제 5 장 퇴각 검색(계속)

- 부분집합의 합
- 그래프 채색

2. 부분집합의 합

- n 개의 서로 다른 양수 w_i, $1 \le i \le n$ 들과 m 에 대하여 합이 m 이 되는 모든 부분 집합들을 찾으려고 한다.
- 가변 길이와 고정 길이의 투플을 사용할 수 있다.
- 여기서는 고정 길이의 투플을 사용한다.
  → $x_i = 1$ 이면 w_i 가 포함되며, $x_i = 0$ 이면 w_i 가 포함되지 않는다.
- 노드의 생성시 왼쪽 자식노드는 $x_i = 1$, 오른쪽 자식노드는 $x_i = 0$ 에 대응된다.
- 한정함수의 선택:
  $B_k(x_1, x_2, \ldots, x_k) = \text{true} \leftrightarrow \sum_{1\le i\le k} w_i x_i + \sum_{k+1\le i\le n} w_i \ge m$
  이 조건을 만족하지 못한다면 해답 노드에 이르지 못한다.
- 한정함수의 추가적인 선택:
  w_i 들이 오름차순으로 되어 있을 때,
  $\sum_{1\le i\le k} w_i x_i + w_{k+1} > m$ 이라면 $(x_1, x_2, \ldots, x_k)$는 해답 노드에 도달할 수 없다.
- 위 두가지 성질을 합친 한정함수:
  $B_k(x_1, x_2, \ldots, x_k) = \text{true} \leftrightarrow \sum_{1\le i\le k} w_i x_i + \sum_{k+1\le i\le n} w_i \ge m$ and
  $\sum_{1\le i\le k} w_i x_i + w_{k+1} \le m$

순환 퇴각검색 알고리즘

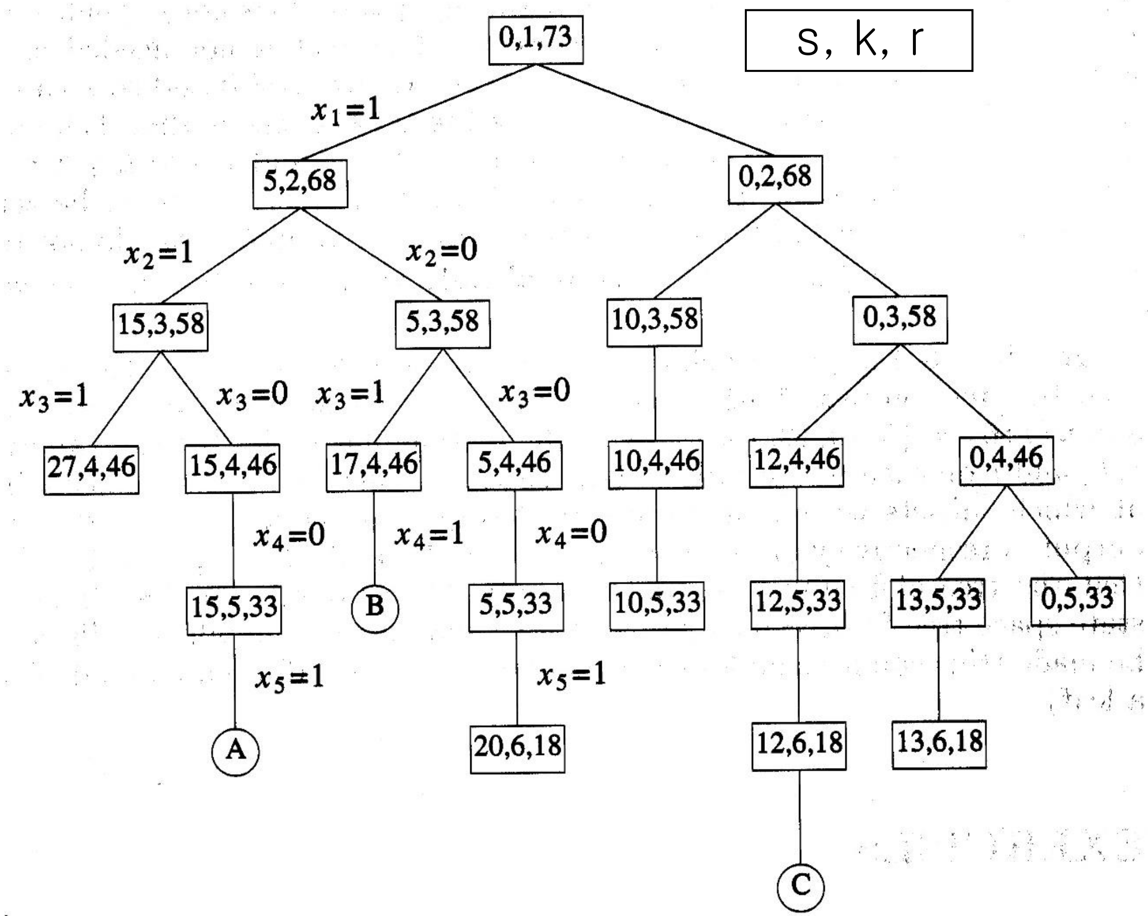
- w_i 들은 오름 차순으로 정렬되어 있다.
- $w_1 \le m$, $\sum_{1\le i\le n} w_i \ge m$ 을 가정한다.
- 노드에서 한정함수의 값을 효율적으로 계산하기 위해서 $\sum_{1\le i\le k} w_i x_i$ 의 값을 변수 s, $\sum_{k+1\le i\le n} w_i$ 의 값을 변수 r에 저장한다.
- 처음 호출 시에 s = 0, r = $\sum_{1\le i\le n} w_i$ 로 설정되며,
  왼쪽 자식 노드로 이동(x_i =1) 시에는 s = s + w_i, r = r − w_i 로 되고,
  오른쪽 자식노드로 이동 (x_i =0) 시에는 s = s, r = r − w_i 로 된다.

부분집합의 합(lec11-1)

```
void SumOfSub(float s, int k, float r)
 {
    // Generate left child. Note that s+w[k] <= m
    // because B_{k-1} is true.
    x[k] = 1;
    if (s+w[k] == m) { // Subset found
       for (int j=1; j<=k; j++) System.out.print(x[j]+" ");
       System.out.println();
    }
        // There is no recursive call here
        // as w[j] > 0, 1 <= j <= n.
    else if (s+w[k]+w[k+1] <= m)
            SumOfSub(s+w[k], k+1, r-w[k]);
    // Generate right child and evaluate B_k.
    x[k] = 0;
    if ((s+r-w[k] >= m) && (s+w[k+1] <= m)) {
            SumOfSub(s, k+1, r-w[k]);
    }
 }
```

예제 6

- n = 6, m = 30, w[1:6] = [5, 10, 12, 13, 15, 18]



A: (1,1,0,0,1)

B: (1,0,1,1)

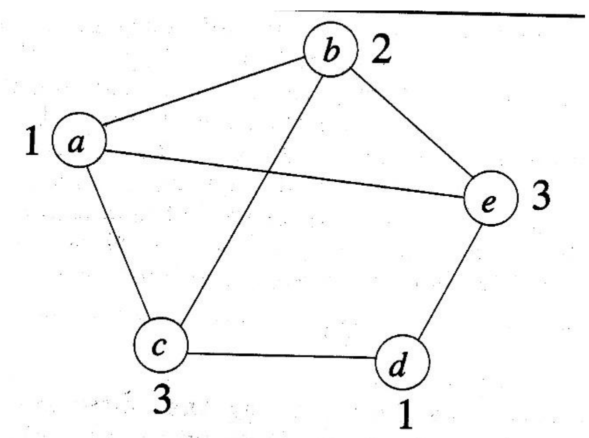
C: (0,0,1,0,0,1)

3. 그래프 채색

- m-채색 가능성 결정 문제(m-colorability decision problem):

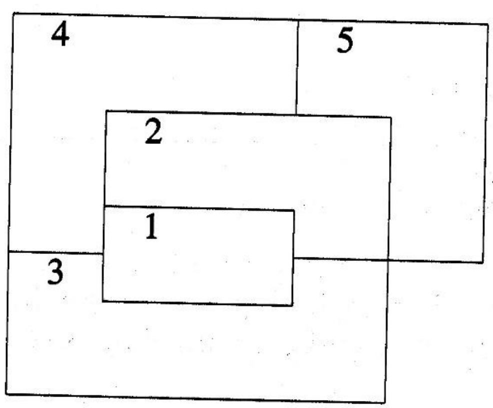
  G를 그래프, m 을 양의 정수라 할 때, G의 노드들을 m 개의 색을 사용하여 인접한 어떠한 두 노드들도 같은 색을 갖지 않도록 색칠할 수 있는가의 여부를 결정한다.
- d 가 주어진 그래프의 차수일 때, d+1 개의 색으로 색칠할 수 있다.
- m-채색 최적화 문제(m-colorability optimization problem):

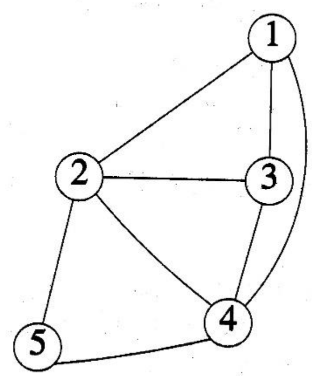
  그래프 G를 색칠할 수 있는 최소의 정수 m을 구하는 것이다.
- 예: 아래 그래프의 최소 채색수는 3이다.



평면 그래프(planar graph)

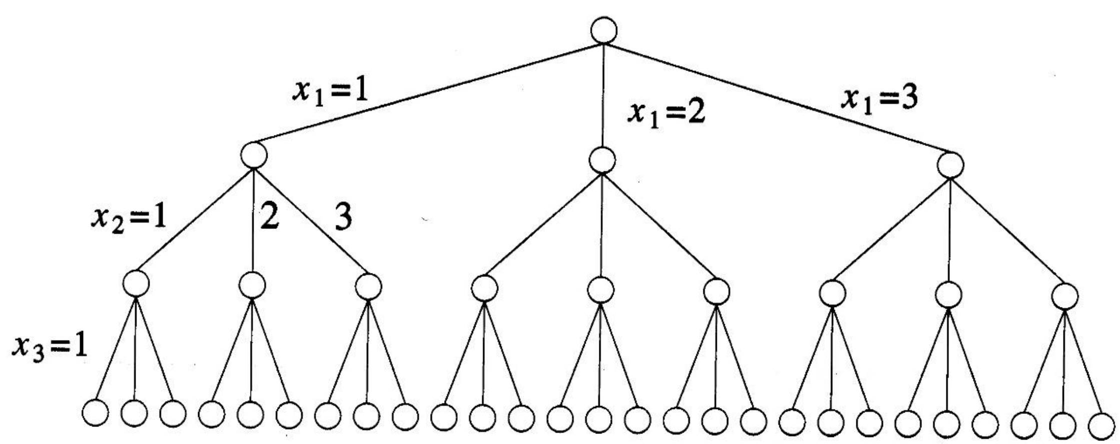
- 한 평면에 어떤 두 간선도 서로 겹치지 않는 그래프이다.
  예: 지도
- 평면 그래프의 m-채색 가능성 문제:
  5-채색 문제: 어떠한 평면 그래프(지도)도 5가지 색으로 채색 가능
  4-채색 문제: 오랫동안 증명되지 못하다 1970년대 중반에 증명됨
- 평면 그래프 채색 문제는 일반 그래프의 정점 채색 문제로 변환할 수 있다.



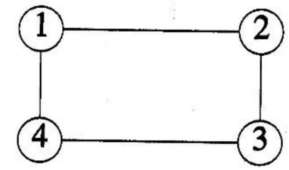


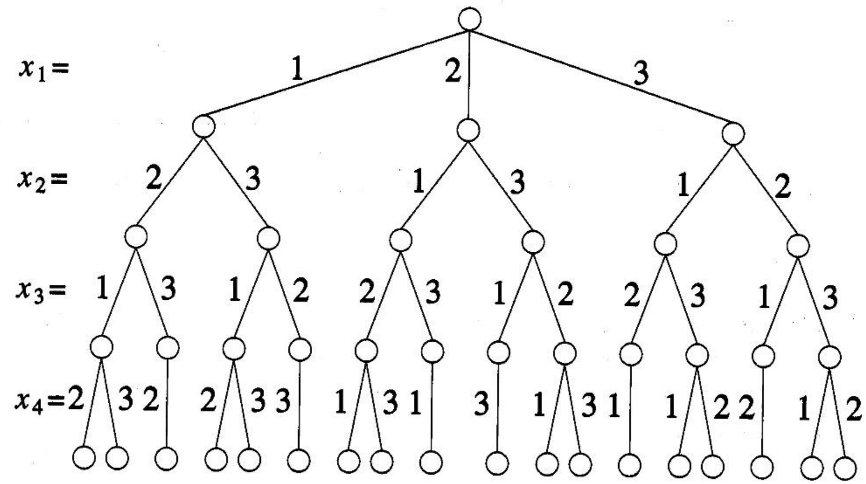
m-채색 가능성 문제

- n 개의 정점을 갖는 그래프는 인접 행렬 G[1:n][1:n] 로 표현된다.
  즉, (i, j)가 간선이면 G[i][j] = 1, 그렇지 않으면 G[i][j] = 0 이다.
- m 개의 색 들은 1, 2, ..., m의 정수로 표현된다.
- 채색 문제의 해들은 n-투플 (x_1, x_2, ..., x_n) 으로 주어지며, x_i는 노드 i 의 색이다.
- 상태 공간 트리는 차수가 m, 높이가 n+1인 트리이다.
  예: n = 3, m = 3 인 경우의 모든 가능한 노드를 가진 상태 공간 트리



- 인접한 노드의 경우 같은 색을 칠할 수 없는 제약조건 때문에 많은 노드들이 제한될 수 있다.
- 예: 오른쪽 그림 (n=4, m=3)
- 단말 노드만이 해답 노드가 된다





그래프 채색(lec11-2)

```
void mColoring(int k) {
  // This program was formed using the recursive backtracking  schema.
  // The graph is represented by its boolean adjacency matrix G[1:n][1:n].
  // All assignments of 1,2,...,m to the vertices of the graph such that adjacent
  // vertices are assigned distinct integers are printed. k is the index of
  // the next vertex to color.
      do { // Generate all legal assignments for x[k].
         NextValue(k); // Assign to x[k] a legal color.
         if (x[k]==0) break; // No new color possible
         if (k == n) { // At most m colors have been used
                          // to color the n vertices.
            for (int i=1; i<=n; i++) System.out.print(x[i]+" ") ;  // 하나의 해 출력
            System.out.println();
         }
         else mColoring(k+1);
      } while(true);
  }
```

- 해를 저장하는 배열 x[1:n]는 초기값으로 0를 가지며, 위 순환함수는 mColoring(1)을 호출함으로써 시작한다.

```
void NextValue(int k)
  // x[1],..., x[k-1] have been assigned integer values in
  // the range [1,m] such that adjacent vertices have distinct
  // integers. A value for x[k] is determined in the range
  // [0,m]. x[k] is assigned the next highest numbered color
  // while maintaining distinctness from the adjacent vertices
  // of vertex k. If no such color exists, then x[k] is zero.
  {
      do {
         x[k] = (x[k]+1) % (m+1); // Next highest color
         if (x[k]==0) break;          // All colors have been used.
         int j ;
         for (j=1; j<=n; j++) {    // Check if this color is
                                      // distinct from adjacent colors.
            if ( G[k][j]==1           // If (k, j) is an edge and if adj.
                  && (x[k] == x[j]))   // vertices have the same color.
                  break;
         }
         if (j == n+1) break; // New color found
      } while (true); // Otherwise try to find another color.
  }
```

시간 복잡도

- 이 알고리즘에서 생성되는 노드(즉 상태)들 중 내부 노드의 수는 최대 $\sum_{0 \le i \le n-1} m^i = m^n-1$ 이다. 각 내부 노드에서는 NextValue 에 의해 적법한 채색에 대응하는 자식 노드를 결정하기 위해 O(m·n) 시간이 소요된다.
- 따라서 전체 시간은 $O(n \cdot m^n)$ 이다.