Programming in C/C++

Patrick Ho
peiqistar@gmail.com

Algorithm 1

Brute Force

- very general problem-solving technique

- consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement

- complete search

- very slow for large case

Why Brute Force ?

- if resource is large enough, brute force can get best solution. So it's used in test case generation.

- a complicate problem cannot find complete solution in short time. At least get some points in small test cases.

- when problem data range is small

- after getting result, think about speed up if there's time

Speed-Up in Brute Force

- Think about Greed decisions (sorting)
- Think about math formula
  (codercharts: Square Within)
- Avoid loop, recursion calls
- Early terminate condition (break)
- Pre-calculate parts of solution and store it in table
- Go to other algorithm
  (backtracking, dynamic-programming, etc.)

Problem Solving(1)

Selection Sort

Given a set of integers. Sort them from small to big. Print out result. Total number is less then 100

(not qsort/sort)

Input:

20 10 30 40 60 50

Output:

10 20 30 40 50 60

Thinking Algorithm:

1. For each i (0 < i < n-2), find the smallest element in A[i..n-1] and swap it with A[i]:

2. Time: O(n^2)
   Memory: in place
   Stable

3. Coding easy

Problem Solving (2)

PrimeNumber

Write a function to check input positive integer is prime number or not. If it is, return true.

bool IsPrime(int iN);

Thinking Algorithm:

1. Search each integer from 1 to N and check N % I == 0
2. Speed up use N/2 or sqrt

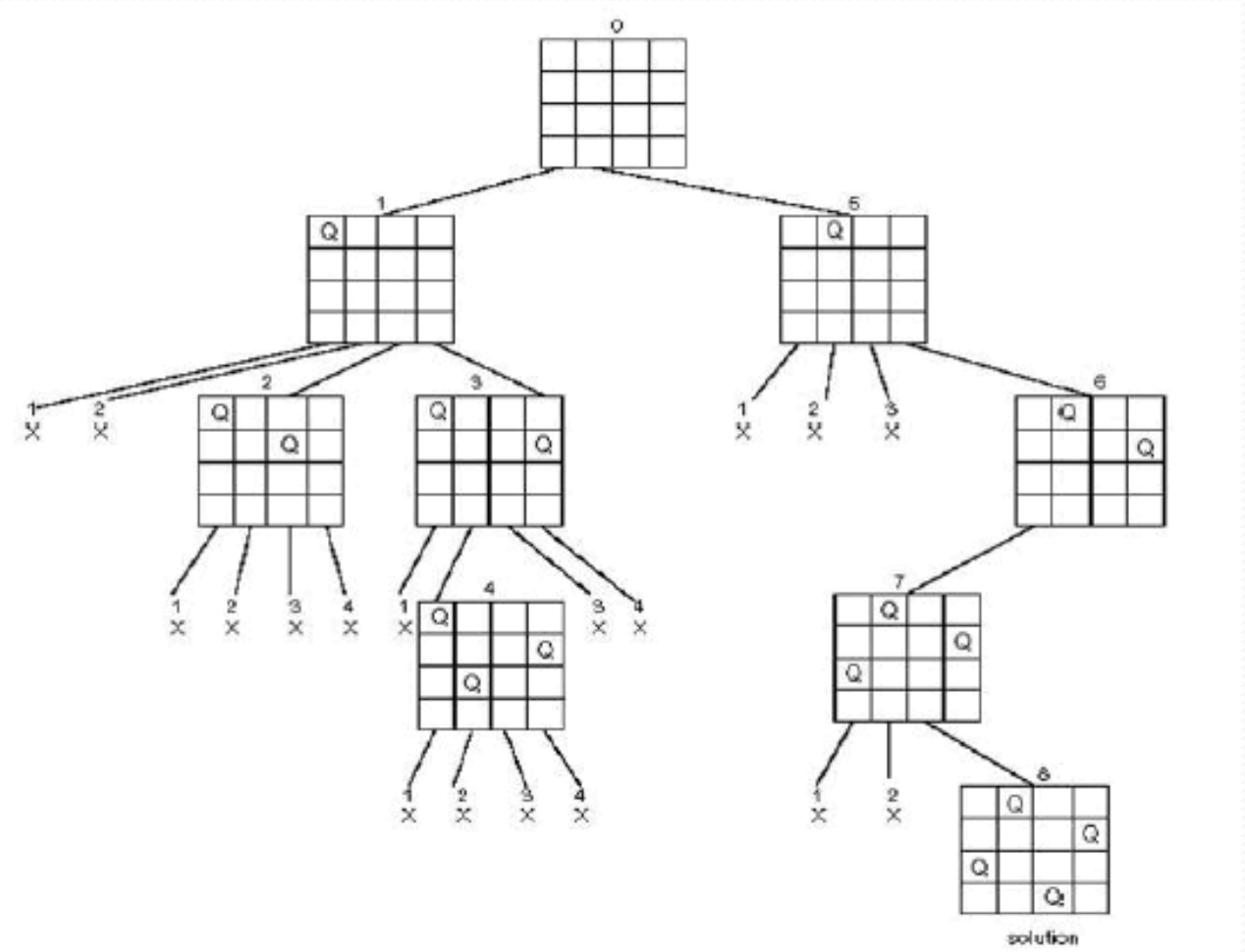
Speed up:

1. Loop to sqrt (prove in math)

```
int iNum=0;
for(int i=2; i*i <= N; i++){
        if (iN % i == 0)
            return false;
}
```

Problem Solving (3)

Four-Queens



Problem Solving (4)

String Match

Given a character string(A) which length is N(N<1000). Search for sub string(B), which length is M(M<=N). Print out it’ s first found position. If not found, print out -1. (string has no space).

Input: 1st line is A, 2nd line is B

THTSATESTSTRING

TEST

Output:

6

Thinking Algorithm:

1. Start search each character from A
2. Do char compare for each char in B
3. O(SizeA * SizeB)

Speed up:

1. Record next first char matched position in sub string match loop, then skip in whole search loop.

Problem Solving (5)

Birthday Cake

You have a cake with radius 100 and 2N (N is a integer, 1<=N<=50) cherries that are placed on the cake.

The center of the cake is Origin(0,0).

You have to cut the cake in 2 parts in a fairly manner that each part has equal cherries and no cherry is on the cutline of division.

The coordinate of cherry is 2 integers(x,y).

The result of a line is in form of 2 integers A,B (stands for Ax+By=0).

The cut line must go through the center.

Input:

2

-20 20

-30 20

-10 -50

10 -5

Output:

0 1

Problem Solving (5)

Thinking Algorithm:

a. Since there are at most 100 cherries and the beeline must go through the center and the radius is 100, it tells (A >= 0 && A <= 100) and (B >= -100 && B <= 100)

b. Each time make a beeline with this A and B (Ax+By=0) that goes through the Origin and check whether there are equal cherries in each side of the beeline and no cherry is on the beeline.
   a. Check if(A*x[i]+B*y[i] > 0) NumA ++;
   b. else if(A*x[i]+B*y[i] < 0) NumB ++;

c. if(NumA == N && NumB == N) print A and B.

Coding:

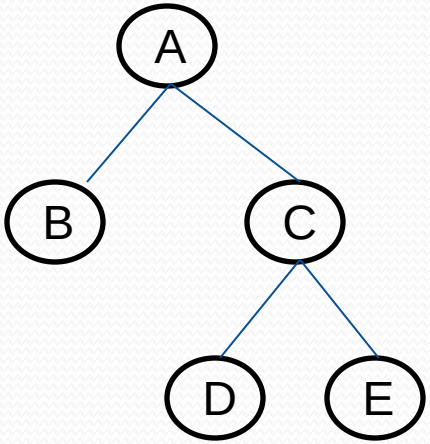
1. array
2. for loop

Tree Structure (Basic 1)

- Non-list data structure
- Abstract model for lots of problems

- Made of node (vertex)

- Node: hold data and link to other nodes
- Root: the most top node (without parent) (level=0) <A>
- Level: the depth to the root <3>
- Edge(branch): the link between 2 nodes
- Parent: node has high level node(s)
- Child: node has low level node
- Sibling: nodes has same parent <D,E>
- Leaf: node without child <B, D, E>
- Sub-Tree: internal node and its children

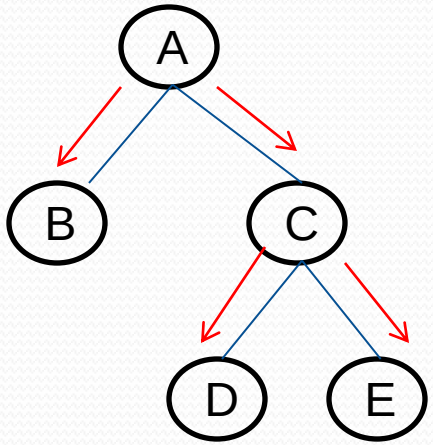


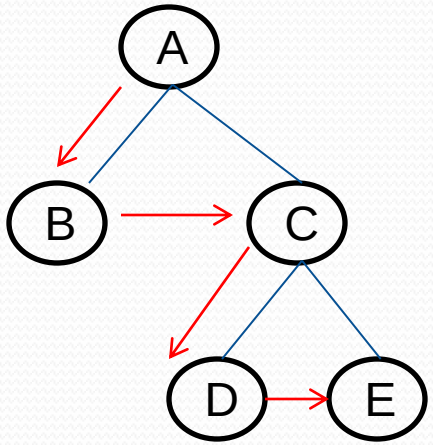
Tree Structure (Basic 2)

- Search/Traverse/Visit a Tree
  - Depth-First Search (DFS)
    starts at the root and explores as far as possible along each branch before backtracking.

  - Breadth-First Search (BFS)
    begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on.

- Implement/Coding use Recursion





Backtracking Algorithm

- finding all (or some) solutions to some computational problem, that incrementally builds candidates to the solutions, and abandons each partial candidate c("backtracks") as soon as it determines that c cannot possibly be completed to a valid solution.

- much faster than brute force enumeration of all complete candidates, since it can eliminate a large number of candidates with a single test.

- Pseudo Code

```
backtrack(node c) {
    if reject(c) then return
    if accept(c) then output(c)
    for (each child u of c)
            backtrack(u);
    reset_state( c );
}
```

Problem Solving (1)

Permutation

Give a set of integers (>0). Print out each permutation set

Example

Solution:

1. Read in data to vector

   Call BTWork(0); // start from array index 0

2. void BTWork (i) function

   if I == iN, print out result and return

   for (each number in array) {

   if it is 0, has been visited, continue;

   save the number (s)for backtracking

   save the number to result

   set array value to 0, as visited

   call BTWork(n), recursion

   set back the s value into array

   erase current result number

   }

Problem Solving (1)

```
int iN;
vector<int> vIn;
vector<int> vRes;
int iPermN=0;

void BTWork(int i) {
    if (vRes.size() == iN) {
        for(int k=0; k<vRes.size(); k++) {
            cout << vRes[k];
            if (k==vIn.size()-1) cout << endl;
            else
                cout << ',';
        }
        iPermN++;  return;
    }
    for(int n=0; n<vIn.size(); n++) {
        if(vIn[n] == 0)                continue;
        int s=vIn[n];
        vRes.push_back(s); vIn[n] = 0;
        BTWork(n+1);
        vIn[n] = s; vRes.pop_back();
    }
}
```

```
int main(int argc, char** argv)
{
    // input N positive integer
    // output Permuation number list
    cin >> iN;
    for(int i=0; i<iN; i++)
    {
        int iD;
        cin >> iD;
        vIn.push_back(iD);
    }

    BTWork(0);
}
```

Problem Solving (2)

KNIGHT

There’ s chess board(size<15). One pawn want to move from A(0,0) to B(n,m). One enemy knight is at C (never move, but can attack pawn). The pawn can only move up or right one step a time. Please count how many ways that pawn can go from A to B.

Input: 4 integers of B and C coordinates

e.g. 6 6 3 3

Output: one integer of ways

e.g. 6

Solution:

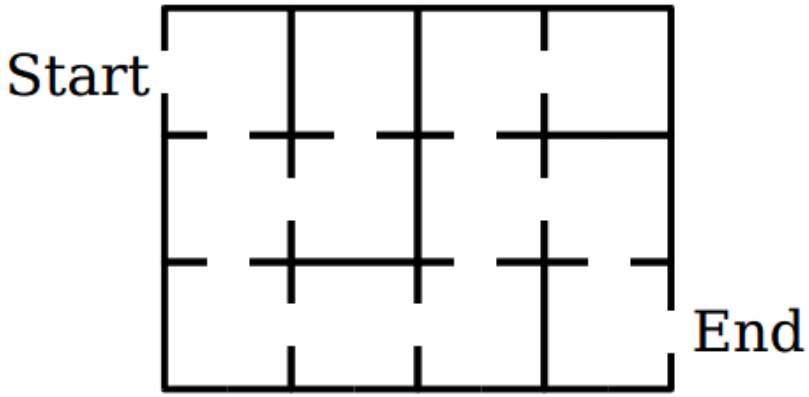
1. Since board size is small, use DFS
2. Use global variable

   It can be used in any function.
3. Set knight possible attack location as block
4. Chess board usually use 2D array to record status, blockage, etc.
5. Think about how to print out each path

Problem Solving (3)

MAZE

Find a path from Start point to End point



Solution:

1. Reading maze map

```
Data Structure maze {
        bool bVisited;
        bool bWall[4];  // true, not  blocked by a wall}
```

2. bool BT (i, j) function

```
if  i,j is outside the maze, return true (solution  is found).
if i,j  bVisited, return false (this path point is tried).
set  i,j  bVisited to true
for (4 wall directions) {
   if (true, not  blocked by a wall) {
      Move one step (i+1 or j+1)
      if  BT(…) is true, return }
}
Reset i,j bVisited to false
Return false
```

Problem Solving (4)

N-Queen Problem

```
#include <cstdio>
#include <cmath>

int n,x[30];
int solution(int k) {
  return k==n;
}

void print(int k) {
  for (int i=1;i<k+1;i++)
      printf("%d ",x[i]);
  printf("\n");
}

int possible(int k) {
  for (int i=1;i<k;i++)
      if (x[i]==x[k] || abs(x[i]-x[k])==k-i)
          return 0;
  return 1;
}
```

```
void back(int k)
{
  if (solution(k))
       print(k);
  else
       for (x[k+1]=1; x[k+1]<=n; x[k+1]++)
              if (possible(k+1))
                back(k+1);
}

void main()
{
  printf("Please enter the size of the NxN
      chessboard: "); scanf("%d",&n);
  printf("\nThe solution: ");
  back(0);
}
```