Problem Solving:

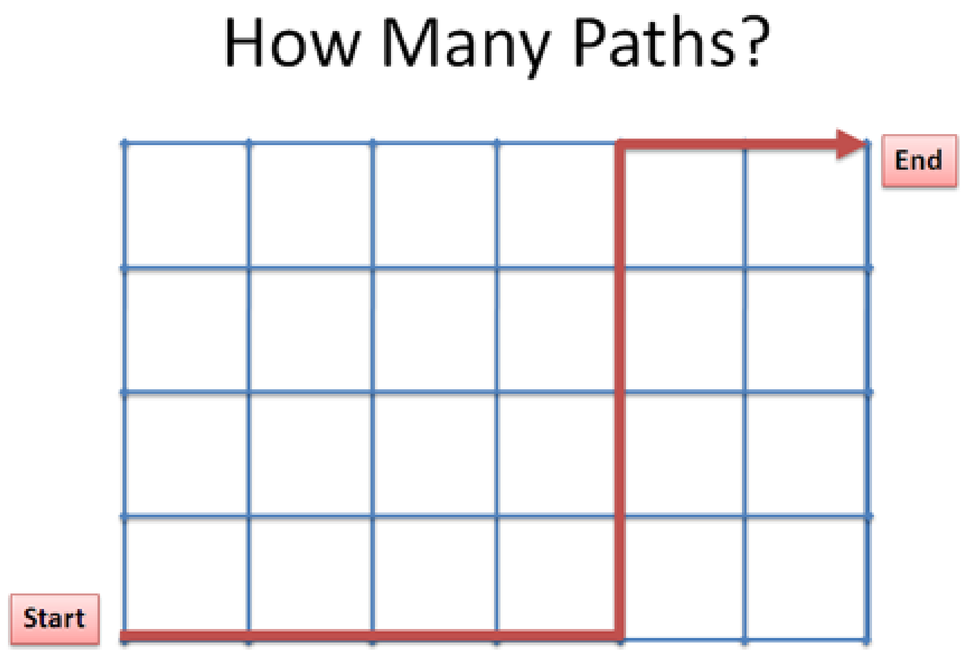
Combinatorics

April 2019

Honguk Woo

Q : Paths of grid

- The number of paths across a grid of (n rows X m columns)
  - e.g. start to end on (4 X 6) grid



Counting Techniques

- Combinatorics is a branch of mathematics concerning the study of finite or countable discrete structures, simply about “Counting”

- Suppose you have 5 shirts and 4 pants in your closet:
- Product rule
  - Different way to wear = Combinations : 5 shirts * 4 pants = total 20
  - $|A| \times |B|$
- Sum rule
  - if any one is missing from the closet, it is one of 9 clothing pieces
  - $|A| + |B|$
- Inclusion-Exclusion Formula
  - $|A \cup B| = |A| + |B| - |A \cap B|$
  - $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |B \cap C| + |A \cap B \cap B|$
  - eliminates double counting

More counting techniques

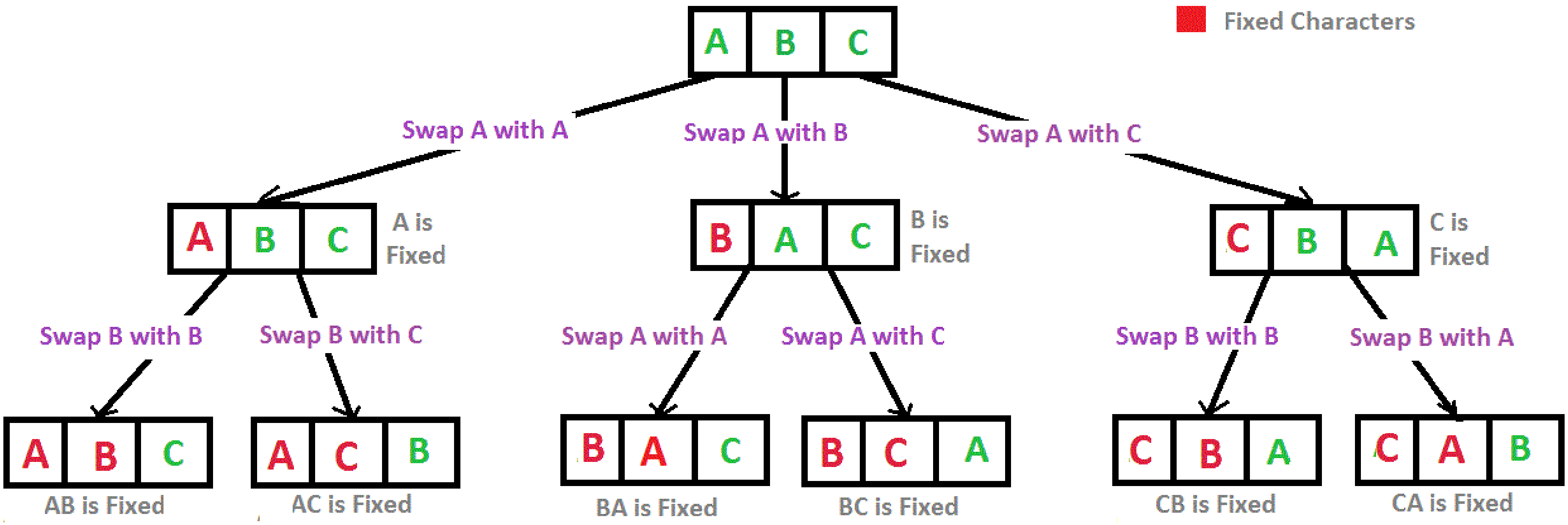
- **(1) Permutation**
  - an **arrangement of *n* items** where every item appears exactly once
    n! = $\prod^{n}_{i=1} i$
  - arrange three letters word using a, b, c
    - abc, acb, bac, bca, cab, cba : 6 words (3!)
    - Very common for exhaustive search methods
    - What about 10! = 3,628,800 …. huge in complexity, approaching to the limits of exhaustive search
  - what if letters are reused ((2) **permutation with repetition**)?
    - aaa, abb, ... on a, b, c: 3X3X3 = 27
    - e.g., r-length **strings** among n characters
      $\mathbf{{}_{n}\prod_{r} = n^{r}}$

- **(3) Subsets**
  - there are 2^n subsets of *n* items
    - a, b, c, ab, bc, ac, abc, and an empty set : 8 subsets

Print all permutations of a given string

- Given a string “abc”, print all the permutations (rearrangements)
- “abc”, 3! = 6
- “abcd”, 4! …
- Many loops ?
- Recursion
  - Base case: each string (the string length == the target length)
  - General case: “a” + permute(“bc”), “b” + permute(“ac”) …

If (base case) then print string
else generate more permutation



Recursion Tree for Permutations of String "ABC"

```
void permute(char *a, int l, int r) {
   if (l == r) printf("%s\n", a);
   else {
       for (i = l; i <= r; i++) {
          swap((a+l), (a+i));
          permute(a, l+1, r);
          swap((a+l), (a+i));
       }
   }
}
```

char str[] = “abc”;
Permute(str, 0, strlen(str) -1);

https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/

Binomial Coefficient

- The number of ways to **choose k things out of n**

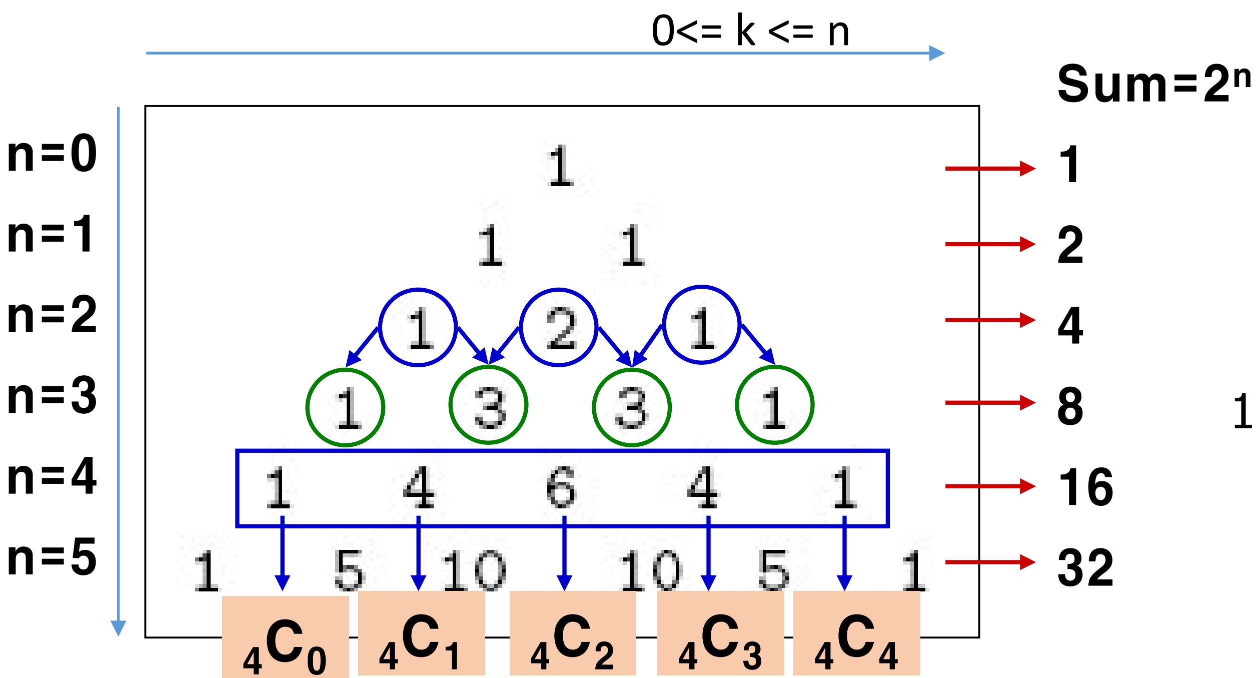
$$_nC_k \equiv \binom{n}{k} \equiv \frac{n!}{(n-k)!\,k!}$$

  - e.g. how many ways to form a k-member committee from n people
  - e.g., how many paths from a grid of k X n-k

Binomial Coefficient (2)

- Coefficient of $(a+b)^n$
  - *$(a + b)^3$ = 1$\boldsymbol{a}^3$ + 3$\boldsymbol{a}^2 b$ + 3ab^2 + 1b^3*
  - *$\boldsymbol{a}^2 b$ → choose 2 a (or 1 b) from all 3, (a+b) X (a+b) X (a+b)*
- what is the coefficient of $a^k b^{n-k}$?
  - how many ways to choose ***k a-terms out of n***
  - $(a + b)^3$ = aaa + 3 aab + 3 abb + bbb

Pascal's Triangle

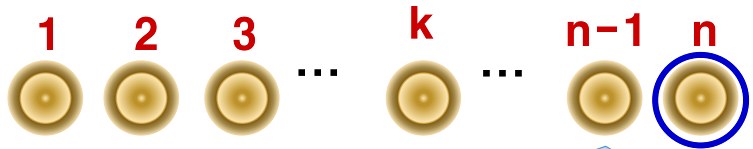


$1a^3 + 3a^2b + 3ab^2 + 1b^3$

$$_nC_k = {}_{(n-1)}C_{(k-1)} + {}_{(n-1)}C_k$$

Binomial Coefficient (3)

Consider : choose k from n items, and two cases about the below n



case1: if this (n) is contained,
same as choosing k-1 from n-1
case2: if this (n) is not contained,
must choose k from n-1

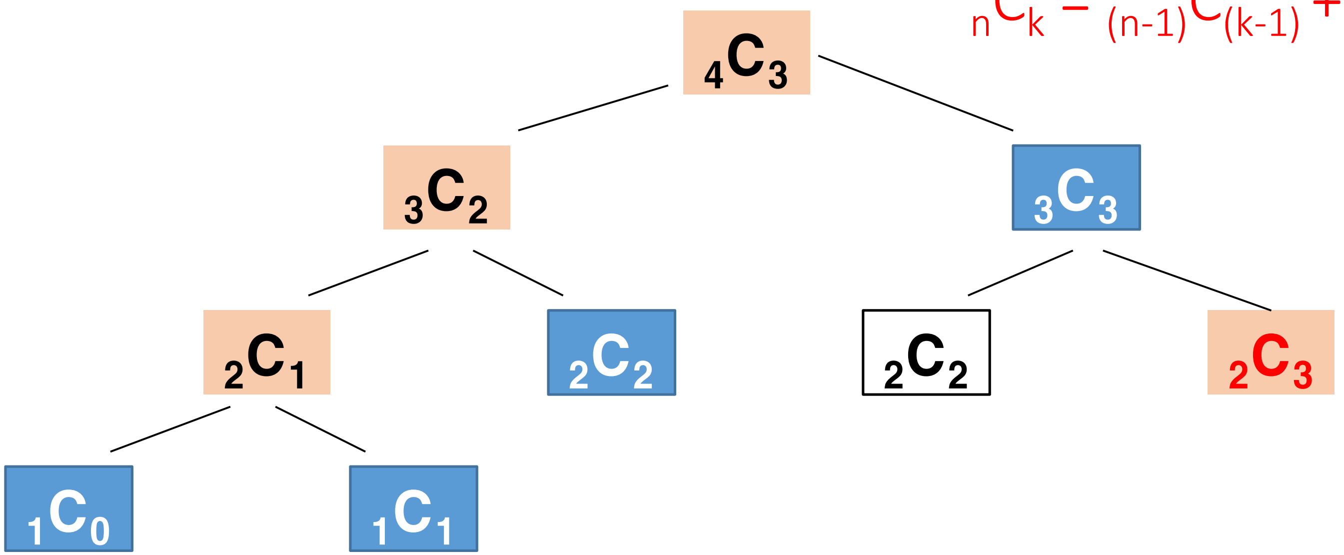
$$ {}_{n}C_{k} = {}_{(n-1)}C_{(k-1)} + {}_{(n-1)}C_{k} $$

- Note that $(n\text{-}k)!k!$ may cause overflow; more stable using recurrence relation.

$$ \begin{bmatrix} n \\ k \end{bmatrix} = \frac{n!}{(n\text{-}k)!k!} $$

Stop the recurrence

$_{n}C_{k} = {}_{(n-1)}C_{(k-1)} + {}_{(n-1)}C_{k}$



- good ending points (base cases)
  - $_{n}C_{0} = 1$
  - $_{n}C_{n} = 1$

Pascal's Triangle in C

recursion

```
int pascal(int r, int c){
  if(c == 0 || c == r) return 1;
  else
    return pascal(r-1, c-1) + pascal(r-1, c);
}

int main(){
  int n = 7;
  for(int i=0; i<n; i++) {
    for(int j=0; j<i+1; j++){
      printf("%d ", pascal(i, j));
    }
    printf("\n");
  }
  return 0;
}
```

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
```

Pascal's Triangle in C

memorization

```
#define MAXN 100  /* largest n or m */
long binomial_coefficient(n,m)    /* computer n choose m */
{
    int i,j; /* counters */
    long bc[MAXN][MAXN]; /* table of binomial coefficients */

    for (i=0; i<=n; i++) bc[i][0] = 1;
    for (j=0; j<=n; j++) bc[j][j] = 1;

     for (i=1; i<=n; i++)
          for (j=1; j<i; j++)
                bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
     return( bc[n][m] );
}
```

Fibonacci Numbers

$F_0 = 0$
$F_1 = 1$
$F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$

```
int fb(int n) {
   if (n == 0 || n == 1) return n;  // f(0) =0,  f(1) = 1
   else return fb(n-1) + fb(n-2);
}
```

Closed form solution

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Eulerian Numbers

- the number of permutations of length n with exactly k ascending sequences
  - E.g. Permutations of the numbers 0 to 9 (length 10) in which exactly 3 elements are greater than the previous element
    - 0, …, 9, k=3
    - 3 1 5 7 6 4 8 2 0

  - case 1: add 9 on solution of 0..8 k=3 w/o changing k
  - case 2: add 9 on solution of 0..8 k=2 increasing k
  - case 3: add 9 on solution of 0..8 k=1 impossible

$$\left\langle {n \atop k} \right\rangle = k\left\langle {n-1 \atop k} \right\rangle + (n-k+1)\left\langle {n-1 \atop k-1} \right\rangle$$

Recurrence Relations

- Recursive relation makes it easy to count “recursively defined structures”
  - Recursively defined structures : tree, list, ...,
    - divide & conquer algorithms : binary search, quick sort, merge sort, …
- Recurrence relation
  - An equation defined in terms of itself

$$a_n = a_{n-1} + 1 \quad , \; a_1 = 1 \quad \rightarrow \quad a_n = n$$

$$a_n = 2a_{n-1} \quad , \; a_1 = 2 \quad \rightarrow \quad a_n = 2^n$$

$$a_n = na_{n-1} \quad , \; a_1 = 1 \quad \rightarrow \quad a_n = n!$$

Mathematical Induction

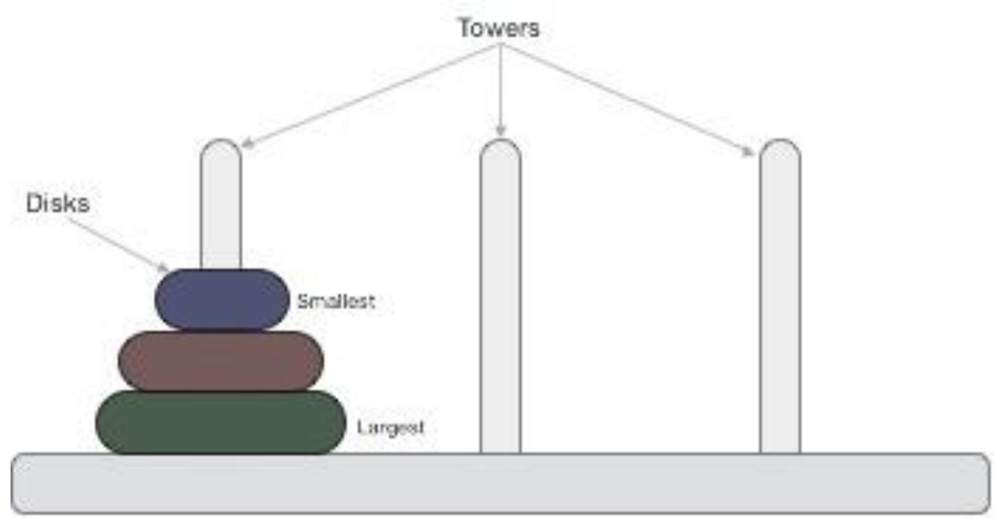
$T_n = 2T_{n-1} + 1, \ T_0=0$

n	0	1	2	3	4	5	6	7	…
T_n	0	1	3	7	15	31	63	127	…

Prove that $T_n=2^n-1$

Q: Tower of Hanoi

- The mission is to move all the disks to some another tower without violating the sequence of arrangement.
  - Only one disk can be moved among the towers at any given time
  - Only the "top" disk can be removed
  - No large disk can sit over a small disk



Tower of Hanoi

- First, we move the smaller (top) disk to aux
- Then, we move the larger (bottom) disk to destination
- And finally, we move the smaller disk from aux to destination

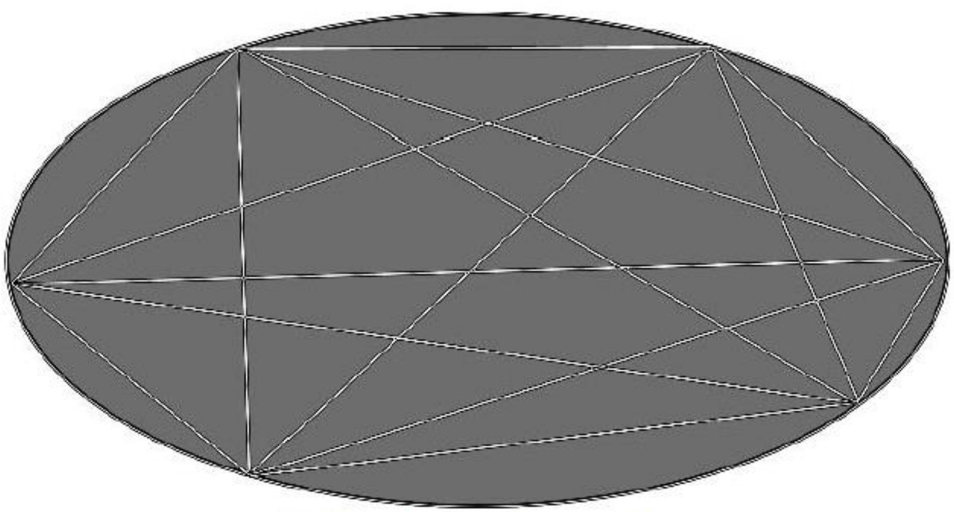
- In general
- **Step 1** − Move n-1 disks from **source** to **aux**
- **Step 2** − Move n^{th} disk from **source** to **dest**
- **Step 3** − Move n-1 disks from **aux** to **dest**

```c
#include <stdio.h>

void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
		if (n == 1)
		{
				printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
				return;
		}
		towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
		printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
		towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}

int main()
{
		int n = 4; // Number of disks
		towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
		return 0;
}
```

Q: How Many Pieces of Land

- You are given a land and you are asked to choose *n* arbitrary points on its boundary. Then you connect each point with every other point using straight lines, forming $n(n-1)/2$ connections.
- What is the maximum number of pieces of land you will get by choosing the points on the boundary carefully?

N (input)	Maximum number of pieces (output)
1	1
2	2
3	4
4	8



Dividing the land when $n = 6$.