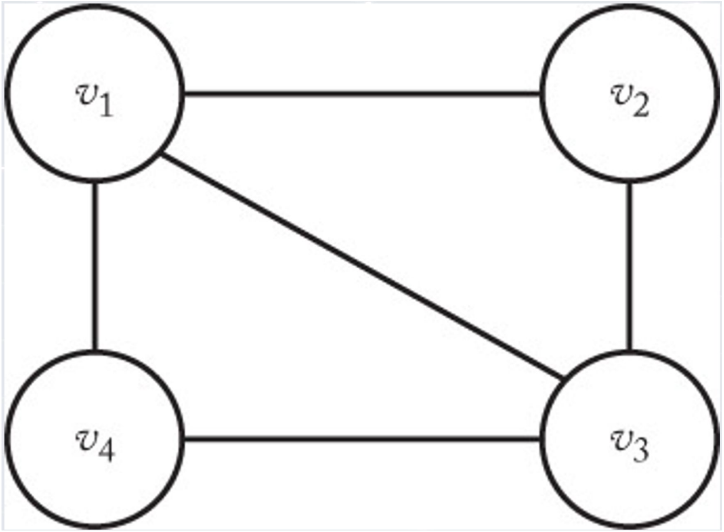
- What is a solution for the following graph for $m = 3$?



Graph Coloring

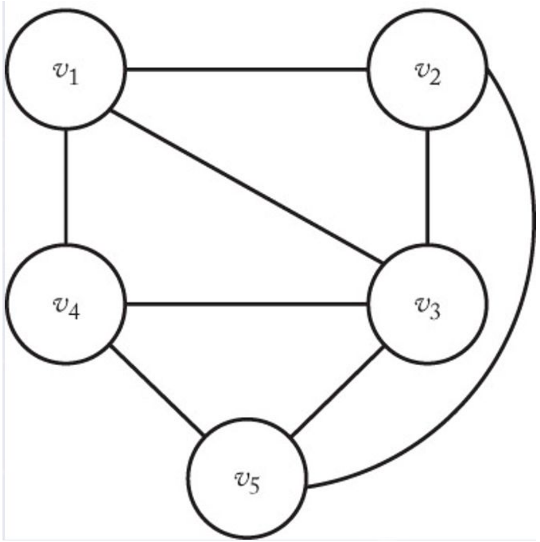
- What is a solution for the following graph for $m = 3$?
  - There are six total solutions to the 3-Coloring problem.
    - However, they only differ in the way the colors are permuted
    - i.e. $\{ c2, c1, c3, c1 \}$ vs $\{ c1, c2, c3, c2 \}$

Vertex	*Color*
v_1	color 1
v_2	color 2
v_3	color 3
v_4	color 2



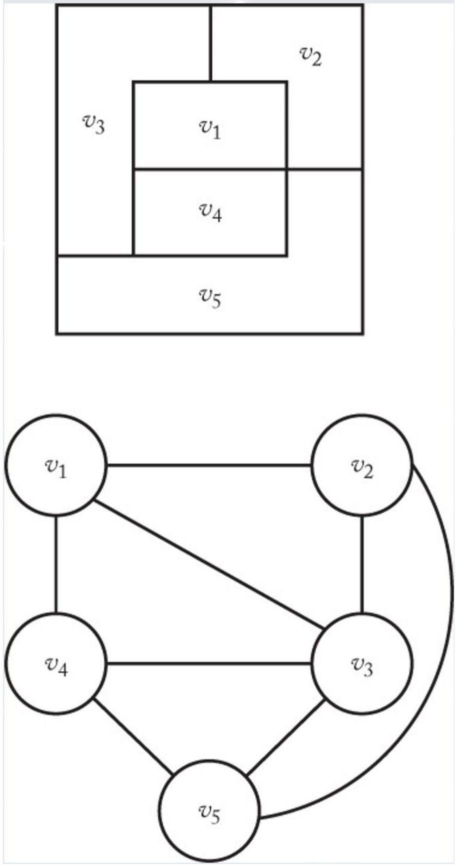
Graph Coloring

- One application of graph coloring is in the coloring of maps.
- A graph is called **planar** if it can be drawn in a plane in such a way that no two edges cross each other.
- The following graph is planar. However, if we added edges (v_1, v_5) and (v_2, v_4), it would no longer be planar.



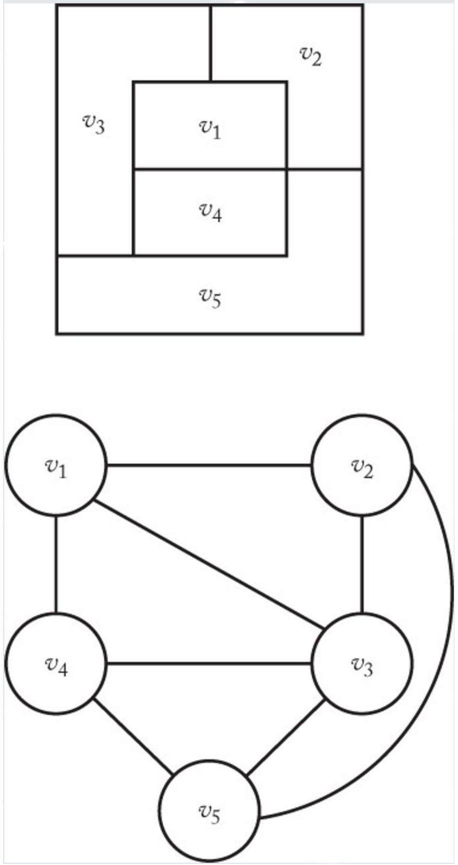
Graph Coloring

- For every map there is a corresponding planar graph.
- Each region in the map is represented by a vertex.
- If one region shares a border with another region, their vertices are connected by an edge.
- The *m*-Coloring problem for planar graphs will determine how many ways we can color a map, using at most *m* colors, so that no two neighboring regions are the same color.



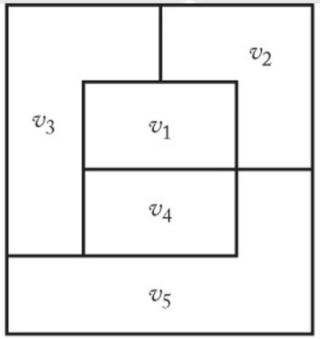
Graph Coloring

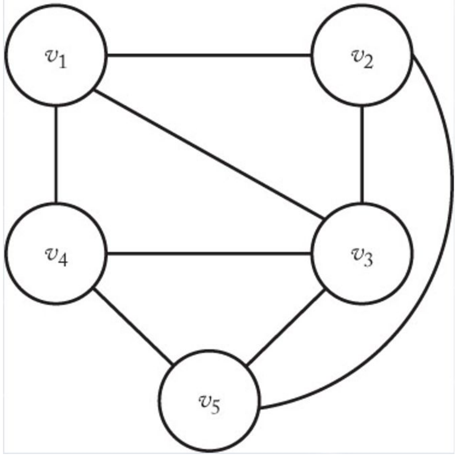
- A straightforward state space tree for the *m*-Coloring problem is one in which each color is tried for v_1 at level 1, each color is tried for v_2 at level 2, etc.
- Once every possible color has been tried for v_n at level n, the tree is complete.
- Each path from the root to a leaf is a candidate solution.
- What makes a vertex *nonpromising*?



Graph Coloring

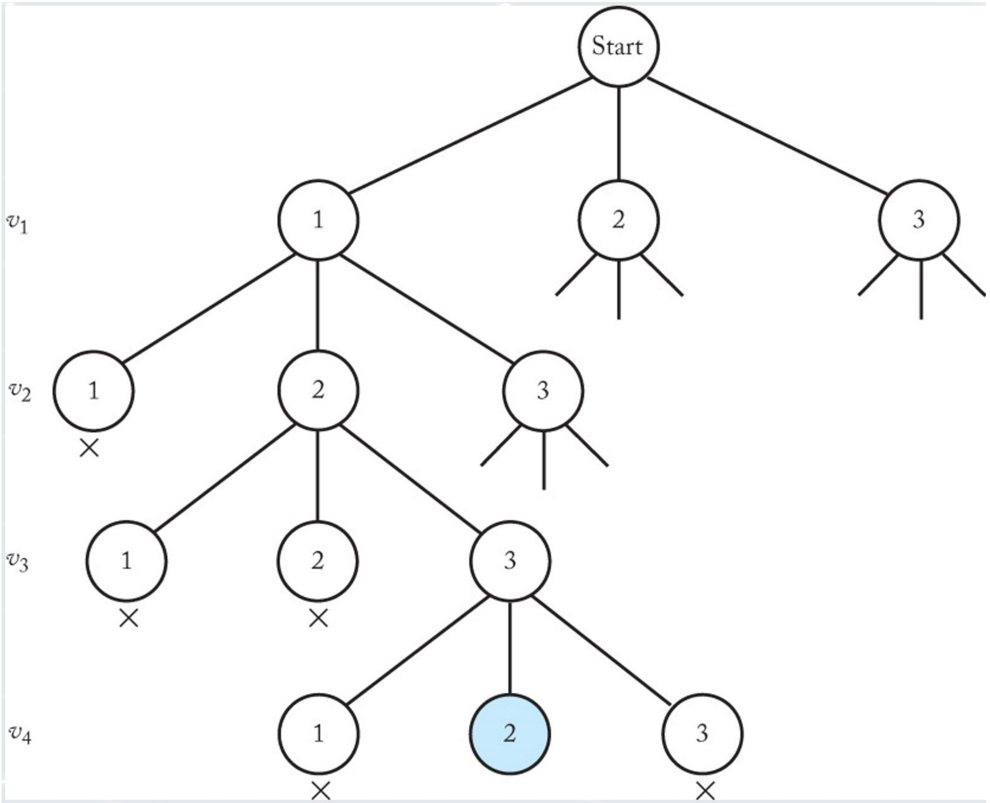
- A straightforward state space tree for the *m*-Coloring problem is one in which each color is tried for v_1 at level 1, each color is tried for v_2 at level 2, etc.
- Once every possible color has been tried for v_n at level n, the tree is complete.
- Each path from the root to a leaf is a candidate solution.
- What makes a vertex *nonpromising*?
  - If we attempt to color it the same color as a vertex adjacent to it.
    - i.e. if we color v_1 c1 and then try to color v_2 c1, we can backtrack and try to color v_2 c2.
  - Which vertices can be colored the same as v_1?

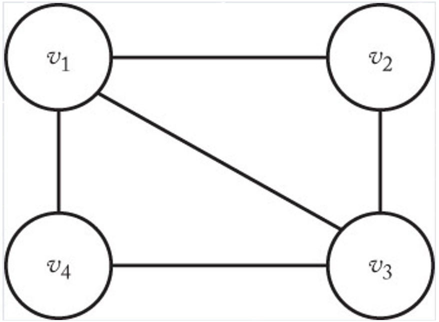




Graph Coloring

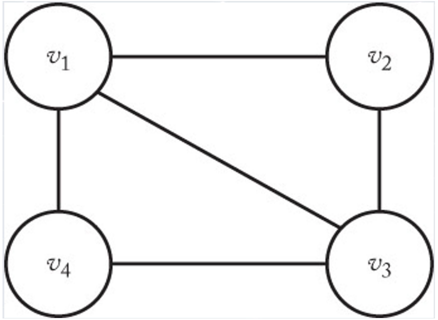
A portion of the pruned state space tree produced using backtracking to do a 3-coloring of the graph pictured below.

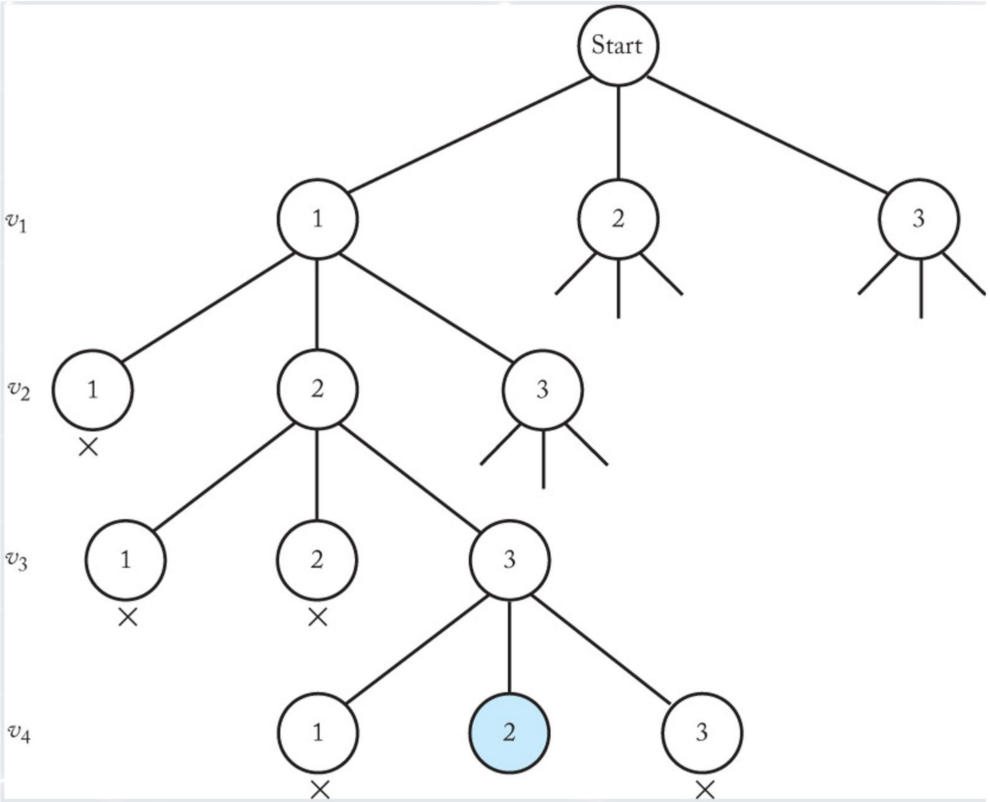




Graph Coloring

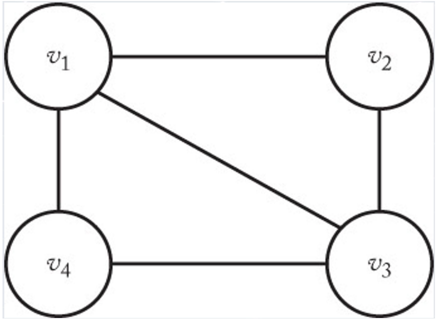
- We start by selecting color 1 for v_1.
- We next select color 1 for v_2, but v_2 is adjacent to v_1, so we can immediately prune that branch.

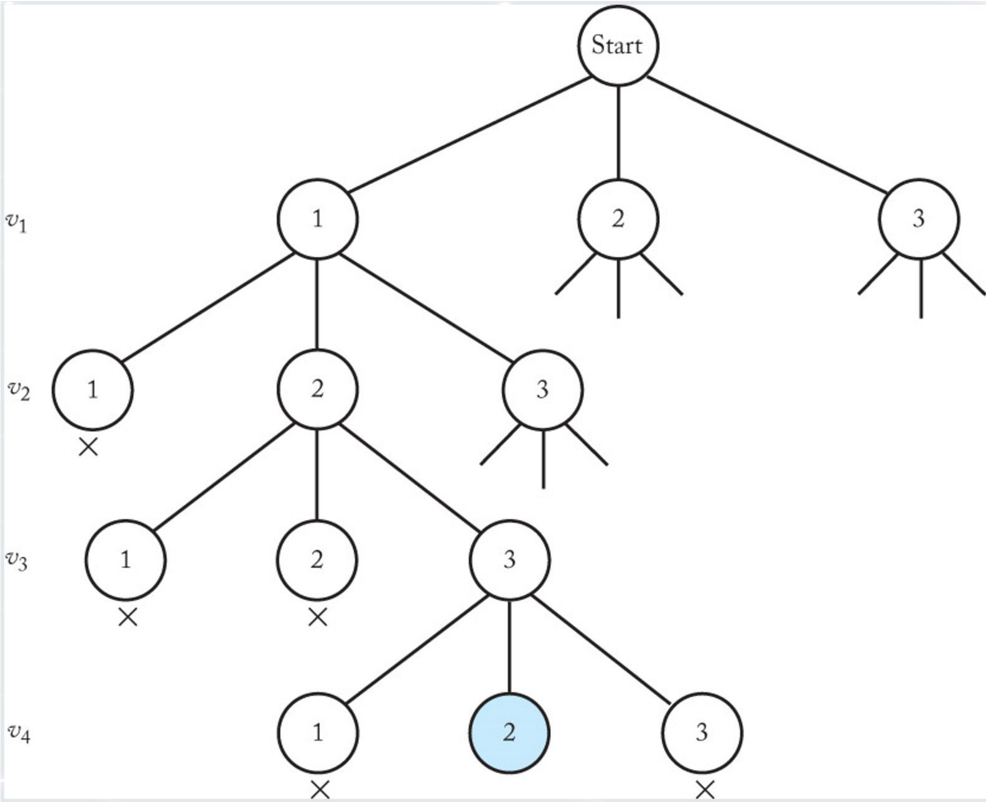




Graph Coloring

- We start by selecting color 1 for v_1.
- We next select color 2 for v_2, which is allowed because no other vertex adjacent to it is color 2.





Graph Coloring

Problem: Determine all ways in which the vertices in an undirected graph can be colored, using only m colors, so that adjacent vertices are not the same color.

Inputs: Positive integers n and m, and an undirected graph containing n vertices. The graph is represented by a two-dimensional array W, which has both its rows and columns indexed from 1 to n, where W[i][j] is true if there is an edge between the ith vertex and the jth vertex and false otherwise.

Outputs: All possible colorings of the graph, using at most m colors, so that no two adjacent vertices are the same color. The output for each coloring is an array `vcolor` indexed from 1 to n, where `vcolor[i]` is the color (an integer between 1 and m) assigned to the ith vertex.

Graph Coloring

```
void m_coloring (index i)
     int color;
     if (promising(i))    // promising returns true if adding the previous color was okay.
          if (i == n)
               cout << vcolor[1] through vcolor[n]; // print a solution
          else
               for (color = 1; color <= m; color++)
                    vcolor[i + 1] = color;             // Try each color for next vertex
                    m_coloring(i + 1);
```

We initially call `m_coloring(0)`

Graph Coloring

```
bool promising(index i)
{
     index j = 1;
     while (j < i)                    // Determine if an adjacent vertex
     {                                // is already this color
          if (W[i][j] && vcolor[i] == vcolor[j])
               return false;
          j++;
     }
     return true;
}
```

In-Class Exercise

1. Use the Backtracking algorithm for the m-Coloring problem to find the first ten possible colorings. m = 3. Show the values in vcolor (example on board)