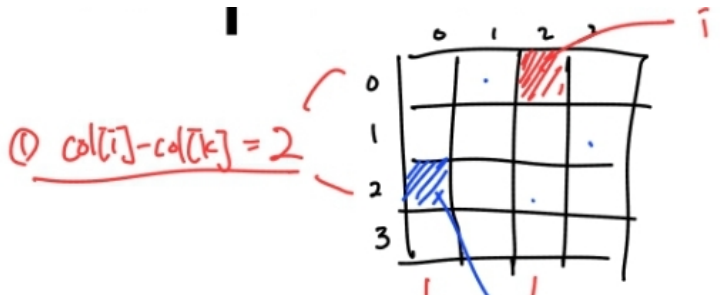
5장 백트래킹

N - queens problem
O(n!)

N-queens problem

```
void queens( index i) {
    index j;

    if (promising(i))
        if (i==n)
            system.out.print ( col[1] .. col[n] )
        else
            for (j=1; j<=n; j++) {
                col[i+1] = j;
                queens(i+1) ;
            }
}
```



❑ N-queens problem

```
public static boolean promising (index i)
{
    index k;   boolean switch ;

    k = 1;
    switch = true ;
    while (k<i && switch) {
        if (col[i]==col[k]) || abs(col[i]-col[k]) == i-k)
            switch = false ;
        k++;
    }
    return switch ;
}
```

같은 줄에 있으면.
같은 대각선에 있으면.

coi

[0]	2
[1]	4
[2]	3
[3]	1

queens 메서드에서는 dfs 방식으로 queens을 호출하고 현재 실행중인 노드들을 col에 저장한다.

같은 열, 같은 대각선이면 promising하지 않다.

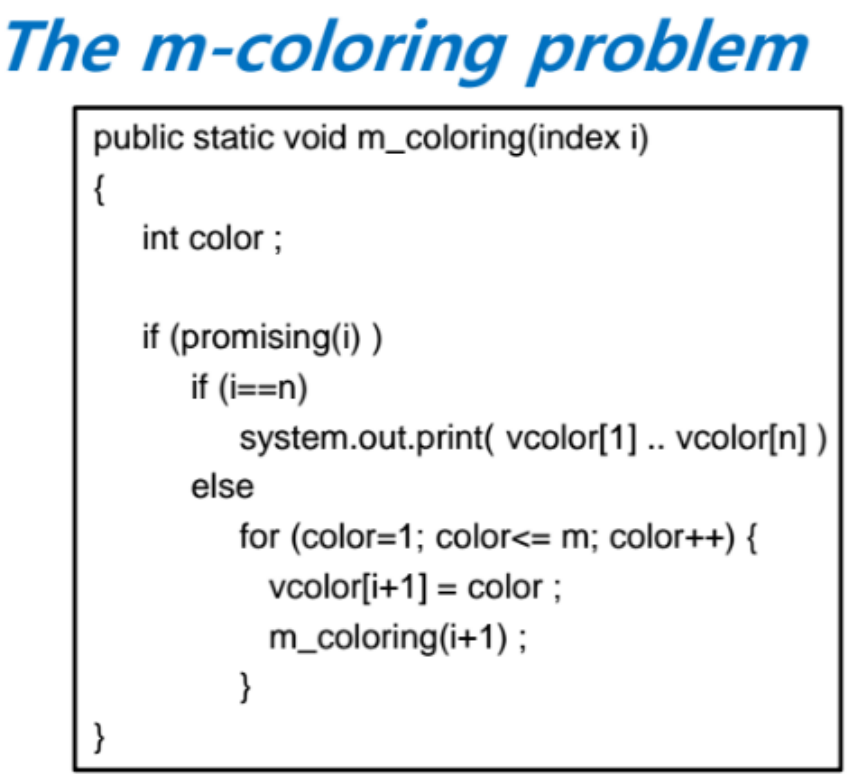
i가 층(행, 레벨)이고 k가 열에 해당한다.

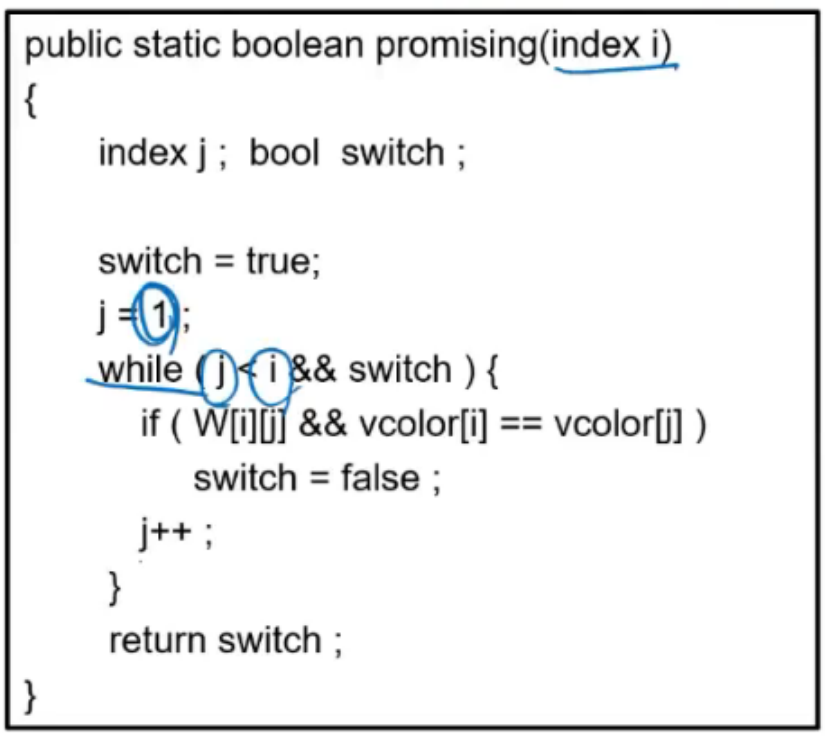
Top - level call queens(0)

coi[i]는 현재 담기는 값과 coi[k]는 이전에 담겨있는 값이 같은지(같은줄인지) or 대각선으로 같은지 확인

그래프 컬러링

O(M^V) - M은 색상의 수, V는 칠할 공간





W[i][j]는 i와 j가 연결되어있는지 true or false로 되어있음.

i와 j가 연결되어있고 (W[i][j] == true) i와 j가 같은 색이면 false를 반환한다.

25분부터 보면 이해 잘됨.

0-1 냅색을 백트래킹

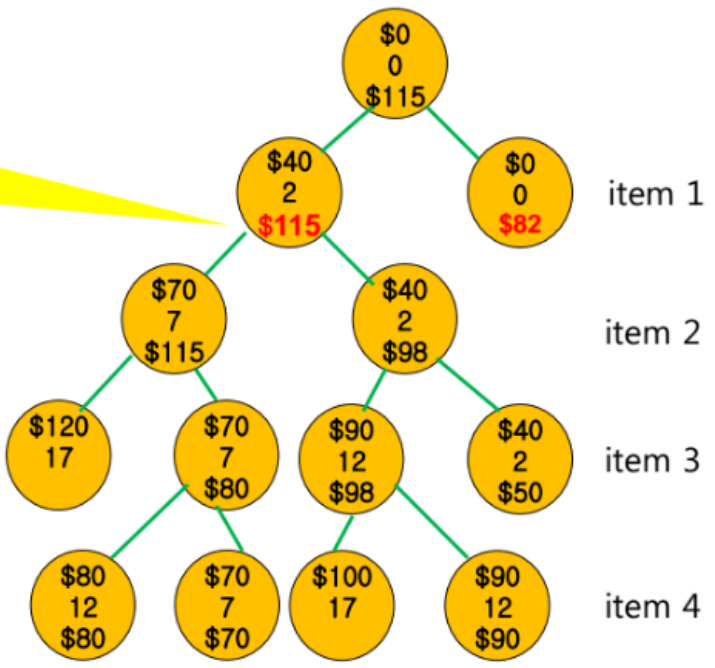
O(2^n)

❑ *Example*

40*2/2
+30*5/5
+50*(9/10)
= 115

	p_i	w_i	p_i/w_i
item 1 :	40	2	20
item 2 :	30	5	6
item 3 :	50	10	5
item 4 :	10	5	2

W = 16




```
public static void knapsack(index i, int profit, int weight)
{
      if ( weight <= W && profit > maxProfit )  {
          maxProfit = profit ;
          numBest = i ;
          bestSet = include ;
      }

      if (promising(i)) {
          include[i+1] = “yes” ;
          knapsack(i+1,profit+p[i+1],weight+w[i+1]);
          include[i+1] = “no” ;
          knapsack(i+1,profit, weight);
      }
}
```

include[i+1] = “yes”로 했으니 i+1 아이템 을 사용한다는 의미이다.
따라서 profit과 weight에 각각 p[i+1], w[i+1]을 더해준다.

```
public static bool promising(index i)  {
   index j,k ;    int totWeight ;    float bound ;

   if (weight >=W) return false ;
   else {
              j = i+1 ;  bound = profit ; totWeight = weight ;
              while ( j<=n  &&  totWeight + w[j] <= W ) {
                    totWeight = totWeight + w[j] ;
                    bound = bound + p[j] ;
                    j++ ;
              }
              k = j ;
              if (k <= n)
                    bound=bound+(W–totWeight)*p[k]/w[k];
              return bound > maxProfit ;
   }
}
```

`return bound > maxProfit` 만약 promising 안에서 구한 bound가 현재 저장된 maxProfit보다 작다면 (조건이 false라면) 아무리 지지고 볶아도 해당 동선에서는 maxProfit보다 높게 나올 수 없으므로 계산하지 않는다.