알고리즘 문제해결기법

제5주 Enumeration (continued)

순열(permutation)

- 임의의 집합 data에 대해서 원소들의 모든 가능한 순열을 출력하라.

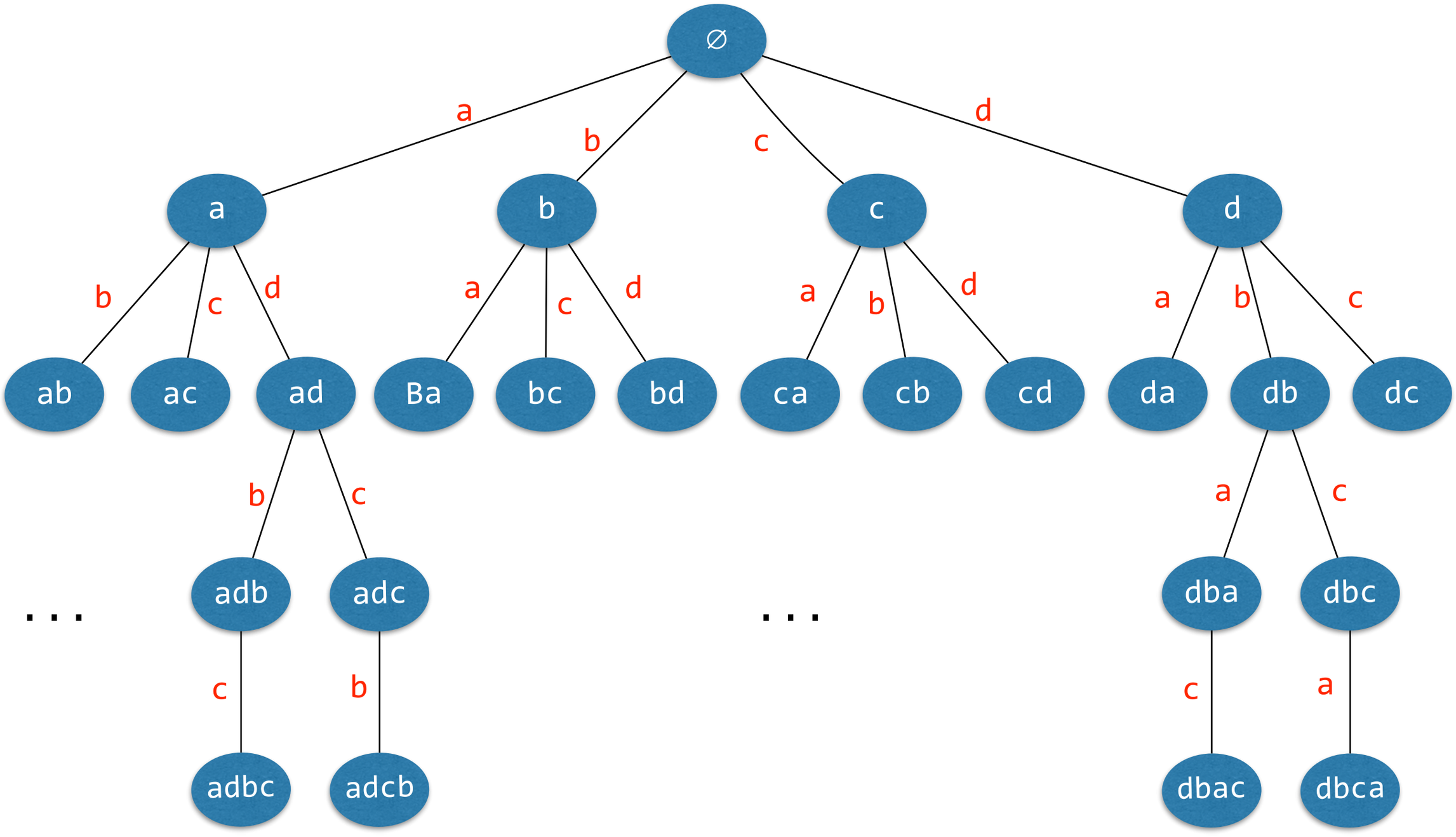
data = {a, b, c, d}

a b c d
a b d c
a c b d
a c d b
a d b c
a d c b
b a c d
b a d c
b c a d
…

4!=24개

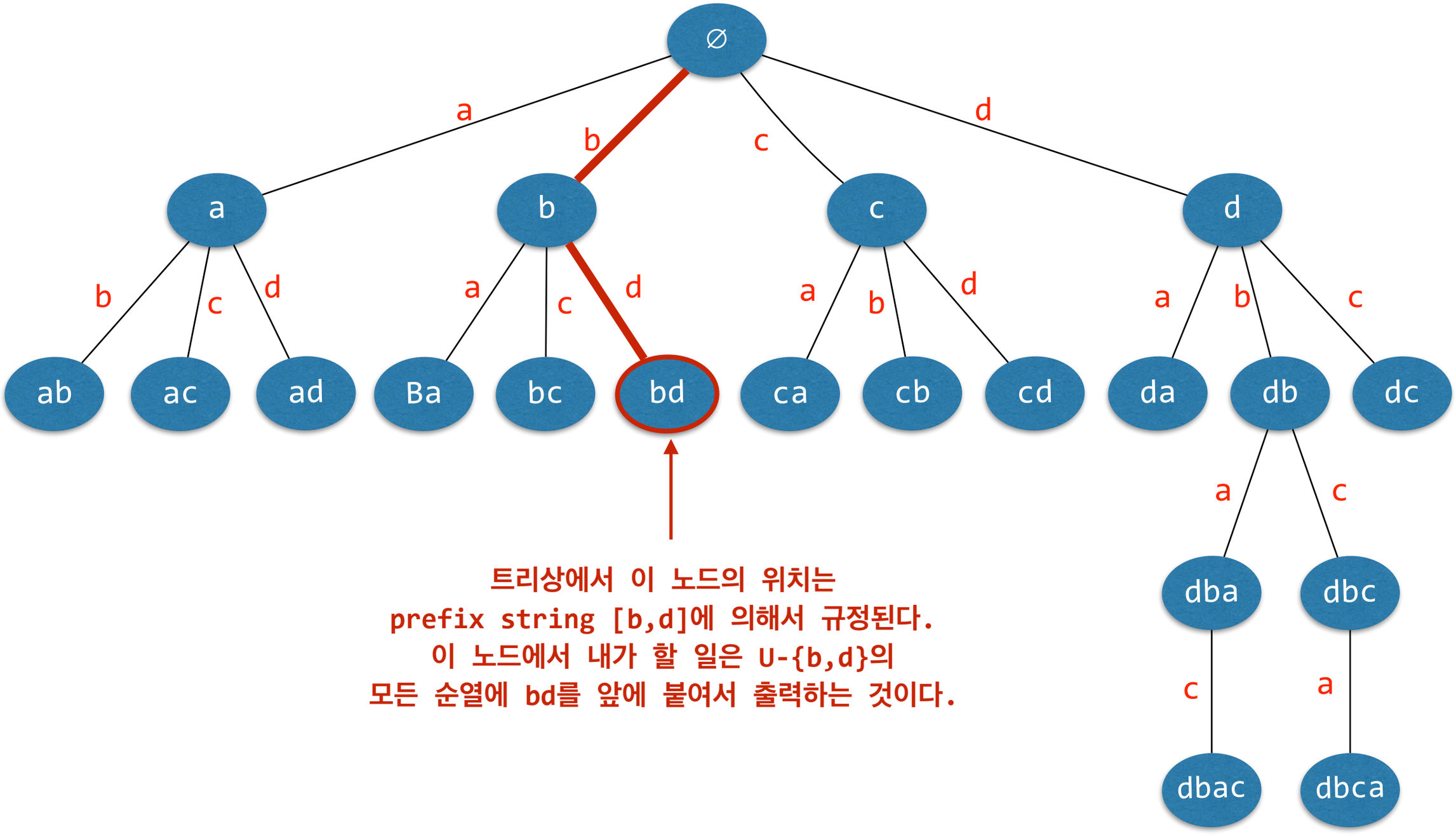
{a,b,c,d}

상태공간트리 (state space tree)



U={a,b,c,d}

상태공간트리 (state space tree)



트리상에서 이 노드의 위치는
prefix string [b,d]에 의해서 규정된다.
이 노드에서 내가 할 일은 U-{b,d}의
모든 순열에 bd를 앞에 붙여서 출력하는 것이다.

순열(permutation)

- {a,b,c,d}의 모든 순열
  - 첫 원소가 a이면서 {b,c,d}의 모든 순열
  - 첫 원소가 b이면서 {a,c,d}의 모든 순열
  - 첫 원소가 c이면서 {a,b,d}의 모든 순열
  - 첫 원소가 d이면서 {a,b,c}의 모든 순열

순열(permutation)

a	b	c	d

이것의 모든 순열을
출력하기 위해서는

a	b	c	d

b,c,d의 모든 순열에
a를 앞에 추가하여 출력하고,

b	a	c	d

a,c,d의 모든 순열에
b를 앞에 추가하여 출력하고,

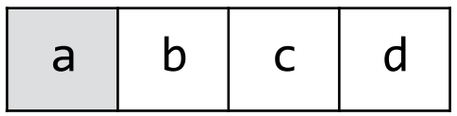
c	b	a	d

b,a,d의 모든 순열에
c를 앞에 추가하여 출력하고,

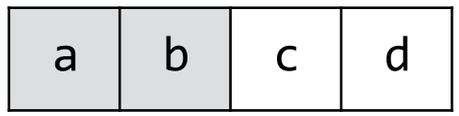
d	b	c	a

b,c,a의 모든 순열에
d를 앞에 추가하여 출력한다.

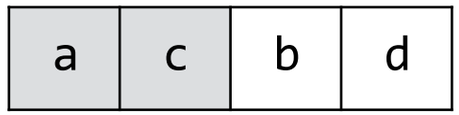
순열(permutation)



b,c,d의 모든 순열에
a를 앞에 추가하여 출력하려면



c,d의 모든 순열에
a,b를 앞에 추가하여 출력하고,



b,d의 모든 순열에
a,c를 앞에 추가하여 출력하고,

c,b의 모든 순열에
a,d를 앞에 추가하여 출력한다.

순열(permutation)

a	b	c	d

c,d의 모든 순열에
a,b를 앞에 추가하여 출력하려면

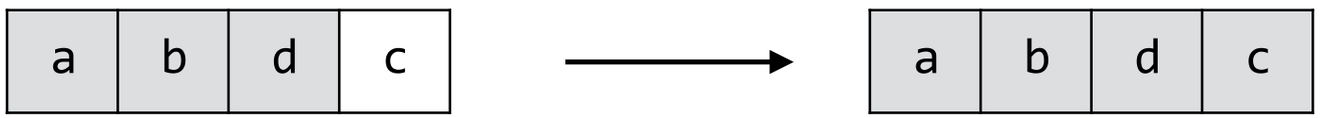
a	b	c	d

d의 모든 순열에
a,b,c를 앞에 추가하여 출력하고,

a	b	d	c

c의 모든 순열에
a,b,d를 앞에 추가하여 출력한다.

순열(permutation)



c의 모든 순열에
a,b,d를 앞에 추가하여
출력하려면

∅의 모든 순열에
a,b,d,c를 앞에 추가하여
출력한다.

순열(permutation)

집합 S={a,b,c,d}의 모든 순열을 출력하려면

S의 각 원소 x에 대해서

S-{x}의 모든 순열들을 생성한 다음 각각의 앞에 x를 추가해서 출력한다.

순열(permutation)

S-{a}의 모든 순열들을 생성한 다음 각각의 맨 앞에 a를 추가해서 출력하려면

S-{a}의 각 원소 y에 대해서

S-{a,y}의 모든 순열들을 생성한 다음 각각의 맨 앞에 (a,y)를 추가해서 출력한다.

The interface should be

Mission: S의 모든 순열들을 생성한 후 각각에 prefix string을 앞에 붙여서 출력한다.

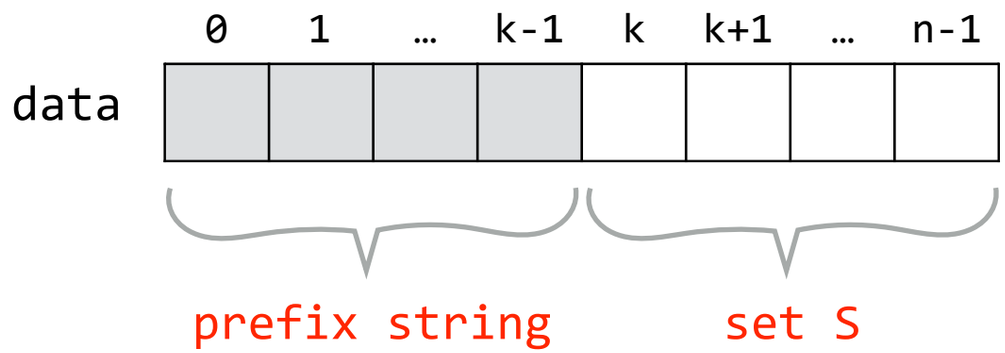
```
void printPerm(a prefix string, a set S)
{
   if |S| is 0
      print the prefix string;
   else
      for each element x in S
         printPerm(the prefix string + x, S-{x});
}
```

순열(permutation)

```
char data[] = {'a','b','c','d'};
int n=4;
```

Prefix string의 길이

```
void perm(int k)  {
    if (k==n) {
        print data[0…n-1];
        return;
    }
    for (int i=k; i<n; i++)  {
        swap(k, i);  // swap data[k] and data[i]
        perm(k+1);
    }
}
```



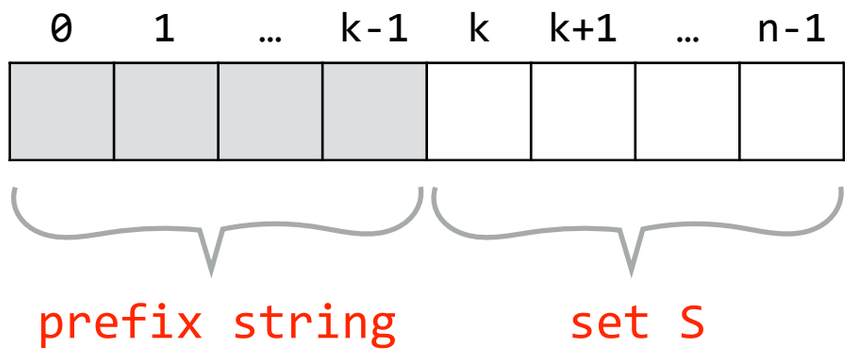
Is it Okay ?

순열(permutation)

```
char data[] = {'a','b','c','d'};
int n=4;

void perm(int k)  {
    if (k==n) {
        print data[0…n-1];
        return;
    }
    for (int i=k; i<n; i++)  {
        swap(k, i);
        perm(k+1);
    }
}
```

이 호출 이후에 data[k+1,…,n-1]의 데이터의 순서가 유지된다는 보장이 없음



Designing recursion

- recursion이 데이터를 변경할 때는 매우 조심해야한다.
- 호출 전후에 데이터가 변경되지 않고 유지되도록 하는게 좋다.

순열(permutation)

```
void perm(int k)  {
   if (k==n)  {
      print data[0…n-1];
      return;
   }
   for (int i=k; i<n; i++)  {
      swap(k, i);
      perm(k+1);
      swap(k, i);
   }
}
```

Tip:
리커젼에 들어가기 전과 후에 데이터의 동일성이 유지되도록 하라.

Mission:
data[0..k-1]을 prefix로 하고,
data[k..n-1]으로 만들 수 있는 모든 순열 앞에 prefix를 붙여서 프린트하되,
배열 data[k..n-1]에 저장된 값들의 순서는 그대로 유지한다.

Lexicographically Next Permutation

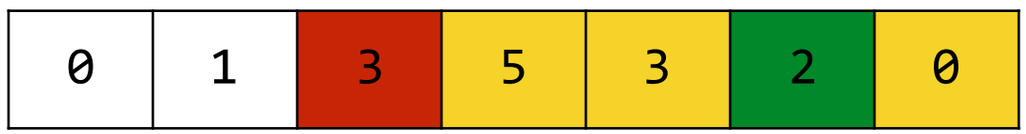
1. Initial

2. Find longest non-increasing suffix.

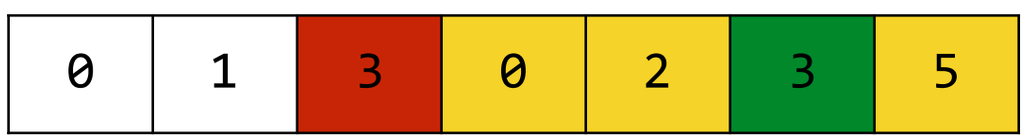
3. Identify pivot

4. Find rightmost successor to pivot in the suffix.

5. Swap with pivot



6. Reverse the suffix



Next Permutation in Java

```
boolean nextPermutation(int [] array) {
  // Find longest non-increasing suffix
  int i = array.length - 1;
  while (i > 0 && array[i - 1] >= array[i])
    i--;
if (i <= 0)
  return false;   // no next permutation exists

// Find rightmost element larger than the pivot array[i-1]
int j = array.length - 1;
while (array[j] <= array[i - 1])
  j—;
```

Next Permutation in Java

```
    // Swap the pivot with j
    int temp = array[i-1];
    array[i-1] = array[j];
    array[j] = temp;

    // Reverse the suffix
    j = array.length - 1;
    while (i < j) {
      temp = array[i];
      array[i] = array[j];
      array[j] = temp;
      i++;
      j--;
    }
    return true;
}
```

사전식 순서로 출력하려면 먼저 데이터를 정렬한 후 next permutation이 존재하지 않을 때까지 반복 호출한다.

Next Permutation in C++

```cpp
#include <iostream>
#include <algorithm>

int main () {
  int data[] = {1,2,3};
  std::sort(data, data+3);
  do {
    std::cout << data[0] << ' ' << data[1] << ' ' << data[2] << '\n';
  } while (std::next_permutation(data, data+3));

  std::cout << data[0] << ' ' << data[1] << ' ' << data[2] << '\n';
  return 0;
}
```

C++ 표준 라이브러리가
next_permutation 함수 제공

1번: 경로의 개수

- 출발 정점 s와 도착 정점 t, 그리고 경로의 길이 L이 주어질 때 s에서 t까지 가는 길이가 L이하인 모든 경로의 개수를 카운트하여 출력하는 프로그램을 작성하라.

1번: 경로의 개수

```
def count_path(self, u, v, len):
    if len < 0:
        return 0
    if u == v:
        return 1
    self.visited.append(u)
    count = 0
    for w in self.adj[u]:
        if w not in self.visited:
            count += self.count_path(w, v)
self.visited.remove(u)
return count
```

```
    self.visited.append(u)
    count = sum([self.count_path(w, v,len-1) for w in self.adj[u] if w not in self.visited])
    self.visited.remove(u)
    return count
```

전형적인 Enumeration의 형태:

Order doesn't matter.➡ Subset

- ## 부분집합의 합 (Subset-Sum)

  N개의 양의 정수의 집합과 또다른 하나의 정수 K가 주어진다. 합이 정확히 K가 되는 부분집합이 존재하는가?

- ## 배낭 문제 (Knapsack)

  무게와 가격이 정해진 N개의 아이템과 용량이 W인 배낭이 있다. 배낭의 용량을 초과하지 않으면서 가격의 합이 최대가 되도록 아이템을 배낭에 넣으려면?

- ## 잔돈 바꾸기

  어떤 나라는 7원, 15원, 120원, 378원, 897원짜리 동전을 사용한다. 잔돈 21,579원을 만들 때 동전의 개수를 최소화하려면 각 동전을 몇개씩 사용해야 하는가?

- ## Maximum Clique / Independent Set

  그래프에서 가장 큰 완전 부그래프(clique) 혹은 서로 아무도 인접하지 않은 가장 큰 정점 부분집합은?

- ## SAT (Satisfiability Problem)

  $(x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee \bar{x_3} \vee x_5) \wedge (\bar{x_2} \vee x_4 \vee x_5)$와 같은 형태의 boolean 식을 참으로 만드는 truth assignment가 존재하는가?

전형적인 Enumeration의 형태:

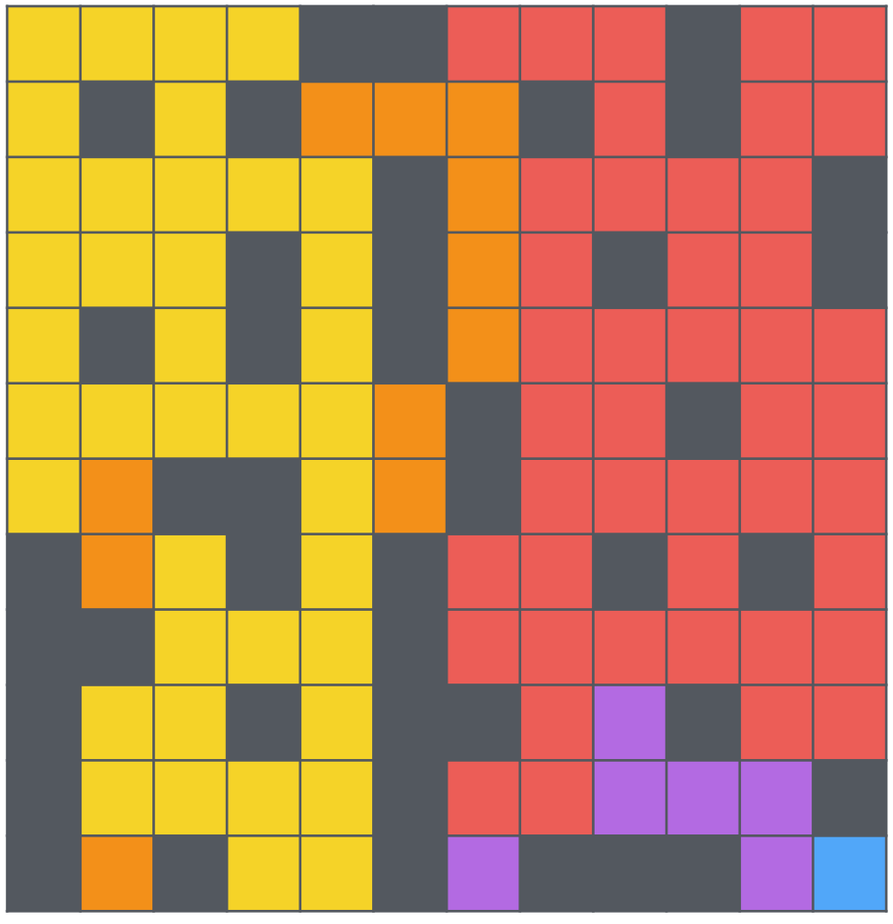
Order matters. ➡ Permutation

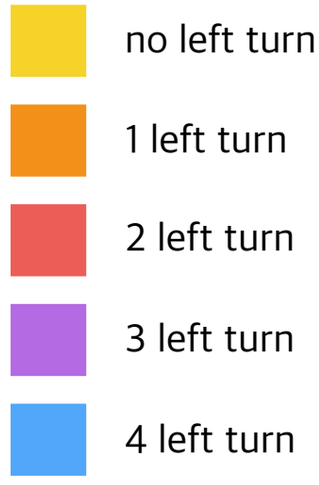
- TSP (Traveling Salesperson's Problem)
- Matching between N men and N women

  목적함수(objective function)이 뭔가에 따라 쉬운 문제일 수도 있고 어려운 문제일 수도 있음
- 그래프에서 두 정점간의 경로의 개수
- Hamiltonian Cycle

  무방향 그래프에서 어떤 정점도 두 번 방문하지 않고 모든 정점을 방문하는 사이클

좌회전 횟수 최소화





유한하고(finite) 이산적인(discrete) 공간을
탐색(search), 나열(enumerate), 혹은 순회(traverse)하는
가장 간단한 두 가지 기법

- **DFS**
  - 스택 혹은 리커전(recursion)으로 구현
- **BFS**
  - 큐(queue)를 이용해서 구현
  - 유한하지 않은 공간에서도 적용 가능
  - **최단 경로를 찾게 됨 (가중치가 없는/동일한 경우에)**

가중치가 있는 경우라면 Dijkstra의 알고리즘 등을 사용해야 함

좌회전 횟수 최소화

1. S_0는 출발점에서 좌회전없이 도달할 수 있는 모든 위치들의 집합이다.
2. 좌회전 횟수 k=1,2,…에 대해서
   (1) $S_k=\varnothing$
   (2) S_{k-1}에 속한 각각의 위치 v에 대해서
      (a) 만약 v에서 좌회전한 위치 w가 유효한 새로운 위치이면
         (i) w에서 출발하여 좌회전 없이 도달 가능한 모든 새로운 위치들의 집합을 S(w)라고 하자.
         (ii) $S_k=S_k \cup S(w)$
   (3) 만약 출구가 S_k에 속하면 k를 반환하고 종료한다.
   (4) 만약 S_k가 ∅이면 “출구로 가는 경로가 없다”고 출력하고 종료한다.

좌회전 횟수 최소화

1. S_0는 출발점에서 좌회전없이 도달할 수 있는 모든 위치들의 집합이다.
2. 좌회전 횟수 k=1,2,…에 대해서 ← 좌회전 횟수에 대해서 BFS
   (1) $S_k=\varnothing$
   (2) S_{k-1}에 속한 각각의 위치 v에 대해서 ← 큐(queue)를 사용
      (a) 만약 v에서 좌회전한 위치 w가 유효한 새로운 위치이면
         (i) w에서 출발하여 좌회전 없이 도달 가능한 모든 새로운 위치들의 집합을 S(w)라고 하자. ← 지난 주에 DFS로 구현해 둔 것을 그대로 사용하자.
         (ii) $S_k=S_k \cup S(w)$
   (3) 만약 출구가 S_k에 속하면 k를 반환하고 종료한다.
   (4) 만약 S_k가 ∅이면 "출구로 가는 경로가 없다"고 출력하고 종료한다.

좌회전 횟수 최소화

```
void solveInstance() {
    Position s = {.x = 0, .y = 0, .z = 0};

    bool success = maze(s, 1);    // s에서 좌회전 없이 갈수 있는 모든 위치에 1을 mark하고
                                  // 그 위치들을 전부 queue에 넣는다.
                                  // 출구에 도달하면 true를 return한다.
    if (success) {
        printf("0\n");
        return;
    }
    while (!isEmpty()) {
        Position cur = dequeue();
        Position w = next_pos(cur, 3);    // cur에서 죄회전한 위치를 w라고 하고

        bool success = maze(w, visited[cur.x][cur.y][cur.z]+1);

        if (success) {
            printf("%d\n", visited[cur.x][cur.y][cur.z]);
            return;
        }
    }
    printf("-1\n");
}
```

w에서 좌회전 없이 갈수 있는 모든 위치에 1증가된 값을 mark하고 그 위치들을 전부 queue에 넣는다.

좌회전 횟수 최소화

p에서 좌회전 없이 갈수 있는 모든
위치에 num_turn으로 mark하고
그 위치들을 전부 queue에 넣는다.

```
bool maze(Position p, int num_turn) {
    if (p.x < 0 || p.x >= n || p.y < 0 || p.y >= n ||
                board[p.x][p.y] != 0 || visited[p.x][p.y][p.z] != 0)
        return false;

    if (p.x == n-1 && p.y == n-1)
        return true;

    visited[p.x][p.y][p.z] = num_turn;
    enqueue(p);
    for (int turn = 0; turn < 2; turn++) {  // 0 for straight, 1 for right turn
        Position q = next_pos(p, turn);
        if (maze(q, num_turn))
            return true;
    }
    return false;
}
```